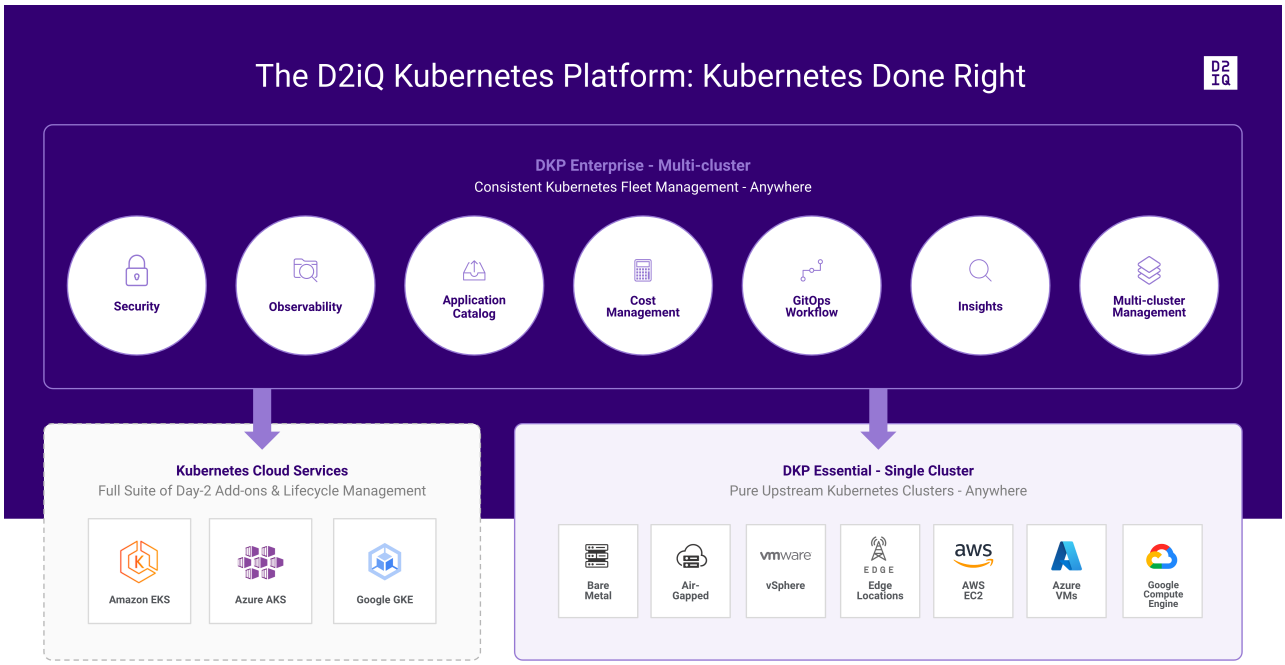
Introducing the D2iQ Kubernetes Platform (DKP)

D2iQ Kubernetes Platform (DKP)

Exported on 09/21/2023

Table of Contents

As the leading independent Kubernetes Management Platform in production, the D2iQ Kubernetes Platform (DKP) provides a holistic approach and a complete set of enterprise-grade technologies, services, training, and support to build and run applications in production at scale. Built around the open-source Cluster API, the new version of DKP becomes the single, centralized point of control for an organization's application infrastructure, empowering organizations to more easily deploy, manage, and scale Kubernetes workloads in Day 2 production environments.



Features	Benefits
Container Orchestration	Leverage an industry standard distribution of open-source Kubernetes for cluster and container management.
Application Management and Deployment	Deploy applications and services within Kubernetes clusters with Helm[1].
Observability	Gain deep insight into your Kubernetes clusters and applications with open source metrics leveraging Telegraf, Prometheus, and Grafana[2].

1 https://helm.sh/
2 https://grafana.com/docs/

Features	Benefits
Cluster API	Automate cluster lifecycle management using Cluster API (see page 94) to simplify the provisioning, upgrading, and operating multiple Kubernetes clusters across a wide range of distributions and virtual and physical environments.
Cluster Autoscaling	Save operational costs by automatically scaling down (see page 997) capacity when it's not needed and adding capability when there is greater demand, with CAPI enabled autoscaling groups.
Logging	Collect and analyze logs and metrics (see page 796) to ensure optimal performance and troubleshooting with Grafana (see page 877), Loki, and Fluentbit (see page 818).
Cloud-Native Scale Testing	Extensive integration and workload testing at massive scale with a wide range of workloads to ensure real-world preparedness.
Networking and Routing	Easily automate and expose application endpoints (see page 844) with Calico, Traefik (see page 841), and CoreDNS.
Fine-Grained Cluster	Upgrades (see page 1365) reduce operational overhead with non-disruptive patching or parallel worker node upgrades.
Backup and Recovery	Ensure business continuity and disaster recovery (see page 772) with Velero[3].
Declarative Automated Installer With Day 2 Platform Services	Accelerate time-to-production on any infrastructure (see page 124) with consistency and reliability with the required platform applications (see page 564) needed for Day 2 production (see page 510).
Operate in Air-gapped Environments (see page 274)	Leverage declarative APIs to optimize cluster resources for cost, resilience, and performance.

3 https://velero.io/docs/main/

Features	Benefits
End-to-End Support	Enterprise-grade support[4] and services for both Kubernetes and its supported platform applications.

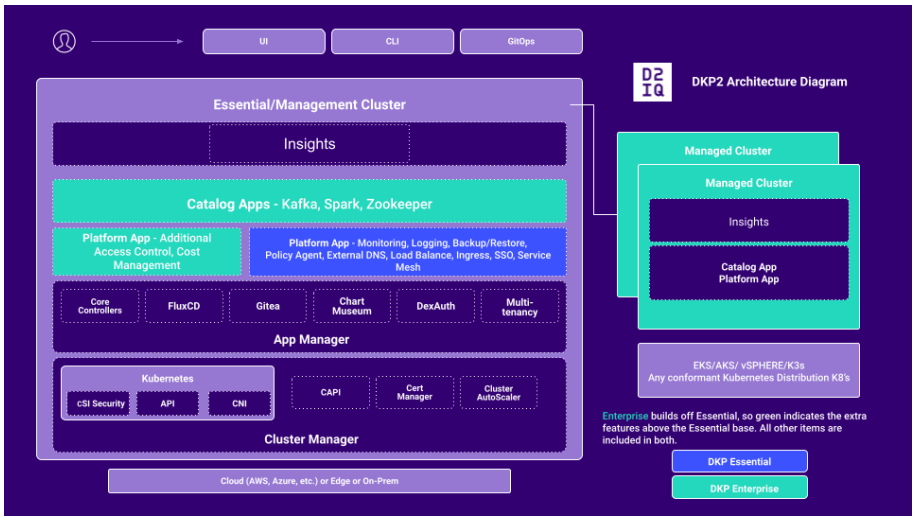
4 https://support.d2iq.com/hc/en-us

1 Architecture

1.1 Learn the key concepts and architectural components of a DKP cluster

Kubernetes provides the foundation for a DKP cluster. Because of this fundamental relationship, you should be familiar with a few key concepts and terms. The topic in this section provide a brief overview of the native Kubernetes architecture, a simplified view of the DKP architecture - both Essential and Enterprise versions. Also, it shows the operational workflow for a DKP cluster.

The following diagram provides a simplified architectural overview to help you visualize the flow of the key components:



DKP Architectural overview

1.2 Components for the Kubernetes Control Plane

The native Kubernetes cluster consists of **components** that provide the cluster's **control plane** and **worker nodes** that run users' containers and maintain the runtime environment.

DKP supplements the native Kubernetes cluster by providing a predefined and pre-configured set of applications. Because this predefined set of applications provides critical features for managing a Kubernetes cluster in a production environment, the default set is identified as DKP **platform services**.

See Platform applications (see page 510) for the full set of DKP platform services.

1.3 Related Information

For information on related topics or procedures, refer to the following Kubernetes[5] documentation.

1.4 Next Steps

- DKP Ports (see page 60)
- Supported Infrastructure Operating Systems (see page 62)

1.5 DKP Ports

This section describes ports used by the different Kubernetes components in your DKP cluster. For more information regarding ports, refer to the Kubernetes Documentation on this page: Ports and Protocols[6]

1.5.1 Control-plane nodes

Port	DKP Component	Notes
22		ssh
179	calico-node	BGP
1338	Containerd	metrics
2379	etcd	client
2380	etcd	peer
6443	kube-apiserver	
9091	calico-node	felix metrics
9092	calico-node	bird metrics

5 https://kubernetes.io/docs/concepts/overview/components/
6 https://kubernetes.io/docs/reference/networking/ports-and-protocols/

Port	DKP Component	Notes
9099	calico-node	felix liveliness
9100	prometheus node-exporter	metrics
10248	kubelet	health
10249	kube-proxy	metrics
10250	kubelet	
10256	kube-proxy	health
10257	kube-controller-manager	secure port
10259	kube-scheduler	secure port
30000-32767	Kubernetes NodePorts	

1.5.2 Worker nodes

Port	DKP Component	Notes
22	Ansible	ssh
179	calico-node	BGP
1338	Containerd	metrics
5473	calico-typha	syncserver
9091	calico-node	felix metrics
9092	calico-node	bird metrics

Port	DKP Component	Notes
9093	calico-typha	metrics
9099	calico-node	felix liveliness
9100	prometheus node-exporter	metrics
9400	NVIDIA GPU DCGM	metrics
10248	kubelet	health
10249	kube-proxy	metrics
10250	kubelet	
10256	kube-proxy	health
30000-32767	Kubernetes NodePorts	

1.6 Supported Infrastructure Operating Systems

DKP supports the following base Operating Systems in DKP 2.5.0.

If the full table is not showing below, select the box icon in the upper-right corner to expand the view:

1.6.1 Amazon Web Services (AWS)

Operating System	Kernel	Default Config (see page 262)	FIPS (see page 289)	Air Gapped (see page 274)	FIPS with Air Gapped (see page 302)	GPU Support (see page 316)	GPU with Air Gapped (see page 332)	Konvoy Image Builder (see page 1282)
CentOS 7.9[7]	3.10.0-1160.el7.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 7.9[8]	3.10.0-1160.80.1.el7.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 8.4[9]	4.18.0-305.12.1.el8_4.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 8.6[10]	4.18.0-372.9.1.el8.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Ubuntu 18.04 (Bionic Beaver)[11]	5.4.0-1088-aws	Yes						Yes
Ubuntu 20.04	5.15.0-1022-aws	Yes				Yes		Yes

7 https://wiki.centos.org/action/show/Manuals/ReleaseNotes/CentOS7.2003
8 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/7.9_release_notes/index
9 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/8.4_release_notes/index
10 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/8.6_release_notes/index
11 https://wiki.ubuntu.com/BionicBeaver/ReleaseNotes

Operating System	Kernel	Default Config (see page 262)	FIPS (see page 289)	Air Gapped (see page 274)	FIPS with Air Gapped (see page 302)	GPU Support (see page 316)	GPU with Air Gapped (see page 332)	Konvoy Image Builder (see page 1282)
(Focal Fossa)[12]								
SLES 15 SP3[13]	5.3.18-150300.59.63-default	Yes				Yes		Yes
Oracle Linux RHCK 7.9[14]	3.10.0-1160.80.1.0.1.el7.x86_64	Yes	Yes					Yes
Rocky Linux 9.1[15]	5.14.0-162.12.1.el9_1.0.2.x86_64	Yes		Yes				Yes

1.6.2 Microsoft Azure

Operating System	Kernel	Default Config (see page 465)	FIPS	Air Gapped	FIPS with Air Gapped	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1295)
Ubuntu 20.04	5.15.0-1034-azure	Yes						Yes

12 https://wiki.ubuntu.com/FocalFossa/ReleaseNotes
13 https://documentation.suse.com/en-us/sles/15-SP3/
14 https://docs.oracle.com/en/operating-systems/oracle-linux/7/relnotes7.9/
15 https://docs.rockylinux.org/release_notes/9_1/

Operating System	Kernel	Default Config (see page 465)	FIPS	Air Gapped	FIPS with Air Gapped	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1295)
(Focal Fossa)[16]								
Rocky Linux 9.1[17]	5.14.0-162.12.1.el9_1.0.2.x86_64	Yes						Yes

1.6.3 Google Cloud Platform (GCP)

Operating System	Kernel	Default Config (see page 494)	FIPS	Air Gapped	FIPS with Air Gapped	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1299)
CentOS 7.9[18]	3.10.0-1160.62.1.el7.x86_64	Yes						Yes
Ubuntu 18.04[19]	5.4.0-1072-gcp	Yes						Yes
Ubuntu 20.04[20]	5.13.0-1024-gcp	Yes						Yes

16 https://wiki.ubuntu.com/FocalFossa/ReleaseNotes
17 https://docs.rockylinux.org/release_notes/9_0/
18 https://wiki.centos.org/action/show/Manuals/ReleaseNotes/CentOS7.2003
19 https://wiki.ubuntu.com/BionicBeaver/ReleaseNotes
20 https://wiki.ubuntu.com/FocalFossa/ReleaseNotes

1.6.4 Pre-Provisioned

Pre-provisioned (see page 125) includes on-premises, vSphere, AWS, Azure, and GCP infrastructures and is described in more detail under What is Pre-provisioned Infrastructure (see page 96).

Operating System	Kernel	Default Config (see page 127)	FIPS (see page 167)	Air Gapped (see page 145)	FIPS with Air-Gapped (see page 186)	GPU Support (see page 209)	GPU with Air Gapped (see page 231)	Konvoy Image Builder (see page 1312)
CentOS 7.9[21]	3.10.0-1160.el7.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 7.9[22]	3.10.0-1160.66.1.el7.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 8.4[23]	4.18.0-305.12.1.el8_4.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 8.6[24]	4.18.0-372.9.1.el8.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Flatcar 3033.3.[25] x	5.10.107-flatcar 5.10.154-flatcar	Yes						Yes

21 https://wiki.centos.org/action/show/Manuals/ReleaseNotes/CentOS7.2003
22 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/7.9_release_notes/index
23 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/8.4_release_notes/index
24 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/8.6_release_notes/index
25 https://www.flatcar.org/releases/#lts-release

Operating System	**Kernel**	Default Config (see page 127)	FIPS (see page 167)	Air Gapped (see page 145)	FIPS with Air-Gapped (see page 186)	GPU Support (see page 209)	GPU with Air Gapped (see page 231)	Konvoy Image Builder (see page 1312)
Ubuntu 20.04[26]	5.15.0-1020-aws	Yes				Yes		Yes
SLES 15 SP3[27]	5.3.18-150300.59.63-default	Yes				Yes		Yes
Oracle Linux RHCK 7.9[28]	5.4.17-2011.6.2.el7uek.x86_64	Yes						Yes
Rocky Linux 9.1[29]	5.14.0-162.6.1.el9_1.x86_64	Yes		Yes				Yes

26 https://wiki.ubuntu.com/FocalFossa/ReleaseNotes
27 https://documentation.suse.com/en-us/sles/15-SP3/
28 https://docs.oracle.com/en/operating-systems/oracle-linux/7/relnotes7.9/
29 https://docs.rockylinux.org/release_notes/9_1/

1.6.5 Pre-provisioned/Azure

Operating System	Kernel	Default Config (see page 1078)	FIPS (see page 167)	Air Gapped (see page 145)	FIPS with Air Gapped (see page 186)	GPU Support[30]	GPU with Air Gapped[31]	Konvoy Image Builder (see page 1282)
RHEL 8.6[32]	4.18.0-372.41.1.el8_6.x86_64	Yes	Yes	Yes	Yes			Yes

1.6.6 vSphere

Operating System	Kernel	Default Config (see page 383)	FIPS (see page 424)	Air Gapped (see page 402)	FIPS with Air Gapped (see page 443)	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1305)
RHEL 7.9[33]	3.10.0-1160.el7.x86_64	Yes	Yes	Yes	Yes			Yes
RHEL 8.4[34]	4.18.0-305.el8.x86_64	Yes	Yes	Yes	Yes			Yes

30 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273387183/2.6+Pre-provisioned+with+GPU+Install
31 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273319903/2.6+Pre-provisioned+Air-gapped+with+GPU+Install
32 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/8.6_release_notes/index
33 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/7.9_release_notes/index
34 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/8.4_release_notes/index

Operating System	Kernel	Default Config (see page 383)	FIPS (see page 424)	Air Gapped (see page 402)	FIPS with Air Gapped (see page 443)	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1305)
RHEL 8.6[35]	4.18.0-372.9.1.el8.x86_64	Yes	Yes	Yes	Yes			Yes
Ubuntu 20.04[36]	5.4.0-125-generic	Yes		Yes				Yes
Rocky Linux 9.1[37]	5.14.0-162.6.1.el9_1.x86_64	Yes		Yes				Yes

1.6.7 Amazon Elastic Kubernetes (EKS)

Operating System	Kernel	Default Config (see page 351)	FIPS	Air Gapped	FIPS with Air Gapped	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1294)
Amazon Linux 2 v7.9[38]	5.10.178-162.673.amzn2.x86_64	Yes						

35 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/8.6_release_notes/index
36 https://wiki.ubuntu.com/FocalFossa/ReleaseNotes
37 https://docs.rockylinux.org/release_notes/9_1/
38 https://docs.aws.amazon.com/AL2/latest/relnotes/relnotes-al2.html

1.6.8 Azure Kubernetes Service (AKS)

Operating System	Kernel	Default Config (see page 478)	FIPS	Air Gapped	FIPS with Air Gapped	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1299)
Ubuntu 22.04.2 LTS[39]	5.15.0-1037-azure	Yes						

39 https://discourse.ubuntu.com/t/jammy-jellyfish-release-notes/24668

2 Download DKP

To download a new version of DKP, you have 2 options:

2.1 Download from the Support Website

Download DKP

From the Download site, select the DKP binary for either Darwin(MacOS) or Linux. After download, extract for your system.

To extract a tar file, either:

- Darwin - Use a file manager program like 7-Zip and right-click the tar file. Select “Open With” and then 7-Zip File Manager. Select “Extract” and choose a location to save the extracted files.
- Linux - Use the command line and type `tar -xvf filename.tar` to extract the files to the current directory. You can add tar `-xvzf` option if the file is compressed with gzip.

2.1.1 AWS Marketplace Only Download

Follow the instructions on the AWS console to download the container image.

After downloading the image, run this command to copy the binaries to your host:

```
docker run -it --rm -u $(id -u):$(id -g) -v $(pwd):/dkp $CONTAINER_IMAGES
```

A successful operation returns this output:

```
dkp binary is placed in the local directory, to run:
./dkp --help
```

And you can view the `dkp` binary in your working directory. Follow the DKP installation (see page 117) instructions using these binaries, and then add your license (see page 85) to DKP. If you have problems downloading or installing DKP, contact your sales representative or sales@d2iq.com[40].

40 mailto:sales@d2iq.com

3 Licenses

This section contains overviews of the feature sets for all the available DKP licenses and how to acquire and use your license.

- *DKP Essential* (see page 78) supports your single-cluster environment, or a single Management cluster.
- *DKP Enterprise* (see page 75) supports a multi-cluster environment, with a Management Cluster and one or more Attached or Managed Clusters.
- *DKP Government Essential* (see page 83) consists of DKP Essential with Confirmed State Side Support (CSS)
- *DKP Government Advanced* (see page 80) consists of DKP Enterprise with Confirmed State Side Support (CSS)

All DKP Enterprise and DKP Government Advanced features are marked with the following badges:

Enterprise Gov Advanced

The type of license you purchase determines how much of DKP you can use. Some features that are compatible with all versions of DKP can be activated by purchasing additional Add-on licenses. The following table shows the differences between the different DKP licenses:

Feature	DKP Essential	DKP Enterprise	DKP Gov Essential	DKP Gov Advanced
Applications				
Workspace Platform Applications	✅	✅	✅	✅
Project Platform Applications (see page 611)		✅		✅
Catalog Applications (Kafka, Zookeeper, Spark)		✅		✅
Custom Applications		✅		✅
DKP Insights[41]				

41 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/213028009/%282.5%29+DKP+Insights

Feature	DKP Essential	DKP Enterprise	DKP Gov Essential	DKP Gov Advanced
Cluster Management				
Single Cluster	✅	✅	✅	✅
Multiple Clusters		✅		✅
Attaching Cluster		✅		✅
Provisioning an additional cluster		✅		✅
Upgrades (see page 1365)	✅	✅	✅	✅
GitOps				
Continuous Deployment (FluxCD)		✅		✅
UX				
DKP CLI (see page 1436)	✅	✅	✅	✅
DKP UI	✅	✅	✅	✅
Workspaces		✅		✅
Projects		✅		✅
Logging				

Feature	DKP Essential	DKP Enterprise	DKP Gov Essential	DKP Gov Advanced
Workspace Level Logging	✅	✅	✅	✅
Multi-tenant Logging		✅		✅
Monitoring	✅	✅	✅	✅
Backup & Restore (see page 772)	✅	✅	✅	✅
GPU	✅	✅	✅	✅
Security				
Authentication using Dex	✅	✅	✅	✅
Policy control using Gatekeeper	✅	✅	✅	✅
Cluster Provisioning				
DKP on AWS (see page 261)	✅	✅	✅	✅
DKP on Azure (see page 465)	✅	✅	✅	✅
DKP on GCP (see page 493)	✅	✅	✅	✅
DKP on vSphere (see page 378)	✅	✅	✅	✅
Pre-provisioned (see page 125)	✅	✅	✅	✅
On-Prem (see page 125)	✅	✅	✅	✅

Feature	DKP Essential	DKP Enterprise	DKP Gov Essential	DKP Gov Advanced
EKS Provisioning (see page 350)		✅		✅
AKS Provisioning (see page 478)		✅		✅
FIPS Compliance	✅	✅	✅	✅
Konvoy Image Builder (see page 1282)	✅	✅	✅	✅
Air-Gapped Deployments	✅	✅	✅	✅
Confirmed Stateside Support (CSS)			✅	✅

3.1 Buy a License

DKP Licenses are sold in units of cores. To learn more about both, and to obtain a valid license:

- Contact your sales representative at sales@d2iq.com[42] to buy one or more licenses.
- Purchase a license via the AWS Marketplace[43]. After buying your license, the information (ARN) is accessible in the AWS License Manager console.

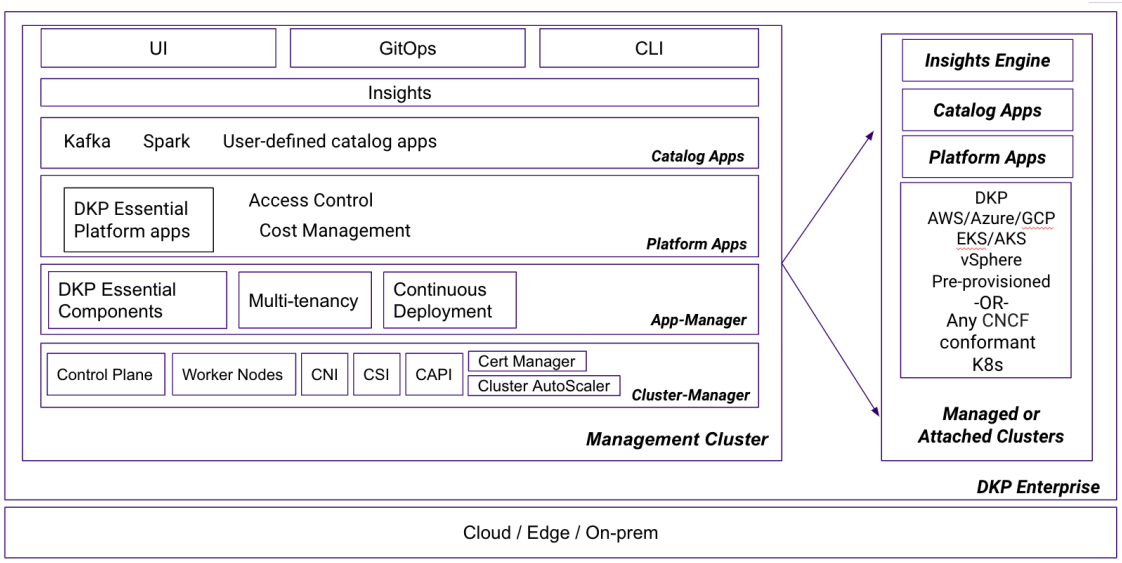
After purchasing, download the binary files (see page 71).

3.2 DKP Enterprise

Enterprise Gov Advanced

Features and capabilities that are specific to DKP Enterprise are marked with this badge at the top of each page. These items are NOT available to DKP Essential customers.

42 mailto:sales@d2iq.com
43 https://aws.amazon.com/marketplace/



DKP Enterprise is a multi-cluster lifecycle management Kubernetes solution centered around a management cluster that manages multiple attached or managed Kubernetes clusters via a centralized management dashboard. The management dashboard gives you a single point of observability and control throughout all of your attached or managed clusters. The DKP Enterprise license gives the user access to the entire Konvoy cluster environment, the DKP UI dashboard that deploys platform and catalog applications, and multi-cluster management, and comprehensive compatibility with our full range of infrastructure deployment options.

3.2.1 Compatible Infrastructure

DKP Enterprise operates across D2iQ's entire range of cloud, on-premises, edge, and air-gapped infrastructures and has support for various OSs, including immutable OSs. See Supported Operating Systems (see page 62) for a full list of compatible infrastructure.

For the basics on standing up a DKP Enterprise cluster in one of the listed environments of your choice, see Advanced Configuration (see page 1282).

3.2.2 Platform Applications

Applications can be deployed in any DKP managed cluster, giving you complete flexibility to operate across cloud, on-premises, edge, and air-gapped scenarios. Customers can also use the UI with Kommander to customize which platform applications to deploy to the cluster in a given workspace.

3.2.3 Catalog Applications

Quickly and easily deploy applications and complex data services from a centralized service catalog to specific or multiple clusters, with governance. Fast data pipelines can be provisioned automatically from the following catalog of DKP Applications:

- **Kafka:** Primarily used to build real-time streaming data pipelines and applications that adapt to the data streams. It combines messaging, storage, and stream processing to allow storage and analysis of both historical and real-time data.

- **Spark:** An industry standard analytics engine for big data processing and machine learning. Spark enables you to process data for both batch and streaming workloads.
- **Zookeeper:** A centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

For instructions on how to deploy catalog applications, see Workspace Catalog Applications (see page 586)

3.2.4 Cluster Manager

Konvoy is the Kubernetes installer component of DKP Enterprise that uses industry standard tools to produce a certified Kubernetes cluster. These industry standard tools create a cluster management system that includes:

- **Control Plane:** Manages the worker nodes and pods in the cluster.
- **Worker Nodes:** Used to run containerized applications and handle networking to ensure that traffic between applications across the cluster and from outside of the cluster can be properly facilitated.
- **Container Networking Interface (CNI):** Calico's open source networking and network security solution for containers, virtual machines, and native host-based workloads.
- **Container Storage Interface (CSI):** A common abstraction to container orchestrators for interacting with storage subsystems of various types.
- **Kubernetes Cluster API (CAPI):** Cluster API uses Kubernetes-style APIs and patterns to automate cluster lifecycle management for platform operators. For more on how CAPI is integrated into DKP Enterprise, see Understanding CAPI Concepts and Terms (see page 94)
- **Cert Manager:** A Kubernetes addon to automate the management and issuance of TLS certificates from various issuing sources.
- **Cluster Autoscaler:** A component that automatically adjusts the size of a Kubernetes cluster so that all pods have a place to run and there are no unneeded nodes.

3.2.5 Built-in GitOps

DKP Enterprise comes bundled with GitOps, an operating model for Kubernetes and other cloud native technologies, providing a set of best practices that unify Git deployment, management and monitoring for containerized clusters and applications. GitOps works by using Git as a single source of truth for declarative infrastructure and applications. With GitOps, the use of software agents can alert on any divergence between Git with what is running in a cluster, and if there's a difference, Kubernetes reconcilers automatically update or rollback the cluster depending on the case.

3.2.6 DKP Enterprise Multi-cluster UI

Bundled with DKP Enterprise is a multi-cluster management view in the UI with Kommander that can be used in lieu of the bundled CLI. From the UI you can:

- **Connect to an infrastructure provider:** DKP Enterprise supports on-premises and cloud infrastructure providers such as AWS and Azure for your Konvoy clusters. To automate their provisioning, DKP requires authentication keys to your preferred infrastructure provider entered on the Add Infrastructure Provider form.

- **Setup identity providers:** DKP Enterprise supports GitHub, LDAP, SAML, and standard OIDC identity providers such as Google. These identity management providers support the login and authentication process for your Kubernetes cluster. See Identity Providers (see page 524) for more information.
- **Configure access control:** Role-based authorization control (RBAC) is central to DKP Enterprise and controls access to resources on all connected clusters. The resources are similar to Kubernetes RBAC. You add an identity provider group, and in that group add cluster roles and cluster role bindings for those roles.
- **Deploy applications:** The DKP Enterprise UI allows you to customize your workspace application deployments via the Applications page within the UI.
- **Create a project:** Create projects within a workspace and deploy project-scoped applications. Projects enable teams to deploy configurations and services to their clusters consistently. After configuring roles, ConfigMaps, secrets, and applications for a project, DKP distributes this configuration to each project namespace. For more information concerning projects, see Projects (see page 608).
- **Add a license:** To add a license via the UI with Kommander, see Add a License (see page 85)
- **Kubernetes cost monitoring and management**: The kubecost platform application provides real-time cost visibility and insights for external cloud services such as AWS, helping you continuously reduce your cloud costs.

For more information concerning the global and workspace-level UI with Kommander, see Workspaces (see page 573)

3.3 DKP Essential

DKP Essential is a self-managed single cluster Kubernetes solution that gives you a feature-rich, easy-to-deploy, and easy-to-manage entry-level cloud container platform. The DKP Essential license gives the user access to the entire Konvoy cluster environment, and to the Kommander platform application manager.

> Features and capabilities that are specific to DKP Enterprise are marked with this badge at the top of each page. These items are NOT available to DKP Essential users.

3.3.1 Compatible Infrastructure

DKP Essential operates across a range of cloud, on-premise, edge, and air-gapped infrastructures and has support for various OSs, including immutable OSs. See Supported Operating Systems (see page 62) for a full list of compatible infrastructure.

For the basics on standing up a DKP Essential cluster in one of the listed environments of your choice, see Advanced Configuration (see page 1282).

3.3.2 Platform Applications

When creating a cluster, the application manager deploys certain platform applications on the newly created cluster. DKP Essential users can use the Kommander UI to customize which platform applications to deploy to the cluster in a given workspace. For a list of available platform applications that are included with DKP, see Workspace Platform Applications (see page 114).

> **NOTE:** The platform application `kubecost` is not included with DKP Essential, but is included with DKP Enterprise.

3.3.3 Cluster Manager

Konvoy is the Kubernetes installer component of DKP Essential that uses industry standard tools to produce a certified Kubernetes cluster. These industry standard tools create a cluster management system that includes:

- **Control Plane:** Manages the worker nodes and pods in the cluster.
- **Worker Nodes:** Used to run containerized applications and handle networking to ensure that traffic between applications across the cluster and from outside of the cluster can be properly facilitated.
- **Container Networking Interface (CNI):** Calico's open source networking and network security solution for containers, virtual machines, and native host-based workloads.
- **Container Storage Interface (CSI):** A common abstraction to container orchestrators for interacting with storage subsystems of various types.
- **Kubernetes Cluster API (CAPI):** Cluster API uses Kubernetes-style APIs and patterns to automate cluster lifecycle management for platform operators. For more on how CAPI is integrated into DKP Essential, see Understanding CAPI Concepts and Terms (see page 94)
- **Cert Manager:** A Kubernetes addon to automate the management and issuance of TLS certificates from various issuing sources.
- **Cluster Autoscaler:** A component that automatically adjusts the size of a Kubernetes cluster so that all pods have a place to run and there are no unneeded nodes.

3.3.4 Built-in GitOps

DKP Essential comes bundled with GitOps, an operating model for Kubernetes and other cloud native technologies, providing a set of best practices that unify Git deployment, management and monitoring for containerized clusters and applications. GitOps works by using Git as a single source of truth for declarative infrastructure and applications. With GitOps, the use of software agents can alert on any divergence between Git with what is running in a cluster, and if there's a difference, Kubernetes reconcilers automatically update or rollback the cluster depending on the case.

3.3.5 DKP Essential Single Cluster UI

Bundled with DKP Essential is a single cluster management view of DKP UI (see page 1280) that can be used in lieu of the bundled CLI. From the UI you can:

- **Setup identity providers:** DKP Essential supports GitHub, LDAP, SAML, and standard OIDC identity providers such as Google. These identity management providers support the login and authentication process for your Kubernetes cluster. See Identity Providers (see page 524) for more information.
- **Deploy applications:** The DKP Essential UI with Kommander allows you to customize your workspace application deployments via the Applications page within the UI.
- **Add a license:** To add a license via the UI with Kommander, see Add a license to Kommander (see page 85)

For more information concerning the global and workspace-level UI, see Workspaces (see page 573)

3.4 DKP Government Advanced

Features and capabilities that are specific to DKP Government Advanced are marked with this badge at the top of each page. These items are NOT available to DKP Government Essential users. In general, every feature available in DKP Enterprise (see page 75)is also available in DKP Government Advanced.

Enterprise Gov Advanced

DKP Government (Gov) Advanced is a multi-cluster Kubernetes management platform that is optimized for deployment in the government sector. With top-to-bottom declarative programming, DKP Government Advanced provides the most advanced and feature-rich Kubernetes management capability across any infrastructure, whether on-premise, in the cloud, in air-gapped environments, or at the edge.

Ease of use is the hallmark of D2iQ Kubernetes solutions. DKP Government Advanced enables you to be up and running in minutes and hours rather than weeks and months, with complete stability, reliability, military-grade security, and rapid time-to-value. Complexity is reduced through packaging, automation, integration, and elegant design.

Built on pure open-source Kubernetes, DKP Government Advanced enables you to avoid vendor lock-in, easily upgrade services, and take full advantage of open-source community innovation. DKP Government Advanced provides platform services for networking, storage, logging, monitoring, and more, all of which have been carefully selected from the Cloud Native Computing Foundation (CNCF) landscape and rigorously tested to work together.

DKP Government Advanced is a multi-cluster lifecycle management Kubernetes solution centered around a management cluster that manages multiple attached or managed Kubernetes clusters via a centralized management dashboard. The management dashboard gives you a single point of observability and control throughout all of your attached or managed clusters. The DKP Government Advanced license gives the user access to the entire Konvoy cluster environment, the DKP UI dashboard that deploys platform and catalog applications, and multi-cluster management, and comprehensive compatibility with our full range of infrastructure deployment options.

3.4.1 Compatible Infrastructure

DKP Government Advanced operates across D2iQ's entire range of cloud, on-premises, edge, and air-gapped infrastructures and has support for various OSs, including immutable OSs. See Supported Operating Systems (see page 62) for a full list of compatible infrastructure.

For the basics on standing up a DKP Government Advanced cluster in one of the listed environments of your choice, see Additional Infrastructure Customized Configurations (see page 919).

3.4.2 Platform Applications

Applications can be deployed in any DKP managed cluster, giving you complete flexibility to operate across cloud, on-premises, edge, and air-gapped scenarios. Customers can also use the UI with Kommander to customize which platform applications to deploy to the cluster in a given workspace.

3.4.3 Catalog Applications

Quickly and easily deploy applications and complex data services from a centralized service catalog to specific or multiple clusters, with governance. Fast data pipelines can be provisioned automatically from the following catalog of DKP Applications:

- **Kafka:** Primarily used to build real-time streaming data pipelines and applications that adapt to the data streams. It combines messaging, storage, and stream processing to allow storage and analysis of both historical and real-time data.
- **Spark:** An industry standard analytics engine for big data processing and machine learning. Spark enables you to process data for both batch and streaming workloads.
- **Zookeeper:** A centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

For instructions on how to deploy catalog applications, see Workspace Catalog Applications (see page 586)

3.4.4 Cluster Manager

Konvoy is the Kubernetes installer component of DKP Government Advanced that uses industry standard tools to produce a certified Kubernetes cluster. These industry standard tools create a cluster management system that includes:

- **Control Plane:** Manages the worker nodes and pods in the cluster.
- **Worker Nodes:** Used to run containerized applications and handle networking to ensure that traffic between applications across the cluster and from outside of the cluster can be properly facilitated.
- **Container Networking Interface (CNI):** Calico's open source networking and network security solution for containers, virtual machines, and native host-based workloads.
- **Container Storage Interface (CSI):** A common abstraction to container orchestrators for interacting with storage subsystems of various types.
- **Kubernetes Cluster API (CAPI):** Cluster API uses Kubernetes-style APIs and patterns to automate cluster lifecycle management for platform operators. For more on how CAPI is integrated into DKP Government Advanced, see Understanding CAPI Concepts and Terms (see page 94)

- **Cert Manager:** A Kubernetes addon to automate the management and issuance of TLS certificates from various issuing sources.
- **Cluster Autoscaler:** A component that automatically adjusts the size of a Kubernetes cluster so that all pods have a place to run and there are no unneeded nodes.

3.4.5 Built-in GitOps

DKP Government Advanced comes bundled with GitOps, an operating model for Kubernetes and other cloud native technologies, providing a set of best practices that unify Git deployment, management and monitoring for containerized clusters and applications. GitOps works by using Git as a single source of truth for declarative infrastructure and applications. With GitOps, the use of software agents can alert on any divergence between Git with what is running in a cluster, and if there's a difference, Kubernetes reconcilers automatically update or rollback the cluster depending on the case.

3.4.6 DKP Government Advanced Multi-cluster UI

Bundled with DKP Government Advanced is a multi-cluster management view in the UI with Kommander that can be used in lieu of the bundled CLI. From the UI you can:

- **Connect to an infrastructure provider:** DKP Government Advanced supports on-premises and cloud infrastructure providers such as AWS and Azure for your Konvoy clusters. To automate their provisioning, DKP requires authentication keys to your preferred infrastructure provider entered on the Add Infrastructure Provider form.
- **Setup identity providers:** DKP Government Advanced supports GitHub, LDAP, SAML, and standard OIDC identity providers such as Google. These identity management providers support the login and authentication process for your Kubernetes cluster. See Identity Providers (see page 524) for more information.
- **Configure access control:** Role-based authorization control (RBAC) is central to DKP Government Advanced and controls access to resources on all connected clusters. The resources are similar to Kubernetes RBAC. You add an identity provider group, and in that group add cluster roles and cluster role bindings for those roles.
- **Deploy applications:** The DKP Government Advanced UI allows you to customize your workspace application deployments via the Applications page within the UI.
- **Create a project:** Create projects within a workspace and deploy project-scoped applications. Projects enable teams to deploy configurations and services to their clusters consistently. After configuring roles, ConfigMaps, secrets, and applications for a project, DKP distributes this configuration to each project namespace. For more information concerning projects, see Projects (see page 608).
- **Add a license:** To add a license via the UI with Kommander, see Add a License (see page 85)
- **Kubernetes cost monitoring and management**: The kubecost platform application provides real-time cost visibility and insights for external cloud services such as AWS, helping you continuously reduce your cloud costs.

For more information concerning the global and workspace-level UI with Kommander, see Workspaces (see page 573)

3.4.7 Confirmed Stateside Support (CSS)

Because many governmental applications require CSS, DKP Government Advanced includes access to D2iQ's CSS team of U.S. citizens working at U.S. support centers.

3.4.8 Military-Grade Security

DKP Government Advanced can be configured to meet defined security standards. Each component is tested and certified before release. DKP also provides instant platform engineering that reduces the burden of security on DevOps and DevSecOps teams. Being based on pure upstream open-source Kubernetes enables easy and trouble-free upgrades, patches, and bug fixes.

3.5 DKP Government Essential

DKP Government (Gov) Essential is a single-cluster Kubernetes management platform that is optimized for deployment in the government sector. Based on the D2iQ Kubernetes Platform (DKP), DKP Government Essential can easily and seamlessly expand to a multi-cluster Kubernetes management platform as your needs grow.

With top-to-bottom declarative programming, DKP Government Essential provides the most advanced and feature-rich single-cluster Kubernetes management capability across any infrastructure, whether on-premise, in the cloud, in air-gapped environments, or at the edge.

Ease of use is the hallmark of D2iQ Kubernetes solutions. DKP Government Essential enables you to be up and running in minutes and hours rather than weeks and months, with complete stability, reliability, military-grade security, and rapid time-to-value. Complexity is reduced through packaging, automation, integration, and elegant design.

Built on pure open-source Kubernetes, DKP Government Essential enables you to avoid vendor lock-in, easily upgrade services, and take full advantage of open-source community innovation. DKP Government Essential provides platform services for networking, storage, logging, monitoring, and more, all of which have been carefully selected from the Cloud Native Computing Foundation (CNCF) landscape and rigorously tested to work together.

DKP Government Essential is a self-managed single cluster Kubernetes solution that gives you a feature-rich, easy-to-deploy, and easy-to-manage entry-level cloud container platform. The DKP Government Essential license gives the user access to the entire Konvoy cluster environment, and to the Kommander platform application manager. In general, every feature available in DKP Essential (see page 78)is also available in DKP Government Essential

3.5.1 Compatible Infrastructure

DKP Government Essential operates across a range of cloud, on-premise, edge, and air-gapped infrastructures and has support for various OSs, including immutable OSs. See Supported Operating Systems (see page 62) for a full list of compatible infrastructure.

For the basics on standing up a DKP Government Essential cluster in one of the listed environments of your choice, see Advanced Configuration (see page 1282).

3.5.2 Platform Applications

When creating a cluster, the application manager deploys certain platform applications on the newly created cluster. DKP Government Essential users can use the Kommander UI to customize which platform applications to deploy to the cluster in a given workspace. For a list of available platform applications that are included with DKP, see Workspace Platform Applications (see page 114).

> **NOTE:** The platform application `kubecost` is not included with DKP Government Essential, but is included with DKP Enterprise (see page 78) and DKP Government Advanced (see page 80).

3.5.3 Cluster Manager

Konvoy is the Kubernetes installer component of DKP Government Essential that uses industry standard tools to produce a certified Kubernetes cluster. These industry standard tools create a cluster management system that includes:

- **Control Plane:** Manages the worker nodes and pods in the cluster.
- **Worker Nodes:** Used to run containerized applications and handle networking to ensure that traffic between applications across the cluster and from outside of the cluster can be properly facilitated.
- **Container Networking Interface (CNI):** Calico's open source networking and network security solution for containers, virtual machines, and native host-based workloads.
- **Container Storage Interface (CSI):** A common abstraction to container orchestrators for interacting with storage subsystems of various types.
- **Kubernetes Cluster API (CAPI):** Cluster API uses Kubernetes-style APIs and patterns to automate cluster lifecycle management for platform operators. For more on how CAPI is integrated into DKP Government Essential, see Understanding CAPI Concepts and Terms (see page 94)
- **Cert Manager:** A Kubernetes addon to automate the management and issuance of TLS certificates from various issuing sources.
- **Cluster Autoscaler:** A component that automatically adjusts the size of a Kubernetes cluster so that all pods have a place to run and there are no unneeded nodes.

3.5.4 Built-in GitOps

DKP Government Essential comes bundled with GitOps, an operating model for Kubernetes and other cloud native technologies, providing a set of best practices that unify Git deployment, management and monitoring for containerized clusters and applications. GitOps works by using Git as a single source of truth for declarative infrastructure and applications. With GitOps, the use of software agents can alert on any divergence between Git with what is running in a cluster, and if there's a difference, Kubernetes reconcilers automatically update or rollback the cluster depending on the case.

3.5.5 DKP Government Essential Single Cluster UI

Bundled with DKP Government Essential is a single cluster management view of DKP UI (see page 1280) that can be used in lieu of the bundled CLI. From the UI you can:

- **Setup identity providers:** DKP Government Essential supports GitHub, LDAP, SAML, and standard OIDC identity providers such as Google. These identity management providers support the login and authentication process for your Kubernetes cluster. See Identity Providers (see page 524) for more information.
- **Deploy applications:** The DKP Government Essential UI with Kommander allows you to customize your workspace application deployments via the Applications page within the UI.
- **Add a license:** To add a license via the UI with Kommander, see Add a license to Kommander (see page 85)

For more information concerning the global and workspace-level UI, see Workspaces (see page 573)

3.5.6 Confirmed Stateside Support (CSS)

Because many governmental applications require CSS, DKP Government Essential includes access to D2iQ's CSS team of U.S. citizens working at U.S. support centers.

3.5.7 Military-Grade Security

DKP Government Essential can be configured to meet defined security standards. Each component is tested and certified before release. DKP also provides instant platform engineering that reduces the burden of security on DevOps and DevSecOps teams. Being based on pure upstream open-source Kubernetes enables easy and trouble-free upgrades, patches, and bug fixes.

3.6 Add a DKP license

3.6.1 Prerequisites

For licenses that you purchase from the AWS Marketplace[44], an AWS administrator must attach the AWS managed policy[45] `AWSLicenseManagerConsumptionPolicy` to the `control-plane.cluster-api-provider-aws.sigs.k8s.io` role created when configuring AWS IAM policies for Konvoy (see page 531). If an Administrator does not attach this policy attached to the role, DKP cannot verify the license information provided in the procedure steps that follow.

The machine onto which you install DKP must meet these requirements:

44 https://aws.amazon.com/marketplace/
45 https://docs.aws.amazon.com/license-manager/latest/userguide/security-iam-awsmanpol.html#security-iam-AWSLicenseManagerConsumptionPolicy

- Docker installed - DKP uses Kubernetes-in-Docker (Kind) to create a bootstrap cluster for creating Management clusters, and thus Docker installation is required. Docker is not used to run your Management or workload clusters.
- Active Internet connectivity
- Existing AWS account
- AWS Administrator logged into the AWS account used to purchase DKP

3.6.2 Download the Container Image and Extract the Binaries

1. When you complete the sales transaction in the AWS Marketplace, select a DKP version from the **Choose a fulfillment option** dropdown. We recommend that you select the latest version.
2. Select **Continue to Launch** to obtain the download instructions, available when you select **Usage Instructions**.
3. Complete the steps in each of the procedures under **Usage instructions**. The links in these steps take you to DKP documentation pages that provide the steps you need to set up AWS to work with DKP, and the download and installation of DKP itself.
4. Expand the **Create a license token and IAM role** link in the **Container images** section of the window. This gives you access to the Create token button to generate the license token and IAM role and exposes an inset code window with the token and role credentials.
5. Use the generated command to log in to AWS using the token and role you just generated. Note that you can omit this step if you have already configured the AWS CLI with valid credentials.
   Tip: Using the Copy button at the upper right of this inset code window helps ensure you copy the entire command.
6. Copy the commands at the bottom of the page, and run them sequentially to download the container images. It will look similar to this:

   ```
   aws ecr get-login-password \
     --region <insert-region> | docker login \
     --username <username> \
     --password-stdin <ecr-address>
   ```

   ```
   CONTAINER_IMAGES="<ecr-address>/mesosphere/d2iq-dkp-essential-premium-
   support:v2.5.0"
   ```

   ```
   for i in $(echo $CONTAINER_IMAGES | sed "s/,/ /g"); do docker pull $i; done
   ```

7. Access a terminal window on the Linux machine where you plan to run the DKP CLI cluster, and execute the command above, using your environment's specific values. This downloads a container that contains the DKP binary.

8. After downloading the container, run the following command to copy the binaries onto your Linux machine.

```
docker run -it --rm -u $(id -u):$(id -g) -v $(pwd):/dkp $CONTAINER_IMAGES
```

You will then see the following output:

```
dkp binary is placed in the local directory, to run:
./dkp --help
```

You will now see the `dkp` binary in your working directory. Follow the DKP installation (see page 117) instructions using these binaries, and then add your license (see page 85) to DKP.

3.6.3 Obtain the Amazon Resource Number (ARN) from the AWS Marketplace UI

At this point, you need to use the AWS Marketplace user interface to navigate to the license table and configure its settings to display the ARN for your purchased license.

1. From the **Launch this software** page, navigate back to the **Product Detail** page.
2. Select the **View Subscription** button at the upper right (in the blue information box labeled "***You have access to this product***").
3. Select **Manage subscriptions** from the left-hand navigation pane, and then click the **Manage** button on the specific license. The details page for that license displays.
4. Select the **License** number hyperlink to display the license page, then select **Granted licenses** from the left-hand navigation pane and find your license in the list.
5. Use the **Settings** icon in the upper right corner to display the columns displayed, and then enable the **License arn** value.
6. Select the **Confirm** button to return to the license list with the License ARN displayed in the last column. You can now copy and paste this ARN as needed.

If you have difficulty obtaining your ARN, contact D2iQ Support for assistance.

3.6.4 Enter License Information in the DKP UI

For licenses purchased directly from D2iQ, you can obtain the license token through the Support Portal[46]. Insert this token in the last step below.

For licenses purchased through the AWS marketplace, you can find the license in the "Granted Licenses" table of AWS License Manager inside the AWS console. If necessary, modify the table view preferences to show the "License ARN." Copy this value for use in the last step below.

46 https://support.d2iq.com/hc/en-us

You must be an administrator to add licenses to DKP.

From the the UI with Kommander, complete the following steps:

1. Select **Global** in the workspace header drop-down.
2. In the sidebar menu, select **Administration** > **Licensing**.
3. Select + Activate License to enter the Activate License form.
4. On the Activate License form page, select **D2iQ** or **AWS Marketplace** depending on where you acquired your license.
5. Paste your license token, or the AWS ARN, in the text box and select **Save**.

If there is an error submitting a license acquired directly from D2iQ, you can activate the license directly through `kubectl`.

3.6.5 Enter a DKP License using kubectl

You can activate a license acquired from D2iQ directly using the `kubectl` utility.

1. Create a secret, replacing the value `MY_LICENSE` in the command that follows with your actual, D2iQ-provided DKP license token:

```
kubectl create secret generic my-license-secret --from-literal=jwt=MY_LICENSE
-n kommander
kubectl label secret my-license-secret kommanderType=license -n kommander
```

2. Create a license object with this command:

```
cat <<EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: License
metadata:
  name: my-license
  namespace: kommander
spec:
  licenseRef:
    name: my-license-secret
EOF
```

3. Return to the license page in the DKP UI to see your valid license display.

3.6.6 Activate a DKP Insights License Key

Instructions for activating an Insights License Key in the DKP UI

3.6.6.1 Prerequisites

Prior to activating an Insights license key, ensure that you have a valid DKP Essential or DKP Enterprise license.

3.6.6.2 Obtain a License Key

DKP Insights license keys to activate the product are provided directly by D2iQ. Contact your account rep to receive a license key.

After you receive a license key, follow the instructions in the next section to activate DKP Insights in your environment.

3.6.6.3 Enter License Information

An administrator must activate licenses in DKP.

From the DKP UI, complete the following steps:

1. Select **Global** in the workspace header drop-down.
2. In the sidebar menu, select **Administration** > **Licensing**.
3. Select **+ Activate Add-on License**.
4. In the Add-on License form page, input the Insight license key that was provided to you by DKP.
5. Select **Save**.

Once you activate an Insights license key, the prompt to activate an Add-on License is removed.

3.6.6.4 Remove an Insights License

If your Insights license information has changed, you may need to remove an existing license from DKP to activate a new one. Only DKP administrators can remove licenses.

In the DKP UI, perform the following:

1. Select **Global** in the workspace header drop-down.
2. In the sidebar menu, select **Administration** > **Licensing**.
3. Select **Remove License** in the DKP Insights card, then follow the prompts.

3.7 Remove a DKP license

If your license information has changed, you may need to remove an existing license from DKP to add a new one. Only DKP administrators have the ability to remove licenses.

Original license information can still be obtained from D2iQ or the AWS License Manager console even after removing from DKP.

3.7.1 Remove a License via the UI

In the DKP UI, do the following:

1. Select **Global** in the workspace header drop-down.
2. In the sidebar menu, select **Administration** > **Licensing**.
3. Your existing licenses will be listed. Click **Remove License** on the license you would like to remove, and follow the prompts.

3.7.2 Manually Remove a License using kubectl

To remove a license from DKP using `kubectl`, you have to delete the `Secret` and `License` objects. In this example, the secret is named “my-license-secret”.

1. Validate that the secret exists in the `kommander` namespace:

   ```
   kubectl describe secret -n kommander my-license-secret
   ```

   Expected output:

   ```
   Name:         my-license-secret
   Namespace:    kommander
   Labels:       kommanderType=license
   Annotations:  <none>

   Type:         Opaque
   ```

```
Data
====
jwt:  455 bytes
```

2. Delete the secret from the `kommander` namespace:

```
kubectl delete secret -n kommander my-license-secret
```

Expected output:

```
secret "my-license-secret" deleted
```

3. We do the same with the `License` object. Validate that it exists in the `kommander` namespace:

```
kubectl describe license -n kommander my-license
```

Expected output:

```
Name:         my-license
Namespace:    kommander
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"kommander.mesosphere.io/v1beta1","kind":"License
","metadata":{"annotations":{},"name":"my-license", "namespace":"kommand...
API Version:  kommander.mesosphere.io/v1beta1
Kind:         License
Metadata:
  Creation Timestamp:  2020-03-25T14:57:31Z
  Generate Name:       license-
  Generation:          1
  Resource Version:    17895
  Self Link:           /apis/kommander.mesosphere.io/v1beta1/namespaces/
kommander/licenses/my-license
  UID:                 35ee9254-4094-40eb-a2d8-4687c5d212d9
Spec:
  License Ref:
    Name:  my-license-secret
Status:
  Cluster Capacity:  500
  Customer Id:       mesosphere-developer
  End Date:          2020-10-02T14:00:09Z
  License Id:        mesosphere-developer
  Start Date:        2019-10-02T14:00:09Z
  Valid:             true
  Version:           1.0
```

```
Events:
  Type    Reason                 Age                   From               Message
  ----    ------                 ----                  ----               -------
  Normal  LicenseUpdateSuccess  7m7s (x2 over 7m7s)  LicenseSignature  License
updated successfully
```

4. Delete the license from the `kommander` namespace:

   ```
   kubectl delete license -n kommander my-license
   ```

   Expected output:

   ```
   license.kommander.mesosphere.io "my-license" deleted
   ```

You have now successfully removed a license.

4 Day 0 - Get Started with DKP

At D2iQ, we partner with you throughout the entire cloud-native journey, at the beginning on Day 0 all the way through your Day 2 operations. Here is the breakdown:

- **Day 0** is the planning phase where we introduce definitions and concepts.
- **Day 1** is to install the software and get it up and running.
- **Day 2** involves customizations, application management, and operations management.

When installing DKP, the first step is to determine the infrastructure on which you want to deploy. Then, ensure you understand the basic concepts and have the prerequisites met, like storage and resource requirements, using the information contained in this section of the documentation.

For example, you can:

- Install on a public cloud infrastructure, such as Amazon Web Services (AWS), GCP, or Azure.
- Install on an internal network, on-premises environment, with a physical or virtual infrastructure.
- Install on an Air-gapped environment.
- Install with or without FIPS and GPU.

After you choose your environment, you download DKP[47], and then select the Day 1 - Basic Install instructions[48] for your Infrastructure provider and environment. The basic installs set up a cluster with the Konvoy component, and then install the Kommander component, so that you can access the dashboards through the DKP UI. You can use this basic cluster to explore DKP and prepare to take your clusters into *production*, where you will deploy and enable the applications that support Day 2 operations. Then you can deploy and test your workload, and put Kubernetes to work!

After you finish the basic install and are ready to customize, move to Additional Infrastructure Configuration (see page 919), if desired. Last, proceed to Day 2 Cluster Operations Management (see page 510) to make the software exactly what you want it to be.

Follow this list, contained in the Get Started section to ensure you have all the resource requirements met for a successful implementation:

- CAPI Concepts and Terms (see page 94)
- DKP Concepts and Terms (see page 95)
- What is Pre-provisioned Infrastructure (see page 96)
- Cluster Types and Concepts (see page 97)
- Provide Context for Commands with a kubeconfig File (see page 99)
- Storage (see page 101)
- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

47 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29917289
48 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/161021963

4.1 CAPI Concepts and Terms

DKP uses CAPI (ClusterAPI) technology for creating and managing the lifecycle of Kubernetes Clusters. A basic understanding of CAPI concepts and terms is helpful in understanding how to install and maintain DKP. You can find a deeper discussion of the architecture in the ClusterAPI Book[49].

CAPI makes use of a bootstrap cluster for provisioning and starting clusters. A bootstrap cluster handles the following:

1. Generating the cluster certificates, if they are not otherwise specified.
2. Initializing the control plane, and managing the creation of other nodes until it is complete.
3. Joining control plane and worker nodes to the cluster.
4. Installing and configuring networking plugin (Calico CNI), CSI volume provisioners, cluster-autoscaler and other core Kubernetes components.

BootstrapData is machine or node role-specific data, such as cloud initialization data, used to bootstrap a “machine” onto a node.

For customers using Kommander for multi-cluster management, a **management cluster** that manages the lifecycle of workload clusters. As the management cluster, DKP Kommander works with bootstrap providers, infrastructure providers, and maintains cluster resources such as bootstrap configurations and templates. If you are working with only one cluster, Kommander will provide you with addon (platform application) management for that cluster, but not others.

A **workload cluster** is a Kubernetes cluster whose lifecycle is managed by a management cluster, and provides the platform on which you deploy and execute your workloads.

As part of a collection of **Custom Resource Definitions** or **CRDs** that extend the Kubernetes API, these additional concepts are important for understanding the upgrade.

A **ClusterResourceSet** Kubernetes cluster created by CAPI is functionally minimal in nature. Crucial components like CSI and CNI are not a part of the default cluster spec. A ClusterResourceSet is a CRD that can be used to group and deploy core cluster components post-Kubernetes cluster install.

When you create a bootstrap cluster. You can find all the components in the default namespace and we move them to the workload cluster during the process of making the cluster self-managed.

A **machine** is a declarative spec for a platform or infrastructure component that hosts a Kubernetes node such as a bare metal server or a VM. CAPI uses provider-specific controllers to provision and install new hosts which then register as nodes. When you update a machine spec other than for certain values, such as annotations, status, and labels, the controller deletes the host and creates a new one that conforms to the new spec. This is called machine immutability. If you delete a machine, the controller deletes both the infrastructure and the node. Provider-specific information is not portable between providers.

Within CAPI, you use declarative **MachineDeployments** to handle changes to machines by replacing them in much the same way that a core Kubernetes Deployment replaces Pods. MachineDeployments reconcile changes to machine specs by rolling out changes to two **MachineSets** (similar to a ReplicaSet), both the old and the newly-updated.

A **MachineHealthCheck** identifies unhealthy node conditions, and initiates remediation for nodes owned by a MachineSet.

---

49 https://cluster-api.sigs.k8s.io/user/concepts.html

4.1.1 Next Topic:

DKP Concepts and Terms (see page 95)

4.2 DKP Concepts and Terms

Understand the terminology used in association with DKP with the information on this page - DKP concepts and terms.

DKP is comprised of three main components, Konvoy, Kommander, and Konvoy Image Builder (KIB) that work seamlessly together to provide a single and centralized point of control for an organization's application infrastructure. DKP empowers organizations to more easily deploy, manage, and scale Kubernetes workloads in Day 2 production environments.

Each main component specifically manages the following:

1. **Konvoy** is the cluster lifecycle manager component of DKP. Konvoy relies on Cluster API, Calico, and other open-source and proprietary software to provide simple cluster lifecycle management for conformant Kubernetes clusters with networking and storage capabilities.

   Konvoy uses industry-standard tools to provision certified Kubernetes clusters on multiple cloud providers, vSphere, and on-premises hardware in connected and air-gapped environments. Konvoy contains the following components:

   - **Cluster Manager** - Cluster API, CSI, CNI, Cluster AutoScaler, Cert Manager, MetalLB

2. **Kommander** is the fleet management component of DKP. Kommander delivers centralized observability, control, governance, unified policy, and better operational insights. With DKP Essential, Kommander manages a single Kubernetes cluster.
   In DKP Enterprise, Kommander supports attaching workload clusters and lifecycle management of clusters using Cluster API. DKP Enterprise also offers lifecycle management of applications through FluxCD. Kommander contains the following components:

   - User interface, Security, Observability, Networking, Application Management
   - *Platform Applications*: Applications that come out of the box with DKP providing functionality like observability, cost management, monitoring, logging, making DKP clusters Day 2 production ready right from install. Platform applications are D2iQ's choice of selected applications from the open source community that are consumed by the platform.
     Essential Platform Applications: Monitoring, Logging, Backup/Restore, Policy Agent, External DNS, Load Balance, Ingress, SSO, Service Mesh.
   - **Enterprise Platform Applications:** All above Essential Platform Applications, plus additional Access Control and Centralized Cost Management.
   - *Catalog Applications*: Applications in DKP Enterprise that are deployed to be used for customer workloads, such as Kafka, Spark and ZooKeeper.

3. **Konvoy Image Builder** (KIB) creates Cluster API compliant machine images. It configures those images to contain all the necessary software to deploy Kubernetes cluster nodes.

DKP Essential and Enterprise also provide a helpful add-on called DKP Insights[50].

4.2.1 Next Topic:

What is Pre-provisioned Infrastructure (see page 96)

4.3 What is Pre-provisioned Infrastructure

Pre-provisioned as an infrastructure allows the deployment of Kubernetes DKP to pre-existing machines. Other providers such as vSphere, AWS, or Azure create, or provision, the machines themselves before Kubernetes is deployed. On most infrastructures (including vSphere and cloud providers), DKP provisions the actual nodes automatically as part of deploying a cluster. It creates the VMs using the appropriate image, then handles the networking and installation of Kubernetes. However, DKP can also work with pre-provisioned infrastructure in which you provision the VMs for the nodes themselves. Note that you can pre-provision nodes for DKP on bare metal, on vSphere, or even in the the cloud. Pre-provisioned and vSphere is a combination of the physical (on premises bare metal) and virtual servers (VMware vSphere).

Pre-provisioned environments are often used in bare metal deployments, where you deploy your OS (see page 62) (such as RHEL, CentOS, Ubuntu, etc) on physical machines. Creating a pre-provisioned cluster means that you, as an **Infrastructure Ops Manager** are responsible for allocating your compute resources, setting up networking, collecting IP and SSH information to be provided to DKP, etc. You can then provide all details to the pre-provisioned provider in order to deploy Kubernetes. These operations are done manually or with the help of other tools.

In pre-provisioned environments, DKP handles your cluster's lifecycle (installation, upgrade, node management, etc.). DKP installs Kubernetes, monitoring and logging apps as well as the UI.

The main use cases for the pre-provisioned provider are:

- On-premises clusters
- Cloud or IAAS environments that do not currently have a D2iQ supported infrastructure provider
- Cloud environments where you must use predefined infrastructure instead of having one of the supported cloud providers create it for you

In an environment with access to the internet, you retrieve artifacts from specialized repositories dedicated to them such as Docker images contained in DockerHub and Helm Charts that come from a dedicated Helm Chart repository. However, in an air-gapped environment, you need local repositories to store Helm charts, Docker images and other artifacts. Tools such as jFrog, Harbor and Nexus handle multiple types of artifacts in one local repository.

4.3.1 Next Topic:

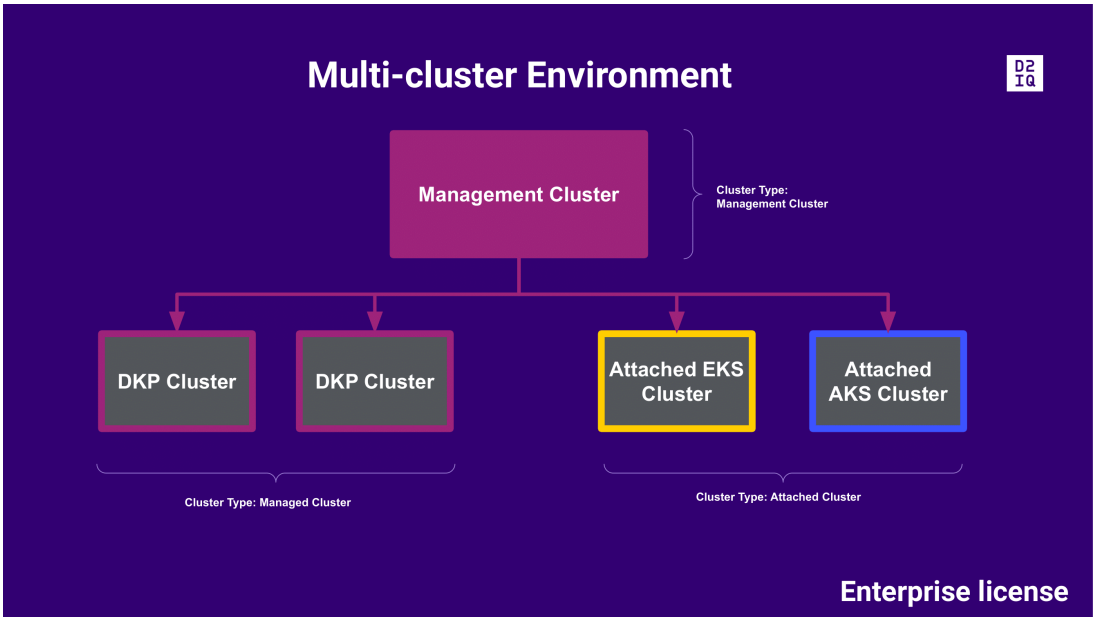
Cluster Types and Concepts (see page 97)

50 https://d2iq.atlassian.net/wiki/spaces/DINS

4.4 Cluster Types and Concepts

Cluster types such as Management clusters, Managed clusters, and Attached clusters are key concepts in understanding and getting the most out of DKP Essential versus Enterprise environments.

4.4.1 Multi-cluster Environment



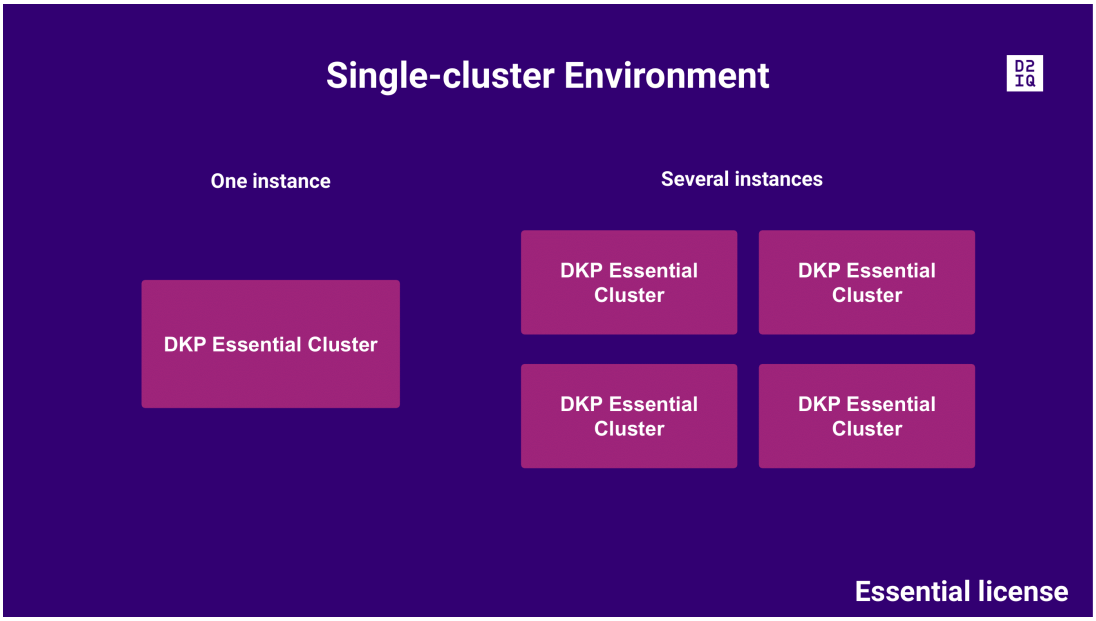
Management Cluster: Is the cluster where you install DKP, and it is self-managed. In a multi-cluster environment, the Management cluster also manages other clusters.

Customers with an Enterprise license (see page 75) should run workloads on *Managed* and *Attached* clusters, and not on the *Management cluster*.

Managed Cluster: A type of workload cluster that you can create with DKP. The *DKP Management cluster* manages its infrastructure, its lifecycle, and its applications.

Attached Cluster: A type of workload cluster that is created outside of DKP, but is then connected to the *DKP Management Cluster* so that it can be managed by DKP. In these cases, the DKP Management cluster only manages the attached cluster's applications.

4.4.2 Single-cluster Environment



DKP Essential Cluster: Is the cluster where you install DKP. A DKP Essential cluster is a stand-alone cluster. It is self-managed and therefore capable of provisioning itself. In this single-cluster environment, you cannot attach other clusters and all workloads are run on your *DKP Essential cluster*. You can, however, have several separate DKP Essential instances, each with its own license.

Customers with an Essential license (see page 78) can run workloads on their *DKP Essential cluster*.

> If you have not decided which license to get, but plan on adding one or several clusters to your environment, and manage them centrally, D2iQ recommends obtaining an Enterprise license (see page 75).

4.4.3 Other important concepts

Self-managed Cluster: In a DKP landscape, only *DKP Essential* and *DKP Enterprise Management* clusters are self-managed. *Self-managed clusters* are clusters that manage the provisioning, and deployment of its own nodes through CAPI controllers. The CAPI controllers are a managing entity, which automatically manages the lifecycle of a cluster's nodes based on a customizable definition of the resources.

A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it, are running on the same cluster they are managing.

4.4.4 Next Topic:

Provide Context for Commands with a kubeconfig File (see page 99)

4.5 Provide Context for Commands with a kubeconfig File

This page contains some basic recommendations regarding `kubeconfig` as it relates to target clusters and the `--kubeconfig=${CLUSTER_NAME}.conf` flag. Refer to the Kubernetes documentation[51] for more information.

For `kubectl` and `dkp` commands to run, it is often necessary to specify the environment or cluster in which you want to run them. This specially applies to commands that create, delete, or update a cluster's resources.

To specify the context, there are two options:

Export an environment variable	Specify the target cluster in the command
Export an environment variable from a cluster's `kubeconfig` file, which sets the environment for the commands you run after exporting it.	Specify an environment variable for one command at the time by running it with the `--kubeconfig=${CLUSTER_NAME}.conf` flag.
• Better suited for single-cluster environments	• Better suited for multi-cluster environments

4.5.1 Single-cluster Environment

In a single-cluster environment, you don't need to switch between clusters to execute commands and perform operations. However, it is still necessary to specify an environment for each terminal session, so the DKP CLI runs the operations on the cluster with the Kommander component.

Use a kubeconfig file to set the environment variable for all your operations

51 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

4.5.1.1 Set the environment variable for all your operations

1. When you create a cluster, a `kubeconfig` file is generated automatically. Get this `kubeconfig` file and write it to the `${CLUSTER_NAME}.conf` variable:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

   After setting this configuration, whenever you use run a command with the variable `${CLUSTER_NAME}.conf`, it will automatically run on your cluster.

2. Set the context by exporting your `kubeconfig` from the source file:
   :note: Execute this command for each terminal session.

   ```
   export KUBECONFIG=${CLUSTER_NAME}.conf
   ```

   For the current terminal session, you don't need to provide the `--kubeconfig` flag.

4.5.2 Multi-cluster Environment

Since having multiple clusters means switching between them to run operations, D2iQ recommends using the flag to set the context.

Use a flag to reference the target cluster

4.5.2.1 Use a flag to reference the target cluster

The `--kubeconfig ${CLUSTER_NAME}.conf` flag defines the configuration file for the cluster that you are configuring and accessing.

When operating and using multiple clusters, this is the easiest way to ensure that you are working on the correct cluster. If you create additional clusters and do not store the name as an environment variable, you can type out the name of the cluster followed by `.conf` to access your cluster.

> Ensure you run `dkp get kubeconfig` for each cluster you create to generate a `kubeconfig` file.

Example flag:

```
--kubeconfig azurecluster1.conf
```

Example command with flag:

```
dkp install kommander --kubeconfig azurecluster1.conf
```

Advanced: Use a kubeconfig file to configure access to multiple clusters using contexts

For advanced users only.

It is possible to set up a `kubeconfig` file to manage access to several clusters. For more information, refer to https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/.

You can also set your environment variable to multiple `kubeconfig` files by merging them. For more information, refer to https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/.

4.5.3 Next Topic:

Storage (see page 101)

4.6 Storage

An introduction to persistent storage in Kubernetes

This document describes the model used in Kubernetes for managing persistent, cluster-scoped storage for workloads requiring access to persistent data.

A workload on Kubernetes typically requires two types of storage:

4.6.1 Ephemeral Storage

Ephemeral storage, by its name, is ephemeral in the sense that it is cleaned up when the workload is deleted or the container crashes. For example, the following are examples of ephemeral storage provided by Kubernetes:

EmptyDir volume.	Managed by kubelet under `/var/lib/kubelet`.
Container logs.	Typically under `/var/logs/containers`.

Container image layers.	Managed by container runtime (e.g., under `/var/lib/containerd`).
Container writable layers.	Managed by container runtime (e.g., under `/var/lib/containerd`).

Ephemeral storage is automatically managed by Kubernetes, and typically does not require explicit settings. You may need to express the capacity requests for ephemeral storage so that `kubelet` can use that information to make sure it does not run out of ephemeral storage space on each node.

4.6.2 Persistent Volume

Persistent Volumes are storage resources that can be used by the cluster. Persistent Volumes are volume plug-ins that have lifecycle capabilities that are independent of any Kubernetes Pod or Deployment. A Kubernetes persistent volume (PV) is an object that allows pods to access persistent storage on a storage device and defined via a Kubernetes StorageClass. Unlike regular volumes, which are transient in nature, PVs are persistent, supporting stateful application use cases.

You may have stateful workloads requiring persistent storage whose lifecycle is longer than that of Pods or containers. For instance, a database server needs to recover database files after it crashes. For those cases, the workloads need to use PersistentVolumes (PV).

Persistent Volumes are resources that represent storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. Unlike ephemeral storage, the lifecycle of a PersistentVolume is independent of that of the workload that uses it.

The Persistent Volume API objects capture the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system. In order to use a Persistent Volume (PV), your application needs to invoke a Persistent Volume Claim (PVC).

4.6.2.1 Persistent Volume Claim

A PersistentVolumeClaim is a request for storage. For a workload that requires persistent volumes, the workload should use PersistentVolumeClaim (PVC) to express its request on persistent storage. A PersistentVolumeClaim can request specific size and Access Modes[52] (for example, they can be mounted once read/write or many times read-only).

Any workload can specify a PersistentVolumeClaim. For example, a Pod may need a volume that is at least 4Gi large or a volume mounted under `/data` in the container's filesystem. If there is a PersistentVolume (PV) that satisfies the specified requirements in the PersistentVolumeClaim (PVC), it will be bound to the PVC before the Pod starts.

52 https://kubernetes.io/docs/concepts/storage/persistent-volumes/#access-modes

4.6.2.2 Related Information

- Kubernetes Storage[53]
- Kubernetes persistent storage design document[54]
- Default Storage Providers in DKP (see page 103)

4.6.3 Next Step:

Resource Requirements (see page 110)

4.6.4 Default Storage Providers in DKP

Default storage providers in DKP

When deploying DKP using a supported cloud provisioner (AWS, Azure, GCP), DKP automatically configures native storage drivers for the target platform. In addition, DKP deploys a default StorageClass[55] for dynamic persistent volume (PV)[56] creation. The table below lists the driver and default StorageClass for each supported cloud provisioner.

Cloud Provisioner	Version	Driver	Default Storage Class
AWS	`v1.16`	aws-ebs-csi-driver[57]	`ebs-sc`
Azure	`v1.27.0`	azuredisk-csi-driver[58]	`azuredisk-sc`
Pre-provisioned	`v2.5.0`	local-static-provisioner[59]	`localvolumeprovisioner`
vSphere	`v2.7.0`	vsphere-csi-driver[60]	`vsphere-raw-block-sc`
GCP	`v1.9.0`	gcp-compute-persistent-disk-csi-driver[61]	`csi-gce-pd`

53 https://kubernetes.io/docs/concepts/storage/
54 https://github.com/kubernetes/design-proposals-archive/tree/main/storage
55 https://kubernetes.io/docs/concepts/storage/storage-classes/
56 https://kubernetes.io/docs/concepts/storage/dynamic-provisioning/
57 https://github.com/kubernetes-sigs/aws-ebs-csi-driver
58 https://github.com/kubernetes-sigs/azuredisk-csi-driver
59 https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner
60 https://github.com/kubernetes-sigs/vsphere-csi-driver
61 https://github.com/kubernetes-sigs/gcp-compute-persistent-disk-csi-driver/releases

When creating a pre-provisioned infrastructure cluster, DKP uses the local static provisioner as the default storage provider. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[62] compatible storage that is suitable for production.

You can choose from any of the storage options[63] available for Kubernetes. To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your newly created StorageClass to be the default by following the commands in the Kubernetes documentation called Changing the Default Storage Class[64].

When a default StorageClass is specified, persistent volume claims (PVCs) can be created without needing to specify the storage class. For instance, to request a volume using the default provisioner, create a PVC with the following:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pv-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 4Gi
```

To start the provisioning of a volume, launch a pod which references the PVC:

```
...
    volumeMounts:
    - mountPath: /data
      name: persistent-storage
...
  volumes:
  - name: persistent-storage
    persistentVolumeClaim:
      claimName: my-pv-claim
```

62 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
63 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
64 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

4.6.4.1 Change or Manage Multiple Storage Classes

The default `StorageClass` provisioned with DKP is acceptable for install, but may not suitable for production. If your workload has different requirements, you can create additional `StorageClass` types with specific configurations. You can change the default `StorageClass` using these steps from the Kubernetes site:

- Changing the default storage class[65]

Ceph can also be used as CSI storage. Refer to that section in the documentation for information on how to use Rook Ceph (see page 894).

4.6.4.2 Driver Information

All default drivers implement the Container Storage Interface[66] (CSI). The CSI provides a common abstraction to container orchestrators for interacting with storage subsystems of various types. Each driver has specific configuration parameters which effect PV provisioning. This section details the default configuration for drivers used with DKP. This section also has links to driver documentation, if further customization is required.

NOTE: `StorageClass` parameters cannot be changed after creation. To use a different volume configuration, you must create a new `StorageClass`.

4.6.4.2.1 Amazon Elastic Block Store (EBS) CSI Driver

DKP EBS default `StorageClass`:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true" # This tells kubernetes to
make this the default storage class
  name: ebs-sc
provisioner: ebs.csi.aws.com
reclaimPolicy: Delete  # volumes are automatically reclaimed when no longer in use
and PVCs are deleted
volumeBindingMode: WaitForFirstConsumer #  Physical volumes will not be created until
a pod is created that uses the PVC, required to use CSI's Topology feature
```

65 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/
66 https://github.com/container-storage-interface/spec/blob/master/spec.md

```
parameters:
  csi.storage.k8s.io/fstype: ext4
  type: gp3 # General Purpose SSD
```

DKP deploys with gp3 (general purpose SSDs) EBS volumes.

- Driver documentation: aws-ebs-csi-driver[67]
- Volume types and pricing: volume types[68]

4.6.4.2.2 Azure CSI Driver

DKP deploys with StandardSSD_LRS for Azure Virtual Disks.

- Driver documentation: azuredisk-csi-driver[69]
- Volume types and pricing: volume types[70]
- Specifics for Azure using Pre-provisioning can be found here: Pre-provisioned Azure only Configurations (see page 1106)

4.6.4.2.3 vSphere CSI Driver

DKP default storage class for vSphere supports dynamic provisioning and static provisioning of block volumes.

- Driver documentation: https://docs.vmware.com/en/VMware-vSphere-Container-Storage-Plug-in/index.html
- Specifics for using vSphere storage driver: https://docs.vmware.com/en/VMware-vSphere-Container-Storage-Plug-in/2.0/vmware-vsphere-csp-getting-started/GUID-5D144DA0-4806-4DEB-8819-10A1C42E38AB.html

4.6.4.2.4 Pre-provisioned CSI Driver

In a pre-provisioned (see page 96) environment, DKP will also deploy a CSI compatible driver and configure a default StorageClass - `localvolumeprovisioner`.

- Driver documentation: local-static-provisioner[71]

67 https://github.com/kubernetes-sigs/aws-ebs-csi-driver
68 https://aws.amazon.com/ebs/features/
69 https://github.com/kubernetes-sigs/azuredisk-csi-driver
70 https://learn.microsoft.com/en-us/azure/aks/azure-disk-csi
71 https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner

DKP uses local-static-provisioner[72] (`localvolumeprovisioner`) as the default storage provider for a pre-provisioned environment. However, local-static-provisioner[73] is not suitable for production use. You should use a different Kubernetes CSI[74] compatible storage that is suitable for production.

To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your newly created StorageClass by following the steps in the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/. You can choose from any of the storage options available for Kubernetes and make your storage choice[75] the default storage.

Ceph can also be used as CSI storage. Refer to that section in the documentation for information on how to use Rook Ceph (see page 901).

4.6.4.2.5 GCP CSI Driver

This driver allows volumes backed by Google Cloud Filestore instances to be dynamically created and mounted by workloads.

- Driver documentation: gcp-filestore-csi-driver[76]
- Persistent volumes and dynamic provisioning: volume types[77]

4.6.4.3 Related Information

- Kubernetes Storage[78]
- Kubernetes CSI Storage Drivers[79]
- Kubernetes Local Persistent Volumes[80]
- DKP vSphere Storage (see page 381)

4.6.5 Provision a Static Local Volume

Learn how to provision a static local volume for a DKP cluster

72 https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner
73 https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner
74 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
75 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
76 https://github.com/kubernetes-sigs/gcp-filestore-csi-driver
77 https://cloud.google.com/kubernetes-engine/docs/concepts/persistent-volumes
78 https://kubernetes.io/docs/concepts/storage/
79 https://kubernetes-csi.github.io/docs/drivers.html
80 https://kubernetes.io/blog/2019/04/04/kubernetes-1.14-local-persistent-volumes-ga/

When creating a pre-provisioned infrastructure cluster, DKP uses `localvolumeprovisioner` as the default storage provider (see page 103). However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[81] compatible storage that is suitable for production.

You can choose from any of the storage options available for Kubernetes. To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your newly created StorageClass by following the steps in the Kubernetes documentation called Changing the Default Storage Class[82].

The `localvolumeprovisioner` component uses the local volume static provisioner[83] to manage persistent volumes for pre-allocated disks. It does this by watching the `/mnt/disks` folder on each host and creating persistent volumes in the `localvolumeprovisioner` storage class for each disk that it discovers in this folder.

- Persistent volumes with a 'Filesystem' volume-mode are discovered if you mount them under `/mnt/disks`.
- Persistent volumes with a 'Block' volume-mode are discovered if you create a symbolic link to the block device in `/mnt/disks`.

For additional DKP documentation regarding StorageClass, see: Default Storage Providers in DKP (see page 103)

4.6.5.1 Before you Begin

Before starting, verify the following:

- You have access to a Linux, macOS, or Windows computer with a supported operating system version.
- You have a provisioned `dkp` cluster that uses the `localvolumeprovisioner` platform application, but have not added any other Kommander applications to the cluster yet.

This distinction between provisioning and deployment is important because some applications depend on the storage class provided by the `localvolumeprovisioner` component and can fail to start if it is not configured yet.

4.6.5.2 Provision the Cluster and a Volume

1. Create a pre-provisioned cluster by following the steps outlined in the Pre-provisioned Infrastructure (see page 1078) topic.

81 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
82 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/
83 https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner

As volumes are created/mounted on the nodes, the local volume provisioner detects each volume in the `/mnt/disks` directory and adds it as a persistent volume with the `localvolumeprovisioner` storage class. For more information, see the documentation regarding Kubernetes Local Storage[84].

2. Create at least one volume in `/mnt/disks` on each host.

   For example, mount a `tmpfs` volume:

   ```
   mkdir -p /mnt/disks/example-volume && mount -t tmpfs example-volume /mnt/disks/
   example-volume
   ```

3. Verify the persistent volume by running the following command:

   ```
   kubectl get pv
   ```

   The command displays output similar to the following:

   ```
   NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
   CLAIM   STORAGECLASS             REASON   AGE
   local-pv-4c7fc8ba   3986Mi     RWO            Delete           Available
   localvolumeprovisioner            2s
   ```

4. Claim the persistent volume using a PersistentVolumeClaim, by running the following command:

   ```
   cat <<EOF | kubectl create -f -
   kind: PersistentVolumeClaim
   apiVersion: v1
   metadata:
     name: example-claim
   spec:
     accessModes:
     - ReadWriteOnce
     resources:
       requests:
         storage: 100Mi
     storageClassName: localvolumeprovisioner
   EOF
   ```

5. Reference the persistent volume claim in a pod by running the following command:

   ```
   cat <<EOF | kubectl create -f -
   ```

84 https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner/blob/master/docs/operations.md#link-devices-into-directory-to-be-discovered-as-block-pvs

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-persistent-volume
spec:
  containers:
    - name: frontend
      image: nginx
      volumeMounts:
        - name: data
          mountPath: "/var/www/html"
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: example-claim
EOF
```

6. Verify the persistent volume claim by running the following command:

   ```
   kubectl get pvc
   ```

   The command displays output similar to the following:

   ```
   NAME            STATUS   VOLUME              CAPACITY   ACCESS MODES
   STORAGECLASS             AGE
   example-claim   Bound    local-pv-4c7fc8ba   3986Mi     RWO
   localvolumeprovisioner   78s
   ```

   And you can also check the persistent volume:

   ```
   NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
   CLAIM                   STORAGECLASS             REASON   AGE
   local-pv-4c7fc8ba   3986Mi     RWO            Delete           Bound
   default/example-claim   localvolumeprovisioner            15m
   ```

Upon deletion of the persistent volume claim, the corresponding persistent volume resource uses the *Delete* reclaim policy, which removes all data on the volume.

4.7 Resource Requirements

To ensure a successful DKP install, there are certain requirements to be met in regards to Control Plan nodes as well as Worker nodes. These resource requirements can be slightly different for different Infrastructure Providers.

- Managed Cluster Requirements (see page 112)
- Management Cluster Application Requirements (see page 113)
- Workspace Platform Application Defaults and Resource Requirements (see page 114)

The specific infrastructure provider resources are listed in their install sections which are referenced below.

4.7.1 General Resource Requirements

To Install DKP (see page 117) with the correct amount of resources, please see the list below before beginning installation.

4.7.2 Control Plane Nodes

You must have at least three control plane nodes. Each control plane node should have at least:

- 4 cores
- 16 GiB memory
- Approximately 80 GiB of free space for the volume used for `/var/lib/kubelet` and `/var/lib/containerd`.
- Disk usage must be below 85% on the root volume.

4.7.3 Worker Nodes

You must have at least four worker nodes. The specific number of worker nodes required for your environment can vary depending on the cluster workload and size of the nodes. Each worker node should have at least:

- 8 cores
- 32 GiB memory
- Around 80 GiB of free space for the volume used for `/var/lib/kubelet` and `/var/lib/containerd`.
- Disk usage must be below 85% on the root volume.

If you use the instructions to create a cluster using the DKP default settings without any edits to configuration files or additional flags, your cluster is deployed on an Ubuntu 20.04 operating system image (see page 62) with 3 control plane nodes, and 4 worker nodes which match the requirements above.

4.7.4 Infrastructure Provider Specific

For Specific Infrastructure Providers additional requirements may apply. For example, DKP on Azure defaults to deploying a `Standard_D4s_v3` virtual machine with an 128 GiB volume for the OS and an 80GiB volume for etcd storage, which meets the above requirements. See Main Page for Provider for infrastructure provider specific additional resource information:

- Pre-provisioned Install Options (see page 125)
- AWS Install Options (see page 261)

- Azure Install Options (see page 465)
- vSphere Install Options (see page 378)

4.7.5 Kommander Component Requirements:

- Management Cluster Application Requirements (see page 113)
- Workspace Platform Application Defaults and Resource Requirements (see page 114)

4.7.5.1 Next Topic:

Install Overview (see page 117)

4.7.6 Managed Cluster Requirements

To create additional clusters that will be managed by your Management cluster (see page 97), please ensure you have at least the minimal recommendation of required resources below.

4.7.6.1 Minimum Recommendation for Managed Clusters:

Worker Nodes - At least three worker nodes with:

- 8 CPU cores each
- 12Gi of memory
- A storage class and disk mounts that can accommodate at least four persistent volumes

Control Plane Nodes:

- 8 CPU cores each
- 12Gi of memory

NOTE: One control plane node is acceptable for a non-critical test environment, but any production workload must have at least three control planes.

The cluster needs:

- default Storage Slass and four volumes of 32G, 32G, 2G, and 100G or the ability to create those volumes depending on the Storage Class:

```
$ kubectl get pv -A
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY
STATUS   CLAIM
STORAGECLASS   REASON   AGE
```

```
pvc-08de8c06-bd66-40a3-9dd4-b0aece8ccbe8   32Gi       RWO            Delete
Bound    kommander-default-workspace/kubecost-cost-analyzer
ebs-sc                  124m
pvc-64552486-7f4c-476a-a35d-19432b3931af   32Gi       RWO            Delete
Bound    kommander-default-workspace/kubecost-prometheus-server
ebs-sc                  124m
pvc-972c3ee3-20bd-449b-84d9-25b7a06a6630   2Gi        RWO            Delete
Bound    kommander-default-workspace/kubecost-prometheus-alertmanager
ebs-sc                  124m
pvc-98ab93f1-2c2f-46b6-b7d3-505c55437fbb   100Gi      RWO            Delete
Bound    kommander-default-workspace/db-prometheus-kube-prometheus-stack-prometheus-0
  ebs-sc                  123m
```

NOTE: Actual workloads may demand more resources depending on usage.

4.7.7 Management Cluster Application Requirements

4.7.7.1 Management cluster application minimum resources and storage requirements

This section only details requirements for management cluster-specific applications in the Kommander component. For the list of all platform applications, see Platform Application Configuration Requirements (see page 564).

This table describes the workspace platform applications specific to the management cluster, minimum resource requirements, and minimum persistent storage requirements.

Common Name	App ID (for App versions, see the Release Notes[85])	Deployed by default	Minimum Resources Suggested	Minimum Persistent Storage Required
Centralized Grafana	centralized-grafana	Yes	cpu: 200m memory: 100Mi	
Centralized Kubecost	centralized-kubecost	Yes	cpu: 1200m memory: 4151Mi	# of PVs: 1 PV sizes: 32Gi

85 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/110723415/DKP+2.4.0+Components+and+Applications#applications

Common Name	App ID (for App versions, see the Release Notes)	Deployed by default	Minimum Resources Suggested	Minimum Persistent Storage Required
Chartmuseum	chartmuseum	Yes		# of PVs: 1 PV sizes: 2Gi
Dex	dex	Yes	cpu: 100m memory: 50Mi	
Dex Authenticator	dex-k8s-authenticator	Yes	cpu: 100m memory: 128Mi	
Gitea	gitea	Yes	cpu: 500m memory: 512Mi	# of PVs: 2 PV sizes: 10Gi
Karma	karma	Yes		
Flux	kommander-flux	Yes	cpu: 5000m memory: 5Gi	
Kubefed	kubefed	Yes	cpu: 300m memory: 192Mi	
Thanos	thanos	Yes		
Traefik ForwardAuth	traefik-forward-auth-mgmt	Yes	cpu: 100m memory: 128Mi	

4.7.8 Workspace Platform Application Defaults and Resource Requirements

The following table provides a list of platform applications that are available in DKP with the Kommander component. Some of them are deployed by default on attachment, others require manual installation (see page 564).

Workspace platform applications require more resources than solely deploying or attaching clusters into a workspace. Your cluster must have sufficient resources when deploying or attaching to ensure that the platform services are installed successfully.

The following table describes all the workspace platform applications that are available to the clusters in a workspace, minimum resource requirements, and whether they are enabled by default.

Common Name	App ID (for App versions, see the Release Notes (see page 1544))	Deployed by default	Minimum Resources Suggested	Minimum Persistent Storage Required
Cert Manager	cert-manager	Yes	cpu: 10m memory: 32Mi	
External DNS	external-dns	No		
Fluent Bit	fluent-bit	No	cpu: 350m memory: 350Mi	
Gatekeeper	gatekeeper	Yes	cpu: 300m memory: 768Mi	
Grafana	grafana-logging	No	cpu: 200m memory: 100Mi	
Loki	grafana-loki	No		# of PVs: 8 PV sizes: 10Gi x 8 (total: 80Gi)
Istio	istio	No	cpu: 1270m memory: 4500Mi	
Jaeger	jaeger	No		
Kiali	kiali	No	cpu: 20m memory: 128Mi	
Knative	knative	No	cpu: 610m memory: 400Mi	
Kube OIDC Proxy	kube-oidc-proxy	Yes		
Kube Prometheus Stack	kube-prometheus-stack	Yes	cpu: 1300m memory: 4300Mi	# of PVs: 1 PV sizes: 100Gi

Common Name	App ID (for App versions, see the Release Notes (see page 1544))	Deployed by default	Minimum Resources Suggested	Minimum Persistent Storage Required
Kubecost	kubecost	Yes	cpu: 700m memory: 1700Mi	# of PVs: 3 PV sizes: 2Gi, 32Gi, 32Gi (total: 66Gi)
Kubernetes Dashboard	kubernetes-dashboard	Yes	cpu: 250m memory: 300Mi	
Logging Operator	logging-operator	No	cpu: 350m * # of nodes + 600m memory: 228Mi + 350Mi * # of nodes	# of PVs: 1 PV sizes: 10Gi
NFS Server Provisioner	nfs-server-provisioner	No	# of PVs: 1 PV sizes: 100Gi	
NVIDIA GPU Operator	nvidia-gpu-operator	No	cpu: 100m memory: 128Mi	
Prometheus Adapter	prometheus-adapter	Yes	cpu: 1000m memory: 1000Mi	
Reloader	reloader	Yes	cpu: 100m memory: 128Mi	
Rook Ceph	rook-ceph	Yes	cpu: 100m memory: 128Mi	
Rook Ceph Cluster	rook-ceph-cluster	Yes	cpu 2500m mem 8Gi	# of PVs: 4 PV sizes: 40Gi
Traefik	traefik	Yes	cpu: 500m	

Common Name	App ID (for App versions, see the Release Notes (see page 1544))	Deployed by default	Minimum Resources Suggested	Minimum Persistent Storage Required
Traefik ForwardAuth	traefik-forward-auth	Yes	cpu: 100m memory: 128Mi	
Velero	velero	No	cpu: 1000m memory: 1024Mi	

- Currently, DKP only supports a single deployment of `cert-manager` per cluster. Because of this, `cert-manager` cannot be installed on any Konvoy managed clusters or clusters that come with `cert-manager` pre-installed.
- Only a single deployment of `traefik` per cluster is supported.
- DKP automatically manages the deployment of `traefik-forward-auth` and `kube-oidc-proxy` when clusters are attached to the workspace. These applications are not shown in the DKP UI.
- Applications are enabled in DKP and then deployed to attached clusters. To confirm that your enabled application has successfully deployed, you should verify via the CLI (see page 596).

4.7.8.1 Next Topic:

Install Overview (see page 117)

4.8 Install Overview

4.8.1 Prepare Your Environment for Install:

Follow the steps below to install the basic package requirements for your environment to achieve a successful install of DKP. Then install DKP and finally you can begin any custom configuration based on your environment.

1. Install required packages. In most cases, you can install the required software using your preferred package manager. For example, on a macOS computer, you can use Homebrew[86] to install `kubectl` and the `aws` command-line utility by running the following command:

   ```
   brew install kubernetes-cli awscli
   ```

2. Check the Kubernetes client version. Many important Kubernetes functions **do not work** if your client is outdated. You can verify that the version of `kubectl` you have installed is supported by running the following command:

   ```
   kubectl version --short=true
   ```

3. For air-gapped, create a bastion host (see page 1162) for the cluster nodes to use within the air-gapped network. This bastion host needs access to a Docker registry in lieu of an Internet connection for pulling Docker images. The recommended template naming pattern is `../folder-name/dkp-e2e-bastion-template` or similar. Each infrastructure provider has its own set of bastion host instructions. Refer to your provider's site for details - Azure[87], AWS[88], GCP[89], or vSphere[90].
4. For creating DKP machine images, you need to download Konvoy Image Builder (see page 71) and extract it.
5. To download DKP, see the Download (see page 71) topic for information. You will need to download and extract the DKP binary package tarball.
6. Verify you have valid **cloud provider security credentials** to deploy the cluster on that platform.

> This step regarding provider security credentials is not required if you are installing DKP on an on-premises environment. For information about installing in an on-premises environment, see Install Pre-provisioned (see page 1078).

7. Install Konvoy depending on which infrastructure you have. Consult the Day 1 - Basic Installs by Infrastructure (see page 124) for provider specific installs. To use customized YAML and other advanced features, consult the Additional Infrastructure Configuration (see page 919) section of the documentation for steps on creating a customized cluster with the Konvoy component of DKP using YAML. Find your infrastructure provider listed in that section.

86 https://docs.brew.sh/Installation
87 https://learn.microsoft.com/en-us/azure/bastion/quickstart-host-portal
88 https://aws.amazon.com/solutions/implementations/linux-bastion/
89 https://blogs.vmware.com/cloud/2021/06/02/intro-google-cloud-vmware-engine-bastion-host-access-iap/
90 https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.security.doc/GUID-6975426F-56D0-4FE2-8A58-580B40D2F667.html

8. Now that you have a basic Kubernetes cluster installed through Konvoy, configure the Kommander (see page 1227) component.
9. Lastly, continue with initializing the configuration file under the Kommander Installer Configuration File (see page 1227) component of DKP.

You may also want to test operations by deploying a simple, sample application, customizing the cluster configuration, or checking the status of cluster components.

4.8.2 Next Step:

Prerequisites for Install (see page 119)

4.9 Prerequisites for Install

Before you create a DKP image and deploy the initial DKP cluster, the operator's machine is required to be either an OSX or Linux based machine of a supported version. Ensure you have all the prerequisites met for both the Konvoy and Kommander component.

> Additional prerequisites are necessary for air-gapped, so verify that you have all non-air-gapped met and then any additional air-gapped prerequisites as well.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)

4.9.1 Prerequisites for the Konvoy Component

For DKP and Konvoy Image Builder to run, the operator machine requires:

4.9.1.1 Non-air-gapped (all environments)

In a non-air-gapped environment, your environment has two-way access to and from the **Internet**. Below are the prerequisites required if installing in a non-air-gapped environment:

- An x86_64-based Linux or macOS machine.
- The `dkp` binary on the bastion by downloading (see page 71). To check which version of DKP you installed for compatibility reasons, run the `dkp version -o` command (dkp version (see page 1534)).
- A Container engine/runtime installed is required to install DKP:

  - Version Docker®[91] container engine version 18.09.2 or higher installed for Linux or MacOS - On macOS, Docker runs in a virtual machine which needs configured with at least 8 GB of memory.
  - Version 4.0 of Podman[92] or higher for Linux. Host requirements found here: Host Requirements[93]

On macOS, Docker runs in a virtual machine. Configure this virtual machine with at least 8 GB of memory.

- kubectl[94] for interacting with the running cluster on the host where the DKP Konvoy command line interface (CLI) runs.
- Konvoy Image Builder (see page 1282) (KIB)
- Valid Provider account with credentials configured.
  - AWS credentials configured[95] that can manage CloudFormation Stacks, IAM Policies, IAM Roles, and IAM Instance Profiles
  - Azure credentials configured[96]
- CLI tooling of the cloud provider being used to deploy DKP commands:
  - aws-cli[97]
  - googlecloud-cli[98]
  - azure-cli[99]
- EKS and AKS only a self-managed cluster is required:
  - If you follow Day 1 - Basic Installation (see page 124) instructions for install, you use the `--self-managed` flag and therefore have a self-managed cluster. If you use the instructions under Additional Infrastructure Configuration (see page 919), ensure you perform the self-managed process on your new cluster:
  - A Self-managed AWS Cluster (see page 976)
  - A Self-managed Azure Cluster (see page 1054)
- Pre-provisioned on Premises only:
  - Pre-provisioned hosts with SSH access enabled.
  - An unencrypted SSH private key, whose public key is configured on the above hosts.

91 https://docs.docker.com/get-docker/
92 https://podman.io/getting-started/installation
93 https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
94 https://kubernetes.io/docs/tasks/tools/#kubectl
95 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html
96 https://github.com/kubernetes-sigs/cluster-api-provider-azure/blob/master/docs/book/src/topics/getting-started.md#prerequisites
97 https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html
98 https://cloud.google.com/sdk/docs/install
99 https://learn.microsoft.com/en-us/cli/azure/install-azure-cli

    - Pre-provisioned Override Files (see page 1341) if needed.
- vSphere only:
    - A valid VMware vSphere account with credentials configured.

> DKP requires permissions for the cloud provider that is being used. See the Supported Infrastructure Operating Systems (see page 62) page for further setup requirements.

4.9.1.2 Air-gapped (**additional** prerequisites)

In an air-gapped environment, your environment is **isolated** from unsecured networks, like the Internet, and therefore require additional considerations for installation. Below are the **additional** prerequisites required if installing in an air-gapped environment:

- Linux machine (bastion) that has access to the existing VPC. (Instead of An x86_64-based Linux or macOS machine).
- Ability to download artifacts from the internet and then copy those onto your bastion machine.
- Download (see page 71) the Complete DKP Air-gapped Bundle for this release - `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz`.
- An existing local registry (see page 1349).

4.9.2 Prerequisites for Kommander Component

To install the Kommander component of DKP, ensure you meet the following prerequisites:

4.9.2.1 Non-air-gapped (all environments)

- The version of the CLI that matches the DKP version you want to install.
- Review the Management Cluster Application Requirements (see page 113) and Workspace Platform Application Defaults and Resource Requirements (see page 114) to ensure that your cluster will have sufficient resources.
- Ensure you have a default StorageClass (see page 1254) configured (the Konvoy component is responsible for configuring one)
- A load balancer to route external traffic. In cloud environments, this is provided by your cloud provider. In on-premises and vSphere deployments, you can configure MetalLB. It is possible to use Virtual IP (see page 1114) also. See Load Balancing (see page 854) topic for more details.

NOTE: If you want to customize your cluster's domain or certificate, ensure you review the respective documentation sections:

- Configure custom domains and certificates during the Kommander installation (see page 1230)
- Configure custom domains and certificates after Kommander has been installed (see page 765)

- For pre-provisioned on-premise environments:
  - Ensure you meet the Storage requirements (see page 101), default storage class (see page 103)and Application Storage (see page 894).
  - Ensure you have added at least 40 GB of raw storage to each of your worker nodes (see page 897) in your clusters.

4.9.2.2 Air-gapped (**additional** prerequisites)

In an air-gapped environment, your environment is **isolated** from unsecured networks, like the Internet, and therefore require additional considerations for installation. Below are the **additional** prerequisites required if installing in an air-gapped environment:

- A local registry (see page 1349) containing all the necessary installation images, including the Kommander images. See install scenarios for how to push the necessary images to this registry.
- Connectivity with the clusters attaching to the management cluster:
  - Both management and attached clusters must be able to connect to the Docker registry.
  - The management cluster must be able to connect to all attached cluster's API servers.
  - The management cluster must be able to connect to any load balancers created for platform services on the management cluster.

NOTE: If you want to customize your cluster's domain or certificate, ensure you review the respective documentation sections:

- Configure custom domains and certificates during the Kommander installation (see page 1230)
- Configure custom domains and certificates after Kommander has been installed (see page 765)

- For pre-provisioned on-premise environments:
  - Ensure you meet the Storage requirements (see page 896).

- Ensure you have added at least 40 GB of raw storage to each of your worker nodes (see page 897) in your cluster.

4.9.2.3 Next Step or Return:

If using the Day 1- Basic Install instructions, proceed (or return) to that section Day 1 - Basic Installs by Infrastructure (see page 124) combinations to install and setup DKP based on your infrastructure environment provider.

If using the Custom Install instructions, proceed (or return) to that section and select the infrastructure provider you are using under Additional Infrastructure Customized Configuration (see page 919).

4.9.3 Container Runtime

DKP supports both Podman and Docker as container engines for cluster creation. If you choose to use Podman, it works with Linux on DKP.

> Installation of Podman will be different for different OS environments, so please refer to the Podman Installation[100] site to install.

Version requirements to create a container engine:

- Version 4.0 of Podman[101] or higher for Linux. Host requirements found here: https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
- Version 20.10.0 of Docker[102] or higher for Linux or MacOS. On macOS, Docker runs in a virtual machine. Configure this VM with at least 8 GB of memory.

4.9.3.1 TIPS

- Set[103] `alias docker=podman` if desired to ensure any Docker specific commands will run without problem, but our testing shows it is not required.
- You will need to restart your machine after following the Host Requirements[104] to add the `/etc/systemd/system/user@.service.d/delegate.conf` and the `/etc/modules-load.d/iptables.conf` files.

100 https://podman.io/getting-started/installation
101 https://podman.io/getting-started/installation
102 https://www.docker.com/
103 https://docs.podman.io/en/latest/
104 https://kind.sigs.k8s.io/docs/user/rootless/

5 Day 1 - Basic Installs by Infrastructure

This section provides basic installation instructions for your infrastructure using combinations of providers and other variables. A “basic” cluster contains nodes and a running instance of DKP, but is not yet a *production* cluster. While you will not yet be ready to deploy a workload after these procedures, you will be able to get familiar with DKP and see the cluster structure. You can think of the install procedures in this section as a sort of quick start guide.

Depending on your selected infrastructure, and the requirements of your specific environment, there are additional configuration tasks (see page 919) to complete. Production cluster configuration lets you deploy and enable the Day 2 applications, and your workload applications, that you need for production operations.

For virtualized environments, DKP can provision the virtual machines necessary to run Kubernetes clusters. If you allow DKP to manage your infrastructure, look for your supported infrastructure provider installation choices below.

In bare metal environments or any time you would like to provision your nodes manually, see the corresponding Pre-provisioned section.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

5.1 Scenario Based Installation Instructions

5.2 Pre-provisioned Install Options

- Pre-provisioned Non-air-gapped Install (see page 127)
- Pre-provisioned Air-gapped Install (see page 145)
- Pre-provisioned FIPS Install (see page 167)
- Pre-provisioned FIPS Air-gapped Install (see page 186)
- Pre-provisioned with GPU Install (see page 209)
- Pre-provisioned Air-gapped with GPU Install (see page 231)

5.3 AWS Install Options

- AWS Install (see page 262)
- AWS Air-gapped Install (see page 274)

- AWS with FIPS Install (see page 289)
- AWS Air-gapped FIPS Install (see page 302)
- AWS with GPU Install (see page 316)
- AWS Air-gapped with GPU Install (see page 332)

5.4 EKS Install Options

- EKS Install (see page 351)

5.5 vSphere Install Options

- vSphere Prerequisites: All Installation Types (see page 379)
- vSphere Install (see page 383)
- vSphere Air-gapped Install (see page 402)
- vSphere FIPS Install (see page 424)
- vSphere FIPS Air-gapped Install (see page 443)

5.6 Azure Install Options

- Azure Install (see page 465)

5.7 AKS Install Options

- AKS Install (see page 478)

5.8 GCP Install Options

- GCP Install (see page 494)

5.9 Pre-provisioned Install Options

For an environment that is Pre-provisioned (see page 96) and either non-air-gapped or air-gapped, install options are provided for you in this one location. Remember, there are always more options in the Additional Infrastructure Configurations (see page 1078) section, but this will get you operative with the most common scenarios.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

Below are all the DKP supported environment combinations: (Refer to this page to check if your OS is a Supported Operating System (see page 62).)

- Pre-provisioned Non-air-gapped Install (see page 127)
- Pre-provisioned Air-gapped Install (see page 145)
- Pre-provisioned FIPS Install (see page 167)
- Pre-provisioned FIPS Air-gapped Install (see page 186)
- Pre-provisioned with GPU Install (see page 209)
- Pre-provisioned Air-gapped with GPU Install (see page 231)

Important Pre-provisioned Topics:

- Pre-provisioned includes on-premises, vSphere, AWS, Azure, and GCP infrastructures and is described in more detail under What is Pre-provisioned Infrastructure (see page 96).
- For more information regarding CSI Disk storage and changing default StorageClass, see Default Storage Providers in DKP (see page 103).
- For Azure environment using Pre-provisioned specifics, see Pre-provisioned Azure only Configurations (see page 1106)

5.9.1 Resources

Additional resource information specific to On Premises and Pre-provisioned is listed below:

- **Control plane machines** -
  - 15% free space on the root file system.
  - Multiple ports open, as described in DKP Ports (see page 60).
  - `firewalld` systemd service disabled. If it exists and is enabled, use the commands `systemctl stop firewalld` then `systemctl disable firewalld`, so that `firewalld` remains disabled after the machine restarts.

Swap is disabled. The `kubelet` does not have generally-available support for swap. Due to variable commands, refer to your operating system documentation.

- **Worker machines** -
  - 15% free space on the root file system
  - If you plan to use local volume provisioning to provide persistent volumes for your workloads, you must mount at least four volumes to the `/mnt/disks/` mount point on each machine. Each volume must have at least 55 GiB of capacity.

- Ensure your disk meets the resource requirements for Rook Ceph in `Block` mode for ObjectStorageDaemons[105] as specified in the requirements table (see page 897).
- Multiple ports open, as described in DKP Ports (see page 60).
- `firewalld` systemd service disabled. If it exists and is enabled, use the commands `systemctl stop firewalld` then `systemctl disable firewalld`, so that `firewalld` remains disabled after the machine restarts.

Swap is disabled. The `kubelet` does not have generally-available support for swap. Due to variable commands, refer to your operating system documentation.

5.9.2 Pre-provisioned Non-air-gapped Install

This section provides instructions to install DKP in a Pre-provisioned non-air-gapped environment.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

5.9.2.1 Next Step:

Pre-provisioned: Define Infrastructure (see page 127)

5.9.2.2 Pre-provisioned: Define Infrastructure

Define the cluster hosts and infrastructure

Konvoy needs to know how to access your cluster hosts. This is done using inventory resources. For initial cluster creation, you must define a control-plane and at least one worker pool.

5.9.2.2.1 Define your Infrastructure

1. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
```

105 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

```
export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
export WORKER_1_ADDRESS="<worker-address-1>"
export WORKER_2_ADDRESS="<worker-address-2>"
export WORKER_3_ADDRESS="<worker-address-3>"
export WORKER_4_ADDRESS="<worker-address-4>"
export SSH_USER="<ssh-user>"
export SSH_PRIVATE_KEY_SECRET_NAME="$CLUSTER_NAME-ssh-key"
```

2. Use the following template to help you define your infrastructure. The environment variables that you set in the previous step automatically replace the variable names when the inventory YAML file is created.

```
cat <<EOF > preprovisioned_inventory.yaml
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-control-plane
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    # Create as many of these as needed to match your infrastructure
    # Note that the command line parameter --control-plane-replicas determines
how many control plane nodes will actually be used.
    #
    - address: $CONTROL_PLANE_1_ADDRESS
    - address: $CONTROL_PLANE_2_ADDRESS
    - address: $CONTROL_PLANE_3_ADDRESS
  sshConfig:
    port: 22
    # This is the username used to connect to your infrastructure. This user
must be root or
    # have the ability to use sudo without a password
    user: $SSH_USER
    privateKeyRef:
      # This is the name of the secret you created in the previous step. It
must exist in the same
      # namespace as this inventory object.
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-md-0
  namespace: default
  labels:
```

```
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    - address: $WORKER_1_ADDRESS
    - address: $WORKER_2_ADDRESS
    - address: $WORKER_3_ADDRESS
    - address: $WORKER_4_ADDRESS
  sshConfig:
    port: 22
    user: $SSH_USER
    privateKeyRef:
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
EOF
```

5.9.2.2.2 Next Step:

Pre-provisioned: Define Control Plane Endpoint (see page 129)

5.9.2.3 Pre-provisioned: Define Control Plane Endpoint

5.9.2.3.1 Define the Control Plane Endpoint for your cluster as well as connection mechanism

A control plane should have three, five, or seven nodes, so it can remain available if one or more nodes fail. A control plane with one node should not be used in production.

In addition, the control plane should have an endpoint that remains available if some nodes fail.

```
                        -------- cp1.example.com:6443
                        |
lb.example.com:6443 ---------- cp2.example.com:6443
                        |
                        -------- cp3.example.com:6443
```

In this example, the control plane endpoint host is `lb.example.com`, and the control plane endpoint port is `6443`. The control plane nodes are `cp1.example.com`, `cp2.example.com`, and `cp3.example.com`. The port of each API server is `6443`.

5.9.2.3.2 Select your Connection Mechanism

A virtual IP is the address that the client uses to connect to the service. A load balancer is the device that distributes the client connections to the backend servers. Before you create a new DKP cluster, choose an external load balancer(LB) or virtual IP.

- **External load balancer**

It is recommended that an external load balancer be the control plane endpoint. To distribute request load among the control plane machines, configure the load balancer to send requests to all the control plane machines. Configure the load balancer to send requests only to control plane machines that are responding to API requests.

- **Built-in virtual IP**

If an external load balancer is not available, use the built-in virtual IP (see page 1114). The virtual IP is *not* a load balancer; it does not distribute request load among the control plane machines. However, if the machine receiving requests does not respond to them, the virtual IP automatically moves to another machine.

5.9.2.3.3 Single-Node Control Plane

Do not use a single-node control plane in a production cluster.

A control plane with one node can use its single node as the endpoint, so you will not require an external load balancer, or a built-in virtual IP. At least one control plane node must always be running. Therefore, to upgrade a cluster with one control plane node, a spare machine must be available in the control plane inventory. This machine is used to provision the new node before the old node is deleted.

Modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1351).

When the API server endpoints are defined, you can create the cluster using the link in Next Step below.

5.9.2.3.4 Known Limitations

Be aware of these limitations in the current release of DKP.

The control plane endpoint port is also used as the API server port on each control plane machine. The default port is 6443. Before you create the cluster, ensure the port is available for use on each control plane machine.

5.9.2.3.5 Next Step:

(2.5) Pre-provisioned: Create a New Cluster (see page 131)

5.9.2.4 Pre-provisioned: Create a Management Cluster

Create a new Pre-provisioned Kubernetes cluster in a non-air-gapped environment with the steps below.

5.9.2.4.1 Name Your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[106] for more naming information.

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-example>
```

5.9.2.4.2 Create a Kubernetes Cluster

Once you've defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster.

Before you create a new DKP cluster below, choose an external load balancer (LB) or virtual IP (see page 1114) and use the corresponding `dkp create cluster` command.

In a pre-provisioned environment, use the Kubernetes CSI and third party drivers for local volumes and other storage devices in your data center.

106 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

DKP uses local static provisioner as the default storage provider (see page 103) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[107] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory YAML (see page 127) previously created.

The create cluster command below includes the `--self-managed` flag. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

1. This command uses the default external load balancer (LB) option (see alternative Step 1 for virtual IP):

```
dkp create cluster preprovisioned \
  --cluster-name ${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --self-managed
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

1. **ALTERNATIVE Virtual IP** - if you don't have an external LB, and wish to use a VIRTUAL IP provided by kube-vip, specify these flags example below:

```
dkp create cluster preprovisioned \
    --cluster-name ${CLUSTER_NAME} \
    --control-plane-endpoint-host 196.168.1.10 \
```

107 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

```
    --virtual-ip-interface eth1
```

The output from this command is shortened here for reading clarity, but should start like this:

```
Generating cluster resources
cluster.cluster.x-k8s.io/preprovisioned-example created
cont.........
```

2. Use the wait command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

Output:

```
cluster.cluster.x-k8s.io/preprovisioned-example condition met
```

Depending on the cluster size, it will take a few minutes to create.

When the command completes, you will have a running Kubernetes cluster! Use this command to get the Kubernetes `kubeconfig` for the new cluster and proceed to installing the DKP Kommander UI:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

If changing the Calico encapsulation (see page 1110), D2iQ recommends changing it after cluster creation, but before production.

If you need to increase Docker Hub's rate limit[108], use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.

108 https://docs.docker.com/docker-hub/download-rate-limit/

5.9.2.4.3 Audit logs

To modify Control Plane Audit logs settings using the information contained in the page Configuring the Control Plane (see page 1351).

5.9.2.4.4 Further Steps:

For more customized cluster creation, access the Additional Infrastructure Configurations for Pre-Provisioned (see page 1078) section. That section is for Pre-Provisioned Override Files (see page 1341), custom flags, and more that specify the secret as part of the create cluster command. If these are not specified, the overrides for your nodes will not be applied.

5.9.2.4.5 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1360).

5.9.2.4.6 Next Step:

Pre-provisioned: Configure MetalLB (see page 134)

5.9.2.5 Pre-provisioned: Configure MetalLB

5.9.2.5.1 Create a MetalLB configmap for your pre-provisioned infrastructure.

Choose one of the following two protocols you want to use to announce service IPs. If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can continue the installation process with Pre-provisioned: Install Kommander (see page 136) .

To use MetalLB, create a MetalLB configMap for your Pre-provisioned infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

5.9.2.5.2 Layer 2 configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to ARP requests on your local network directly, to give the machine's MAC address to clients.

- MetalLB IP address ranges/CIDRs should be within the node's primary network subnet (see page 929).
- MetalLB IP address ranges/CIDRs and node subnet should not conflict with the Kubernetes cluster pod and service subnets.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

5.9.2.5.3 BGP configuration

For a basic configuration featuring one BGP router and one IP address range, you need 4 pieces of information:

- The router IP address that MetalLB should connect to,
- The router's AS number,
- The AS number MetalLB should use,
- An IP address range expressed as a CIDR prefix.

As an example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

5.9.2.5.3.1 Next Step:

Pre-provisioned: Install Kommander (see page 136)

5.9.2.6 Pre-provisioned: Install Kommander

5.9.2.6.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 119).
- Ensure you have a default StorageClass (see page 1254).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

5.9.2.6.2 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1227) for the deployment:

   ```
   dkp install kommander --init > kommander.yaml
   ```

4. Edit the installer file to include configuration overrides for the `rook-ceph-cluster`. DKP's default configuration ships Ceph with PVC based storage[109] which requires your CSI provider to support PVC with type `volumeMode: Block`. As this is not possible with the default local static provisioner (see page 103), you can install Ceph in host storage[110] mode.

   You can choose whether Ceph's object storage daemon (osd) pods should consume all or just some of the devices on your nodes. Include **one** of the following Overrides:

   a. To automatically assign all raw storage devices on all nodes to the Ceph cluster:

      ```
      rook-ceph-cluster:
          enabled: true
          values: |
            cephClusterSpec:
              storage:
                storageClassDeviceSets: []
                useAllDevices: true
                useAllNodes: true
                deviceFilter: "<<value>>"
      ```

   b. To assign specific storage devices on all nodes to the Ceph cluster:

      ```
      rook-ceph-cluster:
          enabled: true
          values: |
            cephClusterSpec:
              storage:
      ```

109 https://rook.io/docs/rook/v1.10/CRDs/Cluster/pvc-cluster/
110 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/

```
          storageClassDeviceSets: []
          useAllNodes: true
          useAllDevices: false
          deviceFilter: "^sdb."
```

Note: If you want to assign specific devices to specific nodes using the `deviceFilter` option, refer to Specific Nodes and Devices[111]. For general information on the `deviceFilter` value, refer to Storage Selection Settings[112].

5. *If required:* Customize your `kommander.yaml`.
   See Kommander Additional Install Configurations (see page 1227) for customization options. Some of them include:
   Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, etc.

5.9.2.6.3 Enable DKP Catalog Applications and Install Kommander

If you have an Enterprise license and would like to use the DKP Catalog Applications function, follow these steps:

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

   ```
   apiVersion: config.kommander.mesosphere.io/v1alpha1
   kind: Installation
   catalog:
     repositories:
       - name: dkp-catalog-applications
         labels:
           kommander.d2iq.io/project-default-catalog-repository: "true"
           kommander.d2iq.io/workspace-default-catalog-repository: "true"
           kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
         gitRepositorySpec:
           url: https://github.com/mesosphere/dkp-catalog-applications
           ref:
             tag: v2.5.0
   ```

   If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

111 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/#specific-nodes-and-devices
112 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.co
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 99)
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

5.9.2.6.4 Next Step:

Pre-provisioned: Verify Install and Log in to UI (see page 139)

5.9.2.7 Pre-provisioned: Verify Install and Log in to UI

Verify Kommander Install and Log in to the Dashboard UI

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander. After the CLI successfully installs the components, it will wait for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Released helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Released` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

5.9.2.7.1 Failed HelmReleases

If any application fails to successfully deploy, you can check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

5.9.2.7.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 524). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 524):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

5.9.2.7.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 510) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

5.9.2.7.4 Next Step:

Pre-provisioned: Subsequent Cluster Creation (see page 142)

5.9.2.8 Pre-provisioned: Create Managed Clusters Using the DKP CLI

After initial cluster creation, you have the ability to create additional clusters from the CLI. In a previous step, the new cluster was created as Self-managed which allows it to be a Management cluster or a stand alone cluster. Subsequent new clusters are not self-managed as they will likely be Managed or Attached clusters to this Management Cluster.

When creating Managed clusters (see page 97), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 97)!

5.9.2.8.1 Choose a Workspace for the New Cluster

1. If you have an existing Workspace name, run this command to find the name:
    NOTE: If you need to create a new Workspace, follow the instructions to Create a Workspace[113].

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

5.9.2.8.2 Name Your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[114] for more naming information.

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

113 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919315/Workspaces#Create-a-Workspace
114 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-additional>
```

5.9.2.8.3 Create a Kubernetes Cluster

Once you've defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster.

> Before you create a new DKP cluster below, choose an external load balancer (LB) or virtual IP (see page 1114) and use the corresponding `dkp create cluster` command.

In a pre-provisioned environment, use the Kubernetes CSI and third party drivers for local volumes and other storage devices in your data center.

> DKP uses local static provisioner as the default storage provider (see page 103) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[115] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory YAML (see page 127) previously created.

1. This command uses the default external load balancer (LB) option:

```
dkp create cluster preprovisioned \
  --cluster-name ${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
```

115 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

```
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --namespace=${WORKSPACE_NAMESPACE}
```

2. Use the wait command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

Output:

```
cluster.cluster.x-k8s.io/preprovisioned-additional condition met
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

NOTE: Depending on the cluster size, it will take a few minutes to create.

5.9.2.8.4 Manually Attach a DKP CLI Cluster to the Management Cluster

1. Find out the `name` of the created `Cluster`, so you can reference it later:

```
kubectl -n <workspace_namespace> get clusters
```

2. Attach the cluster by creating a `KommanderCluster`:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: <cluster_name>
```

```
  namespace: <workspace_namespace>
spec:
  kubeconfigRef:
    name: <cluster_name>-kubeconfig
  clusterRef:
    capiCluster:
      name: <cluster_name>
EOF
```

5.9.2.8.5 Next Step:

Day 2 - Cluster Operations Management (see page 510)

5.9.3 Pre-provisioned Air-gapped Install

This section provides instructions to install DKP in a Pre-provisioned air-gapped environment.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

NOTE: For air-gapped, ensure you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz` , so you can extract the tarball on next page.

5.9.3.1 Next Step:

Pre-provisioned Air-gapped: Configure Environment (see page 145)

5.9.3.2 Pre-provisioned Air-gapped: Configure Environment

The instructions below outline how to fulfill the requirements for a pre-provisioned infrastructure when using an air-gapped environment. In order to create a cluster, you must first setup the environment with necessary artifacts. All artifacts for Pre-provisioned Air-gapped environments need to get onto the bastion host. Artifacts needed by nodes must be unpacked and distributed on the bastion before other provisioning will work in the absence of an internet connection.

There is a new complete DKP air-gapped bundle available to download (see page 71) which contains all the DKP components needed for air-gapped installation. (i.e. `dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz`)

5.9.3.2.1 Setup Process:

1. The bootstrap image must be extracted and loaded onto the bastion host (see page 1162).
2. Artifacts must be copied onto cluster hosts for nodes to access.
3. If using GPU, those artifacts must be positioned locally.
4. Docker Registry seeded with images locally.

5.9.3.2.2 Load the Bootstrap Image

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz` from the download site (see page 71) mentioned above, extract the tarball to a local directory:

```
tar -xzvf dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz && cd dkp-v2.5.2
```

2. Load the bootstrap Docker image on your bastion machine:

```
docker load -i konvoy-bootstrap-image-v2.5.2.tar
```

5.9.3.2.3 Copy Air-gapped Artifacts onto Cluster Hosts

Using the Konvoy Image Builder (see page 1282), you can copy the required artifacts onto your cluster hosts.

Assuming you have downloaded `dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz` , continue below:

1. The kubernetes image bundle will be located in `kib/artifacts/images` and you will want to verify image and artifacts.
   a. Verify the image bundles exist in `artifacts/images` :

```
$ ls artifacts/images/
kubernetes-images-1.25.4-d2iq.1.tar kubernetes-images-1.25.4-d2iq.1-
fips.tar
```

   b. Verify the artifacts for your OS exist in the `artifacts/` directory and export the appropriate variables:

```
$ ls artifacts/
```

```
1.25.4_centos_7_x86_64.tar.gz       1.25.4_redhat_8_x86_64_fips.tar.gz
containerd-1.6.17-d2iq.1-rhel-7.9-x86_64.tar.gz   containerd-1.6.17-
d2iq.1-rhel-8.6-x86_64_fips.tar.gz  pip-packages.tar.gz
1.25.4_centos_7_x86_64_fips.tar.gz  1.25.4_rocky_9_x86_64.tar.gz
containerd-1.6.17-d2iq.1-rhel-7.9-x86_64_fips.tar.gz  containerd-1.6.17-
d2iq.1-rocky-9.0-x86_64.tar.gz
1.25.4_redhat_7_x86_64.tar.gz       1.25.4_ubuntu_20_x86_64.tar.gz
containerd-1.6.17-d2iq.1-rhel-8.4-x86_64.tar.gz   containerd-1.6.17-
d2iq.1-rocky-9.1-x86_64.tar.gz
1.25.4_redhat_7_x86_64_fips.tar.gz  containerd-1.6.17-d2iq.1-centos-7.9-
x86_64.tar.gz       containerd-1.6.17-d2iq.1-rhel-8.4-x86_64_fips.tar.gz
containerd-1.6.17-d2iq.1-ubuntu-20.04-x86_64.tar.gz
1.25.4_redhat_8_x86_64.tar.gz       containerd-1.6.17-d2iq.1-centos-7.9-
x86_64_fips.tar.gz  containerd-1.6.17-d2iq.1-rhel-8.6-x86_64.tar.gz
images
```

c. For example, for RHEL 8.4 you would set:

```
export OS_PACKAGES_BUNDLE=1.25.4_redhat_8_x86_64.tar.gz
export CONTAINERD_BUNDLE=containerd-1.6.17-d2iq.1-rhel-8.4-x86_64.tar.gz
```

2. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
export WORKER_1_ADDRESS="<worker-address-1>"
export WORKER_2_ADDRESS="<worker-address-2>"
export WORKER_3_ADDRESS="<worker-address-3>"
export WORKER_4_ADDRESS="<worker-address-4>"
export SSH_USER="<ssh-user>"
export SSH_PRIVATE_KEY_FILE="<private key file>"
```

`SSH_PRIVATE_KEY_FILE` must be either the name of the SSH private key file in your working directory or an absolute path to the file in your user's home directory.

3. Generate an `inventory.yaml` which is automatically picked up by the `konvoy-image upload` in the next step.

```
cat <<EOF > inventory.yaml
all:
  vars:
    ansible_user: $SSH_USER
    ansible_port: 22
    ansible_ssh_private_key_file: $SSH_PRIVATE_KEY_FILE
  hosts:
    $CONTROL_PLANE_1_ADDRESS:
```

```
      ansible_host: $CONTROL_PLANE_1_ADDRESS
    $CONTROL_PLANE_2_ADDRESS:
      ansible_host: $CONTROL_PLANE_2_ADDRESS
    $CONTROL_PLANE_3_ADDRESS:
      ansible_host: $CONTROL_PLANE_3_ADDRESS
    $WORKER_1_ADDRESS:
      ansible_host: $WORKER_1_ADDRESS
    $WORKER_2_ADDRESS:
      ansible_host: $WORKER_2_ADDRESS
    $WORKER_3_ADDRESS:
      ansible_host: $WORKER_3_ADDRESS
    $WORKER_4_ADDRESS:
      ansible_host: $WORKER_4_ADDRESS
EOF
```

4. Upload the artifacts onto cluster hosts with the following command:

```
konvoy-image upload artifacts \
             --container-images-dir=./artifacts/images/ \
             --os-packages-bundle=./artifacts/$OS_PACKAGES_BUNDLE \
             --containerd-bundle=artifacts/$CONTAINERD_BUNDLE \
             --pip-packages-bundle=./artifacts/pip-packages.tar.gz
```

KIB uses variable overrides (see page 1330) to specify base image and container images to use in your new machine image. The variable overrides files for NVIDIA and FIPS can be ignored unless adding an overlay feature.

5.9.3.2.3.1 Next Step:

Pre-provisioned Air-gapped: Load the Registry (see page 148)

5.9.3.3 Pre-provisioned Air-gapped: Load the Registry

Before creating a Kubernetes cluster, you need the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1162) and either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1349) page for more information.

1. Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz`, extract the tarball to a local directory:

```
tar -xzvf dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz && cd dkp-v2.5.0
```

2. Set an environment variable with your registry address with this command:

```
export REGISTRY_ADDRESS=<registry-address>:<registry-port>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

3. Execute the following command to load the air-gapped image bundle into your private registry:

```
dkp push image-bundle --image-bundle ./container-images/konvoy-image-bundle-
v2.5.0.tar --to-registry $REGISTRY_ADDRESS --to-registry-username
$REGISTRY_USERNAME --to-registry-password $REGISTRY_PASSWORD
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

To increase Docker Hub's rate limit[116] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.

5.9.3.3.1 Kommander Load Images

If you are operating in an air-gapped environment, a local container registry (see page 1349) containing all the necessary installation images, including the Kommander images is required. See below for how to push the necessary images to this registry.

For the basic air-gapped `kommander` image bundle, run the command below:

1. Run the following command to load the image bundle:

```
dkp push image-bundle --image-bundle ./container-images/kommander-image-bundle-
v2.5.0.tar --to-registry $REGISTRY_ADDRESS
```

For **DKP Catalog Applications**, also perform this image load:

116 https://docs.docker.com/docker-hub/download-rate-limit/

1. Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push image-bundle --image-bundle ./container-images/dkp-catalog-
applications-image-bundle-v2.5.0.tar --to-registry $REGISTRY_ADDRESS
```

5.9.3.3.2 Next Step:

Pre-provisioned Air-gapped: Define Infrastructure (see page 150)

5.9.3.4 Pre-provisioned Air-gapped: Define Infrastructure

Define the cluster hosts and infrastructure

Konvoy needs to know how to access your cluster hosts. This is done using inventory resources. For initial cluster creation, you must define a control-plane and at least one worker pool.

5.9.3.4.1 Define your Infrastructure

1. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
export WORKER_1_ADDRESS="<worker-address-1>"
export WORKER_2_ADDRESS="<worker-address-2>"
export WORKER_3_ADDRESS="<worker-address-3>"
export WORKER_4_ADDRESS="<worker-address-4>"
export SSH_USER="<ssh-user>"
export SSH_PRIVATE_KEY_SECRET_NAME="$CLUSTER_NAME-ssh-key"
```

2. Use the following template to help you define your infrastructure. The environment variables that you set in the previous step automatically replace the variable names when the inventory YAML file is created.

```
cat <<EOF > preprovisioned_inventory.yaml
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-control-plane
  namespace: default
  labels:
```

```
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    # Create as many of these as needed to match your infrastructure
    # Note that the command line parameter --control-plane-replicas determines
how many control plane nodes will actually be used.
    #
    - address: $CONTROL_PLANE_1_ADDRESS
    - address: $CONTROL_PLANE_2_ADDRESS
    - address: $CONTROL_PLANE_3_ADDRESS
  sshConfig:
    port: 22
    # This is the username used to connect to your infrastructure. This user
must be root or
    # have the ability to use sudo without a password
    user: $SSH_USER
    privateKeyRef:
      # This is the name of the secret you created in the previous step. It
must exist in the same
      # namespace as this inventory object.
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-md-0
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    - address: $WORKER_1_ADDRESS
    - address: $WORKER_2_ADDRESS
    - address: $WORKER_3_ADDRESS
    - address: $WORKER_4_ADDRESS
  sshConfig:
    port: 22
    user: $SSH_USER
    privateKeyRef:
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
EOF
```

5.9.3.4.2 Next Step:

Pre-provisioned Air-gapped: Define Control Plane Endpoint (see page 152)

5.9.3.5 Pre-provisioned Air-gapped: Define Control Plane Endpoint

5.9.3.5.1 Define the Control Plane Endpoint for your cluster as well as connection mechanism

A control plane should have three, five, or seven nodes, so it can remain available if one or more nodes fail. A control plane with one node should not be used in production.

In addition, the control plane should have an endpoint that remains available if some nodes fail.

```
                        -------- cp1.example.com:6443
                        |
    lb.example.com:6443 ---------- cp2.example.com:6443
                        |
                        -------- cp3.example.com:6443
```

In this example, the control plane endpoint host is `lb.example.com`, and the control plane endpoint port is `6443`. The control plane nodes are `cp1.example.com`, `cp2.example.com`, and `cp3.example.com`. The port of each API server is `6443`.

5.9.3.5.2 Select your Connection Mechanism

A virtual IP is the address that the client uses to connect to the service. A load balancer is the device that distributes the client connections to the backend servers. Before you create a new DKP cluster, choose an external load balancer(LB) or virtual IP.

- **External load balancer**

It is recommended that an external load balancer be the control plane endpoint. To distribute request load among the control plane machines, configure the load balancer to send requests to all the control plane machines. Configure the load balancer to send requests only to control plane machines that are responding to API requests.

- **Built-in virtual IP**

If an external load balancer is not available, use the built-in virtual IP (see page 1114). The virtual IP is *not* a load balancer; it does not distribute request load among the control plane machines. However, if the machine receiving requests does not respond to them, the virtual IP automatically moves to another machine.

5.9.3.5.3 Single-Node Control Plane

Do not use a single-node control plane in a production cluster.

A control plane with one node can use its single node as the endpoint, so you will not require an external load balancer, or a built-in virtual IP. At least one control plane node must always be running. Therefore, to

upgrade a cluster with one control plane node, a spare machine must be available in the control plane inventory. This machine is used to provision the new node before the old node is deleted.

> Modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1351).

When the API server endpoints are defined, you can create the cluster using the link in Next Step below.

5.9.3.5.4 Known Limitations

> Be aware of these limitations in the current release of DKP.

The control plane endpoint port is also used as the API server port on each control plane machine. The default port is 6443. Before you create the cluster, ensure the port is available for use on each control plane machine.

5.9.3.5.5 Next Step:

Pre-provisioned Air-gapped: Create a New Cluster (see page 153)

5.9.3.6 Pre-provisioned Air-gapped: Create a Management Cluster

If your cluster is air-gapped or you have a local registry[117], you must provide additional arguments when creating the cluster. These tell the cluster where to locate the local registry to use by defining the URL.

```
export REGISTRY_URL=<https/http>://<registry-address>:<registry-port>
export REGISTRY_CA=<path to the CA on the bastion>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

- `REGISTRY_URL` : the address of an existing registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
- `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.

117 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/210829371/Registry+Mirror+Tools

- `REGISTRY_PASSWORD` : optional if username is not set.

5.9.3.6.1 Name Your Cluster

> The cluster name may only contain the following characters: `a-z` , `0-9` , `.` , and `-` . Cluster creation will fail if the name has capital letters. See Kubernetes[118] for more naming information.
>
> When specifying the `cluster-name` , you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-example>
```

> Before you create a new DKP cluster below, choose an external load balancer(LB) or virtual IP (see page 1114) and use the corresponding `dkp create cluster` command.

5.9.3.6.2 Create an Air-gapped Kubernetes Cluster

Once you've defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster.

> DKP uses local static provisioner as the default storage provider (see page 103) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[119] compatible storage that is suitable for production.

118 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/
119 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory.

The create cluster command below includes the `--self-managed` flag. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

1. The command uses the default external load balancer (see alternative Step 1 below for virtual IP):

```
dkp create cluster preprovisioned --cluster-name ${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --registry-mirror-url=${REGISTRY_URL} \
  --registry-mirror-cacert=${REGISTRY_CA} \
  --registry-mirror-username=${REGISTRY_USERNAME} \
  --registry-mirror-password=${REGISTRY_PASSWORD} \
  --self-managed
```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

1. **Virtual IP ALTERNATIVE** - if you don't have an external LB, and wish to use a VIRTUAL IP (see page 1114) provided by kube-vip, specify these flags example below:

```
dkp create cluster preprovisioned \
    --cluster-name ${CLUSTER_NAME} \
    --control-plane-endpoint-host 196.168.1.10 \
    --virtual-ip-interface eth1
```

   The output from this command is shortened here for reading clarity, but should start like this:

```
Generating cluster resources
cluster.cluster.x-k8s.io/preprovisioned-example created
cont.........
```

2. Use the wait command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

Output:

```
cluster.cluster.x-k8s.io/preprovisioned-example condition met
```

NOTE: Depending on the cluster size, it will take a few minutes to create.

When the command completes, you will have a running Kubernetes cluster! Use this command to get the Kubernetes `kubeconfig` for the new cluster and proceed to installing the DKP Kommander UI:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

If changing the Calico encapsulation (see page 1110), D2iQ recommends changing it after cluster creation, but before production.

5.9.3.6.3 Audit Logs

To modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1351).

5.9.3.6.4 Further Steps:

For more customized cluster creation, access the Pre-Provisioned Additional Configurations (see page 1078) section.

NOTE: The section mentioned above is for overrides for your clusters (see page 1341), custom flags, and more that specify the secret as part of the create cluster command. If these are not specified, the overrides for your nodes will not be applied.

5.9.3.6.5 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1360).

5.9.3.6.6 Next Step:

Pre-provisioned Air-gapped: Configure MetalLB (see page 157)

5.9.3.7 Pre-provisioned Air-gapped: Configure MetalLB

5.9.3.7.1 Create a MetalLB configmap for your pre-provisioned infrastructure.

Choose one of the following two protocols you want to use to announce service IPs. If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can continue the installation process Pre-provisioned Air-gapped: Install Kommander (see page 159).

To use MetalLB, create a MetalLB configMap for your Pre-provisioned infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

5.9.3.7.2 Layer 2 configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to ARP requests on your local network directly, to give the machine's MAC address to clients.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

5.9.3.7.3 BGP configuration

For a basic configuration featuring one BGP router and one IP address range, you need 4 pieces of information:

- The router IP address that MetalLB should connect to,
- The router's AS number,
- The AS number MetalLB should use,
- An IP address range expressed as a CIDR prefix.

As an example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
```

```
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

5.9.3.7.3.1 Next Step:

Pre-provisioned Air-gapped: Install Kommander (see page 159)

5.9.3.8 Pre-provisioned Air-gapped: Install Kommander

5.9.3.8.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 119) .
- Ensure you have a default StorageClass (see page 1254).
- Ensure you have loaded all necessary images for your configuration. See Load the Images into Your Registry (see page 1258).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

5.9.3.8.2 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1227) for the deployment:

```
dkp install kommander --init --airgapped > kommander.yaml
```

4. Edit the installer file to include configuration overrides for the `rook-ceph-cluster`. DKP's default configuration ships Ceph with PVC based storage[120] which requires your CSI provider to support PVC with type `volumeMode: Block`. As this is not possible with the default local static provisioner (see page 103), you can install Ceph in host storage[121] mode.

   You can choose whether Ceph's object storage daemon (osd) pods should consume all or just some of the devices on your nodes. Include **one** of the following Overrides:

   a. To automatically assign all raw storage devices on all nodes to the Ceph cluster:

   ```
   rook-ceph-cluster:
       enabled: true
       values: |
         cephClusterSpec:
           storage:
             storageClassDeviceSets: []
             useAllDevices: true
             useAllNodes: true
             deviceFilter: "<<value>>"
   ```

   b. To assign specific storage devices on all nodes to the Ceph cluster:

   ```
   rook-ceph-cluster:
       enabled: true
       values: |
         cephClusterSpec:
           storage:
             storageClassDeviceSets: []
             useAllNodes: true
             useAllDevices: false
             deviceFilter: "^sdb."
   ```

   Note: If you want to assign specific devices to specific nodes using the `deviceFilter` option, refer to Specific Nodes and Devices[122]. For general information on the `deviceFilter` value, refer to Storage Selection Settings[123].

5. *If required:* Customize your `kommander.yaml`.

   See Kommander Additional Install Configurations (see page 1227) for customization options. Some of them include:
   Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, etc.

120 https://rook.io/docs/rook/v1.10/CRDs/Cluster/pvc-cluster/
121 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/
122 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/#specific-nodes-and-devices
123 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

5.9.3.8.3 Enable DKP Catalog Applications and Install Kommander

If you have an Enterprise license and would like to use the DKP Catalog Applications function, follow these steps:

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.5.0
```

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.5.0.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.5.0.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.5.0.tar.gz
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 99)

- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

5.9.3.8.4 Next Step:

Pre-provisioned Air-gapped: Verify Install and Log in to UI (see page 162)

5.9.3.9 Pre-provisioned Air-gapped: Verify Install and Log in to UI

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander. After the CLI successfully installs the components, it will wait for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Released helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Released` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
```

```
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

5.9.3.9.1 Failed HelmReleases

If any application fails to successfully deploy, you can check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

5.9.3.9.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 524). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 524):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

5.9.3.9.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 510) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

5.9.3.9.4 Next Step:

Pre-provisioned Air-gapped: Subsequent Cluster Creation (see page 164)

5.9.3.10 Pre-provisioned Air-gapped: Create Managed Clusters Using the DKP CLI

After initial cluster creation, you have the ability to create additional clusters from the CLI. In a previous step, the new cluster was created as Self-managed which allows it to be a Management cluster or a stand alone cluster. Subsequent new clusters are not self-managed as they will likely be Managed or Attached clusters to this Management Cluster.

When creating Managed clusters (see page 97), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 97)!

5.9.3.10.1 Choose a Workspace for the New Cluster

1. If you have an existing Workspace name, run this command to find the name:
   ⚠️ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a Workspace[124].

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

5.9.3.10.2 Name Your Cluster

> The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[125] for more naming information.
>
> When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-additional>
```

5.9.3.10.3 Create a Kubernetes Cluster

Once you've defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster.

---

124 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919315/Workspaces#Create-a-Workspace
125 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

Before you create a new DKP cluster below, choose an external load balancer (LB) or virtual IP (see page 1114) and use the corresponding `dkp create cluster` command.

In a pre-provisioned environment, use the Kubernetes CSI and third party drivers for local volumes and other storage devices in your data center.

DKP uses local static provisioner as the default storage provider (see page 103) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[126] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory YAML (see page 127) previously created.

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

1. This command uses the default external load balancer (LB) option:

```
dkp create cluster preprovisioned --cluster-name ${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --registry-mirror-url=${REGISTRY_URL} \
  --registry-mirror-cacert=${REGISTRY_CA} \
  --registry-mirror-username=${REGISTRY_USERNAME} \
  --registry-mirror-password=${REGISTRY_PASSWORD} \
  --namespace=${WORKSPACE_NAMESPACE}
```

2. Use the wait command to monitor the cluster control-plane readiness:

126 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

Output:

```
cluster.cluster.x-k8s.io/preprovisioned-additional condition met
```

NOTE: Depending on the cluster size, it will take a few minutes to create.

5.9.3.10.4 Manually Attach a DKP CLI Cluster to the Management Cluster

1. Find out the `name` of the created `Cluster`, so you can reference it later:

```
kubectl -n <workspace_namespace> get clusters
```

2. Attach the cluster by creating a `KommanderCluster`:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: <cluster_name>
  namespace: <workspace_namespace>
spec:
  kubeconfigRef:
    name: <cluster_name>-kubeconfig
  clusterRef:
    capiCluster:
      name: <cluster_name>
EOF
```

5.9.3.10.5 Next Step:

Day 2 - Cluster Operations Management (see page 510)

5.9.4 Pre-provisioned FIPS Install

This section provides instructions to install DKP in a Pre-provisioned non-air-gapped environment using FIPS.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

5.9.4.1 Next Step:

Pre-provisioned FIPS: Define Infrastructure (see page 168)

5.9.4.2 Pre-provisioned FIPS: Define Infrastructure

Define the cluster hosts and infrastructure

Konvoy needs to know how to access your cluster hosts. This is done using inventory resources. For initial cluster creation, you must define a control-plane and at least one worker pool.

5.9.4.2.1 Define your Infrastructure

1. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
export WORKER_1_ADDRESS="<worker-address-1>"
export WORKER_2_ADDRESS="<worker-address-2>"
export WORKER_3_ADDRESS="<worker-address-3>"
export WORKER_4_ADDRESS="<worker-address-4>"
export SSH_USER="<ssh-user>"
export SSH_PRIVATE_KEY_SECRET_NAME="$CLUSTER_NAME-ssh-key"
```

2. Use the following template to help you define your infrastructure. The environment variables that you set in the previous step automatically replace the variable names when the inventory YAML file is created.

```
cat <<EOF > preprovisioned_inventory.yaml
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-control-plane
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
```

```
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    # Create as many of these as needed to match your infrastructure
    # Note that the command line parameter --control-plane-replicas determines
how many control plane nodes will actually be used.
    #
    - address: $CONTROL_PLANE_1_ADDRESS
    - address: $CONTROL_PLANE_2_ADDRESS
    - address: $CONTROL_PLANE_3_ADDRESS
  sshConfig:
    port: 22
    # This is the username used to connect to your infrastructure. This user
must be root or
    # have the ability to use sudo without a password
    user: $SSH_USER
    privateKeyRef:
      # This is the name of the secret you created in the previous step. It
must exist in the same
      # namespace as this inventory object.
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-md-0
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    - address: $WORKER_1_ADDRESS
    - address: $WORKER_2_ADDRESS
    - address: $WORKER_3_ADDRESS
    - address: $WORKER_4_ADDRESS
  sshConfig:
    port: 22
    user: $SSH_USER
    privateKeyRef:
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
EOF
```

5.9.4.2.2 Next Step:

Pre-provisioned FIPS: Define Control Plane Endpoint (see page 170)

5.9.4.3 Pre-provisioned FIPS: Define Control Plane Endpoint

5.9.4.3.1 Define the Control Plane Endpoint for your cluster as well as connection mechanism

A control plane should have three, five, or seven nodes, so it can remain available if one or more nodes fail. A control plane with one node should not be used in production.

In addition, the control plane should have an endpoint that remains available if some nodes fail.

```
                        -------- cp1.example.com:6443
                        |
lb.example.com:6443 ----------- cp2.example.com:6443
                        |
                        -------- cp3.example.com:6443
```

In this example, the control plane endpoint host is `lb.example.com`, and the control plane endpoint port is `6443`. The control plane nodes are `cp1.example.com`, `cp2.example.com`, and `cp3.example.com`. The port of each API server is `6443`.

5.9.4.3.2 Select your Connection Mechanism

A virtual IP is the address that the client uses to connect to the service. A load balancer is the device that distributes the client connections to the backend servers. Before you create a new DKP cluster, choose an external load balancer(LB) or virtual IP.

- **External load balancer**

It is recommended that an external load balancer be the control plane endpoint. To distribute request load among the control plane machines, configure the load balancer to send requests to all the control plane machines. Configure the load balancer to send requests only to control plane machines that are responding to API requests.

- **Built-in virtual IP**

If an external load balancer is not available, use the built-in virtual IP (see page 1114). The virtual IP is *not* a load balancer; it does not distribute request load among the control plane machines. However, if the machine receiving requests does not respond to them, the virtual IP automatically moves to another machine.

5.9.4.3.3 Single-Node Control Plane

> ⚠ Do not use a single-node control plane in a production cluster.

A control plane with one node can use its single node as the endpoint, so you will not require an external load balancer, or a built-in virtual IP. At least one control plane node must always be running. Therefore, to

upgrade a cluster with one control plane node, a spare machine must be available in the control plane inventory. This machine is used to provision the new node before the old node is deleted.

> Modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1351).

When the API server endpoints are defined, you can create the cluster using the link in Next Step below.

5.9.4.3.4 Known Limitations

> Be aware of these limitations in the current release of DKP.

The control plane endpoint port is also used as the API server port on each control plane machine. The default port is 6443. Before you create the cluster, ensure the port is available for use on each control plane machine.

5.9.4.3.5 Next Step:

Pre-provisioned FIPS: Create a New Cluster (see page 171)

5.9.4.4 Pre-provisioned FIPS: Create a Management Cluster

5.9.4.4.1 Deploying a Cluster in FIPS mode

In order to create a cluster in FIPS mode, we must inform the bootstrap controllers of the appropriate image repository and version tags of the official D2iQ FIPS builds of Kubernetes.

5.9.4.4.1.1 Supported FIPS Builds

Component	Repository	Version
Kubernetes	docker.io/mesosphere[127]	v1.25.4+fips.0

127 https://hub.docker.com/layers/mesosphere/kube-apiserver/v1.25.4_fips.0/images/sha256-cacb721f96ec8764d187ad3c21557630f5f4d96fea4a03ca35d0388b509d970d?context=explore

Component	Repository	Version
etcd	docker.io/mesosphere[128]	3.5.5+fips.0

5.9.4.4.1.2 Name Your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[129] for more naming information.

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-example>
```

5.9.4.4.1.3 Create a Kubernetes Cluster

Once you've defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster.

Before you create a new DKP cluster below, choose an external load balancer (LB) or virtual IP (see page 1114) and use the corresponding `dkp create cluster` command.

128 https://hub.docker.com/layers/mesosphere/etcd/v3.5.5_fips.0/images/sha256-ba07521fa1875efa0cc623533eb5557e40a6f8fdc8a71a86751a649ce53de1d0?context=explore

129 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

In a pre-provisioned environment, use the Kubernetes CSI and third party drivers for local volumes and other storage devices in your data center.

DKP uses local static provisioner as the default storage provider (see page 103) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[130] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory YAML (see page 127) previously created.

The create cluster command below includes the `--self-managed` flag. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

1. This command uses the default external load balancer (LB) option (see alternative Step 1 below for virtual IP):

```
dkp create cluster preprovisioned \
  --cluster-name ${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --kubernetes-version=v1.25.4+fips.0 \
  --etcd-version=3.5.5+fips.0 \
  --kubernetes-image-repository=docker.io/mesosphere \
  --self-managed
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

130 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

1. **ALTERNATIVE Virtual IP** - if you don't have an external LB, and wish to use a VIRTUAL IP provided by kube-vip, specify these flags example below:

   ```
   dkp create cluster preprovisioned \
       --cluster-name ${CLUSTER_NAME} \
       --control-plane-endpoint-host 196.168.1.10 \
       --virtual-ip-interface eth1
   ```

   The output from this command is shortened here for reading clarity, but should start like this:

   ```
   Generating cluster resources
   cluster.cluster.x-k8s.io/preprovisioned-example created
   cont.........
   ```

2. Use the wait command to monitor the cluster control-plane readiness:

   ```
   kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
   timeout=30m
   ```

   Output:

   ```
   cluster.cluster.x-k8s.io/preprovisioned-example condition met
   ```

Depending on the cluster size, it will take a few minutes to create.

When the command completes, you will have a running Kubernetes cluster! Use this command to get the Kubernetes `kubeconfig` for the new cluster and proceed to installing the DKP Kommander UI:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

If changing the Calico encapsulation (see page 1110), D2iQ recommends changing it after cluster creation, but before production.

If you need to increase Docker Hub's rate limit[131], use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.

5.9.4.4.1.4 Audit logs

To modify Control Plane Audit logs settings using the information contained in the page Configuring the Control Plane (see page 1351).

5.9.4.4.1.5 Further Steps:

For more customized cluster creation, access the Additional Infrastructure Configurations for Pre-Provisioned (see page 1078) section. That section is for Pre-Provisioned Override Files (see page 1341), custom flags, and more that specify the secret as part of the create cluster command. If these are not specified, the overrides for your nodes will not be applied.

5.9.4.4.1.6 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1360).

5.9.4.4.1.7 Next Step:

Pre-provisioned FIPS: Configure MetalLB (see page 175)

5.9.4.5 Pre-provisioned FIPS: Configure MetalLB

5.9.4.5.1 Create a MetalLB configmap for your pre-provisioned infrastructure.

Choose one of the following two protocols you want to use to announce service IPs. If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can continue the installation process Pre-provisioned FIPS: Install Kommander (see page 177) .

To use MetalLB, create a MetalLB configMap for your Pre-provisioned infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

131 https://docs.docker.com/docker-hub/download-rate-limit/

Select one of the following procedures to create your MetalLB manifest for further editing.

5.9.4.5.2 Layer 2 configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to ARP requests on your local network directly, to give the machine's MAC address to clients.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

5.9.4.5.3 BGP configuration

For a basic configuration featuring one BGP router and one IP address range, you need 4 pieces of information:

- The router IP address that MetalLB should connect to,
- The router's AS number,
- The AS number MetalLB should use,
- An IP address range expressed as a CIDR prefix.

As an example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

5.9.4.5.3.1 Next Step:

Pre-provisioned FIPS: Install Kommander (see page 177)

5.9.4.6 Pre-provisioned FIPS: Install Kommander

5.9.4.6.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install[132].
- Ensure you have a default StorageClass[133].
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

132 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/161021953/Prerequisites+for+Install
133 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/165347596

5.9.4.6.2 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

```
export CLUSTER_NAME=<your-management-cluster-name>
```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

```
dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
```

3. Create a configuration file (see page 1227) for the deployment:

```
dkp install kommander --init > kommander.yaml
```

4. Edit the installer file to include configuration overrides for the `rook-ceph-cluster`. DKP's default configuration ships Ceph with PVC based storage[134] which requires your CSI provider to support PVC with type `volumeMode: Block`. As this is not possible with the default local static provisioner (see page 103), you can install Ceph in host storage[135] mode.

   You can choose whether Ceph's object storage daemon (osd) pods should consume all or just some of the devices on your nodes. Include **one** of the following Overrides:

   a. To automatically assign all raw storage devices on all nodes to the Ceph cluster:

```
rook-ceph-cluster:
    enabled: true
    values: |
      cephClusterSpec:
        storage:
          storageClassDeviceSets: []
          useAllDevices: true
          useAllNodes: true
          deviceFilter: "<<value>>"
```

   b. To assign specific storage devices on all nodes to the Ceph cluster:

```
rook-ceph-cluster:
    enabled: true
    values: |
      cephClusterSpec:
        storage:
```

134 https://rook.io/docs/rook/v1.10/CRDs/Cluster/pvc-cluster/
135 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/

```
          storageClassDeviceSets: []
          useAllNodes: true
          useAllDevices: false
          deviceFilter: "^sdb."
```

Note: If you want to assign specific devices to specific nodes using the `deviceFilter` option, refer to Specific Nodes and Devices[136]. For general information on the `deviceFilter` value, refer to Storage Selection Settings[137].

5. *If required:* Customize your `kommander.yaml` .
   See Kommander Additional Install Configurations (see page 1227) for customization options. Some of them include:
   Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, etc.

5.9.4.6.3 Enable DKP Catalog Applications and Install Kommander

If you have an Enterprise license and would like to use the DKP Catalog Applications function, follow these steps:

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications` :

   ```
   apiVersion: config.kommander.mesosphere.io/v1alpha1
   kind: Installation
   catalog:
     repositories:
       - name: dkp-catalog-applications
         labels:
           kommander.d2iq.io/project-default-catalog-repository: "true"
           kommander.d2iq.io/workspace-default-catalog-repository: "true"
           kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
         gitRepositorySpec:
           url: https://github.com/mesosphere/dkp-catalog-applications
           ref:
             tag: v2.5.0
   ```

   If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

136 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/#specific-nodes-and-devices
137 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.co
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 99)
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

5.9.4.6.4 Next Step:

Pre-provisioned FIPS: Verify Install and Log in to UI (see page 180)

5.9.4.7 Pre-provisioned FIPS: Verify Install and Log in to UI

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander. After the CLI successfully installs the components, it will wait for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Released helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Released` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

5.9.4.7.1 Failed HelmReleases

If any application fails to successfully deploy, you can check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
```

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

5.9.4.7.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 524). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 524):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

5.9.4.7.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 510) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

5.9.4.7.4 Next Step:

Pre-provisioned FIPS: Subsequent Cluster Creation (see page 183)

5.9.4.8 Pre-provisioned FIPS: Create Managed Clusters Using the DKP CLI

After initial cluster creation, you have the ability to create additional clusters from the CLI. In a previous step, the new cluster was created as Self-managed which allows it to be a Management cluster or a stand alone cluster. Subsequent new clusters are not self-managed as they will likely be Managed or Attached clusters to this Management Cluster.

When creating Managed clusters (see page 97), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 97)!

5.9.4.8.1 Name Your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[138] for more naming information.

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-additional>
```

138 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

5.9.4.8.2 Choose a Workspace for the New Cluster

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a Workspace[139].

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

5.9.4.8.3 Create a Kubernetes Cluster

Once you've defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster.

ⓘ Before you create a new DKP cluster below, choose an external load balancer (LB) or virtual IP (see page 1114) and use the corresponding `dkp create cluster` command.

In a pre-provisioned environment, use the Kubernetes CSI and third party drivers for local volumes and other storage devices in your data center.

⚠ DKP uses local static provisioner as the default storage provider (see page 103) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[140] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory YAML (see page 127) previously created.

139 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919315/Workspaces#Create-a-Workspace
140 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

1. This command uses the default external load balancer (LB) option:

```
dkp create cluster preprovisioned \
  --cluster-name ${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --kubernetes-version=v1.25.4+fips.0 \
  --etcd-version=3.5.5+fips.0 \
  --kubernetes-image-repository=docker.io/mesosphere \
  --namespace=${WORKSPACE_NAMESPACE}
```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

2. Use the wait command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

   Output:

```
cluster.cluster.x-k8s.io/preprovisioned-additional condition met
```

NOTE: Depending on the cluster size, it will take a few minutes to create.

5.9.4.8.4 Manually Attach a DKP CLI Cluster to the Management Cluster

1. Find out the `name` of the created `Cluster`, so you can reference it later:

```
kubectl -n <workspace_namespace> get clusters
```

2. Attach the cluster by creating a `KommanderCluster`:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
```

```
kind: KommanderCluster
metadata:
  name: <cluster_name>
  namespace: <workspace_namespace>
spec:
  kubeconfigRef:
    name: <cluster_name>-kubeconfig
  clusterRef:
    capiCluster:
      name: <cluster_name>
EOF
```

5.9.4.8.5 Next Step:

Day 2 - Cluster Operations Management (see page 510)

5.9.5 Pre-provisioned FIPS Air-gapped Install

This section provides instructions to install DKP in a Pre-provisioned non-air-gapped environment using FIPS.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

NOTE: For air-gapped, ensure you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz` , so you can extract the tarball on next page.

5.9.5.1 Next Step:

Pre-provisioned Air-gapped FIPS: Configure Environment (see page 186)

5.9.5.2 Pre-provisioned Air-gapped FIPS: Configure Environment

In order to create a cluster in a Pre-provisioned Air-gapped environment with FIPS, you must first prepare the environment.

The instructions below outline how to fulfill the requirements for a pre-provisioned infrastructure when using an air-gapped environment. In order to create a cluster, you must first setup the environment with necessary artifacts. All artifacts for Pre-provisioned Air-gapped environments need to get onto the bastion host. Artifacts needed by nodes must be unpacked and distributed on the bastion before other provisioning will work in the absence of an internet connection.

There is a new complete DKP air-gapped bundle available to download (see page 71) which contains all the DKP components needed for air-gapped installation. (i.e. `dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz`)

5.9.5.2.1 Setup Process:

1. The bootstrap image must be extracted and loaded onto the bastion host (see page 1162).
2. Artifacts must be copied onto cluster hosts for nodes to access.
3. If using GPU, those artifacts must be positioned locally.
4. Docker Registry seeded with images locally.

5.9.5.2.2 Load the Bootstrap Image

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz` from the download site (see page 71) mentioned above, extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz && cd dkp-v2.5.2
   ```

2. Load the bootstrap Docker image on your bastion machine:

   ```
   docker load -i konvoy-bootstrap-image-v2.5.2.tar
   ```

5.9.5.2.3 Copy air-gapped artifacts onto cluster hosts

Using the Konvoy Image Builder (see page 1282), you can copy the required artifacts onto your cluster hosts.

Assuming you have downloaded `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz`, continue below:

1. The kubernetes image bundle will be located in `kib/artifacts/images` and you will want to verify image and artifacts.
   a. Verify the image bundles exist in `artifacts/images`:

      ```
      $ ls artifacts/images/
      kubernetes-images-1.25.4-d2iq.1.tar kubernetes-images-1.25.4-d2iq.1-
      fips.tar
      ```

   b. Verify the artifacts for your OS exist in the `artifacts/` directory and export the appropriate variables:

```
$ ls artifacts/
1.25.4_centos_7_x86_64.tar.gz       1.25.4_redhat_8_x86_64_fips.tar.gz
containerd-1.6.17-d2iq.1-rhel-7.9-x86_64.tar.gz   containerd-1.6.17-
d2iq.1-rhel-8.6-x86_64_fips.tar.gz  pip-packages.tar.gz
1.25.4_centos_7_x86_64_fips.tar.gz  1.25.4_rocky_9_x86_64.tar.gz
containerd-1.6.17-d2iq.1-rhel-7.9-x86_64_fips.tar.gz  containerd-1.6.17-
d2iq.1-rocky-9.0-x86_64.tar.gz
1.25.4_redhat_7_x86_64.tar.gz       1.25.4_ubuntu_20_x86_64.tar.gz
containerd-1.6.17-d2iq.1-rhel-8.4-x86_64.tar.gz   containerd-1.6.17-
d2iq.1-rocky-9.1-x86_64.tar.gz
1.25.4_redhat_7_x86_64_fips.tar.gz  containerd-1.6.17-d2iq.1-centos-7.9-
x86_64.tar.gz       containerd-1.6.17-d2iq.1-rhel-8.4-x86_64_fips.tar.gz
containerd-1.6.17-d2iq.1-ubuntu-20.04-x86_64.tar.gz
1.25.4_redhat_8_x86_64.tar.gz       containerd-1.6.17-d2iq.1-centos-7.9-
x86_64_fips.tar.gz  containerd-1.6.17-d2iq.1-rhel-8.6-x86_64.tar.gz
images
```

c. For example, for RHEL 8.4 you would set:

```
export OS_PACKAGES_BUNDLE=1.25.4_redhat_8_x86_64_fips.tar.gz
export CONTAINERD_BUNDLE=containerd-1.6.10-d2iq.1-rhel-8.4-x86_64.tar.gz
```

2. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
export WORKER_1_ADDRESS="<worker-address-1>"
export WORKER_2_ADDRESS="<worker-address-2>"
export WORKER_3_ADDRESS="<worker-address-3>"
export WORKER_4_ADDRESS="<worker-address-4>"
export SSH_USER="<ssh-user>"
export SSH_PRIVATE_KEY_FILE="<private key file>"
```

`SSH_PRIVATE_KEY_FILE` must be either the name of the SSH private key file in your working directory or an absolute path to the file in your user's home directory.

3. Generate an `inventory.yaml` which is automatically picked up by the `konvoy-image upload` in the next step.

```
cat <<EOF > inventory.yaml
all:
  vars:
    ansible_user: $SSH_USER
    ansible_port: 22
    ansible_ssh_private_key_file: $SSH_PRIVATE_KEY_FILE
```

```
  hosts:
    $CONTROL_PLANE_1_ADDRESS:
      ansible_host: $CONTROL_PLANE_1_ADDRESS
    $CONTROL_PLANE_2_ADDRESS:
      ansible_host: $CONTROL_PLANE_2_ADDRESS
    $CONTROL_PLANE_3_ADDRESS:
      ansible_host: $CONTROL_PLANE_3_ADDRESS
    $WORKER_1_ADDRESS:
      ansible_host: $WORKER_1_ADDRESS
    $WORKER_2_ADDRESS:
      ansible_host: $WORKER_2_ADDRESS
    $WORKER_3_ADDRESS:
      ansible_host: $WORKER_3_ADDRESS
    $WORKER_4_ADDRESS:
      ansible_host: $WORKER_4_ADDRESS
EOF
```

4. Upload the artifacts onto cluster hosts with the following command:

```
konvoy-image upload artifacts \
              --container-images-dir=./artifacts/images/ \
              --os-packages-bundle=./artifacts/$OS_PACKAGES_BUNDLE \
              --containerd-bundle=artifacts/$CONTAINERD_BUNDLE \
              --pip-packages-bundle=./artifacts/pip-packages.tar.gz
```

KIB uses variable overrides (see page 1330) to specify base image and container images to use in your new machine image. The variable overrides files for NVIDIA and FIPS can be ignored unless adding an overlay feature.

- Use the --overrides flag and reference either `fips.yaml` or `offline-fips.yaml` manifests located in the overrides directory[141] or see these pages in the documentation:
  - FIPS Overrides (see page 1331)
  - Create FIPS 140 Images (see page 1344)

5.9.5.2.3.1 Next Step:

Pre-provisioned Air-gapped FIPS: Load the Registry (see page 190)

141 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

5.9.5.3 Pre-provisioned Air-gapped FIPS: Load the Registry

Before creating a Kubernetes cluster, you need the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1162) and either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1349) page for more information.

1. Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz` , extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz && cd dkp-v2.5.0
   ```

2. Set an environment variable with your registry address with this command:

   ```
   export REGISTRY_ADDRESS=<registry-address>:<registry-port>
   export REGISTRY_USERNAME=<username>
   export REGISTRY_PASSWORD=<password>
   ```

3. Execute the following command to load the air-gapped image bundle into your private registry:

   ```
   dkp push image-bundle --image-bundle ./container-images/konvoy-image-bundle-v2.5.0.tar --to-registry $REGISTRY_ADDRESS --to-registry-username $REGISTRY_USERNAME --to-registry-password $REGISTRY_PASSWORD
   ```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

To increase Docker Hub's rate limit[142] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.

5.9.5.3.1 Kommander Load Images

If you are operating in an air-gapped environment, a local container registry (see page 1349) containing all the necessary installation images, including the Kommander images is required. See below for how to push the necessary images to this registry.

For the basic air-gapped `kommander` image bundle, run the command below:

1. Run the following command to load the image bundle:

```
dkp push image-bundle --image-bundle ./container-images/kommander-image-bundle-v2.5.0.tar --to-registry $REGISTRY_ADDRESS
```

For **DKP Catalog Applications**, also perform this image load:

1. Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push image-bundle --image-bundle ./container-images/dkp-catalog-applications-image-bundle-v2.5.0.tar --to-registry $REGISTRY_ADDRESS
```

5.9.5.3.2 Next Step:

Pre-provisioned Air-gapped FIPS: Define Infrastructure (see page 191)

5.9.5.4 Pre-provisioned Air-gapped FIPS: Define Infrastructure

Define the cluster hosts and infrastructure

Konvoy needs to know how to access your cluster hosts. This is done using inventory resources. For initial cluster creation, you must define a control-plane and at least one worker pool.

142 https://docs.docker.com/docker-hub/download-rate-limit/

5.9.5.4.1 Define your Infrastructure

1. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
export WORKER_1_ADDRESS="<worker-address-1>"
export WORKER_2_ADDRESS="<worker-address-2>"
export WORKER_3_ADDRESS="<worker-address-3>"
export WORKER_4_ADDRESS="<worker-address-4>"
export SSH_USER="<ssh-user>"
export SSH_PRIVATE_KEY_SECRET_NAME="$CLUSTER_NAME-ssh-key"
```

2. Use the following template to help you define your infrastructure. The environment variables that you set in the previous step automatically replace the variable names when the inventory YAML file is created.

```
cat <<EOF > preprovisioned_inventory.yaml
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-control-plane
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    # Create as many of these as needed to match your infrastructure
    # Note that the command line parameter --control-plane-replicas determines
how many control plane nodes will actually be used.
    #
    - address: $CONTROL_PLANE_1_ADDRESS
    - address: $CONTROL_PLANE_2_ADDRESS
    - address: $CONTROL_PLANE_3_ADDRESS
  sshConfig:
    port: 22
    # This is the username used to connect to your infrastructure. This user
must be root or
    # have the ability to use sudo without a password
    user: $SSH_USER
    privateKeyRef:
      # This is the name of the secret you created in the previous step. It
must exist in the same
```

```
      # namespace as this inventory object.
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-md-0
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    - address: $WORKER_1_ADDRESS
    - address: $WORKER_2_ADDRESS
    - address: $WORKER_3_ADDRESS
    - address: $WORKER_4_ADDRESS
  sshConfig:
    port: 22
    user: $SSH_USER
    privateKeyRef:
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
EOF
```

5.9.5.4.2 Next Step:

Pre-provisioned Air-gapped FIPS: Define Control Plane Endpoint (see page 193)

5.9.5.5 Pre-provisioned Air-gapped FIPS: Define Control Plane Endpoint

5.9.5.5.1 Define the Control Plane Endpoint for your cluster as well as connection mechanism

A control plane should have three, five, or seven nodes, so it can remain available if one or more nodes fail. A control plane with one node should not be used in production.

In addition, the control plane should have an endpoint that remains available if some nodes fail.

```
                        -------- cp1.example.com:6443
                        |
lb.example.com:6443 ----------- cp2.example.com:6443
                        |
                        -------- cp3.example.com:6443
```

In this example, the control plane endpoint host is `lb.example.com`, and the control plane endpoint port is `6443`. The control plane nodes are `cp1.example.com`, `cp2.example.com`, and `cp3.example.com`. The port of each API server is `6443`.

5.9.5.5.2 Select your Connection Mechanism

A virtual IP is the address that the client uses to connect to the service. A load balancer is the device that distributes the client connections to the backend servers. Before you create a new DKP cluster, choose an external load balancer(LB) or virtual IP.

- **External load balancer**

It is recommended that an external load balancer be the control plane endpoint. To distribute request load among the control plane machines, configure the load balancer to send requests to all the control plane machines. Configure the load balancer to send requests only to control plane machines that are responding to API requests.

- **Built-in virtual IP**

If an external load balancer is not available, use the built-in virtual IP (see page 1114). The virtual IP is *not* a load balancer; it does not distribute request load among the control plane machines. However, if the machine receiving requests does not respond to them, the virtual IP automatically moves to another machine.

5.9.5.5.3 Single-Node Control Plane

Do not use a single-node control plane in a production cluster.

A control plane with one node can use its single node as the endpoint, so you will not require an external load balancer, or a built-in virtual IP. At least one control plane node must always be running. Therefore, to upgrade a cluster with one control plane node, a spare machine must be available in the control plane inventory. This machine is used to provision the new node before the old node is deleted.

Modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1351).

When the API server endpoints are defined, you can create the cluster using the link in Next Step below.

5.9.5.5.4 Known Limitations

Be aware of these limitations in the current release of DKP.

The control plane endpoint port is also used as the API server port on each control plane machine. The default port is 6443. Before you create the cluster, ensure the port is available for use on each control plane machine.

5.9.5.5.5 Next Step:

Pre-provisioned Air-gapped FIPS: Create a New Cluster (see page 195)

5.9.5.6 Pre-provisioned Air-gapped FIPS: Create a Management Cluster

If your cluster is air-gapped or you have a local docker registry, you must provide additional arguments when creating the cluster. These tell the cluster where to locate the local registry to use by defining the URL.

```
export REGISTRY_URL=<https/http>://<registry-address>:<registry-port>
export REGISTRY_CA=<path to the CA on the bastion>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

- `REGISTRY_URL` : the address of an existing registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
- `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
- `REGISTRY_PASSWORD` : optional if username is not set.

5.9.5.6.1 Name Your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[143] for more naming information.

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-example>
```

5.9.5.6.2 Create an Air-gapped Kubernetes Cluster

Once you've defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster.

DKP uses local static provisioner as the default storage provider (see page 103) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[144] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory.

143 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/
144 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

The create cluster command below includes the `--self-managed` flag. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

Before you create a new DKP cluster below, choose an external load balancer (LB) or virtual IP (see page 1114) and use the corresponding `dkp create cluster` command.

1. The command below uses the default external load balancer (See alternative Step 1 for virtual IP):

```
dkp create cluster preprovisioned --cluster-name ${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --registry-mirror-url=${REGISTRY_URL} \
  --registry-mirror-cacert=${REGISTRY_CA} \
  --registry-mirror-username=${REGISTRY_USERNAME} \
  --registry-mirror-password=${REGISTRY_PASSWORD} \
  --kubernetes-version=v1.25.4+fips.0 \
  --etcd-version=3.5.5+fips.0 \
  --kubernetes-image-repository=docker.io/mesosphere \
  --self-managed
```


If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

1. **Virtual IP ALTERNATIVE** - if you don't have an external LB, and wish to use a VIRTUAL IP (see page 1114) provided by kube-vip, specify these flags example below:

```
dkp create cluster preprovisioned \
    --cluster-name ${CLUSTER_NAME} \
    --control-plane-endpoint-host 196.168.1.10 \
    --virtual-ip-interface eth1
```

The output from this command is shortened here for reading clarity, but should start like this:

```
Generating cluster resources
```

```
cluster.cluster.x-k8s.io/preprovisioned-example created
cont.........
```

2. Use the wait command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

Output:

```
cluster.cluster.x-k8s.io/preprovisioned-example condition met
```

NOTE: Depending on the cluster size, it will take a few minutes to create.

When the command completes, you will have a running Kubernetes cluster! Use this command to get the Kubernetes `kubeconfig` for the new cluster and proceed to installing the DKP Kommander UI:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

If changing the Calico encapsulation (see page 1110), D2iQ recommends changing it after cluster creation, but before production.

5.9.5.6.3 Audit Logs

To modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1351).

5.9.5.6.4 Further Steps:

For more customized cluster creation, access the Pre-Provisioned Additional Configurations (see page 1078) section.

NOTE: The section mentioned above is for overrides for your clusters (see page 1341), custom flags, and more that specify the secret as part of the create cluster command. If these are not specified, the overrides for your nodes will not be applied.

5.9.5.6.5 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1360).

5.9.5.6.6 Next Step:

Pre-provisioned Air-gapped FIPS: Configure MetalLB (see page 199)

5.9.5.7 Pre-provisioned Air-gapped FIPS: Configure MetalLB

5.9.5.7.1 Create a MetalLB configmap for your pre-provisioned infrastructure.

Choose one of the following two protocols you want to use to announce service IPs. If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can continue the installation process Pre-provisioned Air-gapped FIPS: Install Kommander (see page 201).

To use MetalLB, create a MetalLB configMap for your Pre-provisioned infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

5.9.5.7.2 Layer 2 configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to ARP requests on your local network directly, to give the machine's MAC address to clients.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

5.9.5.7.3 BGP configuration

For a basic configuration featuring one BGP router and one IP address range, you need 4 pieces of information:

- The router IP address that MetalLB should connect to,
- The router's AS number,
- The AS number MetalLB should use,
- An IP address range expressed as a CIDR prefix.

As an example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
```

```
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

5.9.5.7.3.1 Next Step:

Pre-provisioned Air-gapped FIPS: Install Kommander (see page 201)

5.9.5.8 Pre-provisioned Air-gapped FIPS: Install Kommander

5.9.5.8.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 119) .
- Ensure you have a default StorageClass (see page 1254).
- Ensure you have loaded all necessary images for your configuration. See Load the Images into Your Registry (see page 1258).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

5.9.5.8.2 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1227) for the deployment:

```
dkp install kommander --init --airgapped > kommander.yaml
```

4. Edit the installer file to include configuration overrides for the `rook-ceph-cluster`. DKP's default configuration ships Ceph with PVC based storage[145] which requires your CSI provider to support PVC with type `volumeMode: Block`. As this is not possible with the default local static provisioner (see page 103), you can install Ceph in host storage[146] mode.

   You can choose whether Ceph's object storage daemon (osd) pods should consume all or just some of the devices on your nodes. Include **one** of the following Overrides:

   a. To automatically assign all raw storage devices on all nodes to the Ceph cluster:

   ```
   rook-ceph-cluster:
       enabled: true
       values: |
         cephClusterSpec:
           storage:
             storageClassDeviceSets: []
             useAllDevices: true
             useAllNodes: true
             deviceFilter: "<<value>>"
   ```

   b. To assign specific storage devices on all nodes to the Ceph cluster:

   ```
   rook-ceph-cluster:
       enabled: true
       values: |
         cephClusterSpec:
           storage:
             storageClassDeviceSets: []
             useAllNodes: true
             useAllDevices: false
             deviceFilter: "^sdb."
   ```

   Note: If you want to assign specific devices to specific nodes using the `deviceFilter` option, refer to Specific Nodes and Devices[147]. For general information on the `deviceFilter` value, refer to Storage Selection Settings[148].

5. *If required:* Customize your `kommander.yaml`.

   See Kommander Additional Install Configurations (see page 1227) for customization options. Some of them include:
   Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, etc.

145 https://rook.io/docs/rook/v1.10/CRDs/Cluster/pvc-cluster/
146 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/
147 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/#specific-nodes-and-devices
148 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

5.9.5.8.3 Enable DKP Catalog Applications and Install Kommander

If you have an Enterprise license and would like to use the DKP Catalog Applications function, follow these steps:

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.5.0
```

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.5.0.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.5.0.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.5.0.tar.gz
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 99)
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

5.9.5.8.4 Next Step:

Pre-provisioned Air-gapped FIPS: Verify Install and Log in to UI (see page 204)

5.9.5.9 Pre-provisioned Air-gapped FIPS: Verify Install and Log in to UI

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander. After the CLI successfully installs the components, it will wait for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Released helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Released` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
```

```
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

5.9.5.9.1 Failed HelmReleases

If any application fails to successfully deploy, you can check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

5.9.5.9.2 Next Step:

Pre-provisioned Air-gapped FIPS: Subsequent Clusters (see page 205)

5.9.5.10 Pre-provisioned Air-gapped FIPS: Create Managed Clusters Using the DKP CLI

After initial cluster creation, you have the ability to create additional clusters from the CLI. In a previous step, the new cluster was created as Self-managed which allows it to be a Management cluster or a stand alone

cluster. Subsequent new clusters are not self-managed as they will likely be Managed or Attached clusters to this Management Cluster.

When creating Managed clusters (see page 97), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 97)!

5.9.5.10.1 Choose a Workspace for the New Cluster

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a Workspace[149].

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

5.9.5.10.2 Name Your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[150] for more naming information.

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

149 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919315/Workspaces#Create-a-Workspace
150 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

```
export CLUSTER_NAME=<preprovisioned-additional>
```

5.9.5.10.3 Create a Kubernetes Cluster

Once you've defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster.

Before you create a new DKP cluster below, choose an external load balancer (LB) or virtual IP (see page 1114) and use the corresponding `dkp create cluster` command.

In a pre-provisioned environment, use the Kubernetes CSI and third party drivers for local volumes and other storage devices in your data center.

DKP uses local static provisioner as the default storage provider (see page 103) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[151] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory YAML (see page 127) previously created.

1. This command uses the default external load balancer (LB) option:

```
dkp create cluster preprovisioned --cluster-name ${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --registry-mirror-url=${REGISTRY_URL} \
  --registry-mirror-cacert=${REGISTRY_CA} \
  --registry-mirror-username=${REGISTRY_USERNAME} \
  --registry-mirror-password=${REGISTRY_PASSWORD} \
  --namespace=${WORKSPACE_NAMESPACE}
```

151 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

2. Use the wait command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

Output:

```
cluster.cluster.x-k8s.io/preprovisioned-additional condition met
```

NOTE: Depending on the cluster size, it will take a few minutes to create.

5.9.5.10.4 Manually Attach a DKP CLI Cluster to the Management Cluster

1. Find out the `name` of the created `Cluster`, so you can reference it later:

```
kubectl -n <workspace_namespace> get clusters
```

2. Attach the cluster by creating a `KommanderCluster`:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: <cluster_name>
  namespace: <workspace_namespace>
spec:
  kubeconfigRef:
    name: <cluster_name>-kubeconfig
  clusterRef:
    capiCluster:
      name: <cluster_name>
EOF
```

5.9.5.10.5 Next Step:

Day 2 - Cluster Operations Management (see page 510)

5.9.6 Pre-provisioned with GPU Install

This section provides instructions to install DKP in a Pre-provisioned non-air-gapped environment using GPU.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

5.9.6.1 Next Step:

Pre-provisioned GPU: Nodepool Secrets and Overrides (see page 210)

5.9.6.2 Pre-provisioned GPU: Nodepool Secrets and Overrides

5.9.6.2.1 DKP has introduced the `nvidia-runfile` flag for Air-gapped Pre-provisioned environments. If the NVIDIA runfile[152] installer has not been downloaded, then retrieve and install the download first by running the following command. The first line in the command below downloads and installs the runfile and the second line places it in the artifacts directory (you must create an `artifacts` directory if it doesn't already exist).

```
curl -O https://download.nvidia.com/XFree86/Linux-x86_64/470.82.01/NVIDIA-Linux-x86_64-470.82.01.run
mv NVIDIA-Linux-x86_64-470.82.01.run artifacts
```

DKP supported NVIDIA driver[153] version is 470.x.

1. Create the secret that GPU nodepool would use, this secret is populated from the KIB overrides. In this example we have a file called, `overrides/nvidia.yaml`. It should resemble this:

```
gpu:
  types:
    - nvidia
build_name_extra: "-nvidia"
```

2. Create a secret on the bootstrap cluster that is populated from the above file. We will name it

`${CLUSTER_NAME}-user-overrides`

```
kubectl create secret generic ${CLUSTER_NAME}-user-overrides --from-file=overrides.yaml=overrides/nvidia.yaml
```

3. Create an inventory and nodepool with the instructions below and use the `$CLUSTER_NAME-user-overrides` secret.

Follow these steps:

152 https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html#runfile
153 https://www.nvidia.com/Download/Find.aspx

1. Create an inventory object that has the same name as the node pool you're creating, and the details of the pre-provisioned machines that you want to add to it. For example, to create a node pool named `gpu-nodepool` an inventory named `gpu-nodepool` must be present in the same namespace:

```
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: ${MY_NODEPOOL_NAME}
spec:
  hosts:
    - address: ${IP_OF_NODE}
  sshConfig:
    port: 22
    user: ${SSH_USERNAME}
    privateKeyRef:
      name: ${NAME_OF_SSH_SECRET}
      namespace: ${NAMESPACE_OF_SSH_SECRET}
```

2. (Optional) If your pre-provisioned machines have overrides, you must create a secret that includes all of the overrides you want to provide in one file. Create an override secret using the instructions detailed on this page.

3. Once the `PreprovisionedInventory` object and overrides are created, create a node pool:

```
dkp create nodepool preprovisioned -c ${MY_CLUSTER_NAME} $
{MY_NODEPOOL_NAME} --override-secret-name ${MY_OVERRIDE_SECRET}
```

- Advanced users can use a combination of the `--dry-run` and `--output=yaml` or `--output-directory=<existing-directory>` flags to get a complete set of node pool objects to modify locally or store in version control.

For more information regarding this flag or others, please refer to the dkp create nodepool section of the documentation for either cluster or nodepool and select your provider.

For more information, see:

- Pre-provisioned Create and Delete Node Pools
- Pre-provisioned Air-gapped Define Environment

Next Step:

Pre-provisioned GPU: Define Infrastructure (see page 212)

5.9.6.3 Pre-provisioned GPU: Define Infrastructure

Define the cluster hosts and infrastructure

Konvoy needs to know how to access your cluster hosts. This is done using inventory resources. For initial cluster creation, you must define a control-plane and at least one worker pool.

5.9.6.3.1 Define your Infrastructure

1. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
export WORKER_1_ADDRESS="<worker-address-1>"
export WORKER_2_ADDRESS="<worker-address-2>"
export WORKER_3_ADDRESS="<worker-address-3>"
export WORKER_4_ADDRESS="<worker-address-4>"
export SSH_USER="<ssh-user>"
export SSH_PRIVATE_KEY_SECRET_NAME="$CLUSTER_NAME-ssh-key"
```

2. Use the following template to help you define your infrastructure. The environment variables that you set in the previous step automatically replace the variable names when the inventory YAML file is created.

```
cat <<EOF > preprovisioned_inventory.yaml
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-control-plane
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    # Create as many of these as needed to match your infrastructure
    # Note that the command line parameter --control-plane-replicas determines
how many control plane nodes will actually be used.
    #
    - address: $CONTROL_PLANE_1_ADDRESS
    - address: $CONTROL_PLANE_2_ADDRESS
    - address: $CONTROL_PLANE_3_ADDRESS
```

```
  sshConfig:
    port: 22
    # This is the username used to connect to your infrastructure. This user
must be root or
    # have the ability to use sudo without a password
    user: $SSH_USER
    privateKeyRef:
      # This is the name of the secret you created in the previous step. It
must exist in the same
      # namespace as this inventory object.
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-md-0
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    - address: $WORKER_1_ADDRESS
    - address: $WORKER_2_ADDRESS
    - address: $WORKER_3_ADDRESS
    - address: $WORKER_4_ADDRESS
  sshConfig:
    port: 22
    user: $SSH_USER
    privateKeyRef:
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
EOF
```

5.9.6.3.2 Next Step:

Pre-provisioned GPU: Define Control Plane Endpoint (see page 213)

5.9.6.4 Pre-provisioned GPU: Define Control Plane Endpoint

5.9.6.4.1 Define the Control Plane Endpoint for your cluster as well as connection mechanism

A control plane should have three, five, or seven nodes, so it can remain available if one or more nodes fail. A control plane with one node should not be used in production.

In addition, the control plane should have an endpoint that remains available if some nodes fail.

```
                          -------- cp1.example.com:6443
                          |
      lb.example.com:6443 ---------- cp2.example.com:6443
                          |
                          -------- cp3.example.com:6443
```

In this example, the control plane endpoint host is `lb.example.com`, and the control plane endpoint port is `6443`. The control plane nodes are `cp1.example.com`, `cp2.example.com`, and `cp3.example.com`. The port of each API server is `6443`.

5.9.6.4.2 Select your Connection Mechanism

A virtual IP is the address that the client uses to connect to the service. A load balancer is the device that distributes the client connections to the backend servers. Before you create a new DKP cluster, choose an external load balancer(LB) or virtual IP.

- **External load balancer**

It is recommended that an external load balancer be the control plane endpoint. To distribute request load among the control plane machines, configure the load balancer to send requests to all the control plane machines. Configure the load balancer to send requests only to control plane machines that are responding to API requests.

- **Built-in virtual IP**

If an external load balancer is not available, use the built-in virtual IP (see page 1114). The virtual IP is *not* a load balancer; it does not distribute request load among the control plane machines. However, if the machine receiving requests does not respond to them, the virtual IP automatically moves to another machine.

5.9.6.4.3 Single-Node Control Plane

Do not use a single-node control plane in a production cluster.

A control plane with one node can use its single node as the endpoint, so you will not require an external load balancer, or a built-in virtual IP. At least one control plane node must always be running. Therefore, to upgrade a cluster with one control plane node, a spare machine must be available in the control plane inventory. This machine is used to provision the new node before the old node is deleted.

Modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1351).

When the API server endpoints are defined, you can create the cluster using the link in Next Step below.

5.9.6.4.4 Known Limitations

Be aware of these limitations in the current release of DKP.

The control plane endpoint port is also used as the API server port on each control plane machine. The default port is 6443. Before you create the cluster, ensure the port is available for use on each control plane machine.

5.9.6.4.5 Next Step:

Pre-provisioned GPU: Create a Cluster (see page 215)

5.9.6.5 Pre-provisioned GPU: Create a Management Cluster

5.9.6.5.1 Create a Cluster with GPU AMI

If a custom AMI was created using Konvoy Image Builder, the custom `ami` id is printed and written to `./manifest.json`.

To use the built `ami` with Konvoy, specify it with the `--ami` flag when calling cluster create.

For GPU Steps in Pre-provisioned (see page 0) section of the documentation to use the overrides/nvidia.yaml.

Additional helpful information can be found in the NVIDIA Device Plug-in[154] for Kubernetes instructions and the Installation Guide of Supported Platforms[155].

5.9.6.5.2 Name Your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[156] for more naming information.

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

154 https://github.com/NVIDIA/k8s-device-plugin/blob/master/README.md
155 https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html
156 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-example>
```

5.9.6.5.3 Create a Kubernetes Cluster

Once you've defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster.

Before you create a new DKP cluster below, choose an external load balancer (LB) or virtual IP (see page 1114) and use the corresponding `dkp create cluster` command.

In a pre-provisioned environment, use the Kubernetes CSI and third party drivers for local volumes and other storage devices in your data center.

DKP uses local static provisioner as the default storage provider (see page 103) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[157] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory YAML (see page 127) previously created.

The create cluster command below includes the `--self-managed` flag. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

157 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

1. Execute this command to create a cluster with a GPU AMI using the default external load balancer option:

```
dkp create cluster preprovisioned \
  --cluster-name=${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --ami <ami> \
  --self-managed
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

1. **Virtual IP ALTERNATIVE** - if you don't have an external LB, and wish to use a VIRTUAL IP (see page 1114) provided by kube-vip, specify these flags example below:

```
dkp create cluster preprovisioned \
    --cluster-name ${CLUSTER_NAME} \
    --control-plane-endpoint-host 196.168.1.10 \
    --virtual-ip-interface eth1
```

   The output from this command is shortened here for reading clarity, but should start like this:

```
Generating cluster resources
cluster.cluster.x-k8s.io/preprovisioned-example created
cont.........
```

2. Create the node pool after cluster creation:

```
dkp create nodepool aws -c ${CLUSTER_NAME} \
--instance-type p2.xlarge \
--ami-id=${AMI_ID_FROM_KIB} \
--replicas=1 ${NODEPOOL_NAME} \
--kubeconfig=${CLUSTER_NAME}.conf
```

3. Use the wait command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

Output:

```
cluster.cluster.x-k8s.io/preprovisioned-example condition met
```

Depending on the cluster size, it will take a few minutes to create.

When the command completes, you will have a running Kubernetes cluster! Use this command to get the Kubernetes `kubeconfig` for the new cluster and proceed to installing the DKP Kommander UI:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

If changing the Calico encapsulation (see page 1110), D2iQ recommends changing it after cluster creation, but before production.

To increase Docker Hub's rate limit[158] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.

5.9.6.5.4 Audit logs

To modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1351).

5.9.6.5.4.1 Further Steps:

For more customized cluster creation, access the Pre-Provisioned Additional Configurations (see page 1078) section. That section is for Pre-Provisioned Override Files (see page 215), custom flags, and more that specify

158 https://docs.docker.com/docker-hub/download-rate-limit/

the secret as part of the create cluster command. If these are not specified, the overrides for your nodes will not be applied.

5.9.6.5.4.2 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1360).

5.9.6.5.4.3 Next Step:

Pre-provisioned GPU: Configure MetalLB (see page 219)

5.9.6.6 Pre-provisioned GPU: Configure MetalLB

5.9.6.6.1 Create a MetalLB configmap for your pre-provisioned infrastructure.

Choose one of the following two protocols you want to use to announce service IPs. If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can continue the installation process Pre-provisioned GPU: Install Kommander (see page 221).

To use MetalLB, create a MetalLB configMap for your Pre-provisioned infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

5.9.6.6.2 Layer 2 configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to ARP requests on your local network directly, to give the machine's MAC address to clients.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
```

```
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

5.9.6.6.3 BGP configuration

For a basic configuration featuring one BGP router and one IP address range, you need 4 pieces of information:

- The router IP address that MetalLB should connect to,
- The router's AS number,
- The AS number MetalLB should use,
- An IP address range expressed as a CIDR prefix.

As an example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
```

```
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

5.9.6.6.3.1 Next Step:

Pre-provisioned GPU: Install Kommander (see page 221)

5.9.6.7 Pre-provisioned GPU: Install Kommander

5.9.6.7.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install[159].
- Ensure you have a default StorageClass[160].
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

5.9.6.7.2 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1227) for the deployment:

   ```
   dkp install kommander --init > kommander.yaml
   ```

159 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/161021953/Prerequisites+for+Install
160 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/165347596

4. Edit the installer file to include configuration overrides for the `rook-ceph-cluster`. DKP's default configuration ships Ceph with PVC based storage[161] which requires your CSI provider to support PVC with type `volumeMode: Block`. As this is not possible with the default local static provisioner (see page 103), you can install Ceph in host storage[162] mode.

   You can choose whether Ceph's object storage daemon (osd) pods should consume all or just some of the devices on your nodes. Include **one** of the following Overrides:

   a. To automatically assign all raw storage devices on all nodes to the Ceph cluster:

   ```
   rook-ceph-cluster:
       enabled: true
       values: |
         cephClusterSpec:
           storage:
             storageClassDeviceSets: []
             useAllDevices: true
             useAllNodes: true
             deviceFilter: "<<value>>"
   ```

   b. To assign specific storage devices on all nodes to the Ceph cluster:

   ```
   rook-ceph-cluster:
       enabled: true
       values: |
         cephClusterSpec:
           storage:
             storageClassDeviceSets: []
             useAllNodes: true
             useAllDevices: false
             deviceFilter: "^sdb."
   ```

   Note: If you want to assign specific devices to specific nodes using the `deviceFilter` option, refer to Specific Nodes and Devices[163]. For general information on the `deviceFilter` value, refer to Storage Selection Settings[164].

5. *If required:* Customize your `kommander.yaml`.

   See Kommander Additional Install Configurations (see page 1227) for customization options. Some of them include:
   Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, etc.

161 https://rook.io/docs/rook/v1.10/CRDs/Cluster/pvc-cluster/
162 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/
163 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/#specific-nodes-and-devices
164 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

5.9.6.7.3 Enable GPU Resources

1. In the same `kommander.yaml` file, enable Nvidia platform services.

```
apps:
  nvidia-gpu-operator:
   enabled: true
```

2. Append the correct Toolkit version based on your OS:

   The **NVIDIA Container Toolkit** allows users to run GPU accelerated containers. The toolkit includes a container runtime library and utilities to automatically configure containers to leverage NVIDIA GPU and must be configured correctly according to your base operating system.

   Centos 7.9/RHEL 7.9:
   If you're using Centos 7.9 or RHEL 7.9 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1227) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.10.0-centos7
```

   RHEL 8.4/8.6 and SLES 15 SP3
   If you're using RHEL 8.4/8.6 or SLES 15 SP3 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1227) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.10.0-ubi8
```

   Ubuntu 18.04 and 20.04
   If you're using Ubuntu 18.04 or 20.04 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1227) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.11.0-ubuntu20.04
```

5.9.6.7.4 Enable DKP Catalog Applications and Install Kommander

If you have an Enterprise license and would like to use the DKP Catalog Applications function, follow these steps:

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

   ```
   apiVersion: config.kommander.mesosphere.io/v1alpha1
   kind: Installation
   catalog:
     repositories:
       - name: dkp-catalog-applications
         labels:
           kommander.d2iq.io/project-default-catalog-repository: "true"
           kommander.d2iq.io/workspace-default-catalog-repository: "true"
           kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
         gitRepositorySpec:
           url: https://github.com/mesosphere/dkp-catalog-applications
           ref:
             tag: v2.5.0
   ```

   ⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

   ```
   dkp install kommander --installer-config kommander.yaml --kubeconfig=$
   {CLUSTER_NAME}.co
   ```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 99)
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

5.9.6.7.5 Next Step:

Pre-provisioned GPU: Verify Install and Log in to UI (see page 225)

5.9.6.8 Pre-provisioned GPU: Verify Install and Log in to UI

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander. After the CLI successfully installs the components, it will wait for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Released helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Released` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

5.9.6.8.1 Failed HelmReleases

If any application fails to successfully deploy, you can check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

5.9.6.8.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 524). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 524):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

5.9.6.8.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 510) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

For more GPU information (see page 860), see the section for GPU under Day 2 - Cluster Operations Management.

5.9.6.8.4 Next Step:

Pre-provisioned GPU: Subsequent Cluster Creation (see page 228)

5.9.6.9 Pre-provisioned GPU: Create Managed Clusters Using the DKP CLI

After initial cluster creation, you have the ability to create additional clusters from the CLI. In a previous step, the new cluster was created as Self-managed which allows it to be a Management cluster or a stand alone cluster. Subsequent new clusters are not self-managed as they will likely be Managed or Attached clusters to this Management Cluster.

When creating Managed clusters (see page 97), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 97)!

5.9.6.9.1 Name Your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[165] for more naming information.

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-additional>
```

If a custom AMI was created using Konvoy Image Builder, the custom `ami` id is printed and written to `./manifest.json`. To use the built `ami` with Konvoy, specify it with the `--ami` flag when calling cluster create command below.

165 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

5.9.6.9.2 Choose a Workspace for the New Cluster

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a Workspace[166].

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

5.9.6.9.3 Create a Kubernetes Cluster

Once you've defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster.

ⓘ Before you create a new DKP cluster below, choose an external load balancer (LB) or virtual IP (see page 1114) and use the corresponding `dkp create cluster` command.

In a pre-provisioned environment, use the Kubernetes CSI and third party drivers for local volumes and other storage devices in your data center.

⚠ DKP uses local static provisioner as the default storage provider (see page 103) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[167] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory YAML (see page 127) previously created.

166 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919315/Workspaces#Create-a-Workspace
167 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

1. This command uses the default external load balancer (LB) option:

```
dkp create cluster preprovisioned \
  --cluster-name ${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --ami <ami> \
  --namespace=${WORKSPACE_NAMESPACE}
```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

2. Use the wait command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

   Output:

```
cluster.cluster.x-k8s.io/preprovisioned-additional condition met
```

NOTE: Depending on the cluster size, it will take a few minutes to create.

5.9.6.9.4 Manually Attach a DKP CLI Cluster to the Management Cluster

1. Find out the `name` of the created `Cluster`, so you can reference it later:

```
kubectl -n <workspace_namespace> get clusters
```

2. Attach the cluster by creating a `KommanderCluster`:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
```

```
  name: <cluster_name>
  namespace: <workspace_namespace>
spec:
  kubeconfigRef:
    name: <cluster_name>-kubeconfig
  clusterRef:
    capiCluster:
      name: <cluster_name>
EOF
```

5.9.6.9.5 Next Step:

Day 2 - Cluster Operations Management (see page 510)

5.9.7 Pre-provisioned Air-gapped with GPU Install

This section provides instructions to install DKP in a Pre-provisioned air-gapped environment using GPU.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

> NOTE: For air-gapped, ensure you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz` , so you can extract the tarball on next page.

5.9.7.1 Next Step:

Pre-provisioned Air-gapped GPU: Configure Environment (see page 231)

5.9.7.2 Pre-provisioned Air-gapped GPU: Configure Environment

> If the NVIDIA runfile[168] installer has not been downloaded, then retrieve and install the download first by running the following command. The first line in the command below downloads and installs the runfile and the second line places it in the artifacts directory.

168 https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html#runfile

- ```
  curl -O https://download.nvidia.com/XFree86/Linux-x86_64/470.82.01/NVIDIA-Linux-x86_64-470.82.01.run
  mv NVIDIA-Linux-x86_64-470.82.01.run artifacts
  ```

The instructions below outline how to fulfill the requirements for a pre-provisioned infrastructure when using an air-gapped environment. In order to create a cluster, you must first setup the environment with necessary artifacts. All artifacts for Pre-provisioned Air-gapped environments need to get onto the bastion host. Artifacts needed by nodes must be unpacked and distributed on the bastion before other provisioning will work in the absence of an internet connection.

There is a new complete DKP air-gapped bundle available to download (see page 71) which contains all the DKP components needed for air-gapped installation. (i.e. `dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz`)

5.9.7.2.1 Setup Process:

1. The bootstrap image must be extracted and loaded onto the bastion host (see page 1162).
2. Artifacts must be copied onto cluster hosts for nodes to access.
3. If using GPU, those artifacts must be positioned locally.
4. Docker Registry seeded with images locally.

5.9.7.2.2 Load the Bootstrap Image

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz` from the download site (see page 71) mentioned above, extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz && cd dkp-v2.5.2
   ```

2. Load the bootstrap Docker image on your bastion machine:

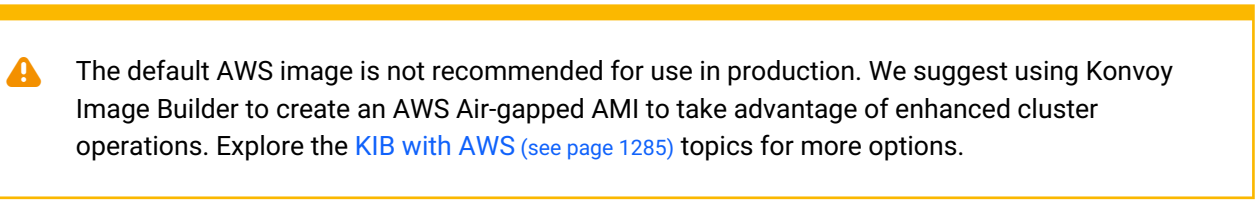
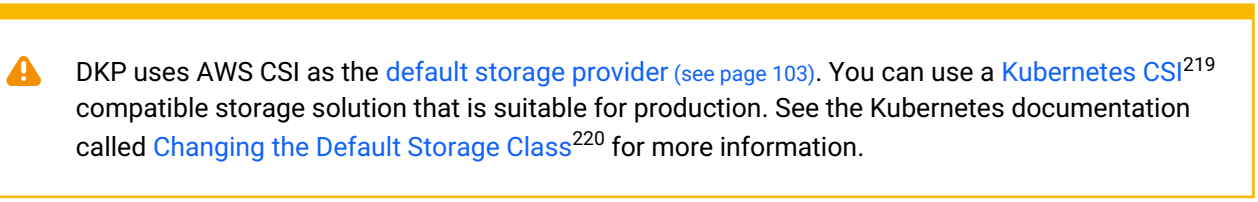
   ```
   docker load -i konvoy-bootstrap-image-v2.5.2.tar
   ```

5.9.7.2.3 Copy Air-gapped Artifacts onto Cluster Hosts

Using the Konvoy Image Builder (see page 1282), you can copy the required artifacts onto your cluster hosts.

Assuming you have downloaded `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz`, continue below:
   ```

1. The Kubernetes image bundle will be located in `kib/artifacts/images` and you will want to verify image and artifacts.
    a. Verify the image bundles exist in `artifacts/images`:

    ```
    $ ls artifacts/images/
    kubernetes-images-1.25.4-d2iq.1.tar kubernetes-images-1.25.4-d2iq.1-
    fips.tar
    ```

    b. Verify the artifacts for your OS exist in the `artifacts/` directory and export the appropriate variables:

    ```
    $ ls artifacts/
    1.25.4_centos_7_x86_64.tar.gz       1.25.4_redhat_8_x86_64_fips.tar.gz
    containerd-1.6.17-d2iq.1-rhel-7.9-x86_64.tar.gz   containerd-1.6.17-
    d2iq.1-rhel-8.6-x86_64_fips.tar.gz  pip-packages.tar.gz
    1.25.4_centos_7_x86_64_fips.tar.gz  1.25.4_rocky_9_x86_64.tar.gz
    containerd-1.6.17-d2iq.1-rhel-7.9-x86_64_fips.tar.gz  containerd-1.6.17-
    d2iq.1-rocky-9.0-x86_64.tar.gz
    1.25.4_redhat_7_x86_64.tar.gz       1.25.4_ubuntu_20_x86_64.tar.gz
    containerd-1.6.17-d2iq.1-rhel-8.4-x86_64.tar.gz   containerd-1.6.17-
    d2iq.1-rocky-9.1-x86_64.tar.gz
    1.25.4_redhat_7_x86_64_fips.tar.gz  containerd-1.6.17-d2iq.1-centos-7.9-
    x86_64.tar.gz       containerd-1.6.17-d2iq.1-rhel-8.4-x86_64_fips.tar.gz
    containerd-1.6.17-d2iq.1-ubuntu-20.04-x86_64.tar.gz  NVIDIA-Linux-
    x86_64-470.82.01.run
    1.25.4_redhat_8_x86_64.tar.gz       containerd-1.6.17-d2iq.1-centos-7.9-
    x86_64_fips.tar.gz  containerd-1.6.17-d2iq.1-rhel-8.6-x86_64.tar.gz
    images
    ```

    c. For example, for RHEL 8.4 you would set:

    ```
    export OS_PACKAGES_BUNDLE=1.25.4_redhat_8_x86_64.tar.gz
    export CONTAINERD_BUNDLE=containerd-1.6.17-d2iq.1-rhel-8.4-x86_64.tar.gz
    ```

2. Export the following environment variables, ensuring that all control plane and worker nodes are included:

    ```
    export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
    export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
    export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
    export WORKER_1_ADDRESS="<worker-address-1>"
    export WORKER_2_ADDRESS="<worker-address-2>"
    export WORKER_3_ADDRESS="<worker-address-3>"
    export WORKER_4_ADDRESS="<worker-address-4>"
    export SSH_USER="<ssh-user>"
    export SSH_PRIVATE_KEY_FILE="<private key file>"
    ```

`SSH_PRIVATE_KEY_FILE` must be either the name of the SSH private key file in your working directory or an absolute path to the file in your user's home directory.

3. Generate an `inventory.yaml` for GPU nodes which is automatically picked up by the `konvoy-image upload` in the next step.

```
cat <<EOF >  gpu_inventory.yaml
all:
  vars:
    ansible_port: 22
    ansible_ssh_private_key_file: $SSH_PRIVATE_KEY_FILE
    ansible_user: $SSH_USER

  hosts:
    $GPU_WORKER_1_ADDRESS:
      ansible_host: $GPU_WORKER_1_ADDRESS
EOF

cat <<EOF > inventory.yaml
all:
  vars:
    ansible_user: $SSH_USER
    ansible_port: 22
    ansible_ssh_private_key_file: $SSH_PRIVATE_KEY_FILE
  hosts:
    $CONTROL_PLANE_1_ADDRESS:
      ansible_host: $CONTROL_PLANE_1_ADDRESS
    $CONTROL_PLANE_2_ADDRESS:
      ansible_host: $CONTROL_PLANE_2_ADDRESS
    $CONTROL_PLANE_3_ADDRESS:
      ansible_host: $CONTROL_PLANE_3_ADDRESS
    $WORKER_1_ADDRESS:
      ansible_host: $WORKER_1_ADDRESS
    $WORKER_2_ADDRESS:
      ansible_host: $WORKER_2_ADDRESS
    $WORKER_3_ADDRESS:
      ansible_host: $WORKER_3_ADDRESS
    $WORKER_4_ADDRESS:
      ansible_host: $WORKER_4_ADDRESS
EOF
```

4. Upload the artifacts to the gpu nodepool with the `nvidia-runfile` flag with the following command:

```
konvoy-image upload artifacts --inventory-file=gpu_inventory.yaml \
              --container-images-dir=./artifacts/images/ \
              --os-packages-bundle=./artifacts/$OS_PACKAGES_BUNDLE \
              --containerd-bundle=artifacts/$CONTAINERD_BUNDLE \
```

```
            --pip-packages-bundle=./artifacts/pip-packages.tar.gz \
            --nvidia-runfile=./artifacts/NVIDIA-Linux-x86_64-470.82.01.run
```

KIB uses variable overrides (see page 1330) to specify base image and container images to use in your new machine image. The variable overrides files for NVIDIA and FIPS can be ignored unless adding an overlay feature.

- Use the `--overrides overrides/fips.yaml,overrides/offline-fips.yaml` flag with manifests located in the overrides directory[169] or see these pages in the documentation:
  - FIPS Overrides (see page 1331)
  - Create FIPS 140 Images (see page 1344)

5.9.7.2.3.1 Next Step:

Pre-provisioned Air-gapped GPU: Load the Registry (see page 235)

5.9.7.3 Pre-provisioned Air-gapped GPU: Load the Registry

Before creating a Kubernetes cluster, you need the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1162) and either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1349) page for more information.

1. Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz`, extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz && cd dkp-v2.5.0
   ```

2. Set an environment variable with your registry address with this command:

169 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

```
export REGISTRY_ADDRESS=<registry-address>:<registry-port>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

3. Execute the following command to load the air-gapped image bundle into your private registry:

```
dkp push image-bundle --image-bundle ./container-images/konvoy-image-bundle-
v2.5.0.tar --to-registry $REGISTRY_ADDRESS --to-registry-username
$REGISTRY_USERNAME --to-registry-password $REGISTRY_PASSWORD
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

To increase Docker Hub's rate limit[170] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.

5.9.7.3.1 Kommander Load Images

If you are operating in an air-gapped environment, a local container registry (see page 1349) containing all the necessary installation images, including the Kommander images is required. See below for how to push the necessary images to this registry.

For the basic air-gapped `kommander` image bundle, run the command below:

1. Run the following command to load the image bundle:

```
dkp push image-bundle --image-bundle ./container-images/kommander-image-bundle-
v2.5.0.tar --to-registry $REGISTRY_ADDRESS
```

For **DKP Catalog Applications**, also perform this image load:

1. Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

170 https://docs.docker.com/docker-hub/download-rate-limit/

```
dkp push image-bundle --image-bundle ./container-images/dkp-catalog-
applications-image-bundle-v2.5.0.tar --to-registry $REGISTRY_ADDRESS
```

5.9.7.3.2 Next Step:

Pre-provisioned Air-gapped GPU: Nodepool Secrets and Overrides (see page 238)

5.9.7.4 Pre-provisioned Air-gapped GPU: Nodepool Secrets and Overrides

5.9.7.4.1 DKP has introduced the `nvidia-runfile` flag for Air-gapped Pre-provisioned environments. If the NVIDIA runfile[171] installer has not been downloaded, then retrieve and install the download first by running the following command. The first line in the command below downloads and installs the runfile and the second line places it in the artifacts directory (you must create an `artifacts` directory if it doesn't already exist).

```
curl -O https://download.nvidia.com/XFree86/Linux-x86_64/470.82.01/NVIDIA-Linux-x86_64-470.82.01.run
mv NVIDIA-Linux-x86_64-470.82.01.run artifacts
```

DKP supported NVIDIA driver[172] version is 470.x.

1. Create the secret that GPU nodepool would use, this secret is populated from the KIB overrides. In this example we have a file called, `overrides/nvidia.yaml`. It should resemble this:

```
gpu:
  types:
    - nvidia
build_name_extra: "-nvidia"
```

2. Create a secret on the bootstrap cluster that is populated from the above file. We will name it

`${CLUSTER_NAME}-user-overrides`

```
kubectl create secret generic ${CLUSTER_NAME}-user-overrides --from-file=overrides.yaml=overrides/nvidia.yaml
```

3. Create an inventory and nodepool with the instructions below and use the `$CLUSTER_NAME-user-overrides` secret.

Follow these steps:

171 https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html#runfile
172 https://www.nvidia.com/Download/Find.aspx

1. Create an inventory object that has the same name as the node pool you're creating, and the details of the pre-provisioned machines that you want to add to it. For example, to create a node pool named `gpu-nodepool` an inventory named `gpu-nodepool` must be present in the same namespace:

   ```
   apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
   kind: PreprovisionedInventory
   metadata:
     name: ${MY_NODEPOOL_NAME}
   spec:
     hosts:
       - address: ${IP_OF_NODE}
     sshConfig:
       port: 22
       user: ${SSH_USERNAME}
       privateKeyRef:
         name: ${NAME_OF_SSH_SECRET}
         namespace: ${NAMESPACE_OF_SSH_SECRET}
   ```

2. (Optional) If your pre-provisioned machines have overrides, you must create a secret that includes all of the overrides you want to provide in one file. Create an override secret using the instructions detailed on this page.

3. Once the `PreprovisionedInventory` object and overrides are created, create a node pool:

   ```
   dkp create nodepool preprovisioned -c ${MY_CLUSTER_NAME} $
   {MY_NODEPOOL_NAME} --override-secret-name ${MY_OVERRIDE_SECRET}
   ```

- Advanced users can use a combination of the `--dry-run` and `--output=yaml` or `--output-directory=<existing-directory>` flags to get a complete set of node pool objects to modify locally or store in version control.

For more information regarding this flag or others, please refer to the dkp create nodepool section of the documentation for either cluster or nodepool and select your provider.

For more information, see:

- Pre-provisioned Create and Delete Node Pools
- Pre-provisioned Air-gapped Define Environment

Next Step:

Pre-provisioned Air-gapped GPU: Define Infrastructure (see page 240)

5.9.7.5 Pre-provisioned Air-gapped GPU: Define Infrastructure

Define the cluster hosts and infrastructure

Konvoy needs to know how to access your cluster hosts. This is done using inventory resources. For initial cluster creation, you must define a control-plane and at least one worker pool.

5.9.7.5.1 Define your Infrastructure

1. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
export WORKER_1_ADDRESS="<worker-address-1>"
export WORKER_2_ADDRESS="<worker-address-2>"
export WORKER_3_ADDRESS="<worker-address-3>"
export WORKER_4_ADDRESS="<worker-address-4>"
export SSH_USER="<ssh-user>"
export SSH_PRIVATE_KEY_SECRET_NAME="$CLUSTER_NAME-ssh-key"
```

2. Use the following template to help you define your infrastructure. The environment variables that you set in the previous step automatically replace the variable names when the inventory YAML file is created.

```
cat <<EOF > preprovisioned_inventory.yaml
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-control-plane
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    # Create as many of these as needed to match your infrastructure
    # Note that the command line parameter --control-plane-replicas determines
how many control plane nodes will actually be used.
    #
    - address: $CONTROL_PLANE_1_ADDRESS
    - address: $CONTROL_PLANE_2_ADDRESS
    - address: $CONTROL_PLANE_3_ADDRESS
```

```
  sshConfig:
    port: 22
    # This is the username used to connect to your infrastructure. This user
must be root or
    # have the ability to use sudo without a password
    user: $SSH_USER
    privateKeyRef:
      # This is the name of the secret you created in the previous step. It
must exist in the same
      # namespace as this inventory object.
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-md-0
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    - address: $WORKER_1_ADDRESS
    - address: $WORKER_2_ADDRESS
    - address: $WORKER_3_ADDRESS
    - address: $WORKER_4_ADDRESS
  sshConfig:
    port: 22
    user: $SSH_USER
    privateKeyRef:
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
EOF
```

5.9.7.5.2 Next Step:

Pre-provisioned Air-gapped GPU: Define Control Plane Endpoint (see page 241)

5.9.7.6 Pre-provisioned Air-gapped GPU: Define Control Plane Endpoint

5.9.7.6.1 Define the Control Plane Endpoint for your cluster as well as connection mechanism

A control plane should have three, five, or seven nodes, so it can remain available if one or more nodes fail. A control plane with one node should not be used in production.

In addition, the control plane should have an endpoint that remains available if some nodes fail.

```
                          -------- cp1.example.com:6443
                          |
      lb.example.com:6443 ---------- cp2.example.com:6443
                          |
                          -------- cp3.example.com:6443
```

In this example, the control plane endpoint host is `lb.example.com`, and the control plane endpoint port is `6443`. The control plane nodes are `cp1.example.com`, `cp2.example.com`, and `cp3.example.com`. The port of each API server is `6443`.

5.9.7.6.2 Select your Connection Mechanism

A virtual IP is the address that the client uses to connect to the service. A load balancer is the device that distributes the client connections to the backend servers. Before you create a new DKP cluster, choose an external load balancer(LB) or virtual IP.

- **External load balancer**

It is recommended that an external load balancer be the control plane endpoint. To distribute request load among the control plane machines, configure the load balancer to send requests to all the control plane machines. Configure the load balancer to send requests only to control plane machines that are responding to API requests.

- **Built-in virtual IP**

If an external load balancer is not available, use the built-in virtual IP (see page 1114). The virtual IP is *not* a load balancer; it does not distribute request load among the control plane machines. However, if the machine receiving requests does not respond to them, the virtual IP automatically moves to another machine.

5.9.7.6.3 Single-Node Control Plane

Do not use a single-node control plane in a production cluster.

A control plane with one node can use its single node as the endpoint, so you will not require an external load balancer, or a built-in virtual IP. At least one control plane node must always be running. Therefore, to upgrade a cluster with one control plane node, a spare machine must be available in the control plane inventory. This machine is used to provision the new node before the old node is deleted.

Modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1351).

When the API server endpoints are defined, you can create the cluster using the link in Next Step below.

5.9.7.6.4 Known Limitations

Be aware of these limitations in the current release of DKP.

The control plane endpoint port is also used as the API server port on each control plane machine. The default port is 6443. Before you create the cluster, ensure the port is available for use on each control plane machine.

5.9.7.6.5 Next Step:

Pre-provisioned Air-gapped GPU: Create a Cluster (see page 243)

5.9.7.7 Pre-provisioned Air-gapped GPU: Create a Management Cluster

If your cluster is in an air-gapped environment or you have a local registry, you must provide additional arguments when creating the cluster. These tell the cluster where to locate the local registry to use by defining the URL.

```
export REGISTRY_URL=<https/http>://<registry-address>:<registry-port>
export REGISTRY_CA=<path to the CA on the bastion>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

- `REGISTRY_URL` : the address of an existing registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
- `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
- `REGISTRY_PASSWORD` : optional if username is not set.

5.9.7.7.1 Create a Cluster with GPU AMI

If a custom AMI was created using Konvoy Image Builder, the custom `ami` id is printed and written to `./manifest.json` .

To use the built `ami` with Konvoy, specify it with the `--ami` flag when calling cluster create in Step 1.

For GPU Steps in Pre-provisioned (see page 0) section of the documentation to use the overrides/nvidia.yaml.

For GPU Steps in Pre-provisioned (see page 0) section of the documentation to use the overrides/nvidia.yaml.

Additional helpful information can be found in the NVIDIA Device Plug-in[173] for Kubernetes instructions and the Installation Guide of Supported Platforms[174].

Before you create a new DKP cluster below, choose an external load balancer or virtual IP (see page 1114) and use the corresponding `dkp create cluster` command.

5.9.7.7.2 Name Your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[175] for more naming information.

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-example>
```

173 https://github.com/NVIDIA/k8s-device-plugin/blob/master/README.md
174 https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html
175 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

5.9.7.7.3 Create an Air-gapped Kubernetes Cluster

Once you've defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster.

> ⚠ DKP uses local static provisioner as the default storage provider (see page 103) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[176] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory.

The create cluster command below includes the `--self-managed` flag. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

1. Execute this command to create a cluster with a GPU AMI using the default external load balancer option:

```
dkp create cluster preprovisioned \
  --cluster-name=${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --registry-mirror-url=${REGISTRY_URL} \
  --registry-mirror-cacert=${REGISTRY_CA} \
  --registry-mirror-username=${REGISTRY_USERNAME} \
  --registry-mirror-password=${REGISTRY_PASSWORD} \
  --ami <ami> \
  --self-managed
```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

176 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

2. **Virtual IP ALTERNATIVE** - if you don't have an external LB, and wish to use a VIRTUAL IP (see page 1114) provided by kube-vip, specify these flags example below::

```
dkp create cluster preprovisioned \
    --cluster-name ${CLUSTER_NAME} \
    --control-plane-endpoint-host 196.168.1.10 \
    --virtual-ip-interface eth1
```

   The output from this command is shortened here for reading clarity, but should start like this:

```
Generating cluster resources
cluster.cluster.x-k8s.io/preprovisioned-example created
cont.........
```

3. Create the node pool after cluster creation:

```
dkp create nodepool aws -c ${CLUSTER_NAME} \
  --instance-type p2.xlarge \
  --ami-id=${AMI_ID_FROM_KIB} \
  --replicas=1 ${NODEPOOL_NAME} \
  --kubeconfig=${CLUSTER_NAME}.conf
```

4. Use the wait command to monitor the cluster control-plane readiness:Use the wait command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

   Output:

```
cluster.cluster.x-k8s.io/preprovisioned-example condition met
```

NOTE: Depending on the cluster size, it will take a few minutes to create.

When the command completes, you will have a running Kubernetes cluster! Use this command to get the Kubernetes `kubeconfig` for the new cluster and proceed to installing the DKP Kommander UI:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

If changing the Calico encapsulation (see page 1110), D2iQ recommends changing it after cluster creation, but before production.

5.9.7.7.4 Audit logs

To modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1351).

5.9.7.7.4.1 Further Steps:

- Modify the Calico installation[177]
- Use the built-in Virtual IP[178]
- Provision on the Flatcar Linux OS[179]
- Use an HTTP proxy[180]
- Use alternate pod or service subnets[181]

1. **Virtual IP ALTERNATIVE** - if you don't have an external LB, and wish to use a VIRTUAL IP (see page 1114) provided by kube-vip, specify these flags example below:

   ```
   dkp create cluster preprovisioned \
       --cluster-name ${CLUSTER_NAME} \
       --control-plane-endpoint-host 196.168.1.10 \
       --virtual-ip-interface eth1
   ```

   The output from this command is shortened here for reading clarity, but should start like this:

   ```
   Generating cluster resources
   cluster.cluster.x-k8s.io/preprovisioned-example created
   cont.........
   ```

2. Use the wait command to monitor the cluster control-plane readiness:

177 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/edit-v2/165347706#
178 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/edit-v2/165347706#
179 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/edit-v2/165347706#
180 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/edit-v2/165347706#
181 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/edit-v2/165347706#

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

Output:

```
cluster.cluster.x-k8s.io/preprovisioned-example condition met
```

NOTE: Depending on the cluster size, it will take a few minutes to create.

When the command completes, you will have a running Kubernetes cluster! Use this command to get the Kubernetes `kubeconfig` for the new cluster and proceed to installing the DKP Kommander UI:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

If changing the Calico encapsulation (see page 1110), D2iQ recommends changing it after cluster creation, but before production.

5.9.7.7.4.2 Audit Logs

To modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1351).

5.9.7.7.4.3 Further Steps:

For more customized cluster creation, access the Pre-Provisioned Additional Configurations (see page 1078) section.

NOTE: The section mentioned above is for overrides for your clusters (see page 1341), custom flags, and more that specify the secret as part of the create cluster command. If these are not specified, the overrides for your nodes will not be applied.

5.9.7.7.4.4 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1360).

5.9.7.7.4.5 Next Step:

Pre-provisioned Air-gapped GPU: Configure MetalLB (see page 249)

5.9.7.8 Pre-provisioned Air-gapped GPU: Configure MetalLB

5.9.7.8.1 Create a MetalLB configmap for your pre-provisioned infrastructure.

Choose one of the following two protocols you want to use to announce service IPs. If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can continue the installation process Pre-provisioned Air-gapped GPU: Install Kommander (see page 251).

To use MetalLB, create a MetalLB configMap for your Pre-provisioned infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

5.9.7.8.2 Layer 2 configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to ARP requests on your local network directly, to give the machine's MAC address to clients.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
```

```
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

5.9.7.8.3 BGP configuration

For a basic configuration featuring one BGP router and one IP address range, you need 4 pieces of information:

- The router IP address that MetalLB should connect to,
- The router's AS number,
- The AS number MetalLB should use,
- An IP address range expressed as a CIDR prefix.

As an example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
```

```
      - 192.168.10.0/24
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

5.9.7.8.3.1 Next Step:

Pre-provisioned Air-gapped GPU: Install Kommander (see page 251)

5.9.7.9 Pre-provisioned Air-gapped GPU: Install Kommander

5.9.7.9.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 119) .
- Ensure you have a default StorageClass (see page 1254).
- Ensure you have loaded all necessary images for your configuration. See Load the Images into Your Registry (see page 1258).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

5.9.7.9.2 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1227) for the deployment:

   ```
   dkp install kommander --init --airgapped > kommander.yaml
   ```

4. Edit the installer file to include configuration overrides for the `rook-ceph-cluster`. DKP's default configuration ships Ceph with PVC based storage[182] which requires your CSI provider to support PVC with type `volumeMode: Block`. As this is not possible with the default local static provisioner (see page 103), you can install Ceph in host storage[183] mode.

   You can choose whether Ceph's object storage daemon (osd) pods should consume all or just some of the devices on your nodes. Include **one** of the following Overrides:

   a. To automatically assign all raw storage devices on all nodes to the Ceph cluster:

   ```
   rook-ceph-cluster:
       enabled: true
       values: |
         cephClusterSpec:
           storage:
             storageClassDeviceSets: []
             useAllDevices: true
             useAllNodes: true
             deviceFilter: "<<value>>"
   ```

   b. To assign specific storage devices on all nodes to the Ceph cluster:

   ```
   rook-ceph-cluster:
       enabled: true
       values: |
         cephClusterSpec:
           storage:
             storageClassDeviceSets: []
             useAllNodes: true
             useAllDevices: false
             deviceFilter: "^sdb."
   ```

   Note: If you want to assign specific devices to specific nodes using the `deviceFilter` option, refer to Specific Nodes and Devices[184]. For general information on the `deviceFilter` value, refer to Storage Selection Settings[185].

5. *If required:* Customize your `kommander.yaml`.

   See Kommander Additional Install Configurations (see page 1227) for customization options. Some of them include:
   Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, etc.

182 https://rook.io/docs/rook/v1.10/CRDs/Cluster/pvc-cluster/
183 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/
184 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/#specific-nodes-and-devices
185 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

5.9.7.9.3 Enable GPU Resources

1. In the same `kommander.yaml` file, enable Nvidia platform services.

```
apps:
  nvidia-gpu-operator:
   enabled: true
```

2. Append the correct Toolkit version based on your OS:

   The **NVIDIA Container Toolkit** allows users to run GPU accelerated containers. The toolkit includes a container runtime library and utilities to automatically configure containers to leverage NVIDIA GPU and must be configured correctly according to your base operating system.

   Centos 7.9/RHEL 7.9:
   If you're using Centos 7.9 or RHEL 7.9 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file*[186][187] or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.13.1-centos7
```

   RHEL 8.4/8.6 and SLES 15 SP3
   If you're using RHEL 8.4/8.6 or SLES 15 SP3 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file*[188][189] or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.13.1-ubi8
```

186 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29892161/Kommander+Customizations
187 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29892161/Kommander+Customizations
188 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29892161/Kommander+Customizations
189 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29892161/Kommander+Customizations

Ubuntu 18.04 and 20.04
If you're using Ubuntu 18.04 or 20.04 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file*[190][191] or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.13.1-ubuntu20.04
```

5.9.7.9.4 Enable DKP Catalog Applications and Install Kommander

If you have an Enterprise license and would like to use the DKP Catalog Applications function, follow these steps:

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

   ```
   apiVersion: config.kommander.mesosphere.io/v1alpha1
   kind: Installation
   catalog:
     repositories:
       - name: dkp-catalog-applications
         labels:
           kommander.d2iq.io/project-default-catalog-repository: "true"
           kommander.d2iq.io/workspace-default-catalog-repository: "true"
           kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
         gitRepositorySpec:
           url: https://github.com/mesosphere/dkp-catalog-applications
           ref:
             tag: v2.5.0
   ```

   ⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

190 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29892161/Kommander+Customizations
191 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29892161/Kommander+Customizations

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.5.0.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.5.0.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.5.0.tar.gz
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 99)
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

5.9.7.9.5 Next Step:

Pre-provisioned Air-gapped GPU: Verify Install and Log in to UI (see page 255)

5.9.7.10 Pre-provisioned Air-gapped GPU: Verify Install and Log in to UI

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander. After the CLI successfully installs the components, it will wait for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Released helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Released` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

5.9.7.10.1 Failed HelmReleases

If any application fails to successfully deploy, you can check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 524). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 524):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

5.9.7.10.2 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 510) section of the documentation. The majority of this customization such as

attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

For more GPU information (see page 860), see the section for GPU under Day 2 - Cluster Operations Management.

5.9.7.10.3 Next Step:

Pre-provisioned Air-gapped with GPU: Subsequent Clusters (see page 258)

5.9.7.11 Pre-provisioned Air-gapped with GPU: Create Managed Clusters Using the DKP CLI

After initial cluster creation, you have the ability to create additional clusters from the CLI. In a previous step, the new cluster was created as Self-managed which allows it to be a Management cluster or a stand alone cluster. Subsequent new clusters are not self-managed as they will likely be Managed or Attached clusters to this Management Cluster.

> ⚠ When creating Managed clusters (see page 97), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 97)!

5.9.7.11.1 Choose a Workspace for the New Cluster

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a Workspace[192].

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

192 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919315/Workspaces#Create-a-Workspace

5.9.7.11.2 Name Your Cluster

> The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[193] for more naming information.
>
> When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-additional>
```

5.9.7.11.3 Create a Cluster with GPU AMI

If a custom AMI was created using Konvoy Image Builder, the custom `ami` id is printed and written to `./manifest.json`. To use the built `ami` with Konvoy, specify it with the `--ami` flag when calling cluster create in Step 1 in the next section where you create your Kubernetes cluster.

5.9.7.11.4 Create a Kubernetes Cluster

Once you've defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster.

> Before you create a new DKP cluster below, choose an external load balancer (LB) or virtual IP (see page 1114) and use the corresponding `dkp create cluster` command.

In a pre-provisioned environment, use the Kubernetes CSI and third party drivers for local volumes and other storage devices in your data center.

193 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

DKP uses local static provisioner as the default storage provider (see page 103) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[194] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory YAML (see page 127) previously created.

1. This command uses the default external load balancer (LB) option:

```
dkp create cluster preprovisioned \
  --cluster-name ${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --registry-mirror-url=${_REGISTRY_URL} \
  --registry-mirror-cacert=${_REGISTRY_CA} \
  --registry-mirror-username=${_REGISTRY_USERNAME} \
  --registry-mirror-password=${_REGISTRY_PASSWORD} \
  --namespace=${WORKSPACE_NAMESPACE}
```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

2. Use the wait command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

   Output:

```
cluster.cluster.x-k8s.io/preprovisioned-additional condition met
```

194 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

NOTE: Depending on the cluster size, it will take a few minutes to create.

5.9.7.11.5 Manually Attach a DKP CLI Cluster to the Management Cluster

1. Find out the `name` of the created `Cluster`, so you can reference it later:

```
kubectl -n <workspace_namespace> get clusters
```

2. Attach the cluster by creating a `KommanderCluster`:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: <cluster_name>
  namespace: <workspace_namespace>
spec:
  kubeconfigRef:
    name: <cluster_name>-kubeconfig
  clusterRef:
    capiCluster:
      name: <cluster_name>
EOF
```

5.9.7.11.6 Next Step:

Day 2 - Cluster Operations Management (see page 510)

5.10 AWS Install Options

For an environment that is on the AWS Infrastructure, install options are provided for you in this one location. Remember, there are always more options in the Additional Infrastructure Configurations (see page 931) sections, but this will get you operative in the most common scenarios.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

Below are all the supported environment combinations: (Refer to this page to check if your OS is a Supported Operating System (see page 62).)

- AWS Install (see page 262)
- AWS Air-gapped Install (see page 274)
- AWS with FIPS Install (see page 289)
- AWS Air-gapped FIPS Install (see page 302)
- AWS with GPU Install (see page 316)
- AWS Air-gapped with GPU Install (see page 332)

Additional Resource Information specific to AWS is below.

- **Control Plane Nodes** - DKP on AWS defaults to deploying an `m5.xlarge` instance with an 80GiB root volume for control plane nodes, which meets the above resource requirements (see page 110).
- **Worker Nodes** - DKP on AWS defaults to deploying a `m5.2xlarge` instance with an 80GiB root volume for worker nodes, which meets the above resource requirements (see page 110).

5.10.1 AWS Install

This section provides instructions to install DKP in an AWS non-air-gapped environment.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

5.10.1.1 AWS Prerequisites

Before you begin using Konvoy with AWS, you must:

1. Follow the steps to create a Minimal Permissions and Role to Create Clusters (see page 940)..
2. Create Cluster IAM Policies and Roles (see page 946).
3. Export the AWS region where you want to deploy the cluster:

   ```
   export AWS_REGION=us-west-2
   ```

4. Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

   ```
   export AWS_PROFILE=<profile>
   ```

If using AWS ECR as your local private registry, more information can be found on the Local Registry Tools[195] page.

To deploy a cluster with a custom image in a region where CAPI images[196] are not provided, you need to use Konvoy Image Builder[197] to create your own image for the region.

5.10.1.2 Next Step:

AWS: Create an Image (see page 263)

5.10.1.3 AWS: Create an Image

5.10.1.3.1 Learn how to build a custom AMI for use with DKP

This procedure describes how to use the Konvoy Image Builder (see page 1282) (KIB) to create a Cluster API[198] compliant Amazon Machine Image (AMI). KIB uses variable overrides (see page 1330) to specify base image and container images to use in your new AMI.

The default AWS image is not recommended for use in production. We suggest using Konvoy Image Builder to create a custom AMI to take advantage of enhanced cluster operations. Explore the KIB with AWS (see page 1285) topics for more options.

5.10.1.3.2 Prerequisites

Before you begin, you must:

- Check the DKP Supported Kubernetes Versions (see page 1543).
- Check the Supported Infrastructure Operating Systems (see page 62)
- Download the KIB (see page 0) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
- Create a working `Docker` setup.
- Ensure you have met the minimal set of permissions from the AWS Image Builder Book[199].

195 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/210829371/Registry+Mirror+Tools
196 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html
197 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/90439791
198 https://cluster-api.sigs.k8s.io/
199 https://image-builder.sigs.k8s.io/capi/providers/aws.html#required-permissions-to-build-the-aws-amis

- A Minimal IAM Permissions for KIB (see page 1285) to create an Image for an AWS account using Konvoy Image Builder.

5.10.1.3.3 Extract KIB Bundle

Extract the bundle and `cd` into the extracted `konvoy-image-bundle-$VERSION` folder. The bundled version of `konvoy-image` contains an embedded `docker` image that contains all the requirements for building.

> The `konvoy-image` binary and all supporting folders are also extracted. When run, `konvoy-image` bind mounts the current working directory (`${PWD}`) into the container to be used.

- Set environment variables for AWS access[200]. The following variables must be set using your credentials including required IAM (see page 1285):

```
export AWS_ACCESS_KEY_ID
export AWS_SECRET_ACCESS_KEY
export AWS_DEFAULT_REGION
```

- If you have an override file (see page 1330) to configure specific attributes of your AMI file, add it.

5.10.1.3.4 Build the Image

Depending on which version of DKP (see page 1282) you are running, steps and flags will be different. To deploy in a region where CAPI images are not provided, you need to use KIB to create your own image for the region. For a list of supported AWS regions, refer to the Published AMI[201]information from AWS.

5.10.1.3.4.1 Execute the following to begin image creation:

Run the `konvoy-image` command to build and validate the image.

```
konvoy-image build aws images/ami/centos-79.yaml
```

By default it builds in the `us-west-2` region. to specify another region set the `--region` flag:

```
konvoy-image build aws --region us-east-1 images/ami/centos-79.yaml
```

200 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-envvars.html
201 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html

Ensure you have named the correct YAML file for your OS in the `konvoy-image build` command.

Once KIB provisions the image successfully, the `ami` id is printed and written to the `manifest.json` file. This file has an `artifact_id` field whose value provides the name of the AMI ID as shown in the example below:

```
{
      "name": "rhel-7.9-fips",
      "builder_type": "amazon-ebs",
      "build_time": 1659486130,
      "files": null,
      "artifact_id": "us-west-2:ami-0f2ef742482e1b829",
      "packer_run_uuid": "0ca500d9-a5f0-815c-6f12-aceb4d46645b",
      "custom_data": {
        "containerd_version": "",
        "distribution": "RHEL",
        "distribution_version": "7.9",
        "kubernetes_cni_version": "",
        "kubernetes_version": "1.24.5+fips.0"
      }
    }
```

5.10.1.3.5 Next Step:

AWS: Cluster Creation (see page 265)

5.10.1.4 AWS: Create the Management Cluster

5.10.1.4.1 Name Your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[202] for more naming information.

202 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable:

```
export CLUSTER_NAME=<aws-example>
```

To increase Docker Hub's rate limit[203] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster` command.

5.10.1.4.2 Create a New AWS Kubernetes Cluster

If you use these instructions to create a cluster on AWS using the DKP default settings without any edits to configuration files or additional flags, your cluster is deployed on an Ubuntu 20.04 operating system image (see page 62) with 3 control plane nodes, and 4 worker nodes.

The default AWS image is not recommended for use in production. D2iQ suggests using Konvoy Image Builder (see page 1285) to create a custom AMI and take advantage of enhanced cluster operations.

Create a Kubernetes cluster with the command(s) below:

DKP uses AWS CSI as the default storage provider (see page 103). You can use a Kubernetes CSI[204] compatible storage solution that is suitable for production. See the Kubernetes documentation called Changing the Default Storage Class[205] for more information.

1. Execute this command to create your Kubernetes cluster using any relevant flags:

203 https://docs.docker.com/docker-hub/download-rate-limit/
204 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
205 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

```
dkp create cluster aws \
--cluster-name=${CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--with-aws-bootstrap-credentials=true \
--self-managed
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

- A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing. As part of the underlying processing using the `--self-managed` flag, the DKP CLI:
  - creates a bootstrap cluster
  - creates a workload cluster
  - moves CAPI controllers from the bootstrap cluster to the workload cluster, making it self-managed
  - deletes the bootstrap cluster

To understand how this process works step by step, you can find a customizable Create a New Customized AWS Cluster (see page 953) under Additional Infrastructure Configuration.

5.10.1.4.3 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1360).

5.10.1.4.4 Next Step:

AWS: Install Kommander (see page 267)

5.10.1.5 AWS: Install Kommander

You have installed the Konvoy component and started creating a Management cluster. Now it is time to Install Kommander which completes the Management cluster and allows you to access the UI and attach new or existing managed clusters to monitor.

5.10.1.5.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 119).
- Ensure you have a default StorageClass (see page 1254).

- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

5.10.1.5.2 Create and customize your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1227) for the deployment:

   ```
   dkp install kommander --init > kommander.yaml
   ```

4. *If required:* Customize your `kommander.yaml`.
   See Kommander Additional Install Configurations (see page 1227) for customization options. Some of them include:
   Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1271)), etc.

5. *If required:* If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

   ```
   apps:
     traefik:
       enabled: true
       values: |
         service:
           annotations:
             service.beta.kubernetes.io/aws-load-balancer-internal: "true"
   ```

5.10.1.5.3 Enable DKP Catalog Applications and Install Kommander

If you have an Enterprise license and would like to use the DKP Catalog Applications function, follow these steps:

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.5.0
```

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.co
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see (Provide Context for Commands with a kubeconfig File (see page 99).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

5.10.1.5.4 Next Step:

AWS: Verify Install and Log in the UI (see page 270)

5.10.1.6 AWS: Verify Install and Log in the UI

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander. After the CLI successfully installs the components, it will wait for all applications to be ready by default.

> **NOTE**: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Released helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Released` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
```

```
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

5.10.1.6.1 Failed HelmReleases

If any application fails to successfully deploy, you can check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a "broken" release state such as "exhausted" or "another rollback/release in progress", you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

5.10.1.6.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 524). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 524):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

5.10.1.6.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 510) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

Now you can create additional new clusters (see page 272), or proceed to Day 2 Cluster Operations (see page 510) for attaching existing cluster..

5.10.1.6.4 Next Step:

AWS: Subsequent New Clusters (see page 272)

5.10.1.7 AWS: Create a Managed Cluster Using the DKP CLI

In previous steps, you created a new Management cluster, which is self-managed. When you use these steps to create new Managed clusters, they will become Attached clusters under your Management Cluster.

> ⚠ When creating Managed clusters (see page 97), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 97)!

5.10.1.7.1 Choose a Workspace for the New Cluster

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a Workspace[206].

206 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919315/Workspaces#Create-a-Workspace

```
kubectl get workspace -A
```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

```
export WORKSPACE_NAMESPACE=<workspace_namespace>
```

5.10.1.7.2 Name Your Cluster

> The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[207] for more naming information.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable:

```
export MANAGED_CLUSTER_NAME=<aws-additional>
```

5.10.1.7.3 Create New Cluster with the CLI

Execute this command to create your additional Kubernetes cluster using any relevant flags. This will create a new non-self-managed cluster that can be managed by management cluster (see page 265) you created in the previous section:

```
dkp create cluster aws \
--cluster-name=${MANAGED_CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--namespace=${WORKSPACE_NAMESPACE}
```

207 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

5.10.1.7.4 Manually Attach a DKP CLI Cluster to the Management Cluster

1. Find out the `name` of the created `Cluster`, so you can reference it later:

```
kubectl -n <workspace_namespace> get clusters
```

2. Attach the cluster by creating a `KommanderCluster`:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: <cluster_name>
  namespace: <workspace_namespace>
spec:
  kubeconfigRef:
    name: <cluster_name>-kubeconfig
  clusterRef:
    capiCluster:
      name: <cluster_name>
EOF
```

If you have existing clusters or want to create new clusters to attach, there are many ways to attach a cluster with various requirements and restrictions. To see all the options, visit the section in documentation Day 2 - Attach an Existing Kubernetes Cluster (see page 673).

5.10.1.7.5 Next Step:

Day 2 - Cluster Operations Management (see page 510)

5.10.2 AWS Air-gapped Install

This section provides the instructions to install DKP in an AWS air-gapped environment.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

5.10.2.1 AWS Prerequisites

Before you begin using Konvoy with AWS, you must:

1. Follow the steps to create a Minimal Permissions and Role to Create Clusters (see page 940)..
2. Create Cluster IAM Policies and Roles (see page 946).
3. Export the AWS region where you want to deploy the cluster:

```
export AWS_REGION=us-west-2
```

4. Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

```
export AWS_PROFILE=<profile>
```

If using AWS ECR as your local private registry, more information can be found on the Local Registry Tools[208][209] page.

To deploy a cluster with a custom image in a region where CAPI images[210] are not provided, you need to use Konvoy Image Builder[211][212] to create your own image for the region.

For an air-gapped environment, ensure you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz` , so you can extract the tarball on next page.

5.10.2.2 Next Step:

AWS Air-gapped: Create an Image (see page 276)

208 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/210829371/Registry+Mirror+Tools
209 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/210829371/Registry+Mirror+Tools
210 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html
211 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/90439791
212 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/90439791

5.10.2.3 AWS Air-gapped: Create an Image

This procedure describes how to use the Konvoy Image Builder (see page 1282) (KIB) to create a Cluster API[213] compliant Amazon Machine Image (AMI). AMI images contain configuration information and software to create a specific, pre-configured, operating environment. KIB uses variable overrides (see page 1330) to specify base image and container images to use in your new AMI.

5.10.2.3.1 Create an Air-gapped AMI

5.10.2.3.1.1 Create an Image using Konvoy Image Builder (KIB) for use in an air-gapped cluster

5.10.2.3.2 Prerequisites

- Before you begin, you must:
  - Check the DKP Supported Kubernetes Versions (see page 1543).
  - Check the Supported Infrastructure Operating Systems (see page 62)
  - Download the KIB (see page 0) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
  - Create a working Docker or other registry setup.
  - Ensure you have met the minimal set of permissions from the AWS Image Builder Book[214].
  - A Minimal IAM Permissions for KIB (see page 1285) to create an Image for an AWS account using Konvoy Image Builder.



Using KIB (see page 1282), you can build an AMI without requiring access to the internet by providing an additional `--override` flag.

1. Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz` , extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz && cd dkp-v2.5.2/kib
   ```

213 https://cluster-api.sigs.k8s.io/
214 https://image-builder.sigs.k8s.io/capi/providers/aws.html#required-permissions-to-build-the-aws-amis

2. Follow the instructions below to build an AMI.

Depending on which version of DKP (see page 1282) you are running, steps and flags will be different. To deploy in a region where CAPI images are not provided, you need to use KIB to create your own image for the region. For a list of supported AWS regions, refer to the Published AMI[215]information from AWS.

5.10.2.3.2.1 Execute the following to begin image creation:

Run the `konvoy-image` command to build and validate the image.

```
konvoy-image build aws images/ami/centos-79.yaml --overrides overrides/offline.yaml
```

By default it builds in the `us-west-2` region. to specify another region set the `--region` flag:

```
konvoy-image build aws --region us-east-1 images/ami/centos-79.yaml --overrides
overrides/offline.yaml
```

When the command is complete the `ami` id is printed and written to `./manifest.json`.

5.10.2.3.3 Next Step:

AWS Air-gapped: Docker Registry (see page 277)

5.10.2.4 AWS Air-gapped: Load the Registry

After you create an image for your air-gapped environment, you will need to load it into your registry.

5.10.2.4.1 Load Images into your Registry

Before creating a Kubernetes cluster, you need the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1162) and either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1349) page for more information.

1. Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz`, extract the tarball to a local directory:

215 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html

```
tar -xzvf dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz && cd dkp-v2.5.0
```

2. Set an environment variable with your registry address with this command:

```
export REGISTRY_ADDRESS=<registry-address>:<registry-port>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

3. Execute the following command to load the air-gapped image bundle into your private registry:

```
dkp push image-bundle --image-bundle ./container-images/konvoy-image-bundle-
v2.5.0.tar --to-registry $REGISTRY_ADDRESS --to-registry-username
$REGISTRY_USERNAME --to-registry-password $REGISTRY_PASSWORD
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

5.10.2.4.2 Kommander Load Images

If you are operating in an air-gapped environment, a local container registry (see page 1349) containing all the necessary installation images, including the Kommander images is required. See below for how to push the necessary images to this registry.

For the basic air-gapped `kommander` image bundle, run the command below:

1. Run the following command to load the image bundle:

```
dkp push image-bundle --image-bundle ./container-images/kommander-image-bundle-
v2.5.0.tar --to-registry $REGISTRY_ADDRESS
```

For **DKP Catalog Applications**, also perform this image load:

1. Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push image-bundle --image-bundle ./container-images/dkp-catalog-
applications-image-bundle-v2.5.0.tar --to-registry $REGISTRY_ADDRESS
```

5.10.2.4.3 Next Step:

AWS Air-gapped: Create the Management Cluster (see page 279)

5.10.2.5 AWS Air-gapped: Create the Management Cluster

Create a new self-managed AWS Kubernetes cluster in an Air-gapped environment on your AWS infrastructure. When you use existing infrastructure, DKP does *not* create, modify, or delete the following AWS resources:

- Internet Gateways
- NAT Gateways
- Routing tables
- Subnets
- VPC
- VPC Endpoints (for subnets without NAT Gateways)

An AWS subnet has Network ACLs that can control traffic in and out of the subnet. DKP does not modify the Network ACLs of an existing subnet. DKP uses Security Groups to control traffic. If a Network ACL denies traffic that is allowed by DKP-managed Security Groups, the cluster may not work correctly.

To increase Docker Hub's rate limit[216] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster` command.

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable to the name you assigned this cluster:

   ```
   export CLUSTER_NAME=<aws-example>
   ```

216 https://docs.docker.com/docker-hub/download-rate-limit/

NOTE: The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[217] for more naming information.

3. Export variables for the existing infrastructure details:

```
export AWS_VPC_ID=<vpc-...>
export AWS_SUBNET_IDS=<subnet-...,subnet-...,subnet-...>
export AWS_ADDITIONAL_SECURITY_GROUPS=<sg-...>
export AWS_AMI_ID=<ami-...>
```

   - `AWS_VPC_ID`: the VPC ID where the cluster will be created. The VPC requires the `ec2`, `elasticloadbalancing`, `secretsmanager` and `autoscaling` VPC endpoints to be already present.
   - `AWS_SUBNET_IDS`: a comma-separated list of one or more private Subnet IDs with each one in a different Availability Zone. The cluster control-plane and worker nodes will automatically be spread across these Subnets.
   - `AWS_ADDITIONAL_SECURITY_GROUPS`: a comma-seperated list of one or more Security Groups IDs to use in addition to the ones automatically created by CAPA[218].
   - `AWS_AMI_ID`: the AMI ID to use for control-plane and worker nodes. The AMI must be created by the konvoy-image-builder (see page 1282).

   ⚠ **IMPORTANT:** You must tag the subnets as described below to allow for Kubernetes to create ELBs for services of type `LoadBalancer` in those subnets. If the subnets are not tagged, they will not receive an ELB and the following error displays: `Error syncing load balancer, failed to ensure load balancer; could not find any suitable subnets for creating the ELB.`.

   The tags should be set as follows, where `<CLUSTER_NAME>` corresponds to the name set in `CLUSTER_NAME` environment variable:

```
kubernetes.io/cluster = <CLUSTER_NAME>
kubernetes.io/cluster/<CLUSTER_NAME> = owned
kubernetes.io/role/internal-elb = 1
```

4. Configure your cluster to use an existing registry as a mirror when attempting to pull images:
   ⚠ If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1349) page for more information.

   ⚠ **IMPORTANT:** The AMI must be created by the konvoy-image-builder (see page 1282) project in order to use the registry mirror feature.

217 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/
218 https://github.com/kubernetes-sigs/cluster-api-provider-aws

```
export REGISTRY_URL=<https/http>://<registry-address>:<registry-port>
export REGISTRY_CA=<path to the CA on the bastion>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

- `REGISTRY_URL` : the address of an existing registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
- `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
- `REGISTRY_PASSWORD` : optional if username is not set.

5. Create a Kubernetes cluster. The following example shows a common configuration. See dkp create cluster aws (see page 1456) reference for the full list of cluster creation options:



DKP uses AWS CSI as the default storage provider (see page 103). You can use a Kubernetes CSI[219] compatible storage solution that is suitable for production. See the Kubernetes documentation called Changing the Default Storage Class[220] for more information.

- ```
  dkp create cluster aws --cluster-name=${CLUSTER_NAME} \
  --additional-tags=owner=$(whoami) \
  --with-aws-bootstrap-credentials=true \
  --vpc-id=${AWS_VPC_ID} \
  --ami=${AWS_AMI_ID} \
  --subnet-ids=${AWS_SUBNET_IDS} \
  --internal-load-balancer=true \
  --additional-security-group-ids=${AWS_ADDITIONAL_SECURITY_GROUPS} \
  --registry-mirror-url=${REGISTRY_URL} \
  --registry-mirror-cacert=${REGISTRY_CA} \
  --registry-mirror-username=${REGISTRY_USERNAME} \
  --registry-mirror-password=${REGISTRY_PASSWORD} \
  --self-managed
  ```

  If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy` , `--https-proxy` , and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

219 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
220 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/
  ```

- A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing. As part of the underlying processing using the `--self-managed` flag, the DKP CLI:
    - creates a bootstrap cluster
    - creates a workload cluster
    - moves CAPI controllers from the bootstrap cluster to the workload cluster, making it self-managed
    - deletes the bootstrap cluster

  To understand how this process works step by step, you can find a customizable Create a new Air-gapped AWS Cluster (see page 953) under Additional Infrastructure Configuration.

5.10.2.5.1 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1360).

5.10.2.5.2 Next Step:

AWS Air-gapped: Install Kommander (see page 282)

5.10.2.6 AWS Air-gapped: Install Kommander

You have installed the Konvoy component and created a cluster. Now it is time to Install Kommander which will allow you to access the UI and attach new or existing clusters to monitor.

5.10.2.6.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 119).
- Ensure you have a default StorageClass (see page 1254).
- Ensure you have loaded all necessary images for your configuration. See load the images into your registry (see page 1258) .
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

5.10.2.6.2 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

```
dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
```

3. Create a configuration file (see page 1227) for the deployment:

```
dkp install kommander --init --airgapped > kommander.yaml
```

4. *If required:* Customize your `kommander.yaml`.
   See Kommander Additional Install Configurations (see page 1227) for customization options. Some of them include:
   Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1271)), etc.
5. *If required*: If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

```
apps:
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true"
```

5.10.2.6.3 Enable DKP Catalog Applications and Install Kommander

If you have an Enterprise license and would like to use the DKP Catalog Applications function, follow these steps:

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
```

```
          tag: v2.5.0
```

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.5.0.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.5.0.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.5.0.tar.gz
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see (Provide Context for Commands with a kubeconfig File (see page 99) .
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

5.10.2.6.4 Next Step:

AWS Air-gapped: Verify Install and Log in the UI (see page 285)

5.10.2.7 AWS Air-gapped: Verify Install and Log in the UI

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander. After the CLI successfully installs the components, it will wait for all applications to be ready by default.

> **NOTE**: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Released helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Released` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
```

```
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

5.10.2.7.1 Failed HelmReleases

If any application fails to successfully deploy, you can check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

5.10.2.7.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 524). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 524):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

5.10.2.7.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 510) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

Now you can create additional new clusters (see page 287), or proceed to Day 2 Operations (see page 510).

5.10.2.7.4 Next Step:

AWS Air-gapped: Subsequent New Clusters (see page 287)

5.10.2.8 AWS Air-gapped: Create a Managed Cluster Using the DKP CLI

In previous steps, you created a new Management cluster, which is self-managed. When you use these steps to create new Managed clusters, they will become Attached clusters under your Management Cluster.

> ⚠ When creating Managed clusters (see page 97), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 97)!

5.10.2.8.1 Choose a Workspace for the New Cluster

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a Workspace[221].

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

221 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919315/Workspaces#Create-a-Workspace

```
export WORKSPACE_NAMESPACE=<workspace_namespace>
```

5.10.2.8.2 Name Your Cluster

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable:

   ```
   export CLUSTER_NAME=<aws-additional>
   ```

> The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[222] for more naming information.

3. Execute this command to create your Kubernetes cluster using any relevant flags but note that the `--self-managed` is absent so that this cluster can be attached to your previously created Management Cluster.

```
dkp create cluster aws --cluster-name=${CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--with-aws-bootstrap-credentials=true \
--vpc-id=${AWS_VPC_ID} \
--ami=${AWS_AMI_ID} \
--subnet-ids=${AWS_SUBNET_IDS} \
--internal-load-balancer=true \
--additional-security-group-ids=${AWS_ADDITIONAL_SECURITY_GROUPS} \
--registry-mirror-url=${REGISTRY_URL} \
--registry-mirror-cacert=${REGISTRY_CA} \
--registry-mirror-username=${REGISTRY_USERNAME} \
--registry-mirror-password=${REGISTRY_PASSWORD} \
--namespace=${WORKSPACE_NAMESPACE}
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

222 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

5.10.2.8.3 Manually Attach a DKP CLI Cluster to the Management Cluster

1. Find out the `name` of the created `Cluster`, so you can reference it later:

```
kubectl -n <workspace_namespace> get clusters
```

2. Attach the cluster by creating a `KommanderCluster`:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: <cluster_name>
  namespace: <workspace_namespace>
spec:
  kubeconfigRef:
    name: <cluster_name>-kubeconfig
  clusterRef:
    capiCluster:
      name: <cluster_name>
EOF
```

After you have existing clusters or want to create new clusters to attach, there are many ways to attach a cluster with various requirements and restrictions. To see all the options, visit the section in documentation Day 2 - Attach an Existing Kubernetes Cluster (see page 673).

5.10.2.8.4 Next Step:

Day 2 - Cluster Operations Management (see page 510)

5.10.3 AWS with FIPS Install

This section provides instructions to install DKP in an AWS non-air-gapped FIPS environment.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

5.10.3.1 AWS Prerequisites

Before you begin using Konvoy with AWS, you must:

1. Follow the steps to create a Minimal Permissions and Role to Create Clusters (see page 940)..
2. Create Cluster IAM Policies and Roles (see page 946).
3. Export the AWS region where you want to deploy the cluster:

```
export AWS_REGION=us-west-2
```

4. Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

```
export AWS_PROFILE=<profile>
```

If using AWS ECR as your local private registry, more information can be found on the Local Registry Tools[223][224] page.

To deploy a cluster with a custom image in a region where CAPI images[225] are not provided, you need to use Konvoy Image Builder[226][227] to create your own image for the region.

5.10.3.2 Next Step:

AWS FIPS Create an Image (see page 291)

223 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/210829371/Registry+Mirror+Tools
224 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/210829371/Registry+Mirror+Tools
225 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html
226 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/90439791
227 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/90439791

5.10.3.3 AWS FIPS: Create an Image

5.10.3.3.1 Learn how to build a custom AMI for use with DKP

This procedure describes how to use the Konvoy Image Builder (see page 1282) (KIB) to create a Cluster API[228] compliant Amazon Machine Image (AMI). KIB uses variable overrides (see page 1330) to specify base image and container images to use in your new AMI.

The default AWS image is not recommended for use in production. We suggest using Konvoy Image Builder to create a custom AMI to take advantage of enhanced cluster operations. Explore the KIB with AWS (see page 1285) topics for more options.

5.10.3.3.2 Prerequisites

Before you begin, you must:

- Check the DKP Supported Kubernetes Versions (see page 1543).
- Check the Supported Infrastructure Operating Systems (see page 62)
- Download the KIB (see page 0) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
- Create a working `Docker` setup.
- Ensure you have met the minimal set of permissions from the AWS Image Builder Book[229].
- A Minimal IAM Permissions for KIB (see page 1285) to create an Image for an AWS account using Konvoy Image Builder.

5.10.3.3.3 Extract KIB Bundle

Extract the bundle and `cd` into the extracted `konvoy-image-bundle-$VERSION` folder. The bundled version of `konvoy-image` contains an embedded `docker` image that contains all the requirements for building.

The `konvoy-image` binary and all supporting folders are also extracted. When run, `konvoy-image` bind mounts the current working directory (`${PWD}`) into the container to be used.

228 https://cluster-api.sigs.k8s.io/
229 https://image-builder.sigs.k8s.io/capi/providers/aws.html#required-permissions-to-build-the-aws-amis

- Set environment variables for AWS access[230]. The following variables must be set using your credentials including required IAM (see page 1285):

```
export AWS_ACCESS_KEY_ID
export AWS_SECRET_ACCESS_KEY
export AWS_DEFAULT_REGION
```

- If you have an override file (see page 1330) to configure specific attributes of your AMI file, add it.

5.10.3.3.4 Non-air-gapped Environment Create FIPS-140 images

KIB can produce images containing FIPS-140 compliant binaries. Use the `fips.yaml` override file (see page 1331) provided with the image bundles.

You can also find these override files in the Konvoy Image Builder repo[231].

- A non-air-gapped example of override file use is the command below, which produces a FIPS-compliant image on RHEL 8.4 for AWS:
  Replace `ami` with your infrastructure provisioner

```
konvoy-image build --overrides overrides/fips.yaml images/ami/rhel-84.yaml
```

5.10.3.3.5 Next Step:

AWS FIPS Cluster Creation (see page 292)

5.10.3.4 AWS FIPS: Create the Management Cluster

Additional considerations exist when using FIPS. Alter the Create a Kubernetes cluster command by using the FIPS example, or similar, when executing the command to create a cluster.

5.10.3.4.1 Deploying a Cluster in FIPS mode

In order to create a cluster in FIPS mode, we must inform the bootstrap controllers of the appropriate image repository and version tags of the official D2iQ FIPS builds of Kubernetes.

230 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-envvars.html
231 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

5.10.3.4.1.1 Supported FIPS Builds

Component	Repository	Version
Kubernetes	docker.io/mesosphere[232]	v1.25.4+fips.0
etcd	docker.io/mesosphere[233]	3.5.5+fips.0

5.10.3.4.1.2 Name Your Cluster

> The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[234] for more naming information.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable:

```
export CLUSTER_NAME=<aws-example>
```

5.10.3.4.1.3 Create a New AWS Kubernetes Cluster

If you use these instructions to create a cluster on AWS using the DKP default settings without any edits to configuration files or additional flags, your cluster is deployed on an Ubuntu 20.04 operating system image (see page 62) with 3 control plane nodes, and 4 worker nodes.

232 https://hub.docker.com/layers/mesosphere/kube-apiserver/v1.25.4_fips.0/images/sha256-cacb721f96ec8764d187ad3c21557630f5f4d96fea4a03ca35d0388b509d970d?context=explore
233 https://hub.docker.com/layers/mesosphere/etcd/v3.5.5_fips.0/images/sha256-ba07521fa1875efa0cc623533eb5557e40a6f8fdc8a71a86751a649ce53de1d0?context=explore
234 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

The default AWS image is not recommended for use in production. D2iQ suggests using Konvoy Image Builder (see page 1285) to create a custom AMI and take advantage of enhanced cluster operations.

Create a Kubernetes cluster with the command(s) below:

DKP uses AWS CSI as the default storage provider (see page 103). You can use a Kubernetes CSI[235] compatible storage solution that is suitable for production. See the Kubernetes documentation called Changing the Default Storage Class[236] for more information.

```
dkp create cluster aws \
--cluster-name=${CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--with-aws-bootstrap-credentials=true \
--kubernetes-version=v1.25.4+fips.0 \
--etcd-version=3.5.5+fips.0 \
--kubernetes-image-repository=docker.io/mesosphere \
--self-managed
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

- A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing. As part of the underlying processing using the `--self-managed` flag, the DKP CLI:
    - creates a bootstrap cluster
    - creates a workload cluster
    - moves CAPI controllers from the bootstrap cluster to the workload cluster, making it self-managed
    - deletes the bootstrap cluster

  To understand how this process works step by step, you can find a customizable Create a new AWS Cluster (see page 953) under the Additional Infrastructure Configuration section.

235 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
236 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

5.10.3.4.1.4 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1360).

5.10.3.4.1.5 Next Step:

AWS FIPS: Install Kommander (see page 295)

5.10.3.5 AWS FIPS: Install Kommander

You have installed the Konvoy component and created a cluster. Now it is time to Install Kommander which will allow you to access the UI and attach new or existing clusters to monitor.

5.10.3.5.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 119).
- Ensure you have a default StorageClass (see page 1254).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

5.10.3.5.2 Create and customize your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1227) for the deployment:

   ```
   dkp install kommander --init > kommander.yaml
   ```

4. *If required:* Customize your `kommander.yaml`.
   See Kommander Additional Install Configurations (see page 1227) for customization options. Some of them include:
   Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1271)), etc.

5. *If required:* If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

```
apps:
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true"
```

5.10.3.5.3 Enable DKP Catalog Applications and Install Kommander

If you have an Enterprise license and would like to use the DKP Catalog Applications function, follow these steps:

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.5.0
```

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.co
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More

information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see (Provide Context for Commands with a kubeconfig File (see page 99) .
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

5.10.3.5.4 Next Step:

AWS FIPS: Verify Install and Log in the UI (see page 297)

5.10.3.6 AWS FIPS: Verify Install and Log in the UI

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander. After the CLI successfully installs the components, it will wait for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Released helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Released` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

5.10.3.6.1 Failed HelmReleases

If any application fails to successfully deploy, you can check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

5.10.3.6.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 524). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 524):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

5.10.3.6.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 510) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

Now you can create additional new clusters (see page 299), or proceed to Day 2 Operations in the link below.

5.10.3.6.4 Next Step:

AWS FIPS: Subsequent New Clusters (see page 299)

5.10.3.7 AWS FIPS: Create a Managed Cluster Using the DKP CLI

In previous steps, you created a new Management cluster, which is self-managed. When you use these steps to create new Managed clusters, they will become Attached clusters under your Management Cluster.

When creating Managed clusters (see page 97), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 97)!

5.10.3.7.1 Choose a Workspace for the New Managed Cluster

Your new managed cluster needs to be part of either a new Workspace or an existing Workspace.

To create a new Workspace, follow the instructions to Create a Workspace[237].

To find the existing Workspaces, use this command:

```
kubectl get workspace -A
```

When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

```
export WORKSPACE_NAMESPACE=<workspace_namespace>
```

5.10.3.7.2 Name Your Cluster

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS, it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable:

   ```
   export CLUSTER_NAME=<aws-additional>
   ```

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[238] for more naming information.

237 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919315/Workspaces#Create-a-Workspace
238 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

3. Execute this command to create your Kubernetes cluster using any relevant flags but note that the `--self-managed` is absent so that this cluster can be attached to your previously created Management Cluster.

Execute this command:

```
dkp create cluster aws \
--cluster-name=${CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--with-aws-bootstrap-credentials=true \
--kubernetes-version=v1.25.4+fips.0 \
--kubernetes-image-repository=docker.io/mesosphere \
--namespace=${WORKSPACE_NAMESPACE}
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

5.10.3.7.3 Manually Attach a DKP CLI Cluster to the Management Cluster

1. Find out the `name` of the created `Cluster`, so you can reference it later:

   ```
   kubectl -n <workspace_namespace> get clusters
   ```

2. Attach the cluster by creating a `KommanderCluster`:

   ```
   cat << EOF | kubectl apply -f -
   apiVersion: kommander.mesosphere.io/v1beta1
   kind: KommanderCluster
   metadata:
     name: <cluster_name>
     namespace: <workspace_namespace>
   spec:
     kubeconfigRef:
       name: <cluster_name>-kubeconfig
     clusterRef:
       capiCluster:
         name: <cluster_name>
   EOF
   ```

Once you have existing clusters or want to create new clusters to attach, there are many ways to attach a cluster with various requirements and restrictions. To see all the options, visit the section in documentation Day 2 - Attach an Existing Kubernetes Cluster (see page 673).

5.10.3.7.4 Next Step:

Day 2 - Cluster Operations Management (see page 510)

5.10.4 AWS Air-gapped FIPS Install

This section provides instructions to install DKP in an AWS air-gapped FIPS environment.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

5.10.4.1 AWS Prerequisites

Before you begin using Konvoy with AWS, you must:

1. Follow the steps to create a Minimal Permissions and Role to Create Clusters (see page 940)..
2. Create Cluster IAM Policies and Roles (see page 946).
3. Export the AWS region where you want to deploy the cluster:

   ```
   export AWS_REGION=us-west-2
   ```

4. Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

   ```
   export AWS_PROFILE=<profile>
   ```

For an air-gapped environment, ensure you have Download DKP (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz` , so you can extract the tarball on next page.

5.10.4.2 Next Step:

AWS Air-gapped FIPS: Create an Image (see page 303)

5.10.4.3 AWS Air-gapped FIPS: Create an Image

5.10.4.3.1 Create an Image using Konvoy Image Builder (KIB) for use in an air-gapped cluster

5.10.4.3.2 Prerequisites

- Before you begin, you must:
  - Check the DKP Supported Kubernetes Versions (see page 1543).
  - Check the Supported Infrastructure Operating Systems (see page 62)
  - Download the KIB (see page 0) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
  - Create a working Docker or other registry setup.
  - Ensure you have met the minimal set of permissions from the AWS Image Builder Book[239].
  - A Minimal IAM Permissions for KIB (see page 1285) to create an Image for an AWS account using Konvoy Image Builder.

The default AWS image is not recommended for use in production. We suggest using Konvoy Image Builder to create an AWS Air-gapped AMI to take advantage of enhanced cluster operations. Explore the KIB with AWS (see page 1285) topics for more options.

Using KIB (see page 1282), you can build an AMI without requiring access to the internet by providing an additional `--override` flag.

1. Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz` , extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz && cd dkp-v2.5.2/kib
   ```

2. Follow the instructions below to build an AMI.

(2.5) Create FIPS 140 Images: Air-gapped Environment

239 https://image-builder.sigs.k8s.io/capi/providers/aws.html#required-permissions-to-build-the-aws-amis

KIB can produce images containing FIPS-140 compliant binaries. Use the `fips.yaml` override file[240][241] provided with the image bundles.

> You can also find these override files in the Konvoy Image Builder repo[242].

- An air-gapped example of override file use is the command below which produces an AWS FIPS-compliant image on RHEL 8.4:

```
konvoy-image build --overrides overrides/offline-fips.yaml images/ami/rhel-84.yaml
```

5.10.4.3.3 Next Step:

AWS Air-gapped FIPS: Load the Registry (see page 304)

5.10.4.4 AWS Air-gapped FIPS: Load the Registry

After you create an image for your air-gapped environment, you will need to load it into your registry.

5.10.4.4.1 Load Images into your Registry

Before creating a Kubernetes cluster, you need the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1162) and either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

> If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1349) page for more information.

1. Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz` , extract the tarball to a local directory:

240 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/46039044
241 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/46039044
242 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

```
tar -xzvf dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz && cd dkp-v2.5.0
```

2. Set an environment variable with your registry address with this command:

```
export REGISTRY_ADDRESS=<registry-address>:<registry-port>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

3. Execute the following command to load the air-gapped image bundle into your private registry:

```
dkp push image-bundle --image-bundle ./container-images/konvoy-image-bundle-
v2.5.0.tar --to-registry $REGISTRY_ADDRESS --to-registry-username
$REGISTRY_USERNAME --to-registry-password $REGISTRY_PASSWORD
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

5.10.4.4.2 Kommander Load Images

If you are operating in an air-gapped environment, a local container registry (see page 1349) containing all the necessary installation images, including the Kommander images is required. See below for how to push the necessary images to this registry.

For the basic air-gapped `kommander` image bundle, run the command below:

1. Run the following command to load the image bundle:

```
dkp push image-bundle --image-bundle ./container-images/kommander-image-bundle-
v2.5.0.tar --to-registry $REGISTRY_ADDRESS
```

For **DKP Catalog Applications**, also perform this image load:

1. Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push image-bundle --image-bundle ./container-images/dkp-catalog-
applications-image-bundle-v2.5.0.tar --to-registry $REGISTRY_ADDRESS
```

5.10.4.4.3 Next Step:

AWS Air-gapped FIPS: Create the Management Cluster (see page 306)

5.10.4.5 AWS Air-gapped FIPS: Create the Management Cluster

Create a new self-managed AWS Kubernetes cluster in an Air-gapped FIPS environment on your AWS infrastructure.

To increase Docker Hub's rate limit[243] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster` command.

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable to the name you assigned this cluster:

   ```
   export CLUSTER_NAME=<aws-example>
   ```

   NOTE: The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[244] for more naming information.

3. Export variables for the existing infrastructure details:

   ```
   export AWS_VPC_ID=<vpc-...>
   export AWS_SUBNET_IDS=<subnet-...,subnet-...,subnet-...>
   export AWS_ADDITIONAL_SECURITY_GROUPS=<sg-...>
   export AWS_AMI_ID=<ami-...>
   ```

   - `AWS_VPC_ID`: the VPC ID where the cluster will be created. The VPC requires the `ec2`, `elasticloadbalancing`, `secretsmanager` and `autoscaling` VPC endpoints to be already present.

243 https://docs.docker.com/docker-hub/download-rate-limit/
244 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

- `AWS_SUBNET_IDS` : a comma-separated list of one or more private Subnet IDs with each one in a different Availability Zone. The cluster control-plane and worker nodes will automatically be spread across these Subnets.
- `AWS_ADDITIONAL_SECURITY_GROUPS` : a comma-seperated list of one or more Security Groups IDs to use in addition to the ones automatically created by CAPA[245].
- `AWS_AMI_ID` : the AMI ID to use for control-plane and worker nodes. The AMI must be created by the konvoy-image-builder (see page 1282).

⚠ **IMPORTANT:** You must tag the subnets as described below to allow for Kubernetes to create ELBs for services of type `LoadBalancer` in those subnets. If the subnets are not tagged, they will not receive an ELB and the following error displays: `Error syncing load balancer, failed to ensure load balancer; could not find any suitable subnets for creating the ELB.` .

The tags should be set as follows, where `<CLUSTER_NAME>` corresponds to the name set in `CLUSTER_NAME` environment variable:

```
kubernetes.io/cluster = <CLUSTER_NAME>
kubernetes.io/cluster/<CLUSTER_NAME> = owned
kubernetes.io/role/internal-elb = 1
```

4. Configure your cluster to use an existing registry as a mirror when attempting to pull images:
⚠ If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1349) page for more information.

⚠ **IMPORTANT:** The AMI must be created by the konvoy-image-builder (see page 1282) project in order to use the registry mirror feature.

```
export REGISTRY_URL=<https/http>://<registry-address>:<registry-port>
export REGISTRY_CA=<path to the CA on the bastion>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

- `REGISTRY_URL` : the address of an existing registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
- `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
- `REGISTRY_PASSWORD` : optional if username is not set.

245 https://github.com/kubernetes-sigs/cluster-api-provider-aws

5. Create a Kubernetes cluster. The following example shows a common configuration. See dkp create cluster aws (see page 1456) reference for the full list of cluster creation options:

DKP uses AWS CSI as the default storage provider (see page 103). You can use a Kubernetes CSI[246] compatible storage solution that is suitable for production. See the Kubernetes documentation called Changing the Default Storage Class[247] for more information.

- ```
  dkp create cluster aws --cluster-name=${CLUSTER_NAME} \
  --additional-tags=owner=$(whoami) \
  --with-aws-bootstrap-credentials=true \
  --vpc-id=${AWS_VPC_ID} \
  --ami=${AWS_AMI_ID} \
  --subnet-ids=${AWS_SUBNET_IDS} \
  --internal-load-balancer=true \
  --additional-security-group-ids=${AWS_ADDITIONAL_SECURITY_GROUPS} \
  --registry-mirror-url=${REGISTRY_URL} \
  --registry-mirror-cacert=${REGISTRY_CA} \
  --registry-mirror-username=${REGISTRY_USERNAME} \
  --registry-mirror-password=${REGISTRY_PASSWORD} \
  --kubernetes-version=v1.25.4+fips.0 \
  --etcd-version=3.5.5+fips.0 \
  --kubernetes-image-repository=docker.io/mesosphere \
  --self-managed
  ```

  If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).
- A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing. As part of the underlying processing, the DKP CLI:
  - creates a bootstrap cluster
  - creates a workload cluster
  - moves CAPI controllers from the bootstrap cluster to the workload cluster, making it self-managed
  - deletes the bootstrap cluster

  To understand how this process works step by step, you can find a customizable Create a New AWS Cluster (see page 953) under Additional Infrastructure Configuration.

246 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
247 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/
  ```

5.10.4.5.1 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1360).

5.10.4.5.2 Next Step:

AWS Air-gapped FIPS: Install Kommander (see page 309)

5.10.4.6 AWS Air-gapped FIPS: Install Kommander

You have installed the Konvoy component and created a cluster. Now it is time to Install Kommander which will allow you to access the UI and attach new or existing clusters to monitor.

5.10.4.6.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 119).
- Ensure you have a default StorageClass (see page 1254).
- Ensure you have loaded all necessary images for your configuration. See load the images into your registry (see page 1258) .
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

5.10.4.6.2 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1227) for the deployment:

   ```
   dkp install kommander --init --airgapped > kommander.yaml
   ```

4. *If required:* Customize your `kommander.yaml` .
   See Kommander Additional Install Configurations (see page 1227) for customization options. Some of them include:

Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1271)), etc.

5. *If required*: If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

```
apps:
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true"
```

5.10.4.6.3 Enable DKP Catalog Applications and Install Kommander

If you have an Enterprise license and would like to use the DKP Catalog Applications function, follow these steps:

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.5.0
```

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.5.0.tar.gz \
```

```
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.5.0.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.5.0.tar.gz
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see (Provide Context for Commands with a kubeconfig File (see page 99) .
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

5.10.4.6.4 Next Step:

AWS Air-gapped FIPS: Verify Install and Log in the UI (see page 311)

5.10.4.7 AWS Air-gapped FIPS: Verify Install and Log in the UI

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander. After the CLI successfully installs the components, it will wait for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Released helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Released` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

5.10.4.7.1 Failed HelmReleases

If any application fails to successfully deploy, you can check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
```

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

5.10.4.7.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 524). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 524):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

5.10.4.7.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 510) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

Now you can create additional new clusters (see page 314), or proceed to Day 2 Operations in the link below.

5.10.4.7.4 Next Step:

AWS Air-gapped FIPS: Subsequent New Clusters (see page 314)

5.10.4.8 AWS Air-gapped FIPS: Create a Managed Cluster Using the DKP CLI

In previous steps, you created a new Management cluster, which is self-managed. When you use these steps to create new Managed clusters, they will become Attached clusters under your Management Cluster.

When creating Managed clusters (see page 97), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 97)!

5.10.4.8.1 Choose a Workspace for the New Cluster

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a Workspace[248].

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

5.10.4.8.2 Name Your Cluster

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable:

   ```
   export CLUSTER_NAME=<aws-additional>
   ```

248 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919315/Workspaces#Create-a-Workspace

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[249] for more naming information.

3. Execute this command to create your Kubernetes cluster using any relevant flags but note that the `--self-managed` is absent so that this cluster can be attached to your previously created Management Cluster:

```
dkp create cluster aws --cluster-name=${CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--with-aws-bootstrap-credentials=true \
--vpc-id=${AWS_VPC_ID} \
--ami=${AWS_AMI_ID} \
--subnet-ids=${AWS_SUBNET_IDS} \
--internal-load-balancer=true \
--additional-security-group-ids=${AWS_ADDITIONAL_SECURITY_GROUPS} \
--registry-mirror-url=${REGISTRY_URL} \
--registry-mirror-cacert=${REGISTRY_CA} \
--registry-mirror-username=${REGISTRY_USERNAME} \
--registry-mirror-password=${REGISTRY_PASSWORD} \
--namespace=${WORKSPACE_NAMESPACE}
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

5.10.4.8.3 Manually Attach a DKP CLI Cluster to the Management Cluster

1. Find out the `name` of the created `Cluster`, so you can reference it later:

   ```
   kubectl -n <workspace_namespace> get clusters
   ```

2. Attach the cluster by creating a `KommanderCluster`:

   ```
   cat << EOF | kubectl apply -f -
   apiVersion: kommander.mesosphere.io/v1beta1
   kind: KommanderCluster
   metadata:
     name: <cluster_name>
     namespace: <workspace_namespace>
   ```

249 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

```
spec:
  kubeconfigRef:
    name: <cluster_name>-kubeconfig
  clusterRef:
    capiCluster:
      name: <cluster_name>
EOF
```

When you have existing clusters or want to create other new clusters to attach, there are many ways to attach a cluster with various requirements and restrictions. To see all the options, visit the section in the Day 2- Cluster Operations Management section for documentation on how to Attach an Existing Kubernetes Cluster (see page 673).

5.10.4.8.4 Next Step:

Day 2 - Cluster Operations Management (see page 510)

5.10.5 AWS with GPU Install

This section provides instructions to install DKP in an AWS non-air-gapped GPU environment.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

5.10.5.1 AWS Prerequisites

Before you begin using Konvoy with AWS, you must:

1. Follow the steps to create a Minimal Permissions and Role to Create Clusters (see page 940)..
2. Create Cluster IAM Policies and Roles (see page 946).
3. Export the AWS region where you want to deploy the cluster:

   ```
   export AWS_REGION=us-west-2
   ```

4. Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

   ```
   export AWS_PROFILE=<profile>
   ```

5.10.5.2 GPU Prerequisites

Before you begin, you must:

- Ensure nodes provide an NVIDIA GPU.
- If you are using a public cloud service such as AWS (see page 988), create an AMI with KIB using the instructions on the KIB for GPU (see page 1302) page.
- If you are deploying in a pre-provisioned environment, ensure that you have created the appropriate secret for your GPU nodepool (see page 210) and have uploaded the appropriate artifacts to each node. See the GPU only steps section (see page 0)on the Pre-provisioned Prerequisites Air-gapped page for additional information.

Specific instructions must be followed for enabling `nvidia-gpu-operator` depending on if you want to deploy the app on a Management cluster or a Attached or a Managed cluster.

- For instructions on enabling the NVIDIA platform application on a Management cluster, follow the instructions in the **NVIDIA Platform Application Management Cluster** (see page 861) section.
- For instructions on enabling the NVIDIA platform application on attached or managed clusters, follow the instructions in the **NVIDIA Platform Application Attached or Managed Cluster** (see page 864) section.

Once `nvidia-gpu-operator` has been enabled depending on the cluster type, proceed to the **Select the correct Toolkit version for your NVIDIA GPU Operator** on each of those pages.

5.10.5.3 Next Step:

AWS GPU: Node Label Automatic Configuration (see page 317)

5.10.5.4 AWS GPU: Use Node Label Automatic Configuration

When using GPU nodes, it is important they have the proper label identifying them as Nvidia GPU nodes. Node feature discovery (NFD), by default labels PCI hardware as:

```
"feature.node.kubernetes.io/pci-<device label>.present": "true"
```

where `<device label>` is by default as defined in this topic[250]:

```
< class > _ < vendor >
```

However, because there is a wide variety in devices and their assigned PCI classes, you may find that the labels assigned to your GPU nodes do not always properly identify them as containing an Nvidia GPU.

If the default detection does not work, you can manually change the daemonset that the GPU operator creates by running the following command:

250 https://kubernetes-sigs.github.io/node-feature-discovery/v0.7/get-started/features.html#pci

```
        nodeSelector:
          feature.node.kubernetes.io/pci-< class > _ < vendor>.present: "true"
```

where `class` is any 4 digit number starting with `03xy` and the vendor for Nvidia is `10de`. If this is already deployed, you can always change the `daemonset` and change the `nodeSelector` field so that it deploys to the right nodes.

5.10.5.4.1 Next Step:

AWS GPU: Create an Image (see page 318)

5.10.5.5 AWS GPU: Create an Image

5.10.5.5.1 Learn how to build a custom AMI for use with DKP

This procedure describes how to use the Konvoy Image Builder (see page 1282) (KIB) to create a Cluster API[251] compliant Amazon Machine Image (AMI). KIB uses variable overrides (see page 1330) to specify base image and container images to use in your new AMI.

> ⚠ The default AWS image is not recommended for use in production. We suggest using Konvoy Image Builder to create a custom AMI to take advantage of enhanced cluster operations. Explore the KIB with AWS (see page 1285) topics for more options.

5.10.5.5.2 Prerequisites

Before you begin, you must:

- Check the DKP Supported Kubernetes Versions (see page 1543).
- Check the Supported Infrastructure Operating Systems (see page 62)
- Download the KIB (see page 0) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
- Create a working `Docker` setup.
- Ensure you have met the minimal set of permissions from the AWS Image Builder Book[252].
- A Minimal IAM Permissions for KIB (see page 1285) to create an Image for an AWS account using Konvoy Image Builder.

251 https://cluster-api.sigs.k8s.io/
252 https://image-builder.sigs.k8s.io/capi/providers/aws.html#required-permissions-to-build-the-aws-amis

5.10.5.5.3 Extract KIB Bundle

Extract the bundle and `cd` into the extracted `konvoy-image-bundle-$VERSION` folder. The bundled version of `konvoy-image` contains an embedded `docker` image that contains all the requirements for building.

> The `konvoy-image` binary and all supporting folders are also extracted. When run, `konvoy-image` bind mounts the current working directory (`${PWD}`) into the container to be used.

- Set environment variables for AWS access[253]. The following variables must be set using your credentials including required IAM (see page 1285):

```
export AWS_ACCESS_KEY_ID
export AWS_SECRET_ACCESS_KEY
export AWS_DEFAULT_REGION
```

- If you have an override file (see page 1330) to configure specific attributes of your AMI file, add it.

253 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-envvars.html

5.10.5.5.4 Build the GPU Image

5.10.5.5.5 Using the Konvoy Image Builder, you can build an image that has support to use NVIDIA GPU hardware to support GPU workloads.

> **NOTE:** The NVIDIA driver requires a specific Linux kernel version. Make sure that the base image for the OS version has the required kernel version.
>
> See Supported Infrastructure Operating Systems for a list of OS versions and the corresponding kernel versions known to work with the NVIDIA driver.

If the NVIDIA runfile[254] installer has not been downloaded, then retrieve and install the download first by running the following command. The first line in the command below downloads and installs the runfile and the second line places it in the artifacts directory (you must create an `artifacts` directory if you don't already have one).

```
curl -O https://download.nvidia.com/XFree86/Linux-x86_64/470.82.01/NVIDIA-Linux-x86_64-470.82.01.run
mv NVIDIA-Linux-x86_64-470.82.01.run artifacts
```

> DKP supported NVIDIA driver[255] version is 470.x.

To build an image for use on GPU enabled hardware, perform the following steps.

1. In your `overrides/nvidia.yaml` file, add the following to enable GPU builds. You can also access and use the overrides repo[256] or in the documentation under Nvidia GPU Override File or Offline Nvidia Override file.
   a. Non-air-gapped GPU override:

```
gpu:
  types:
    - nvidia
build_name_extra: "-nvidia"
```

254 https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html#runfile
255 https://www.nvidia.com/Download/Find.aspx
256 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

b. Air-gapped GPU override:

```
# Use this file when building a machine image, not as a
override secret for preprovisioned environments
nvidia_runfile_local_file: "{{ playbook_dir}}/../artifacts/
{{ nvidia_runfile_installer }}"
gpu:
  types:
    - nvidia

build_name_extra: "-nvidia"
```

2. Build your image using the following Konvoy Image Builder commands, making sure to include the flag `--instance-type` that specifies an AWS instance that has an available GPU:
AWS Example:

```
konvoy-image build --region us-east-1 --instance-type=p2.xlarge --
source-ami=ami-12345abcdef images/ami/centos-7.yaml --overrides
overrides/nvidia.yaml
```

In this example, we chose an instance type with an NVIDIA GPU using the `--instance-type` flag, and we provided the NVIDIA overrides using the `--overrides` flag. See KIB with AWS for more information on creating an AMI.

Additional helpful information can be found in the NVIDIA Device Plug-in[257] for Kubernetes instructions and the Installation Guide of Supported Platforms[258].

See also: NVIDIA documentation[259]

5.10.5.5.6 Next Steps:

AWS GPU Cluster Creation (see page 321)

5.10.5.6 AWS GPU: Create the Management Cluster

After the GPU compatible image is created with KIB, a cluster can be generated using that custom AMI.

257 https://github.com/NVIDIA/k8s-device-plugin/blob/master/README.md
258 https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html
259 https://nvidia.custhelp.com/app/answers/detail/a_id/131/kw/driver%20installation%20docs/related/1

5.10.5.6.1 Name Your Cluster

> The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[260] for more naming information.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable:

```
export CLUSTER_NAME=<aws-example>
```

5.10.5.6.2 Create a New AWS Kubernetes Cluster

If you use these instructions to create a cluster on AWS using the DKP default settings without any edits to configuration files or additional flags, your cluster is deployed on an Ubuntu 20.04 operating system image (see page 62) with 3 control plane nodes, and 4 worker nodes.

> The default AWS image is not recommended for use in production. D2iQ suggests using Konvoy Image Builder (see page 1285) to create a custom AMI and take advantage of enhanced cluster operations.

Create a Kubernetes cluster with the command(s) below:

260 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

DKP uses AWS CSI as the default storage provider (see page 103). You can use a Kubernetes CSI[261] compatible storage solution that is suitable for production. See the Kubernetes documentation called Changing the Default Storage Class[262] for more information.

1. Execute this command to create a cluster with a GPU AMI and `--self-managed` flag. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing:

```
dkp create cluster aws \
--cluster-name=${CLUSTER_NAME} \
--with-aws-bootstrap-credentials=true \
--self-managed
```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

2. Create the node pool after cluster creation:

```
dkp create nodepool aws -c ${CLUSTER_NAME} \
--instance-type p2.xlarge \
--ami-id=${AMI_ID_FROM_KIB} \
--replicas=1 ${NODEPOOL_NAME} \
--kubeconfig=${CLUSTER_NAME}.conf
```

- As part of the underlying processing using the, the DKP CLI:
  - creates a bootstrap cluster
  - creates a workload cluster
  - moves CAPI controllers from the bootstrap cluster to the workload cluster, making it self-managed
  - deletes the bootstrap cluster

  To understand how this process works step by step, you can find a customizable Create a New AWS Cluster (see page 953) under Additional Infrastructure Configuration.

5.10.5.6.3 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1360).

261 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
262 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

5.10.5.6.4 Next Step:

AWS GPU: Install Kommander (see page 324)

5.10.5.7 AWS GPU: Install Kommander

You have installed the Konvoy component and created a cluster. Now it is time to Install Kommander which will allow you to access the UI and attach new or existing clusters to monitor.

5.10.5.7.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 119).
- Ensure you have a default StorageClass (see page 1254).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

5.10.5.7.2 Create and customize your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1227) for the deployment:

   ```
   dkp install kommander --init > kommander.yaml
   ```

4. *If required:* Customize your `kommander.yaml`.
   See Kommander Additional Install Configurations (see page 1227) for customization options. Some of them include:
   Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1271)), etc.

5. *If required:* If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

   ```
   apps:
   ```

```
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true"
```

5.10.5.7.3 Enable GPU Resources

1. In the same `kommander.yaml` file, enable Nvidia platform services.

   ```
   apps:
     nvidia-gpu-operator:
      enabled: true
   ```

2. Append the correct Toolkit version based on your OS:

   The **NVIDIA Container Toolkit** allows users to run GPU accelerated containers. The toolkit includes a container runtime library and utilities to automatically configure containers to leverage NVIDIA GPU and must be configured correctly according to your base operating system.

   Centos 7.9/RHEL 7.9:
   If you're using Centos 7.9 or RHEL 7.9 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1227) or `<kommander.yaml>` to the following:

   ```
   kind: Installation
   apps:
     nvidia-gpu-operator:
      enabled: true
      values: |
        toolkit:
          version: v1.10.0-centos7
   ```

   RHEL 8.4/8.6 and SLES 15 SP3
   If you're using RHEL 8.4/8.6 or SLES 15 SP3 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1227) or `<kommander.yaml>` to the following:

   ```
   kind: Installation
   apps:
     nvidia-gpu-operator:
      enabled: true
      values: |
   ```

```
    toolkit:
      version: v1.10.0-ubi8
```

Ubuntu 18.04 and 20.04
If you're using Ubuntu 18.04 or 20.04 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1227) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.11.0-ubuntu20.04
```

5.10.5.7.4 Enable DKP Catalog Applications and Install Kommander

If you have an Enterprise license and would like to use the DKP Catalog Applications function, follow these steps:

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.5.0
```

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.co
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see (Provide Context for Commands with a kubeconfig File (see page 99) .
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

5.10.5.7.5 Next Step:

AWS GPU: Verify Install and Log in the UI (see page 327)

5.10.5.8 AWS GPU: Verify Install and Log in the UI

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander. After the CLI successfully installs the components, it will wait for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Released helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Released` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

5.10.5.8.1 Failed HelmReleases

If any application fails to successfully deploy, you can check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

5.10.5.8.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 524). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 524):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

5.10.5.8.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 510) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

For more GPU information (see page 860), see the section for GPU under Day 2 - Cluster Operations Management.

5.10.5.8.4 Next Step:

AWS GPU: Subsequent New Clusters (see page 330)

5.10.5.9 AWS GPU: Create a Managed Cluster Using the DKP CLI

In previous steps, you created a new Management cluster, which is self-managed. When you use these steps to create new Managed clusters, they will become Attached clusters under your Management Cluster.

> ⚠ When creating Managed clusters (see page 97), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 97)!

5.10.5.9.1 Choose a Workspace for the New Cluster

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a Workspace[263].

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

5.10.5.9.2 Name Your Cluster

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable:

263 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919315/Workspaces#Create-a-Workspace

```
export CLUSTER_NAME=<aws-additional>
```

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[264] for more naming information.

3. Execute this command to create your Kubernetes cluster using any relevant flags but note that the `--self-managed` is absent so that this cluster can be attached to your previously created Management Cluster.

- Execute this command to create a cluster with a GPU AMI:

```
dkp create cluster aws \
--cluster-name=${CLUSTER_NAME} \
--with-aws-bootstrap-credentials=true \
--namespace=${WORKSPACE_NAMESPACE}
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

- Create the node pool after cluster creation:

```
dkp create nodepool aws -c ${CLUSTER_NAME} \
--instance-type p2.xlarge \
--ami-id=${AMI_ID_FROM_KIB} \
--replicas=1 ${NODEPOOL_NAME} \
--kubeconfig=${CLUSTER_NAME}.conf
```

5.10.5.9.3 Manually Attach a DKP CLI Cluster to the Management Cluster

1. Find out the `name` of the created `Cluster`, so you can reference it later:

```
kubectl -n <workspace_namespace> get clusters
```

2. Attach the cluster by creating a `KommanderCluster`:

264 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: <cluster_name>
  namespace: <workspace_namespace>
spec:
  kubeconfigRef:
    name: <cluster_name>-kubeconfig
  clusterRef:
    capiCluster:
      name: <cluster_name>
EOF
```

5.10.5.9.4 Next Step:

Day 2 - Cluster Operations Management (see page 510)

5.10.6 AWS Air-gapped with GPU Install

5.10.6.1 This section provides instructions to install DKP in an AWS air-gapped GPU environment.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

For an air-gapped environment, ensure you have Download DKP (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz` , so you can extract the tarball on next page.

5.10.6.2 AWS Prerequisites

Before you begin using Konvoy with AWS, you must:

1. Follow the steps to create a Minimal Permissions and Role to Create Clusters (see page 940)..
2. Create Cluster IAM Policies and Roles (see page 946).
3. Export the AWS region where you want to deploy the cluster:

```
export AWS_REGION=us-west-2
```

4. Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

```
export AWS_PROFILE=<profile>
```

5.10.6.3 GPU Prerequisites

Before you begin, you must:

- Ensure nodes provide an NVIDIA GPU.
- If you are using a public cloud service such as AWS (see page 988), create an AMI with KIB using the instructions on the KIB for GPU (see page 1302) page.
- If you are deploying in a pre-provisioned environment, ensure that you have created the appropriate secret for your GPU nodepool (see page 210) and have uploaded the appropriate artifacts to each node. See the GPU only steps section (see page 0)on the Pre-provisioned Prerequisites Air-gapped page for additional information.

Specific instructions must be followed for enabling `nvidia-gpu-operator` depending on if you want to deploy the app on a Management cluster or a Attached or a Managed cluster.

- For instructions on enabling the NVIDIA platform application on a Management cluster, follow the instructions in the **NVIDIA Platform Application Management Cluster** (see page 861) section.
- For instructions on enabling the NVIDIA platform application on attached or managed clusters, follow the instructions in the **NVIDIA Platform Application Attached or Managed Cluster** (see page 864) section.

Once `nvidia-gpu-operator` has been enabled depending on the cluster type, proceed to the **Select the correct Toolkit version for your NVIDIA GPU Operator** on each of those pages.

5.10.6.4 Next Step:

AWS Air-gapped GPU: Use Node Label Automatic Configuration (see page 333)

5.10.6.5 AWS Air-gapped GPU: Use Node Label Automatic Configuration

When using GPU nodes, it is important they have the proper label identifying them as Nvidia GPU nodes. Node feature discovery (NFD), by default labels PCI hardware as:

```
"feature.node.kubernetes.io/pci-<device label>.present": "true"
```

where `<device label>` is by default as defined in this topic[265]:

```
< class > _ < vendor >
```

However, because there is a wide variety in devices and their assigned PCI classes, you may find that the labels assigned to your GPU nodes do not always properly identify them as containing an Nvidia GPU.

If the default detection does not work, you can manually change the daemonset that the GPU operator creates by running the following command:

```
          nodeSelector:
            feature.node.kubernetes.io/pci-< class > _ < vendor>.present: "true"
```

where `class` is any 4 digit number starting with `03xy` and the vendor for Nvidia is `10de`. If this is already deployed, you can always change the `daemonset` and change the `nodeSelector` field so that it deploys to the right nodes.

5.10.6.5.1 Next Step:

AWS Air-gapped GPU: Create an Image (see page 334)

5.10.6.6 AWS Air-gapped GPU: Create an Image

5.10.6.6.1 Create an Image using Konvoy Image Builder (KIB) for use in an air-gapped cluster

5.10.6.6.2 Prerequisites

- Before you begin, you must:
  - Check the DKP Supported Kubernetes Versions (see page 1543).
  - Check the Supported Infrastructure Operating Systems (see page 62)
  - Download the KIB (see page 0) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
  - Create a working Docker or other registry setup.
  - Ensure you have met the minimal set of permissions from the AWS Image Builder Book[266].
  - A Minimal IAM Permissions for KIB (see page 1285) to create an Image for an AWS account using Konvoy Image Builder.

265 https://kubernetes-sigs.github.io/node-feature-discovery/v0.7/get-started/features.html#pci
266 https://image-builder.sigs.k8s.io/capi/providers/aws.html#required-permissions-to-build-the-aws-amis

The default AWS image is not recommended for use in production. We suggest using Konvoy Image Builder to create an AWS Air-gapped AMI to take advantage of enhanced cluster operations. Explore the KIB with AWS (see page 1285) topics for more options.

Using KIB (see page 1282), you can build an AMI without requiring access to the internet by providing an additional `--override` flag.

1. Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz` , extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz && cd dkp-v2.5.2/kib
   ```

2. Follow the instructions below to build an AMI.

5.10.6.6.3 Build the GPU Image

Using the Konvoy Image Builder (see page 1282), you can build an image that has support to use NVIDIA GPU hardware to support GPU workloads.

NOTE: The NVIDIA driver requires a specific Linux kernel version. Make sure that the base image for the OS version has the required kernel version.

See Supported Infrastructure Operating Systems (see page 62) for a list of OS versions and the corresponding kernel versions known to work with the NVIDIA driver.

If the NVIDIA runfile[267] installer has not been downloaded, then retrieve and install the download first by running the following command. The first line in the command below downloads and installs the runfile and the second line places it in the artifacts directory (you must create an `artifacts` directory if you don't already have one).

```
curl -O https://download.nvidia.com/XFree86/Linux-x86_64/470.82.01/NVIDIA-Linux-x86_64-470.82.01.run
mv NVIDIA-Linux-x86_64-470.82.01.run artifacts
```

267 https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html#runfile

DKP supported NVIDIA driver[268] version is 470.x.

To build an image for use on GPU enabled hardware, perform the following steps.

1. In your `overrides/nvidia.yaml` file, add the following to enable GPU builds. You can also access and use the overrides repo[269] or in the documentation under Nvidia GPU Override File (see page 1334) or Offline Nvidia Override file (see page 1335).
   a. Non-air-gapped GPU override:

```
gpu:
  types:
    - nvidia
build_name_extra: "-nvidia"
```

   b. Air-gapped GPU override:

```
# Use this file when building a machine image, not as a override secret
for preprovisioned environments
nvidia_runfile_local_file: "{{ playbook_dir}}/../artifacts/
{{ nvidia_runfile_installer }}"
gpu:
  types:
    - nvidia

build_name_extra: "-nvidia"
```

2. Build your image using the following Konvoy Image Builder commands, making sure to include the flag `--instance-type` that specifies an AWS instance that has an available GPU:
   AWS Example:

```
konvoy-image build --region us-east-1 --instance-type=p2.xlarge --source-
ami=ami-12345abcdef images/ami/centos-7.yaml --overrides overrides/nvidia.yaml
```

   In this example, we chose an instance type with an NVIDIA GPU using the `--instance-type` flag, and we provided the NVIDIA overrides using the `--overrides` flag. See KIB with AWS (see page 1285) for more information on creating an AMI.

268 https://www.nvidia.com/Download/Find.aspx
269 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

Additional helpful information can be found in the NVIDIA Device Plug-in[270] for Kubernetes instructions and the Installation Guide of Supported Platforms[271].

See also: NVIDIA documentation[272]

5.10.6.6.4 Next Steps:

AWS Air-gapped GPU: Load the Registry (see page 337)

5.10.6.7 AWS Air-gapped GPU: Load the Registry

After you create an image for your air-gapped environment, you will need to load it into your registry.

5.10.6.7.1 Load Images into your Registry

Before creating a Kubernetes cluster, you need the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1162) and either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1349) page for more information.

1. Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz`, extract the tarball to a local directory:

```
tar -xzvf dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz && cd dkp-v2.5.0
```

2. Set an environment variable with your registry address with this command:

```
export REGISTRY_ADDRESS=<registry-address>:<registry-port>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

3. Execute the following command to load the air-gapped image bundle into your private registry:

270 https://github.com/NVIDIA/k8s-device-plugin/blob/master/README.md
271 https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html
272 https://nvidia.custhelp.com/app/answers/detail/a_id/131/kw/driver%20installation%20docs/related/1

```
dkp push image-bundle --image-bundle ./container-images/konvoy-image-bundle-
v2.5.0.tar --to-registry $REGISTRY_ADDRESS --to-registry-username
$REGISTRY_USERNAME --to-registry-password $REGISTRY_PASSWORD
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

5.10.6.7.2 Kommander Load Images

If you are operating in an air-gapped environment, a local container registry (see page 1349) containing all the necessary installation images, including the Kommander images is required. See below for how to push the necessary images to this registry.

For the basic air-gapped `kommander` image bundle, run the command below:

1. Run the following command to load the image bundle:

```
dkp push image-bundle --image-bundle ./container-images/kommander-image-bundle-
v2.5.0.tar --to-registry $REGISTRY_ADDRESS
```

For **DKP Catalog Applications**, also perform this image load:

1. Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push image-bundle --image-bundle ./container-images/dkp-catalog-
applications-image-bundle-v2.5.0.tar --to-registry $REGISTRY_ADDRESS
```

5.10.6.7.3 Next Step:

AWS Air-gapped GPU: Create the Management Cluster (see page 338)

5.10.6.8 AWS Air-gapped GPU: Create the Management Cluster

To create a cluster in an AWS Air-gapped environment using GPUs, execute the following:

To increase Docker Hub's rate limit[273], use your Docker Hub credentials when creating the cluster by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster` command.

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable to the name you assigned this cluster:

   ```
   export CLUSTER_NAME=<aws-example>
   ```

   NOTE: The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[274] for more naming information.

3. Export variables for the existing infrastructure details:

   ```
   export AWS_VPC_ID=<vpc-...>
   export AWS_SUBNET_IDS=<subnet-...,subnet-...,subnet-...>
   export AWS_ADDITIONAL_SECURITY_GROUPS=<sg-...>
   export AWS_AMI_ID=<ami-...>
   ```

   - `AWS_VPC_ID`: the VPC ID where the cluster will be created. The VPC requires the `ec2`, `elasticloadbalancing`, `secretsmanager` and `autoscaling` VPC endpoints to be already present.
   - `AWS_SUBNET_IDS`: a comma-separated list of one or more private Subnet IDs with each one in a different Availability Zone. The cluster control-plane and worker nodes will automatically be spread across these Subnets.
   - `AWS_ADDITIONAL_SECURITY_GROUPS`: a comma-seperated list of one or more Security Groups IDs to use in addition to the ones automatically created by CAPA[275].
   - `AWS_AMI_ID`: the AMI ID to use for control-plane and worker nodes. The AMI must be created by the konvoy-image-builder (see page 1282).

   ⚠ **IMPORTANT:** You must tag the subnets as described below to allow for Kubernetes to create ELBs for services of type `LoadBalancer` in those subnets. If the subnets are not tagged, they will

273 https://docs.docker.com/docker-hub/download-rate-limit/
274 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/
275 https://github.com/kubernetes-sigs/cluster-api-provider-aws

not receive an ELB and the following error displays: `Error syncing load balancer, failed to ensure load balancer; could not find any suitable subnets for creating the ELB.`.

The tags should be set as follows, where `<CLUSTER_NAME>` corresponds to the name set in `CLUSTER_NAME` environment variable:

```
kubernetes.io/cluster = <CLUSTER_NAME>
kubernetes.io/cluster/<CLUSTER_NAME> = owned
kubernetes.io/role/internal-elb = 1
```

4. Configure your cluster to use an existing registry as a mirror when attempting to pull images:
⚠ If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1349) page for more information.

⚠ **IMPORTANT:** The AMI must be created by the konvoy-image-builder (see page 1282) project in order to use the registry mirror feature.

```
export REGISTRY_URL=<https/http>://<registry-address>:<registry-port>
export REGISTRY_CA=<path to the CA on the bastion>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

   - `REGISTRY_URL` : the address of an existing registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
   - `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
   - `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
   - `REGISTRY_PASSWORD` : optional if username is not set.

5. Create a Kubernetes cluster. The following example shows a common configuration. See dkp create cluster aws (see page 1456) reference for the full list of cluster creation options:

⚠ DKP uses AWS CSI as the default storage provider (see page 103). You can use a Kubernetes CSI[276] compatible storage solution that is suitable for production. See the Kubernetes documentation called Changing the Default Storage Class[277] for more information.

276 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
277 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

- ```
  dkp create cluster aws \
  --cluster-name=${CLUSTER_NAME} \
  --additional-tags=owner=$(whoami) \
  --with-aws-bootstrap-credentials=true \
  --vpc-id=${AWS_VPC_ID} \
  --ami=${AWS_AMI_ID} \
  --subnet-ids=${AWS_SUBNET_IDS} \
  --internal-load-balancer=true \
  --additional-security-group-ids=${AWS_ADDITIONAL_SECURITY_GROUPS} \
  --registry-mirror-url=${REGISTRY_URL} \
  --registry-mirror-cacert=${REGISTRY_CA} \
  --registry-mirror-username=${REGISTRY_USERNAME} \
  --registry-mirror-password=${REGISTRY_PASSWORD} \
  --self-managed
  ```

  If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

6. After cluster creation, create the node pool after cluster creation:

```
dkp create nodepool aws -c ${CLUSTER_NAME} \
--instance-type p2.xlarge \
--ami-id=${AMI_ID_FROM_KIB} \
--replicas=1 ${NODEPOOL_NAME} \
--kubeconfig=${CLUSTER_NAME}.conf
```

- A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing. As part of the underlying processing, the DKP CLI:
  - creates a bootstrap cluster
  - creates a workload cluster
  - moves CAPI controllers from the bootstrap cluster to the workload cluster, making it self-managed
  - deletes the bootstrap cluster

  To understand how this process works step by step, you can find a customizable Create a New AWS Cluster (see page 953) under Additional Infrastructure Configuration.

5.10.6.8.1 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1360).
```

5.10.6.8.2 Next Step:

AWS Air-gapped GPU: Install Kommander (see page 342)

5.10.6.9 AWS Air-gapped GPU: Install Kommander

You have installed the Konvoy component and created a cluster. Now it is time to Install Kommander which will allow you to access the UI and attach new or existing clusters to monitor.

5.10.6.9.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 119).
- Ensure you have a default StorageClass (see page 1254).
- Ensure you have loaded all necessary images for your configuration. See load the images into your registry (see page 1258) .
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

5.10.6.9.2 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1227) for the deployment:

   ```
   dkp install kommander --init --airgapped > kommander.yaml
   ```

4. *If required:* Customize your `kommander.yaml` .
   See Kommander Additional Install Configurations (see page 1227) for customization options. Some of them include:
   Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1271)), etc.

5. *If required*: If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

```
apps:
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true"
```

5.10.6.9.3 Enable GPU Resources

1. Append the following to the apps section in the `kommander.yaml` file to enable Nvidia platform services.

   ```
   apps:
     nvidia-gpu-operator:
      enabled: true
   ```

2. Append the correct Nvidia Toolkit version based on your OS:

   The **NVIDIA Container Toolkit** allows users to run GPU accelerated containers. The toolkit includes a container runtime library and utilities to automatically configure containers to leverage NVIDIA GPU and must be configured correctly according to your base operating system.

   Centos 7.9/RHEL 7.9:
   If you're using Centos 7.9 or RHEL 7.9 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1227) or `<kommander.yaml>` to the following:

   ```
   kind: Installation
   apps:
     nvidia-gpu-operator:
      enabled: true
      values: |
        toolkit:
          version: v1.10.0-centos7
   ```

   RHEL 8.4/8.6 and SLES 15 SP3
   If you're using RHEL 8.4/8.6 or SLES 15 SP3 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1227) or `<kommander.yaml>` to the following:

   ```
   kind: Installation
   apps:
   ```

```
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.10.0-ubi8
```

Ubuntu 18.04 and 20.04
If you're using Ubuntu 18.04 or 20.04 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1227) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.11.0-ubuntu20.04
```

5.10.6.9.4 Enable DKP Catalog Applications and Install Kommander

If you have an Enterprise license and would like to use the DKP Catalog Applications function, follow these steps:

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.5.0
```

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.5.0.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.5.0.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.5.0.tar.gz
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see (Provide Context for Commands with a kubeconfig File (see page 99) .
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

5.10.6.9.5 Next Step:

AWS Air-gapped GPU: Verify Install and Log in the UI (see page 345)

5.10.6.10 AWS Air-gapped GPU: Verify Install and Log in the UI

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander. After the CLI successfully installs the components, it will wait for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Released helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Released` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

5.10.6.10.1 Failed HelmReleases

If any application fails to successfully deploy, you can check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

5.10.6.10.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 524). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 524):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

5.10.6.10.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 510) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

For more GPU information (see page 860), see the section for GPU under Day 2 - Cluster Operations Management.

5.10.6.10.4 Next Step:

AWS Air-gapped GPU: Subsequent New Clusters (see page 348)

5.10.6.11 AWS Air-gapped GPU: Create a Managed Cluster Using the DKP CLI

In previous steps, you created a new Management cluster, which is self-managed. When you use these steps to create new Managed clusters, they will become Attached clusters under your Management Cluster.

> ⚠ When creating Managed clusters (see page 97), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 97)!

5.10.6.11.1 Choose a Workspace for the New Cluster

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a Workspace[278].

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

278 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919315/Workspaces#Create-a-Workspace

5.10.6.11.2 Name Your Cluster

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account are allowed to have the same name.

2. Set the environment variable:

```
export CLUSTER_NAME=<aws-additional>
```

> The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[279] for more naming information.

3. Execute this command to create your Kubernetes cluster using any relevant flags but note that the `--self-managed` is absent so that this cluster can be attached to your previously created Management Cluster.

- Execute this command to create a cluster with a GPU AMI:

```
dkp create cluster aws --cluster-name=${CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--with-aws-bootstrap-credentials=true \
--vpc-id=${AWS_VPC_ID} \
--ami=${AWS_AMI_ID} \
--subnet-ids=${AWS_SUBNET_IDS} \
--internal-load-balancer=true \
--additional-security-group-ids=${AWS_ADDITIONAL_SECURITY_GROUPS} \
--registry-mirror-url=${REGISTRY_URL} \
--registry-mirror-cacert=${REGISTRY_CA} \
--registry-mirror-username=${REGISTRY_USERNAME} \
--registry-mirror-password=${REGISTRY_PASSWORD} \
--namespace=${WORKSPACE_NAMESPACE}
```

  If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

- Create the node pool after cluster creation:

279 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

```
dkp create nodepool aws -c ${CLUSTER_NAME} \
--instance-type p2.xlarge \
--ami-id=${AMI_ID_FROM_KIB} \
--replicas=1 ${NODEPOOL_NAME} \
--kubeconfig=${CLUSTER_NAME}.conf
```

5.10.6.11.3 Manually Attach a DKP CLI Cluster to the Management Cluster

1. Find out the `name` of the created `Cluster`, so you can reference it later:

```
kubectl -n <workspace_namespace> get clusters
```

2. Attach the cluster by creating a `KommanderCluster`:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: <cluster_name>
  namespace: <workspace_namespace>
spec:
  kubeconfigRef:
    name: <cluster_name>-kubeconfig
  clusterRef:
    capiCluster:
      name: <cluster_name>
EOF
```

5.10.6.11.4 Next Step:

Day 2 - Cluster Operations Management (see page 510)

5.11 EKS Install Options

Enterprise Gov Advanced

For an environment that is EKS on the AWS Infrastructure, install options are provided for you in this one location. Remember, there are always more options in the Additional Infrastructure Configurations (see page 1005) sections, but this will get you operative in the most common scenario.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

Below are the supported environment variables. (Refer to this page to check if your OS is a Supported Operating System (see page 62).)

- EKS Install (see page 351)

> In order to install Kommander, you need to have CAPI components, cert-manager, etc on a self-managed cluster. The CAPI components mean you can control the lifecycle of the cluster, and other clusters. However, because EKS is semi-managed by AWS, the EKS clusters are under AWS control and don't have those components. Therefore, Kommander will not be installed.

5.11.1 EKS Install

Enterprise Gov Advanced

This section provides instructions to install DKP with EKS in an AWS non-air-gapped environment.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

> Ensure that the `KUBECONFIG` environment variable is set to the self-managed cluster by running `export KUBECONFIG={SELF_MANAGED_AWS_CLUSTER}.conf`.

5.11.1.1 AWS prerequisites

Before you begin using DKP with AWS, you must have:

- You have a valid AWS account with credentials configured[280] that can manage CloudFormation Stacks, IAM Policies, and IAM Roles.

280 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html

- You will need to have the AWS CLI utility installed[281].
- Install aws-iam-authenticator[282]. This binary is used to access your cluster using kubectl.

> In order to install Kommander, you need to have CAPI components, cert-manager, etc on a self-managed cluster. The CAPI components mean you can control the lifecycle of the cluster, and other clusters. However, because EKS is semi-managed by AWS, the EKS clusters are under AWS control and don't have those components. Therefore, Kommander will not be installed and these clusters will be attached to the management cluster.

5.11.1.2 Next Step:

EKS: Minimal User Permission for Cluster Creation (see page 352)

5.11.1.3 EKS: Minimal User Permission for Cluster Creation

Enterprise Gov Advanced

The following is a cloudformation stack which adds a policy named `eks-bootstrapper` to manage EKS cluster to the `dkp-bootstrapper-role` created by the cloudformation stack in the Minimal Permissions and Role to Create Cluster (see page 940) section. Consult the Leveraging the Role (see page 944)section for an example of how to use this role and how a system administrator wants to expose using the permissions.

```
AWSTemplateFormatVersion: 2010-09-09
Parameters:
  existingBootstrapperRole:
    Type: CommaDelimitedList
    Description: 'Name of existing minimal role you want to add to add EKS cluster
management permissions to'
    Default: dkp-bootstrapper-role
Resources:
  EKSMinimumPermissions:
    Properties:
      Description: Minimal user policy to manage eks clusters
      ManagedPolicyName: eks-bootstrapper
      PolicyDocument:
        Statement:
          - Action:
```

281 https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html
282 https://docs.aws.amazon.com/eks/latest/userguide/install-aws-iam-authenticator.html

```
              - 'ssm:GetParameter'
            Effect: Allow
            Resource:
              - 'arn:*:ssm:*:*:parameter/aws/service/eks/optimized-ami/*'
          - Action:
              - 'iam:CreateServiceLinkedRole'
            Condition:
              StringLike:
                'iam:AWSServiceName': eks.amazonaws.com
            Effect: Allow
            Resource:
              - >-
                arn:*:iam::*:role/aws-service-role/eks.amazonaws.com/
AWSServiceRoleForAmazonEKS
          - Action:
              - 'iam:CreateServiceLinkedRole'
            Condition:
              StringLike:
                'iam:AWSServiceName': eks-nodegroup.amazonaws.com
            Effect: Allow
            Resource:
              - >-
                arn:*:iam::*:role/aws-service-role/eks-nodegroup.amazonaws.com/
AWSServiceRoleForAmazonEKSNodegroup
          - Action:
              - 'iam:CreateServiceLinkedRole'
            Condition:
              StringLike:
                'iam:AWSServiceName': eks-fargate.amazonaws.com
            Effect: Allow
            Resource:
              - >-
                arn:aws:iam::*:role/aws-service-role/eks-fargate-pods.amazonaws.com/
AWSServiceRoleForAmazonEKSForFargate
          - Action:
              - 'iam:GetRole'
              - 'iam:ListAttachedRolePolicies'
            Effect: Allow
            Resource:
              - 'arn:*:iam::*:role/*'
          - Action:
              - 'iam:GetPolicy'
            Effect: Allow
            Resource:
              - 'arn:aws:iam::aws:policy/AmazonEKSClusterPolicy'
          - Action:
              - 'eks:DescribeCluster'
              - 'eks:ListClusters'
              - 'eks:CreateCluster'
              - 'eks:TagResource'
              - 'eks:UpdateClusterVersion'
              - 'eks:DeleteCluster'
```

```
            - 'eks:UpdateClusterConfig'
            - 'eks:UntagResource'
            - 'eks:UpdateNodegroupVersion'
            - 'eks:DescribeNodegroup'
            - 'eks:DeleteNodegroup'
            - 'eks:UpdateNodegroupConfig'
            - 'eks:CreateNodegroup'
            - 'eks:AssociateEncryptionConfig'
            - 'eks:ListIdentityProviderConfigs'
            - 'eks:AssociateIdentityProviderConfig'
            - 'eks:DescribeIdentityProviderConfig'
            - 'eks:DisassociateIdentityProviderConfig'
          Effect: Allow
          Resource:
            - 'arn:*:eks:*:*:cluster/*'
            - 'arn:*:eks:*:*:nodegroup/*/*/*'
        - Action:
            - 'ec2:AssociateVpcCidrBlock'
            - 'ec2:DisassociateVpcCidrBlock'
            - 'eks:ListAddons'
            - 'eks:CreateAddon'
            - 'eks:DescribeAddonVersions'
            - 'eks:DescribeAddon'
            - 'eks:DeleteAddon'
            - 'eks:UpdateAddon'
            - 'eks:TagResource'
            - 'eks:DescribeFargateProfile'
            - 'eks:CreateFargateProfile'
            - 'eks:DeleteFargateProfile'
          Effect: Allow
          Resource:
            - '*'
        - Action:
            - 'iam:PassRole'
          Condition:
            StringEquals:
              'iam:PassedToService': eks.amazonaws.com
          Effect: Allow
          Resource:
            - '*'
        - Action:
            - 'kms:CreateGrant'
            - 'kms:DescribeKey'
          Condition:
            'ForAnyValue:StringLike':
              'kms:ResourceAliases': alias/cluster-api-provider-aws-*
          Effect: Allow
          Resource:
            - '*'
      Version: 2012-10-17
    Roles: !Ref existingBootstrapperRole
  Type: 'AWS::IAM::ManagedPolicy'
```

If your role is not named `dkp-bootstrapper-role` change the parameter on line 6 of the file.

To create the resources in the cloudformation stack, copy the contents above into a file. Before executing the following command, replace `MYFILENAME.yaml` and `MYSTACKNAME` with the intended values for your system:

```
aws cloudformation create-stack --template-body=file://MYFILENAME.yaml --stack-
name=MYSTACKNAME --capabilities CAPABILITY_NAMED_IAM
```

5.11.1.3.1 Next Step:

EKS: Cluster IAM Policies and Roles (see page 355)

5.11.1.4 EKS: Cluster IAM Policies and Roles

Enterprise Gov Advanced

This section guides a DKP user in creating IAM Policies and Instance Profiles that governs who has access to the cluster. The IAM Role is used by the cluster's control plane and worker nodes using the provided AWS CloudFormation Stack specific to EKS. This CloudFormation Stack has additional permissions that are used to delegate access roles for other users.

5.11.1.4.1 Prerequisites from AWS:

- The user you delegate from your role must have a minimum set of permissions, see User Roles and Instance Profiles (see page 940) page for AWS.
- Create the Cluster IAM Policies (see page 946) in your AWS account.

5.11.1.4.2 EKS IAM Artifacts

5.11.1.4.2.1 Policies

- `controllers-eks.cluster-api-provider-aws.sigs.k8s.io` - enumerates the Actions required by the workload cluster to create and modify EKS clusters in the user's AWS Account. It is attached to the existing `control-plane.cluster-api-provider-aws.sigs.k8s.io` role

- `eks-nodes.cluster-api-provider-aws.sigs.k8s.io` - enumerates the Actions required by the EKS workload cluster's worker machines. It is attached to the existing `nodes.cluster-api-provider-aws.sigs.k8s.io`

5.11.1.4.2.2 Roles

- `eks-controlplane.cluster-api-provider-aws.sigs.k8s.io` - is the Role associated with EKS cluster control planes

> NOTE: `control-plane.cluster-api-provider-aws.sigs.k8s.io` and `nodes.cluster-api-provider-aws.sigs.k8s.io` roles were created by Cluster IAM Policies and Roles (see page 946) in AWS.

Below is a CloudFormation stack[283] that includes IAM policies and roles required to setup EKS Clusters:

```
AWSTemplateFormatVersion: 2010-09-09
Parameters:
  existingControlPlaneRole:
    Type: CommaDelimitedList
    Description: 'Names of existing Control Plane Role you want to add to the newly
created EKS Managed Policy for AWS cluster API controllers'
    Default: control-plane.cluster-api-provider-aws.sigs.k8s.io
  existingNodeRole:
    Type: CommaDelimitedList
    Description: 'ARN of the Nodes Managed Policy to add to the role for nodes'
    Default: nodes.cluster-api-provider-aws.sigs.k8s.io
Resources:
  AWSIAMManagedPolicyControllersEKS:
    Properties:
      Description: For the Kubernetes Cluster API Provider AWS Controllers
      ManagedPolicyName: controllers-eks.cluster-api-provider-aws.sigs.k8s.io
      PolicyDocument:
        Statement:
          - Action:
              - 'ssm:GetParameter'
            Effect: Allow
            Resource:
              - 'arn:*:ssm:*:*:parameter/aws/service/eks/optimized-ami/*'
          - Action:
              - 'iam:CreateServiceLinkedRole'
            Condition:
              StringLike:
```

283 https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html

```
                'iam:AWSServiceName': eks.amazonaws.com
              Effect: Allow
              Resource:
                - >-
                  arn:*:iam::*:role/aws-service-role/eks.amazonaws.com/
AWSServiceRoleForAmazonEKS
            - Action:
                - 'iam:CreateServiceLinkedRole'
              Condition:
                StringLike:
                  'iam:AWSServiceName': eks-nodegroup.amazonaws.com
              Effect: Allow
              Resource:
                - >-
                  arn:*:iam::*:role/aws-service-role/eks-nodegroup.amazonaws.com/
AWSServiceRoleForAmazonEKSNodegroup
            - Action:
                - 'iam:CreateServiceLinkedRole'
              Condition:
                StringLike:
                  'iam:AWSServiceName': eks-fargate.amazonaws.com
              Effect: Allow
              Resource:
                - >-
                  arn:aws:iam::*:role/aws-service-role/eks-fargate-pods.amazonaws.com/
AWSServiceRoleForAmazonEKSForFargate
            - Action:
                - 'iam:GetRole'
                - 'iam:ListAttachedRolePolicies'
              Effect: Allow
              Resource:
                - 'arn:*:iam::*:role/*'
            - Action:
                - 'iam:GetPolicy'
              Effect: Allow
              Resource:
                - 'arn:aws:iam::aws:policy/AmazonEKSClusterPolicy'
            - Action:
                - 'eks:DescribeCluster'
                - 'eks:ListClusters'
                - 'eks:CreateCluster'
                - 'eks:TagResource'
                - 'eks:UpdateClusterVersion'
                - 'eks:DeleteCluster'
                - 'eks:UpdateClusterConfig'
                - 'eks:UntagResource'
                - 'eks:UpdateNodegroupVersion'
                - 'eks:DescribeNodegroup'
                - 'eks:DeleteNodegroup'
                - 'eks:UpdateNodegroupConfig'
                - 'eks:CreateNodegroup'
                - 'eks:AssociateEncryptionConfig'
```

```
            - 'eks:ListIdentityProviderConfigs'
            - 'eks:AssociateIdentityProviderConfig'
            - 'eks:DescribeIdentityProviderConfig'
            - 'eks:DisassociateIdentityProviderConfig'
          Effect: Allow
          Resource:
            - 'arn:*:eks:*:*:cluster/*'
            - 'arn:*:eks:*:*:nodegroup/*/*/*'
        - Action:
            - 'ec2:AssociateVpcCidrBlock'
            - 'ec2:DisassociateVpcCidrBlock'
            - 'eks:ListAddons'
            - 'eks:CreateAddon'
            - 'eks:DescribeAddonVersions'
            - 'eks:DescribeAddon'
            - 'eks:DeleteAddon'
            - 'eks:UpdateAddon'
            - 'eks:TagResource'
            - 'eks:DescribeFargateProfile'
            - 'eks:CreateFargateProfile'
            - 'eks:DeleteFargateProfile'
          Effect: Allow
          Resource:
            - '*'
        - Action:
            - 'iam:PassRole'
          Condition:
            StringEquals:
              'iam:PassedToService': eks.amazonaws.com
          Effect: Allow
          Resource:
            - '*'
        - Action:
            - 'kms:CreateGrant'
            - 'kms:DescribeKey'
          Condition:
            'ForAnyValue:StringLike':
              'kms:ResourceAliases': alias/cluster-api-provider-aws-*
          Effect: Allow
          Resource:
            - '*'
      Version: 2012-10-17
    Roles: !Ref existingControlPlaneRole
  Type: 'AWS::IAM::ManagedPolicy'
AWSIAMManagedEKSNodesPolicy:
  Properties:
    Description: Additional Policies to nodes role to work for EKS
    ManagedPolicyName: eks-nodes.cluster-api-provider-aws.sigs.k8s.io
    PolicyDocument:
      Statement:
        - Action:
            - "ec2:AssignPrivateIpAddresses"
```

```
            - "ec2:AttachNetworkInterface"
            - "ec2:CreateNetworkInterface"
            - "ec2:DeleteNetworkInterface"
            - "ec2:DescribeInstances"
            - "ec2:DescribeTags"
            - "ec2:DescribeNetworkInterfaces"
            - "ec2:DescribeInstanceTypes"
            - "ec2:DetachNetworkInterface"
            - "ec2:ModifyNetworkInterfaceAttribute"
            - "ec2:UnassignPrivateIpAddresses"
          Effect: Allow
          Resource:
            - '*'
        - Action:
            - ec2:CreateTags
          Effect: Allow
          Resource:
            - arn:aws:ec2:*:*:network-interface/*
        - Action:
            - "ec2:DescribeInstances"
            - "ec2:DescribeInstanceTypes"
            - "ec2:DescribeRouteTables"
            - "ec2:DescribeSecurityGroups"
            - "ec2:DescribeSubnets"
            - "ec2:DescribeVolumes"
            - "ec2:DescribeVolumesModifications"
            - "ec2:DescribeVpcs"
            - "eks:DescribeCluster"
          Effect: Allow
          Resource:
            - '*'
      Version: 2012-10-17
    Roles: !Ref existingNodeRole
  Type: 'AWS::IAM::ManagedPolicy'
AWSIAMRoleEKSControlPlane:
  Properties:
    AssumeRolePolicyDocument:
      Statement:
        - Action:
            - 'sts:AssumeRole'
          Effect: Allow
          Principal:
            Service:
              - eks.amazonaws.com
      Version: 2012-10-17
    ManagedPolicyArns:
      - 'arn:aws:iam::aws:policy/AmazonEKSClusterPolicy'
    RoleName: eks-controlplane.cluster-api-provider-aws.sigs.k8s.io
  Type: 'AWS::IAM::Role'
```

To create the resources in the CloudFormation stack, copy the contents above into a file and execute the following command after replacing `MYFILENAME.yaml` and `MYSTACKNAME` with the intended values:

```
 aws cloudformation create-stack --template-body=file://MYFILENAME.yaml --stack-
name=MYSTACKNAME --capabilities  CAPABILITY_NAMED_IAM
```

5.11.1.4.2.3 Add EKS CSI Policy

AWS CloudFormation does not support attaching an existing IAM Policy to an existing IAM Role. Add the necessary IAM policy to your worker instance profile using the `aws` CLI:

```
aws iam attach-role-policy --role-name nodes.cluster-api-provider-aws.sigs.k8s.io --
policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy
```

5.11.1.4.3 Next Step:

EKS: Create an Image (see page 360)

5.11.1.5 EKS: Create an Image

Enterprise Gov Advanced

AWS EKS best practices discourage building custom images. The Amazon EKS Optimized AMI[284] is the preferred way to deploy containers for EKS. If the image is customized, it breaks some of the autoscaling and security capabilities of EKS.

5.11.1.5.1 Next Step:

EKS: Cluster Creation (see page 360)

5.11.1.6 EKS: Create an EKS Cluster

Enterprise Gov Advanced

284 https://docs.aws.amazon.com/eks/latest/userguide/eks-optimized-amis.html

When installing DKP on your EKS infrastructure, you can choose from multiple configuration types. The steps for creating and accessing your cluster are listed below after setting minimal permissions (see page 352) and creating IAM Policies and Roles (see page 355).

5.11.1.6.1 Access Cluster

- Use previously installed aws-iam-authenticator[285] to access your cluster using kubectl. Amazon EKS uses IAM to provide authentication to your Kubernetes cluster.
- Export the AWS region where you want to deploy the cluster:

```
export AWS_REGION=us-west-2
```

- Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

```
export AWS_PROFILE=<profile>
```

See the AWS site for more information about AWS credentials.[286]

Ensure that the `KUBECONFIG` environment variable is set to the self-managed cluster by running `export KUBECONFIG={SELF_MANAGED_AWS_CLUSTER}.conf`

5.11.1.6.2 Name Your Cluster

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable:

```
export CLUSTER_NAME=<aws-example>
```

285 https://docs.aws.amazon.com/eks/latest/userguide/install-aws-iam-authenticator.html
286 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[287] for more naming information.

5.11.1.6.3 Create a New EKS Kubernetes Cluster

By default, the control-plane Nodes will be created in 3 different Availability Zones. However, the default worker Nodes will reside in a single zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Set the environment variable to the name you assigned this cluster.

```
export CLUSTER_NAME=eks-example
```

2. Make sure your AWS credentials are up-to-date. Refresh the credentials command is only necessary if you are using Static Credentials (see page 0) (Access Keys) otherwise, if you are using role-based authentication on a bastion host, proceed to step 3:

```
dkp update bootstrap credentials aws
```

3. Create the cluster:

```
dkp create cluster eks \
  --cluster-name=${CLUSTER_NAME} \
  --additional-tags=owner=$(whoami)
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).
The output displays similar to this:

```
Generating cluster resources
cluster.cluster.x-k8s.io/eks-example created
```

287 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

```
awsmanagedcontrolplane.controlplane.cluster.x-k8s.io/eks-example-control-plane
 created
machinedeployment.cluster.x-k8s.io/eks-example-md-0 created
awsmachinetemplate.infrastructure.cluster.x-k8s.io/eks-example-md-0 created
eksconfigtemplate.bootstrap.cluster.x-k8s.io/eks-example-md-0 created
clusterresourceset.addons.cluster.x-k8s.io/calico-cni-installation-eks-example
 created
configmap/calico-cni-installation-eks-example created
configmap/tigera-operator-eks-example created
clusterresourceset.addons.cluster.x-k8s.io/cluster-autoscaler-eks-example
 created
configmap/cluster-autoscaler-eks-example created
clusterresourceset.addons.cluster.x-k8s.io/node-feature-discovery-eks-example
 created
configmap/node-feature-discovery-eks-example created
clusterresourceset.addons.cluster.x-k8s.io/nvidia-feature-discovery-eks-example
 created
configmap/nvidia-feature-discovery-eks-example created
```

4. Inspect or edit the cluster objects:

   Use your favorite editor.

   NOTE: Editing the cluster objects requires some understanding of Cluster API. Edits can prevent the cluster from deploying successfully.

   The objects are Custom Resources[288] defined by Cluster API components, and they belong in three different categories:

   a. **Cluster**
      A *Cluster* object has references to the infrastructure-specific and control plane objects. Because this is an AWS cluster, there is an *AWSCluster* object that describes the infrastructure-specific cluster properties. Here, this means the AWS region, the VPC ID, subnet IDs, and security group rules required by the Pod network implementation.

   b. **Control Plane**
      A *AWSManagedControlPlane* object describes the control plane, which is the group of machines that run the Kubernetes control plane components, which include the etcd distributed database, the API server, the core controllers, and the scheduler. The object describes the configuration for these components. The object also has a reference to an infrastructure-specific object that describes the properties of all control plane machines.

   c. **Node Pool**
      A Node Pool is a collection of machines with identical properties. For example, a cluster might have one Node Pool with large memory capacity, another Node Pool with GPU support. Each Node Pool is described by three objects: The MachinePool references an object that describes the configuration of Kubernetes components (for example, kubelet) deployed on each node pool machine, and an infrastructure-specific object that describes the properties of all node pool machines. Here, it references a *KubeadmConfigTemplate*, and an *AWSMachineTemplate* object, which describes the instance type, the type of disk used, the size of the disk, among other properties.

288 https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/

For in-depth documentation about the objects, read Concepts[289] in the Cluster API Book.

5. Wait for the cluster control-plane to be ready:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=20m
```

```
cluster.cluster.x-k8s.io/eks-example condition met
```

The `READY` status will become `True` after the cluster control-plane becomes ready in one of the following steps.

6. Once the objects are created on the API server, the Cluster API controllers reconcile them. They create infrastructure and machines. As they progress, they update the Status of each object. Konvoy provides a command to describe the current status of the cluster:

```
dkp describe cluster -c ${CLUSTER_NAME}
```

```
NAME                                                          READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/eks-example                                           True
10m
├──ControlPlane - AWSManagedControlPlane/eks-example-control-plane  True
10m
└──Workers
   └──MachineDeployment/eks-example-md-0                       True
26s
     ├──Machine/eks-example-md-0-78fcd7c7b7-66ntt              True
84s
     ├──Machine/eks-example-md-0-78fcd7c7b7-b9qmc              True
84s
     ├──Machine/eks-example-md-0-78fcd7c7b7-v5vfq              True
84s
     └──Machine/eks-example-md-0-78fcd7c7b7-zl6m2              True
84s
```

7. As they progress, the controllers also create Events. List the Events using this command:

```
kubectl get events | grep ${CLUSTER_NAME}
```

289 https://cluster-api.sigs.k8s.io/user/concepts.html

For brevity, the example uses `grep`. It is also possible to use separate commands to get Events for specific objects. For example, `kubectl get events --field-selector involvedObject.kind="AWSCluster"` and `kubectl get events --field-selector involvedObject.kind="AWSMachine"`.

```
46m         Normal   SuccessfulCreateVPC
awsmanagedcontrolplane/eks-example-control-plane   Created new managed VPC
"vpc-05e775702092abf09"
46m         Normal   SuccessfulSetVPCAttributes
awsmanagedcontrolplane/eks-example-control-plane   Set managed VPC attributes
for "vpc-05e775702092abf09"
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-0419dd3f2dfd95ff8"
46m         Normal   SuccessfulModifySubnetAttributes
awsmanagedcontrolplane/eks-example-control-plane   Modified managed Subnet
"subnet-0419dd3f2dfd95ff8" attributes
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-0e724b128e3113e47"
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-06b2b31ea6a8d3962"
46m         Normal   SuccessfulModifySubnetAttributes
awsmanagedcontrolplane/eks-example-control-plane   Modified managed Subnet
"subnet-06b2b31ea6a8d3962" attributes
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-0626ce238be32bf98"
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-0f53cf59f83177800"
46m         Normal   SuccessfulModifySubnetAttributes
awsmanagedcontrolplane/eks-example-control-plane   Modified managed Subnet
"subnet-0f53cf59f83177800" attributes
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-0878478f6bbf153b2"
46m         Normal   SuccessfulCreateInternetGateway
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Internet
Gateway "igw-09fb52653949d4579"
46m         Normal   SuccessfulAttachInternetGateway
awsmanagedcontrolplane/eks-example-control-plane   Internet Gateway
"igw-09fb52653949d4579" attached to VPC "vpc-05e775702092abf09"
46m         Normal   SuccessfulCreateNATGateway
awsmanagedcontrolplane/eks-example-control-plane   Created new NAT Gateway
"nat-06356aac28079952d"
```

```
46m         Normal   SuccessfulCreateNATGateway
awsmanagedcontrolplane/eks-example-control-plane   Created new NAT Gateway
"nat-0429d1cd9d956bf35"
46m         Normal   SuccessfulCreateNATGateway
awsmanagedcontrolplane/eks-example-control-plane   Created new NAT Gateway
"nat-059246bcc9d4e88e7"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-01689c719c484fd3c"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-01689c719c484fd3c" with subnet "subnet-0419dd3f2dfd95ff8"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-065af81b9752eeb69"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-065af81b9752eeb69" with subnet "subnet-0e724b128e3113e47"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-03eeff810a89afc98"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-03eeff810a89afc98" with subnet "subnet-06b2b31ea6a8d3962"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-0fab36f8751fdee73"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-0fab36f8751fdee73" with subnet "subnet-0626ce238be32bf98"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-0e5c9c7bbc3740a0f"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-0e5c9c7bbc3740a0f" with subnet "subnet-0f53cf59f83177800"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-0bf58eb5f73c387af"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
```

```
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-0bf58eb5f73c387af" with subnet "subnet-0878478f6bbf153b2"
46m         Normal   SuccessfulCreateSecurityGroup
awsmanagedcontrolplane/eks-example-control-plane   Created managed
SecurityGroup "sg-0b045c998a120a1b2" for Role "node-eks-additional"
46m         Normal   InitiatedCreateEKSControlPlane
awsmanagedcontrolplane/eks-example-control-plane   Initiated creation of a new
 EKS control plane default_eks-example-control-plane
37m         Normal   SuccessfulCreateEKSControlPlane
awsmanagedcontrolplane/eks-example-control-plane   Created new EKS control
plane default_eks-example-control-plane
37m         Normal   SucessfulCreateKubeconfig
awsmanagedcontrolplane/eks-example-control-plane   Created kubeconfig for
 cluster "eks-example"
37m         Normal   SucessfulCreateUserKubeconfig
awsmanagedcontrolplane/eks-example-control-plane   Created user kubeconfig for
 cluster "eks-example"
27m         Normal   SuccessfulCreate
awsmachine/eks-example-md-0-4t9nc                  Created new node instance
with id "i-0aecc1897c93df740"
26m         Normal   SuccessfulDeleteEncryptedBootstrapDataSecrets
awsmachine/eks-example-md-0-4t9nc                  AWS Secret entries
containing userdata deleted
26m         Normal   SuccessfulSetNodeRef                            machine/
eks-example-md-0-78fcd7c7b7-fn7x9          ip-10-0-88-24.us-west-2.compute.inte
rnal
26m         Normal   SuccessfulSetNodeRef                            machine/
eks-example-md-0-78fcd7c7b7-g64nv          ip-10-0-110-219.us-west-2.compute.in
ternal
26m         Normal   SuccessfulSetNodeRef                            machine/
eks-example-md-0-78fcd7c7b7-gwc5j          ip-10-0-101-161.us-west-2.compute.in
ternal
26m         Normal   SuccessfulSetNodeRef                            machine/
eks-example-md-0-78fcd7c7b7-j58s4          ip-10-0-127-49.us-west-2.compute.int
ernal
46m         Normal   SuccessfulCreate
machineset/eks-example-md-0-78fcd7c7b7             Created machine "eks-
example-md-0-78fcd7c7b7-fn7x9"
46m         Normal   SuccessfulCreate
machineset/eks-example-md-0-78fcd7c7b7             Created machine "eks-
example-md-0-78fcd7c7b7-g64nv"
46m         Normal   SuccessfulCreate
machineset/eks-example-md-0-78fcd7c7b7             Created machine "eks-
example-md-0-78fcd7c7b7-j58s4"
46m         Normal   SuccessfulCreate
machineset/eks-example-md-0-78fcd7c7b7             Created machine "eks-
example-md-0-78fcd7c7b7-gwc5j"
27m         Normal   SuccessfulCreate
awsmachine/eks-example-md-0-7whkv                  Created new node instance
with id "i-06dfc0466b8f26695"
```

```
26m         Normal   SuccessfulDeleteEncryptedBootstrapDataSecrets
awsmachine/eks-example-md-0-7whkv                  AWS Secret entries
containing userdata deleted
27m         Normal   SuccessfulCreate
awsmachine/eks-example-md-0-ttgzv                  Created new node instance
with id "i-0544fce0350fd41fb"
26m         Normal   SuccessfulDeleteEncryptedBootstrapDataSecrets
awsmachine/eks-example-md-0-ttgzv                  AWS Secret entries
containing userdata deleted
27m         Normal   SuccessfulCreate
awsmachine/eks-example-md-0-v2hrf                  Created new node instance
with id "i-0498906edde162e59"
26m         Normal   SuccessfulDeleteEncryptedBootstrapDataSecrets
awsmachine/eks-example-md-0-v2hrf                  AWS Secret entries
containing userdata deleted
46m         Normal   SuccessfulCreate
machinedeployment/eks-example-md-0                 Created MachineSet "eks-
example-md-0-78fcd7c7b7"
```

5.11.1.6.4 Known Limitations

Be aware of these limitations in the current release of Konvoy.

- The Konvoy version used to create a workload cluster must match the Konvoy version used to delete a workload cluster.
- EKS clusters cannot be Self-managed.
- Konvoy supports deploying one workload cluster.
- Konvoy generates a set of objects for one Node Pool.
- Konvoy does not validate edits to cluster objects.

5.11.1.6.4.1 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1360).

5.11.1.6.4.2 Next Step:

EKS: Grant Cluster Access (see page 369)

5.11.1.7 EKS: Grant Cluster Access

Enterprise Gov Advanced

5.11.1.7.1 How to Grant EKS Cluster Access

You can access your cluster using AWS IAM roles in the dashboard. When you create an EKS cluster, the IAM entity is granted `system:masters` permissions in Kubernetes Role Based Access Control (RBAC) configuration.[290]

> More information about the configuration of the EKS control plane can be found on the EKS Cluster IAM Policies and Roles (see page 1010) page.

If the EKS cluster was created as a cluster using a self-managed AWS cluster that uses IAM Instance Profiles, you will need to modify the `IAMAuthenticatorConfig` field in the `AWSManagedControlPlane` API object to allow other IAM entities to access the EKS workload cluster. Follow the steps below:

1. Run the following command with your `KUBECONFIG` configured to select the self-managed cluster previously used to create the workload EKS cluster. Ensure you substitute `${CLUSTER_NAME}` and `${CLUSTER_NAMESPACE}` with their corresponding values for your cluster.

   ```
   kubectl edit awsmanagedcontrolplane ${CLUSTER_NAME}-control-plane -n $
   {CLUSTER_NAMESPACE}
   ```

2. Edit the `IamAuthenticatorConfig` field with the IAM Role to the corresponding Kubernetes Role. In this example, the IAM role `arn:aws:iam::111122223333:role/PowerUser` is granted the cluster role `system:masters`. Note that this example uses example AWS resource ARNs[291], so these values should be substituted for real values in the corresponding AWS account.

   ```
   iamAuthenticatorConfig:
       mapRoles:
         - groups:
   ```

290 https://kubernetes.io/docs/reference/access-authn-authz/rbac/
291 https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html

```
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws:iam::111122223333:role/my-node-role
      username: system:node:{{EC2PrivateDNSName}}
    - groups:
      - system:masters
      rolearn: arn:aws:iam::111122223333:role/PowerUser
      username: admin
```

For further instructions on changing or assigning `roles` or `clusterroles` to which you can map IAM users or roles, see Amazon Enabling IAM access to your cluster[292].

5.11.1.7.2 Next Step:

EKS: Retrieve kubeconfig for EKS Cluster (see page 370)

5.11.1.8 EKS: Retrieve kubeconfig for EKS Cluster

Enterprise Gov Advanced

This guide explains how to use the command line to interact with your newly deployed Kubernetes cluster.

Before you start, make sure you have created a workload cluster, as described in Create an EKS Cluster (see page 360) .

5.11.1.8.1 Explore the new Kubernetes cluster

Follow these steps:

1. Get a kubeconfig file for the workload cluster:

   When the workload cluster is created, the cluster lifecycle services generate a kubeconfig file for the workload cluster, and write it to a *Secret*. The kubeconfig file is scoped to the cluster administrator.

   Get the kubeconfig from the *Secret*, and write it to a file, using this command:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

2. List the Nodes using this command:

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf get nodes
   ```

292 https://docs.aws.amazon.com/eks/latest/userguide/add-user-role.html

```
NAME                                           STATUS   ROLES    AGE   VERSION
ip-10-0-122-211.us-west-2.compute.internal     Ready    <none>   35m   v1.23.9-
eks-ba74326
ip-10-0-127-74.us-west-2.compute.internal      Ready    <none>   35m   v1.23.9-
eks-ba74326
ip-10-0-71-155.us-west-2.compute.internal      Ready    <none>   35m   v1.23.9-
eks-ba74326
ip-10-0-93-47.us-west-2.compute.internal       Ready    <none>   35m   v1.23.9-
eks-ba74326
```

NOTE: It may take a few minutes for the Status to move to `Ready` while the Pod network is deployed. The Nodes' Status should change to Ready soon after the `calico-node` DaemonSet Pods are Ready.

3. List the Pods using this command:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf get --all-namespaces pods
```

```
NAMESPACE               NAME                                       READY
STATUS     RESTARTS     AGE
calico-system           calico-kube-controllers-7d6749878f-ccsx9   1/1
Running    0            34m
calico-system           calico-node-2r6l8                          1/1
Running    0            34m
calico-system           calico-node-5pdlb                          1/1
Running    0            34m
calico-system           calico-node-n24hh                          1/1
Running    0            34m
calico-system           calico-node-qrh7p                          1/1
Running    0            34m
calico-system           calico-typha-7bbcb87696-7pk45              1/1
Running    0            34m
calico-system           calico-typha-7bbcb87696-t4c8r              1/1
Running    0            34m
calico-system           csi-node-driver-bz48k                      2/2
Running    0            34m
calico-system           csi-node-driver-k5mmk                      2/2
Running    0            34m
calico-system           csi-node-driver-nvcck                      2/2
Running    0            34m
calico-system           csi-node-driver-x4xnh                      2/2
Running    0            34m
kube-system             aws-node-2xp86                             1/1
Running    0            35m
```

```
kube-system                aws-node-5f2kx                                  1/1
Running    0               35m
kube-system                aws-node-6lzm7                                  1/1
Running    0               35m
kube-system                aws-node-pz8c6                                  1/1
Running    0               35m
kube-system                cluster-autoscaler-789d86b489-sz9x2             0/1
Init:0/1   0               36m
kube-system                coredns-57ff979f67-pk5cg                        1/1
Running    0               75m
kube-system                coredns-57ff979f67-sf2j9                        1/1
Running    0               75m
kube-system                ebs-csi-controller-5f6bd5d6dc-bplwm             6/6
Running    0               36m
kube-system                ebs-csi-controller-5f6bd5d6dc-dpjt7             6/6
Running    0               36m
kube-system                ebs-csi-node-7hmm5                              3/3
Running    0               35m
kube-system                ebs-csi-node-l4vfh                              3/3
Running    0               35m
kube-system                ebs-csi-node-mfr7c                              3/3
Running    0               35m
kube-system                ebs-csi-node-v8krq                              3/3
Running    0               35m
kube-system                kube-proxy-7fc5x                                1/1
Running    0               35m
kube-system                kube-proxy-vvkmk                                1/1
Running    0               35m
kube-system                kube-proxy-x6hcc                                1/1
Running    0               35m
kube-system                kube-proxy-x8frb                                1/1
Running    0               35m
kube-system                snapshot-controller-8ff89f489-4cfxv             1/1
Running    0               36m
kube-system                snapshot-controller-8ff89f489-78gg8             1/1
Running    0               36m
node-feature-discovery     node-feature-discovery-master-7d5985467-52fcn   1/1
Running    0               36m
node-feature-discovery     node-feature-discovery-worker-88hr7             1/1
Running    0               34m
node-feature-discovery     node-feature-discovery-worker-h95nq             1/1
Running    0               35m
node-feature-discovery     node-feature-discovery-worker-lfghg             1/1
Running    0               34m
node-feature-discovery     node-feature-discovery-worker-prc8p             1/1
Running    0               35m
tigera-operator            tigera-operator-6dcd98c8ff-k97hq                1/1
Running    0               36m
```

5.11.1.8.2 Next Step:

EKS: Attach a Cluster (see page 373)

5.11.1.9 EKS: Attach a Cluster

Enterprise Gov Advanced

You can attach existing Kubernetes clusters to the Management Cluster. After attaching the cluster, you can use the UI to examine and manage (see page 669) this cluster. The following procedure shows how to attach an existing Amazon Elastic Kubernetes Service (EKS) cluster.

> This procedure assumes you have an existing and spun up Amazon EKS cluster(s) with administrative privileges. Refer to the Amazon EKS[293] for setup and configuration information.

- Install aws-iam-authenticator[294]. This binary is used to access your cluster using kubectl.

5.11.1.9.1 Attach a Pre-existing EKS Cluster

Ensure that the `KUBECONFIG` environment variable is set to the Management cluster before attaching by running:

```
export KUBECONFIG=<Management_cluster_kubeconfig>.conf
```

5.11.1.9.1.1 Access your EKS clusters

1. Ensure you are connected to your EKS clusters. Enter the following commands for each of your clusters:

   ```
   kubectl config get-contexts
   kubectl config use-context <context for first eks cluster>
   ```

2. Confirm `kubectl` can access the EKS cluster:

293 https://aws.amazon.com/eks/
294 https://docs.aws.amazon.com/eks/latest/userguide/install-aws-iam-authenticator.html

```
kubectl get nodes
```

5.11.1.9.1.2 Create a `kubeconfig` File

To get started, ensure you have kubectl[295] set up and configured with ClusterAdmin[296] for the cluster you want to connect to Kommander.

1. Create the necessary service account:

   ```
   kubectl -n kube-system create serviceaccount kommander-cluster-admin
   ```

2. Create a token secret for the `serviceaccount`:

   ```
   kubectl -n kube-system create  -f - <<EOF
   apiVersion: v1
   kind: Secret
   metadata:
     name: kommander-cluster-admin-sa-token
     annotations:
       kubernetes.io/service-account.name: kommander-cluster-admin
   type: kubernetes.io/service-account-token
   EOF
   ```

   For more information on Service Account Tokens, refer to this article[297] in our blog.

3. Verify that the `serviceaccount` token is ready by running this command:

   ```
   kubectl -n kube-system get secret kommander-cluster-admin-sa-token -oyaml
   ```

   Verify that the `data.token` field is populated. The output should be similar to this:

   ```
   apiVersion: v1
   data:
     ca.crt: LS0tLS1CRUdJTiBDR...
     namespace: ZGVmYXVsdA==
     token: ZXlKaGJHY2lPaUpTVX...
   kind: Secret
   metadata:
     annotations:
       kubernetes.io/service-account.name: kommander-cluster-admin
   ```

295 https://kubernetes.io/docs/tasks/tools/#kubectl
296 https://kubernetes.io/docs/concepts/cluster-administration/
297 https://eng.d2iq.com/blog/service-account-tokens-in-kubernetes-v1.24/#whats-changed-in-kubernetes-v124

```
    kubernetes.io/service-account.uid: b62bc32e-b502-4654-921d-94a742e273a8
  creationTimestamp: "2022-08-19T13:36:42Z"
  name: kommander-cluster-admin-sa-token
  namespace: default
  resourceVersion: "8554"
  uid: 72c2a4f0-636d-4a70-9f1c-55a75f15e520
type: kubernetes.io/service-account-token
```

4. Configure the new service account for `cluster-admin` permissions:

```
cat << EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kommander-cluster-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: kommander-cluster-admin
  namespace: kube-system
EOF
```

5. Set up the following environment variables with the access data that is needed for producing a new kubeconfig file:

```
export USER_TOKEN_VALUE=$(kubectl -n kube-system get secret/kommander-cluster-
admin-sa-token -o=go-template='{{.data.token}}' | base64 --decode)
export CURRENT_CONTEXT=$(kubectl config current-context)
export CURRENT_CLUSTER=$(kubectl config view --raw -o=go-
template='{{range .contexts}}{{if eq .name "'''${CURRENT_CONTEXT}'''"}}
{{ index .context "cluster" }}{{end}}{{end}}')
export CLUSTER_CA=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}"{{with
index .cluster "certificate-authority-data" }}{{.}}{{end}}"{{ end }}{{ end }}')
export CLUSTER_SERVER=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}
{{ .cluster.server }}{{end}}{{ end }}')
```

6. Confirm these variables have been set correctly:

```
export -p | grep -E 'USER_TOKEN_VALUE|CURRENT_CONTEXT|CURRENT_CLUSTER|
CLUSTER_CA|CLUSTER_SERVER'
```

7. Generate a kubeconfig file that uses the environment variable values from the previous step:

```
cat << EOF > kommander-cluster-admin-config
apiVersion: v1
kind: Config
current-context: ${CURRENT_CONTEXT}
contexts:
- name: ${CURRENT_CONTEXT}
  context:
    cluster: ${CURRENT_CONTEXT}
    user: kommander-cluster-admin
    namespace: kube-system
clusters:
- name: ${CURRENT_CONTEXT}
  cluster:
    certificate-authority-data: ${CLUSTER_CA}
    server: ${CLUSTER_SERVER}
users:
- name: kommander-cluster-admin
  user:
    token: ${USER_TOKEN_VALUE}
EOF
```

8. This process produces a file in your current working directory called `kommander-cluster-admin-config`. The contents of this file are used in Kommander to attach the cluster.
   Before importing this configuration, verify the `kubeconfig` file can access the cluster:

```
kubectl --kubeconfig $(pwd)/kommander-cluster-admin-config get all --all-namespaces
```

5.11.1.9.1.3 Finish Attach EKS Cluster from the UI

Now that you have kubeconfig, go to the DKP UI and follow these steps below:

1. From the top menu bar, select your target workspace.
2. On the Dashboard page, select the **Add Cluster** option in the **Actions** dropdown menu at the top right.
3. Select **Attach Cluster**.
4. Select the **No additional networking restrictions** card. Alternatively, if you must use network restrictions, stop following the steps below, and see the instructions on the page Attach a cluster WITH network restrictions (see page 693).
5. Upload the kubeconfig file you created in the previous section (or copy its contents) into the **Cluster Configuration** section.
6. The **Cluster Name** field automatically populates with the name of the cluster in the kubeconfig. You can edit this field with the name you want for your cluster.
7. Add labels to classify your cluster as needed.

8. Select **Create** to attach your cluster.

> If a cluster has limited resources to deploy all the federated platform services, it will fail to stay attached in the DKP UI. If this happens, ensure your system has sufficient resources for all pods.

5.11.1.9.2 Manually Attach a DKP CLI Cluster to the Management Cluster

1. Find out the `name` of the created `Cluster`, so you can reference it later:

```
kubectl -n <workspace_namespace> get clusters
```

2. Attach the cluster by creating a `KommanderCluster`:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: <cluster_name>
  namespace: <workspace_namespace>
spec:
  kubeconfigRef:
    name: <cluster_name>-kubeconfig
  clusterRef:
    capiCluster:
      name: <cluster_name>
EOF
```

5.11.1.9.3 Related Information

For information on related topics or procedures, refer to the following:

- Create an EKS Cluster from the UI (see page 1022)
- Configuring and Running Amazon EKS Clusters[298]
- Manage Clusters (see page 669)

5.11.1.9.4 Next Step:

Day 2 - Cluster Operations Management (see page 510)

298 https://aws.amazon.com/eks/

5.12 vSphere Install Options

For vSphere, an installation scenario is provided for you in this location. Remember, there are always more options in the Additional Infrastructure Configurations (see page 1132) sections, but this will get you operative in the most common scenarios.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

Below are all the supported environment variable combinations: (Refer to this page to check if your OS is a Supported Operating System (see page 62).)

- vSphere Prerequisites: All Installation Types (see page 379)
- vSphere Install (see page 383)
- vSphere Air-gapped Install (see page 402)
- vSphere FIPS Install (see page 424)
- vSphere FIPS Air-gapped Install (see page 443)

5.12.1 Overview

vSphere is a more complex setup than some of the other providers and infrastructures, so an overview of steps has been provided to help.

The overall process for configuring vSphere and DKP together includes the following steps:

1. Configure vSphere to provide the needed elements, described in the vSphere Prerequisites: All Installation Types (see page 379).
2. **For air-gapped environments:** Create a bastion VM host (see page 1162).
3. Create a base OS image (for use in the OVA package containing the disk images packaged with the OVF).
4. Create a CAPI VM image template that uses the base OS image and adds the needed Kubernetes cluster components.
5. Create a bootstrap cluster.
6. Create a new DKP cluster on vSphere.
7. Make the cluster self-managing.
8. Explore the cluster and perform other functions as needed.
9. Configure MetalLB.
10. Install Kommander
11. Verify and log in to the UI

5.12.2 Next Step:

vSphere Prerequisites: All Installation Types (see page 379)

5.12.3 vSphere Prerequisites: All Installation Types

5.12.3.1 vSphere: Minimum User Permissions

5.12.3.1.1 Create minimum required roles for provisioning and installing in vSphere

When a user needs permissions less than Admin, a role must be created. The process for configuring a vSphere role with the least permissions for provisioning nodes and installing includes the following steps:

1. Open a vSphere Client connection to the vCenter Server, described in the Prerequisites (see page 1133).
2. Select Home > Administration > Roles > Add Role.
3. Give the new role a name, then select these Privileges:

Cns		
	☑	Searchable
Datastore		
	☑	Allocate space
	☑	Low level file operations
Host		
	• Configuration	
	☑	Storage partition configuration
Profile-driven storage		

	☑	Profile-driven storage view
Network		
	☑	Assign network
Resource		
	☑	Assign virtual machine to resource pool
Virtual machine		
	• Change Configuration - from the list in that section, select these permissions below:	
	☑	Add new disk
	☑	Add existing disk
	☑	Add or remove device
	☑	Advanced configuration
	☑	Change CPU count
	☑	Change Memory
	☑	Change Settings
	☑	Reload from path
	Edit inventory	
	☑	Create from existing
	☑	Remove
	Interaction	

	☑	Power off
	☑	Power on
	Provisioning	
	☑	Clone template
	☑	Deploy template
	Session	
	☑	ValidateSession

Add the permission at the highest level and set to propagate the permissions.

5.12.3.1.2 Next Step:

vSphere: Storage Options (see page 381)

5.12.3.2 vSphere: Storage Options

5.12.3.2.1 Explore storage options and considerations for using DKP with VMware vSphere

The vSphere Container Storage[299] plugin supports shared NFS, vNFS, and vSAN. You need to provision your storage options in vCenter prior to creating a CAPI VM Template (see page 385) for non-air-gapped or in air-gapped (see page 407) in DKP for use with vSphere.

DKP has integrated the CSI 2.x driver[300] used in vSphere. When creating your DKP cluster, DKP uses whatever configuration you provide for the Datastore name. vSAN is not required. Using NFS can reduce the amount of tagging and permission granting required to configure your cluster.

5.12.3.2.2 Next Step:

vSphere: VMware Prerequisites (see page 382)

299 https://docs.vmware.com/en/VMware-vSphere-Container-Storage-Plug-in/2.0/vmware-vsphere-csp-getting-started/GUID-74AF02D7-1562-48BD-A9FE-C81A53342AC3.html
300 https://github.com/kubernetes-sigs/vsphere-csi-driver

5.12.3.3 vSphere: VMware Prerequisites

Before installing, verify that your VMware vSphere Client environment[301] meets the following basic requirements:

- Access to a bastion (see page 1162) VM, or other network-connected host, running vSphere Client version v6.7.x with Update 3 or later version.
    - You must be able to reach the vSphere API endpoint from where the Konvoy command line interface (CLI) runs.
- vSphere account with credentials configured - this account must have Administrator privileges (see page 379).
- A RedHat® subscription with username and password for downloading DVD ISOs.
- Valid vSphere values for the following:
    - vCenter API server URL
    - Datacenter name
    - Zone name that contains ESXi hosts[302] for your cluster's nodes.
    - Datastore name for the shared storage resource to be used for the VMs in the cluster.
        - Use of PersistentVolumes in your cluster depends on Cloud Native Storage (CNS), available in vSphere v6.7.x with Update 3 and later versions. CNS depends on this shared Datastore's configuration.
    - Datastore URL from the datastore record for the shared datastore you want your cluster to use.
        - You need this URL value to ensure that the correct Datastore is used when DKP creates VMs for your cluster in vSphere.
    - Folder name
    - Base template name, such as base-rhel-8, or base-rhel-7.
    - Name of a Virtual Network that has DHCP enabled for both air-gapped and non air-gapped environments.
    - Resource Pools - at least one resource pool is needed, with every host in the pool having access to shared storage, such as VSAN.
        - Each host in the resource pool needs access to shared storage, such as NFS or VSAN, to make use of MachineDeployments and high-availability control planes.

5.12.3.3.1 Next Step:

vSphere Install (see page 383)

---

301 https://docs.vmware.com/en/VMware-vSphere/6.7/com.vmware.vsphere.vm_admin.doc/GUID-55238059-912E-411F-A0E9-A7A536972A91.html

302 https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.esxi.install.doc/GUID-B2F01BF5-078A-4C7E-B505-5DFFED0B8C38.html

5.12.4 vSphere Install

This section provides instructions to install DKP in a vSphere non-air-gapped environment.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

5.12.4.1 Further vSphere Prerequisites

- vSphere: VMware Prerequisites (see page 382)
- https://d2iq.atlassian.net/wiki/spaces/DENT/pages/162430985/vSphere+Prerequisites+-+Storage+Options (see page 381)
- https://d2iq.atlassian.net/wiki/spaces/DENT/pages/162430977/vSphere+Prerequisites+-+Minimum+User+Permissions (see page 379)

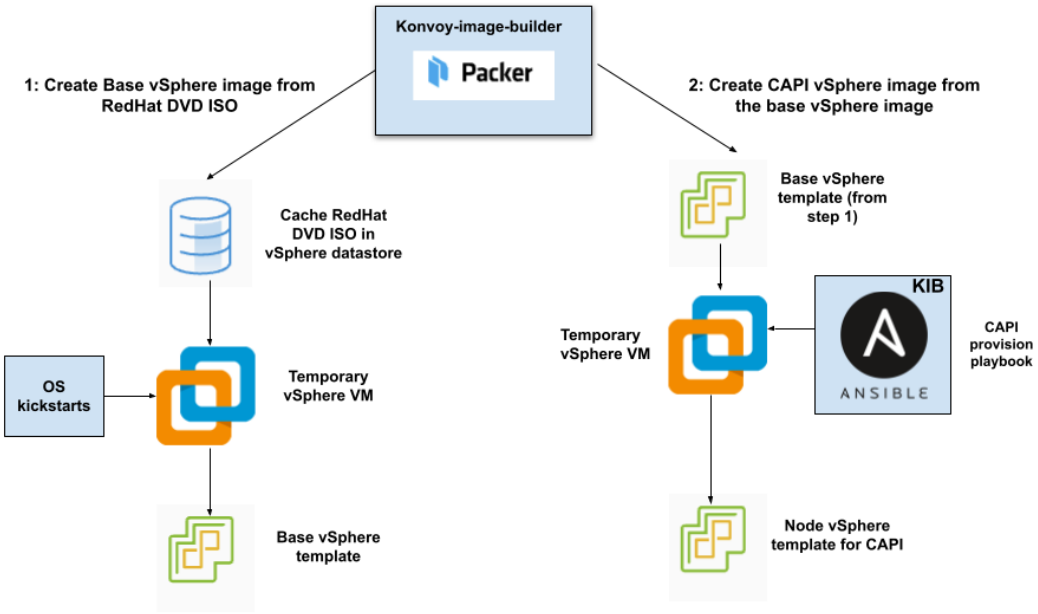
5.12.4.2 Next Step:

vSphere: Image Creation Overview (see page 383)

5.12.4.3 vSphere: Image Creation Overview

5.12.4.3.1 Image Creation Process

This diagram illustrates the image creation process:



The workflow on the left shows the creation of a base OS image in the vCenter vSphere client using inputs from Packer. The workflow on the right shows how DKP uses that same base OS image to create CAPI-enabled VM images for your cluster.

After creating the base image, the DKP image builder uses it to create a CAPI-enabled vSphere template that includes the Kubernetes objects for the cluster. You can use that resulting template with the DKP `create cluster` command to create the VM nodes in your cluster directly on a vCenter server. From that point, you can use DKP to provision and manage your cluster.

5.12.4.3.2 Next Step:

vSphere: Create an Image (see page 384)

5.12.4.4 vSphere: Create an Image

Creating a base OS image from DVD ISO files is a one-time process. The base OS image file is created in the vSphere Client for use in the vSphere VM template. Therefore, the base OS image is used by Konvoy Image Builder (see page 1282)(KIB) to create a VM template to configure Kubernetes nodes by the DKP vSphere provider.

5.12.4.4.1 Create the Base OS Image

For vSphere, a username is populated by `SSH_USERNAME` and the user can use authorization via `SSH_PASSWORD` or `SSH_PRIVATE_KEY_FILE` environment variables and required by default by packer. This user should have administrator privileges. It is possible to configure a custom user and password when building the OS image, however, that requires the Konvoy Image Builder (KIB) configuration to be overridden (see page 1330).

While creating the base OS image, it is important to take into consideration the following elements:

- Storage configuration: D2iQ recommends customizing disk partitions and not configuring a SWAP partition.
- Network configuration: as KIB must download and install packages, activating the network is required.
- Connect to Red Hat: if using RHEL, registering with Red Hat is required to configure software repositories and install software packages.
- Software selection: D2iQ recommends choosing **Minimal Install**.
- DKP recommends to install with the packages provided by the operating system package managers. Use the version that corresponds to the major version of your operating system.

vSphere: Create a CAPI VM Template Image (see page 385)

5.12.4.5 vSphere: Create a CAPI VM Template Image

5.12.4.5.1 Prerequisites

- Users need to vSphere: Create an Image (see page 384) before starting this procedure.
- Build image template with Konvoy Image Builder (KIB)

- If using GPU, prepare an OS image for your NVIDIA GPU nodepool, using the Konvoy Image Builder instructions here: KIB for GPU (see page 1302).

5.12.4.5.2 Create a vSphere Template for Your Cluster from a Base OS Image

Using the base OS image created in a previous procedure, DKP creates the new vSphere template directly on the vCenter server.

1. Set the following vSphere environment variables on the bastion VM host:

```
export VSPHERE_SERVER=your_vCenter_APIserver_URL
export VSPHERE_USERNAME=your_vCenter_user_name
export VSPHERE_PASSWORD=your_vCenter_password
```

2. Copy the base OS image file created in the vSphere Client to your desired location on the bastion VM host, and make a note of the path and file name.
3. Create an `image.yaml` file and add the following variables for vSphere. DKP uses this file and these variables as inputs in the next step.
   NOTE: You will need to replace OS name below based on your OS - EX: “rhel-79” to “rhel-86”. See other YAML examples[303] for copy and paste below last step.

```
---
download_images: true
build_name: "rhel-79"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-
init-vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-rhel-7.9" # change default value with your base template name
  vsphere_guest_os_type: "rhel7_64Guest"
  guest_os_type: "rhel7-64"
  # goss params
  distribution: "RHEL"
  distribution_version: "7.9"
# Use following overrides to select the authentication method that can be used
with base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # if set to true, ssh_password and ssh_private_key
will be ignored
```

303 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

4. Create a vSphere VM template with your variation of the following command:

```
konvoy-image build images/ova/<image.yaml>
```

Any additional configurations can be added to this command using `--overrides` flags as shown below:

   a. for FIPS, add this flag - `--overrides overrides/fips.yaml`
   b. for air-gapped, add this flag `--overrides overrides/offline-fips.yaml`

5. The Konvoy Image Builder (see page 1305) (KIB) uses the values in `image.yaml` and the input base OS image to create a vSphere template directly on the vCenter server. This template contains the required artifacts needed to create a Kubernetes cluster.
   When KIB provisions the OS image[304] successfully, it creates a manifest file. The `artifact_id` field of this file contains the name of the AMI ID (AWS), template name (vSphere), or image name (GCP/Azure), for example:

```
{
      "name": "vsphere-clone",
      "builder_type": "vsphere-clone",
      "build_time": 1644985039,
      "files": null,
      "artifact_id": "konvoy-ova-vsphere-rhel-84-1.21.6-1644983717",
      "packer_run_uuid": "260e8110-77f8-ca94-e29e-ac7a2ae779c8",
      "custom_data": {
        "build_date": "2022-02-16T03:55:17Z",
        "build_name": "vsphere-rhel-84",
        "build_timestamp": "1644983717",
        [...]
      }
    }
```

6. Next steps are to deploy a DKP cluster using your vSphere template.

5.12.4.5.2.1 Additional OS YAML files for copy/paste:

RHEL 8.6

```
---
download_images: true
build_name: "rhel-86"
packer_builder_type: "vsphere"
```

304 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

```
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "os-qualification-templates/d2iq-base-RHEL-86" # change default value
with your base template name
  vsphere_guest_os_type: "rhel8_64Guest"
  guest_os_type: "rhel8-64"
  # goss params
  distribution: "RHEL"
  distribution_version: "8.6"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

Ubuntu 20.04

```
---
download_images: true
build_name: "ubuntu-2004"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "os-qualification-templates/d2iq-base-Ubuntu-20.04" # change default
 value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
```

```
  guest_os_type: "ubuntu2004-64"
  # goss params
  distribution: "ubuntu"
  distribution_version: "20.04"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

Rocky Linux 9.1

```
---
download_images: true
build_name: "rocky-91"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "os-qualification-templates/d2iq-base-RockyLinux-9.1" # change default
 value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "rocky9-64"
  # goss params
  distribution: "rocky"
  distribution_version: "9.1"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

5.12.4.5.3 Next Step:

vSphere: Cluster Creation (see page 390)

5.12.4.6 vSphere: Create the Management Cluster

Use this procedure to create a vSphere Management cluster with DKP.

5.12.4.6.1 Name your Cluster

1. Give your cluster a unique name suitable for your environment.
2. Set the CLUSTER_NAME environment variable with the command:

```
export CLUSTER_NAME=<my-vsphere-cluster>
```

5.12.4.6.2 Create a New vSphere Kubernetes Cluster

DKP uses the vsphere CSI driver as the default storage provider (see page 103). Use a Kubernetes CSI[305] compatible storage that is suitable for production.

Follow these steps to create the cluster:

1. Use the following command to set the environment variables for vSphere:

```
export VSPHERE_SERVER=example.vsphere.url
export VSPHERE_USERNAME=user@example.vsphere.url
export VSPHERE_PASSWORD=example_password
```

2. Generate the Kubernetes cluster objects by copying and editing this command to include the correct values, including the VM template name you assigned in the previous procedure:

305 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

- To increase Dockerhub's rate limit[306] use your Dockerhub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.
- The following example shows a common configuration. See dkp create cluster (see page 1454) reference for the full list of cluster creation options:

```
dkp create cluster vsphere \
  --cluster-name ${CLUSTER_NAME} \
  --network <NETWORK_NAME> \
  --control-plane-endpoint-host <xxx.yyy.zzz.000> \
  --data-center <DATACENTER_NAME> \
  --data-store <DATASTORE_NAME> \
  --folder <FOLDER_NAME> \
  --server <VCENTER_API_SERVER_URL> \
  --ssh-public-key-file <SSH_PUBLIC_KEY_FILE> \
  --resource-pool <RESOURCE_POOL_NAME> \
  --vm-template <TEMPLATE_NAME> \
  --virtual-ip-interface <ip_interface_name> \
  --self-managed
```

A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

5.12.4.6.3 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1360).

306 https://docs.docker.com/docker-hub/download-rate-limit/

5.12.4.6.4 Known Limitations

Be aware of these limitations in the current release of DKP Konvoy.

- The DKP Konvoy version used to create a bootstrap cluster must match the DKP Konvoy version used to create a workload cluster.
- DKP Konvoy supports deploying one workload cluster.
- DKP Konvoy generates a set of objects for one Node Pool.
- DKP Konvoy does not validate edits to cluster objects.

5.12.4.6.5 Next Step:

vSphere: Configure MetalLB (see page 392)

5.12.4.7 vSphere: Configure MetalLB

If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can proceed to vSphere: Install Kommander (see page 395) and continue the installation process.

To use MetalLB, create a MetalLB configMap for your vSphere infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

Create a MetalLB configMap for your vSphere infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP)

Select one of the following procedures to create your MetalLB manifest for further editing and deployment.

5.12.4.7.1 Layer 2 configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to Address Resolution Protocol (ARP) requests on your local network directly, to give the machine's MAC address to clients.

- MetalLB IP address ranges/CIDRs should be within the node's primary network subnet (see page 929).
- MetalLB IP address ranges/CIDRs and node subnet should not conflict with the Kubernetes cluster pod and service subnets.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

1. Create a `metallb-conf.yaml` file for editing with the command:

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

2. Edit the file to contain values specific to your environment, then run the following `kubectl` command:

```
kubectl apply -f metallb-conf.yaml
```

5.12.4.7.2 BGP configuration

For a basic configuration featuring one BGP router and one IP address range, you need four pieces of information:

- The router IP address to which MetalLB should connect
- The router's AS number

- The AS number MetalLB should use
- An IP address range expressed as a CIDR prefix

For example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like the code snippet below.

The following example values are generic, enter your specific values into the fields where applicable.

1. Extract the kubeconfig for your cluster and deploy a configMap for MetalLB using the following command:

```
dkp get kubeconfig -c ${DKP_CLUSTER_NAME} > ${DKP_CLUSTER_NAME}.conf
```

2. Create a `metallb-conf.yaml` file for editing with the command:

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

3. Edit the file to contain values specific to your environment, then run the following `kubectl` command:

```
kubectl apply -f metallb-conf.yaml
```

5.12.4.7.3 Next Step:

vSphere: Install Kommander (see page 395)

5.12.4.8 vSphere: Install Kommander

You have installed the Konvoy component and created a cluster. Now it is time to Install Kommander which will allow you to access the UI and attach new or existing clusters to monitor.

5.12.4.8.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 119).
- Ensure you have a default StorageClass (see page 1254).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

5.12.4.8.2 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1227) for the deployment:

   ```
   dkp install kommander --init > kommander.yaml
   ```

4. *If required:* Customize your `kommander.yaml`.
   See Kommander customizations (see page 1227) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1271)), etc.

5.12.4.8.3 Enable DKP Catalog Applications and Install Kommander

If you have an Enterprise license and would like to use the DKP Catalog Applications function, follow these steps:

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.5.0
```

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.co
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see (Provide Context for Commands with a kubeconfig File (see page 99) .
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

5.12.4.8.4 Next Step:

vSphere: Verify Install and Log in to the UI (see page 397)

5.12.4.9 vSphere: Verify Install and Log in to the UI

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander. After the CLI successfully installs the components, it will wait for all applications to be ready by default.

> **NOTE**: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Released helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Released` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
```

```
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

5.12.4.9.1 Failed HelmReleases

If any application fails to successfully deploy, you can check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a "broken" release state such as "exhausted" or "another rollback/release in progress", you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

5.12.4.9.2 Log in to the UI with Kommander

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 524). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 524):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

5.12.4.9.2.1 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 510) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

5.12.4.9.2.2 Next Steps:

vSphere: Create a Managed Cluster Using the DKP CLI (see page 399)

5.12.4.10 vSphere: Create a Managed Cluster Using the DKP CLI

Creating a vSphere Managed cluster with the DKP CLI assumes that you already fulfilled all of the prerequisites and successfully created a vSphere Management cluster. Use this process and its procedures to create a Managed vSphere cluster.

> ⚠ When creating Managed clusters (see page 97), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 97)!

The procedures to follow for creating a managed vSphere cluster from the DKP CLI include:

5.12.4.10.1 Choose a Workspace for the New Managed Cluster

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a Workspace[307].

307 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919315/Workspaces#Create-a-Workspace

```
kubectl get workspace -A
```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

```
export WORKSPACE_NAMESPACE=<workspace_namespace>
```

5.12.4.10.2 Name your Cluster

Use an environment variable to contain your managed cluster's name to make working with later YAML files and CLI commands easier.

1. Give your cluster a unique name suitable for your environment.
2. Set the CLUSTER_NAME environment variable with the command:

```
export CLUSTER_NAME=<my-managed-vsphere-cluster-name>
```

5.12.4.10.3 Create a New vSphere Kubernetes Cluster

DKP uses local static provisioner as the default storage provider (see page 103). However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[308] compatible storage that is suitable for production.

You can choose from any of the storage options[309] available for Kubernetes. To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your newly created StorageClass to be the default by following the commands in the Kubernetes documentation called Changing the Default Storage Class[310].

Follow these steps to create the cluster:

1. Use the following command to set the environment variables for vSphere:

```
export VSPHERE_SERVER=example.vsphere.url
export VSPHERE_USERNAME=user@example.vsphere.url
export VSPHERE_PASSWORD=example_password
```

308 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
309 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
310 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

2. Ensure your vSphere credentials are up-to-date by refreshing the credentials with the command:

```
dkp update bootstrap credentials vsphere
```

3. Generate the Kubernetes cluster objects by copying and editing this command to include the correct values, including the VM template name you assigned in the previous procedure:

- To increase Dockerhub's rate limit[311] use your Dockerhub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.
- The following example shows a common configuration. See dkp create cluster (see page 1454) reference for the full list of cluster creation options:

The command to create a basic vSphere managed cluster is:

```
dkp create cluster vsphere \
  --cluster-name ${CLUSTER_NAME} \
  --network <NETWORK_NAME> \
  --control-plane-endpoint-host <xxx.yyy.zzz.000> \
  --data-center <DATACENTER_NAME> \
  --data-store <DATASTORE_NAME> \
  --folder <FOLDER_NAME> \
  --server <VCENTER_API_SERVER_URL> \
  --ssh-public-key-file <SSH_PUBLIC_KEY_FILE> \
  --resource-pool <RESOURCE_POOL_NAME> \
  --vm-template <TEMPLATE_NAME> \
  --virtual-ip-interface <ip_interface_name> \
  --namespace ${WORKSPACE_NAMESPACE}
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

311 https://docs.docker.com/docker-hub/download-rate-limit/

5.12.4.10.4 Manually Attach a DKP CLI Cluster to the Management Cluster

1. Find out the `name` of the created `Cluster`, so you can reference it later:

```
kubectl -n <workspace_namespace> get clusters
```

2. Attach the cluster by creating a `KommanderCluster`:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: <cluster_name>
  namespace: <workspace_namespace>
spec:
  kubeconfigRef:
    name: <cluster_name>-kubeconfig
  clusterRef:
    capiCluster:
      name: <cluster_name>
EOF
```

If you have existing clusters or want to create other new, managed clusters to attach, there are many ways to attach a cluster with various requirements and restrictions. To see all the options, visit the section, Day 2 - Attach an Existing Kubernetes Cluster[312].

5.12.4.10.5 Next Steps:

At this point, you can:

- Create more clusters, by following this procedure again
- Perform other configuration tasks (see page 919)
- Proceed to Day 2 Operations[313].

5.12.5 vSphere Air-gapped Install

This section provides instructions to install DKP in a vSphere **air-gapped** environment.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

312 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29920384/Attach+a+Kubernetes+Cluster
313 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29918898/Day+2+-+Cluster+Operations+Management

NOTE: For an air-gapped environment, ensure you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz` , so you can extract the tarball on next page.

5.12.5.1 Further vSphere Prerequisites

- vSphere: VMware Prerequisites (see page 382)
- https://d2iq.atlassian.net/wiki/spaces/DENT/pages/213321127/2.5%2BvSphere%2BStorage%2BOptions (see page 381)
- https://d2iq.atlassian.net/wiki/spaces/DENT/pages/213321150/2.5%2BvSphere%2BMinimum%2BUser%2BPermissions (see page 379)

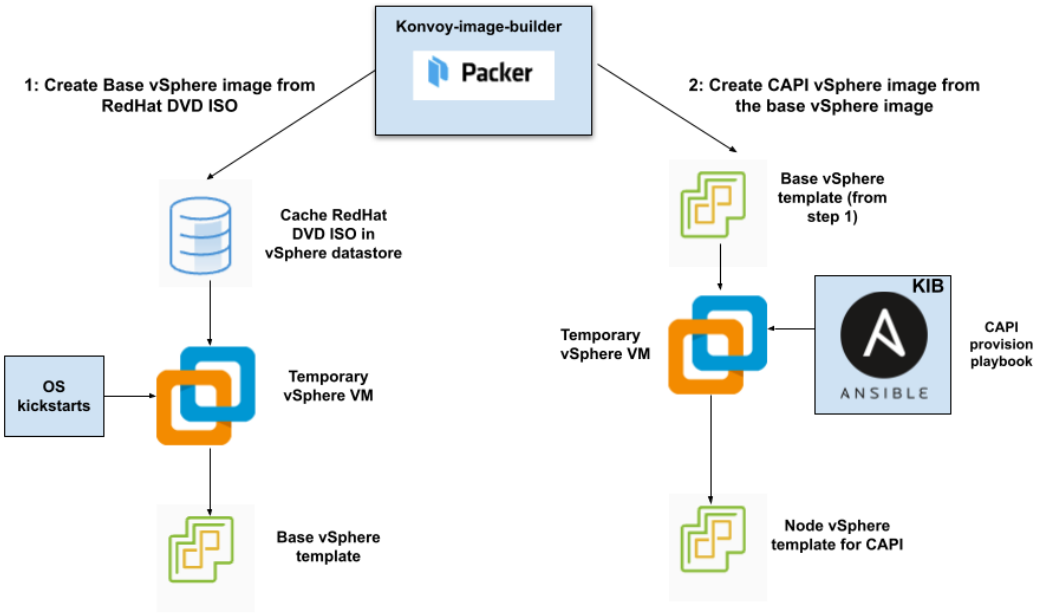
5.12.5.2 Next Step:

vSphere Air-gapped: Image Creation Overview (see page 403)

5.12.5.3 vSphere Air-gapped: Image Creation Overview

5.12.5.3.1 Image Creation Process

This diagram illustrates the image creation process:



The workflow on the left shows the creation of a base OS image in the vCenter vSphere client using inputs from Packer. The workflow on the right shows how DKP uses that same base OS image to create CAPI-enabled VM images for your cluster.

After creating the base image, the DKP image builder uses it to create a CAPI-enabled vSphere template that includes the Kubernetes objects for the cluster. You can use that resulting template with the DKP `create cluster` command to create the VM nodes in your cluster directly on a vCenter server. From that point, you can use DKP to provision and manage your cluster.

5.12.5.3.2 Next Step:

vSphere Air-gapped: Create an Image (see page 404)

5.12.5.4 vSphere Air-gapped: Create an Image

Creating a base OS image from DVD ISO files is a one-time process. The base OS image file is created in the vSphere Client for use in the vSphere VM template. Therefore, the base OS image is used by Konvoy Image Builder (see page 1282)(KIB) to create a VM template to configure Kubernetes nodes by the DKP vSphere provider.

5.12.5.4.1 Create the Base OS Image

For vSphere, a username is populated by `SSH_USERNAME` and the user can use authorization via `SSH_PASSWORD` or `SSH_PRIVATE_KEY_FILE` environment variables and required by default by packer. This user should have administrator privileges. It is possible to configure a custom user and password when building the OS image, however, that requires the Konvoy Image Builder (KIB) configuration to be overridden (see page 1330).

While creating the base OS image, it is important to take into consideration the following elements:

- Storage configuration: D2iQ recommends customizing disk partitions and not configuring a SWAP partition.
- Network configuration: as KIB must download and install packages, activating the network is required.
- Connect to Red Hat: if using RHEL, registering with Red Hat is required to configure software repositories and install software packages.
- Software selection: D2iQ recommends choosing **Minimal Install**.
- DKP recommends to install with the packages provided by the operating system package managers. Use the version that corresponds to the major version of your operating system.

5.12.5.4.2 Next Step:

vSphere Air-gapped: Docker Registry (see page 405)

5.12.5.5 vSphere Air-gapped: Load the Registry

Before creating a Kubernetes cluster, you need the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1162) and either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1349) page for more information.

1. Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz` , extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz && cd dkp-v2.5.0
   ```

2. Set an environment variable with your registry address with this command:

```
export REGISTRY_ADDRESS=<registry-address>:<registry-port>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

3. Execute the following command to load the air-gapped image bundle into your private registry:

```
dkp push image-bundle --image-bundle ./container-images/konvoy-image-bundle-
v2.5.0.tar --to-registry $REGISTRY_ADDRESS --to-registry-username
$REGISTRY_USERNAME --to-registry-password $REGISTRY_PASSWORD
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

5.12.5.5.1 Kommander Load Images

If you are operating in an air-gapped environment, a local container registry (see page 1349) containing all the necessary installation images, including the Kommander images is required. See below for how to push the necessary images to this registry.

For the basic air-gapped `kommander` image bundle, run the command below:

1. Run the following command to load the image bundle:

```
dkp push image-bundle --image-bundle ./container-images/kommander-image-bundle-
v2.5.0.tar --to-registry $REGISTRY_ADDRESS
```

For **DKP Catalog Applications**, also perform this image load:

1. Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push image-bundle --image-bundle ./container-images/dkp-catalog-
applications-image-bundle-v2.5.0.tar --to-registry $REGISTRY_ADDRESS
```

5.12.5.5.2 Next Step:

vSphere Air-gapped: Create a CAPI VM Template Image (see page 407)

5.12.5.6 vSphere Air-gapped: Create a CAPI VM Template Image

You must have at least one image before creating a new cluster. As long as you have an image, this step in your configuration is not required each time since that image can be used to spin up a new cluster. However, if you need different images for different environments or providers, you will need to create a new custom image.

Create a vSphere template for your cluster from a base OS image

Using KIB (see page 1282), you can create your VM image without requiring access to the internet by providing an additional `--override` flag.

1. Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz`, extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz && cd dkp-v2.5.0/kib
   ```

2. Follow the instructions to build a vSphere template below and set the override `--overrides overrides/offline.yaml` flag.

5.12.5.6.1 Create a vSphere Template for Your Cluster from a Base OS Image

Using the base OS image created in a previous procedure, DKP creates the new vSphere template directly on the vCenter server.

1. Set the following vSphere environment variables on the bastion VM host:

   ```
   export VSPHERE_SERVER=your_vCenter_APIserver_URL
   export VSPHERE_USERNAME=your_vCenter_user_name
   export VSPHERE_PASSWORD=your_vCenter_password
   ```

2. Copy the base OS image file created in the vSphere Client to your desired location on the bastion VM host, and make a note of the path and file name.
3. Create an `image.yaml` file and add the following variables for vSphere. DKP uses this file and these variables as inputs in the next step.
   NOTE: You will need to replace OS name below based on your OS - EX: “rhel-79” to “rhel-86”. See other YAML examples[314] for copy and paste below last step.

   ```
   ---
   download_images: true
   ```

314 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

```
build_name: "rhel-79"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-
init-vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-rhel-7.9" # change default value with your base template name
  vsphere_guest_os_type: "rhel7_64Guest"
  guest_os_type: "rhel7-64"
  # goss params
  distribution: "RHEL"
  distribution_version: "7.9"
# Use following overrides to select the authentication method that can be used
with base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # if set to true, ssh_password and ssh_private_key
will be ignored
```

4. Create a vSphere VM template with your variation of the following command:

   ```
   konvoy-image build images/ova/<image.yaml>
   ```

   Any additional configurations can be added to this command using `--overrides` flags as shown below:

   a. for FIPS, add this flag - `--overrides overrides/fips.yaml`
   b. for air-gapped, add this flag `--overrides overrides/offline-fips.yaml`

5. The Konvoy Image Builder (see page 1305) (KIB) uses the values in `image.yaml` and the input base OS image to create a vSphere template directly on the vCenter server. This template contains the required artifacts needed to create a Kubernetes cluster.
   When KIB provisions the OS image[315] successfully, it creates a manifest file. The `artifact_id` field of this file contains the name of the AMI ID (AWS), template name (vSphere), or image name (GCP/Azure), for example:

315 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

```
{
      "name": "vsphere-clone",
      "builder_type": "vsphere-clone",
      "build_time": 1644985039,
      "files": null,
      "artifact_id": "konvoy-ova-vsphere-rhel-84-1.21.6-1644983717",
      "packer_run_uuid": "260e8110-77f8-ca94-e29e-ac7a2ae779c8",
      "custom_data": {
        "build_date": "2022-02-16T03:55:17Z",
        "build_name": "vsphere-rhel-84",
        "build_timestamp": "1644983717",
        [...]
      }
    }
```

6. Next steps are to deploy a DKP cluster using your vSphere template.

5.12.5.6.1.1 Additional OS YAML files for copy/paste:

RHEL 8.6

```
---
download_images: true
build_name: "rhel-86"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-rhel-8.6" # change default value with your base template name
  vsphere_guest_os_type: "rhel8_64Guest"
  guest_os_type: "rhel8-64"
  # goss params
  distribution: "RHEL"
  distribution_version: "8.6"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
```

```
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

Ubuntu 20.04

```
---
download_images: true
build_name: "ubuntu-2004"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-ubuntu-20.04" # change default value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "ubuntu2004-64"
  # goss params
  distribution: "ubuntu"
  distribution_version: "20.04"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

Rocky Linux 9.1

```
---
download_images: true
build_name: "rocky-91"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
```

```
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-rocky-9.1" # change default value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "rocky9-64"
  # goss params
  distribution: "rocky"
  distribution_version: "9.1"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

5.12.5.6.2 Next Step:

vSphere Air-gapped: Cluster Creation (see page 411)

5.12.5.7 vSphere Air-gapped: Create the Management Cluster

5.12.5.7.1 Name your Cluster

1. Give your cluster a unique name suitable for your environment.
2. Set the CLUSTER_NAME environment variable with the command:

```
export CLUSTER_NAME=<my-vsphere-cluster>
```

DKP uses the vsphere CSI driver as the default storage provider (see page 103). Use a Kubernetes CSI[316] compatible storage that is suitable for production.

316 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

5.12.5.7.2 Create a New vSphere Kubernetes Cluster

Use the following steps to create a new, air-gapped vSphere cluster.

1. Configure your cluster to use an existing registry as a mirror when attempting to pull images: **IMPORTANT:** The image must be created by Konvoy Image Builder (see page 1305) in order to use the registry mirror feature.

   ```
   export REGISTRY_URL=<https/http>://<registry-address>:<registry-port>
   export REGISTRY_CA=<path to the CA on the bastion>
   export REGISTRY_USERNAME=<username>
   export REGISTRY_PASSWORD=<password>
   ```

   - `REGISTRY_URL` : the address of an existing registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
   - `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
   - `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
   - `REGISTRY_PASSWORD` : optional if username is not set.

2. Load the container image, using either the `docker` or `podman` command:

   ```
   docker load -i konvoy-bootstrap-image-v2.5.2.tar
   ```

   OR

   ```
   podman load -i konvoy-bootstrap-image-v2.5.2.tar
   ```

3. Create a Kubernetes cluster by copying the following command and substituting the valid values for your environment:

```
dkp create cluster vsphere \
  --cluster-name ${CLUSTER_NAME} \
  --network <NETWORK_NAME> \
  --control-plane-endpoint-host <CONTROL_PLANE_IP> \
  --data-center <DATACENTER_NAME> \
  --data-store <DATASTORE_NAME> \
  --folder <FOLDER_NAME> \
  --server <VCENTER_API_SERVER_URL> \
  --ssh-public-key-file </path/to/key.pub> \
  --resource-pool <RESOURCE_POOL_NAME> \
  --vm-template konvoy-ova-vsphere-os-release-k8s_release-vsphere-timestamp \
```

```
  --virtual-ip-interface <ip_interface_name> \
  --extra-sans "127.0.0.1" \
  --registry-mirror-url=${REGISTRY_URL} \
  --registry-mirror-cacert=${REGISTRY_CA} \
  --registry-mirror-username=${REGISTRY_USERNAME} \
  --registry-mirror-password=${REGISTRY_PASSWORD} \
  --self-managed
```

A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

5.12.5.7.3 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1360).

As they progress, the controllers create Events, which you can also monitor using the command:

```
kubectl get events | grep ${CLUSTER_NAME}
```

For brevity, this example uses `grep`. You can also use separate commands to get Events for specific objects, such as `kubectl get events --field-selector involvedObject.kind="VSphereCluster"` and `kubectl get events --field-selector involvedObject.kind="VSphereMachine"`.

5.12.5.7.4 Next Step:

vSphere Air-gapped: Configure MetalLB (see page 413)

5.12.5.8 vSphere Air-gapped: Configure MetalLB

If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can proceed to vSphere Air-gapped: Install Kommander (see page 416) and continue the installation process.

To use MetalLB, create a MetalLB configMap for your vSphere infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

Create a MetalLB configMap for your vSphere infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP)

Select one of the following procedures to create your MetalLB manifest for further editing and deployment.

5.12.5.8.1 Layer 2 configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to Address Resolution Protocol (ARP) requests on your local network directly, to give the machine's MAC address to clients.

- MetalLB IP address ranges/CIDRs should be within the node's primary network subnet (see page 929).
- MetalLB IP address ranges/CIDRs and node subnet should not conflict with the Kubernetes cluster pod and service subnets.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

1. Create a `metallb-conf.yaml` file for editing with the command:

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
```

```
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

2. Edit the file to contain values specific to your environment, then run the following `kubectl` command:

```
kubectl apply -f metallb-conf.yaml
```

5.12.5.8.2 BGP configuration

For a basic configuration featuring one BGP router and one IP address range, you need four pieces of information:

- The router IP address to which MetalLB should connect
- The router's AS number
- The AS number MetalLB should use
- An IP address range expressed as a CIDR prefix

For example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like the code snippet below.

The following example values are generic, enter your specific values into the fields where applicable.

1. Extract the kubeconfig for your cluster and deploy a configMap for MetalLB using the following command:

```
dkp get kubeconfig -c ${DKP_CLUSTER_NAME} > ${DKP_CLUSTER_NAME}.conf
```

2. Create a `metallb-conf.yaml` file for editing with the command:

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
```

```
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

3. Edit the file to contain values specific to your environment, then run the following `kubectl` command:

```
kubectl apply -f metallb-conf.yaml
```

5.12.5.8.3 Next Step:

vSphere Air-gapped: Install Kommander (see page 416)

5.12.5.9 vSphere Air-gapped: Install Kommander

5.12.5.9.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 119).
- Ensure you have a default StorageClass (see page 1254).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

5.12.5.9.2 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

```
export CLUSTER_NAME=<your-management-cluster-name>
```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

```
dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
```

3. Create a configuration file (see page 1227) for the deployment:

```
dkp install kommander --init --airgapped > kommander.yaml
```

4. *If required:* Customize your `kommander.yaml`.
   See Kommander Additional Install Configurations (see page 1227) for customization options. Some of them include:
   Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1271)), etc.

5.12.5.9.3 Enable DKP Catalog Applications and Install Kommander

If you have an Enterprise license and would like to use the DKP Catalog Applications function, follow these steps:

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.5.0
```

   ⚠ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.5.0.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.5.0.tar.gz
\
```

```
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.5.0.tar.gz
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 99) .
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

5.12.5.9.4 Next Step:

vSphere Air-gapped: Verify Install and Log in to the UI (see page 418)

5.12.5.10 vSphere Air-gapped: Verify Install and Log in to the UI

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander. After the CLI successfully installs the components, it will wait for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Released helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Released` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

5.12.5.10.1 Failed HelmReleases

If any application fails to successfully deploy, you can check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
```

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

5.12.5.10.2 Log in to the UI with Kommander

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 524). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 524):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

5.12.5.10.2.1 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 510) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

5.12.5.10.2.2 Next Steps:

vSphere Air-gapped: Create a Managed Cluster Using the DKP CLI (see page 421)

5.12.5.11 vSphere Air-gapped: Create a Managed Cluster Using the DKP CLI

Creating an air-gapped vSphere Managed cluster with the DKP CLI assumes that you already fulfilled all of the prerequisites and successfully created a vSphere Management cluster. Use this procedure to create a Managed vSphere cluster.

When creating Managed clusters (see page 97), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 97)!

5.12.5.11.1 Choose a Workspace for the New Cluster

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a Workspace[317].

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

5.12.5.11.2 Name your Cluster

1. Give your cluster a unique name suitable for your environment.
2. Set the CLUSTER_NAME environment variable with the command:

   ```
   export CLUSTER_NAME=<my-vsphere-cluster>
   ```

317 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919315/Workspaces#Create-a-Workspace

DKP uses local static provisioner as the default storage provider (see page 103). However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[318] compatible storage that is suitable for production.

- You can choose from any of the storage options[319] available for Kubernetes. To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your newly created StorageClass to be the default by following the commands in the Kubernetes documentation called Changing the Default Storage Class[320].

5.12.5.11.3 Create a New vSphere Kubernetes Cluster

Use the following steps to create a new, air-gapped vSphere cluster.

1. Configure your cluster to use an existing registry as a mirror when attempting to pull images: **IMPORTANT:** The image must be created by Konvoy Image Builder (see page 1282) so that it uses the registry mirror feature.

   ```
   export REGISTRY_URL=<https/http>://<registry-address>:<registry-port>
   export REGISTRY_CA=<path to the CA on the bastion>
   export REGISTRY_USERNAME=<username>
   export REGISTRY_PASSWORD=<password>
   ```

   - `REGISTRY_URL` : the address of an existing registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
   - `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
   - `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
   - `REGISTRY_PASSWORD` : optional if username is not set.

2. Create a Kubernetes cluster by copying the following command and substituting the valid values for your environment:

```
dkp create cluster vsphere \
  --cluster-name ${CLUSTER_NAME} \
  --network <NETWORK_NAME> \
  --control-plane-endpoint-host <CONTROL_PLANE_IP> \
  --data-center <DATACENTER_NAME> \
```

318 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
319 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
320 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

```
--data-store <DATASTORE_NAME> \
--folder <FOLDER_NAME> \
--server <VCENTER_API_SERVER_URL> \
--ssh-public-key-file </path/to/key.pub> \
--resource-pool <RESOURCE_POOL_NAME> \
--vm-template konvoy-ova-vsphere-os-release-k8s_release-vsphere-timestamp \
--virtual-ip-interface <ip_interface_name> \
--extra-sans "127.0.0.1" \
--registry-mirror-url=${REGISTRY_URL} \
--registry-mirror-cacert=${REGISTRY_CA} \
--registry-mirror-username=${REGISTRY_USERNAME} \
--registry-mirror-password=${REGISTRY_PASSWORD} \
--namespace=${WORKSPACE_NAMESPACE}
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

5.12.5.11.4 Manually Attach a DKP CLI Cluster to the Management Cluster

1. Find out the `name` of the created `Cluster`, so you can reference it later:

   ```
   kubectl -n <workspace_namespace> get clusters
   ```

2. Attach the cluster by creating a `KommanderCluster`:

   ```
   cat << EOF | kubectl apply -f -
   apiVersion: kommander.mesosphere.io/v1beta1
   kind: KommanderCluster
   metadata:
     name: <cluster_name>
     namespace: <workspace_namespace>
   spec:
     kubeconfigRef:
       name: <cluster_name>-kubeconfig
     clusterRef:
       capiCluster:
         name: <cluster_name>
   EOF
   ```

If you have existing clusters or want to create other new clusters to attach, there are many ways to attach a cluster with various requirements and restrictions. To see all the options, visit the section in documentation Day 2 - Attach an Existing Kubernetes Cluster (see page 673).

At this point, you can create more clusters, perform other configuration tasks (see page 919), or proceed to Day 2 Operations (see page 510).

5.12.6 vSphere FIPS Install

This section provides instructions to install DKP in a vSphere with FIPS.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

5.12.6.1 Further vSphere Prerequisites

- vSphere: VMware Prerequisites (see page 382)
- https://d2iq.atlassian.net/wiki/spaces/DENT/pages/162430985/vSphere+Prerequisites+-+Storage+Options (see page 381)
- https://d2iq.atlassian.net/wiki/spaces/DENT/pages/162430977/vSphere+Prerequisites+-+Minimum+User+Permissions (see page 379)

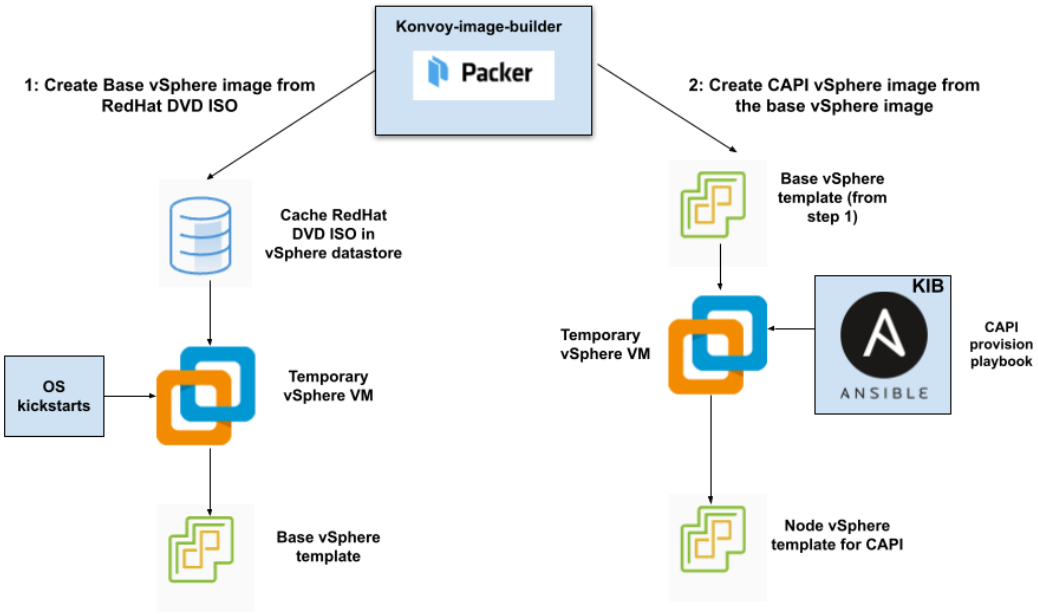
5.12.6.2 Next Step:

vSphere FIPS: Image Creation Overview (see page 424)

5.12.6.3 vSphere FIPS: Image Creation Overview

5.12.6.3.1 Image Creation Process

This diagram illustrates the image creation process:



The workflow on the left shows the creation of a base OS image in the vCenter vSphere client using inputs from Packer. The workflow on the right shows how DKP uses that same base OS image to create CAPI-enabled VM images for your cluster.

After creating the base image, the DKP image builder uses it to create a CAPI-enabled vSphere template that includes the Kubernetes objects for the cluster. You can use that resulting template with the DKP `create cluster` command to create the VM nodes in your cluster directly on a vCenter server. From that point, you can use DKP to provision and manage your cluster.

5.12.6.3.2 Next Step:

vSphere FIPS: Create an Image (see page 425)

5.12.6.4 vSphere FIPS: Create an Image

Creating a base OS image from DVD ISO files is a one-time process. The base OS image file is created in the vSphere Client for use in the vSphere VM template. Therefore, the base OS image is used by Konvoy Image Builder (see page 1282)(KIB) to create a VM template to configure Kubernetes nodes by the DKP vSphere provider.

5.12.6.4.1 Create the Base OS Image

For vSphere, a username is populated by `SSH_USERNAME` and the user can use authorization via `SSH_PASSWORD` or `SSH_PRIVATE_KEY_FILE` environment variables and required by default by packer. This user should have administrator privileges. It is possible to configure a custom user and password when building the OS image, however, that requires the Konvoy Image Builder (KIB) configuration to be overridden (see page 1330).

While creating the base OS image, it is important to take into consideration the following elements:

- Storage configuration: D2iQ recommends customizing disk partitions and not configuring a SWAP partition.
- Network configuration: as KIB must download and install packages, activating the network is required.
- Connect to Red Hat: if using RHEL, registering with Red Hat is required to configure software repositories and install software packages.
- Software selection: D2iQ recommends choosing **Minimal Install**.
- DKP recommends to install with the packages provided by the operating system package managers. Use the version that corresponds to the major version of your operating system.

5.12.6.4.2 Next Step:

The next step is to vSphere FIPS: Create a CAPI VM Template Image (see page 426). The image contains the CAPI and Kubernetes objects.

5.12.6.5 vSphere FIPS: Create a CAPI VM Template Image

5.12.6.5.1 Prerequisites

- Users need to create a vSphere FIPS: Create an Image (see page 425) before starting this procedure.
- Build image with Konvoy Image Builder (KIB)
- In step 4 of the following section, ensure you use `--overrides overrides/fips.yaml`

5.12.6.5.2 Create a vSphere Template for Your Cluster from a Base OS Image

Using the base OS image created in a previous procedure, DKP creates the new vSphere template directly on the vCenter server.

1. Set the following vSphere environment variables on the bastion VM host:

```
export VSPHERE_SERVER=your_vCenter_APIserver_URL
export VSPHERE_USERNAME=your_vCenter_user_name
export VSPHERE_PASSWORD=your_vCenter_password
```

2. Copy the base OS image file created in the vSphere Client to your desired location on the bastion VM host, and make a note of the path and file name.

3. Create an `image.yaml` file and add the following variables for vSphere. DKP uses this file and these variables as inputs in the next step.
   NOTE: You will need to replace OS name below based on your OS - EX: “rhel-79” to “rhel-86”. See other YAML examples[321] for copy and paste below last step.

```
---
download_images: true
build_name: "rhel-79"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-
init-vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-rhel-7.9" # change default value with your base template name
  vsphere_guest_os_type: "rhel7_64Guest"
  guest_os_type: "rhel7-64"
  # goss params
  distribution: "RHEL"
  distribution_version: "7.9"
# Use following overrides to select the authentication method that can be used
with base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # if set to true, ssh_password and ssh_private_key
will be ignored
```

4. Create a vSphere VM template with your variation of the following command:

```
konvoy-image build images/ova/<image.yaml>
```

321 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

Any additional configurations can be added to this command using `--overrides` flags as shown below:

a. for FIPS, add this flag - `--overrides overrides/fips.yaml`
b. for air-gapped, add this flag `--overrides overrides/offline-fips.yaml`

5. The Konvoy Image Builder (see page 1305) (KIB) uses the values in `image.yaml` and the input base OS image to create a vSphere template directly on the vCenter server. This template contains the required artifacts needed to create a Kubernetes cluster.
   When KIB provisions the OS image[322] successfully, it creates a manifest file. The `artifact_id` field of this file contains the name of the AMI ID (AWS), template name (vSphere), or image name (GCP/Azure), for example:

```
{
      "name": "vsphere-clone",
      "builder_type": "vsphere-clone",
      "build_time": 1644985039,
      "files": null,
      "artifact_id": "konvoy-ova-vsphere-rhel-84-1.21.6-1644983717",
      "packer_run_uuid": "260e8110-77f8-ca94-e29e-ac7a2ae779c8",
      "custom_data": {
        "build_date": "2022-02-16T03:55:17Z",
        "build_name": "vsphere-rhel-84",
        "build_timestamp": "1644983717",
        [...]
      }
    }
```

6. Next steps are to deploy a DKP cluster using your vSphere template.

5.12.6.5.2.1 Additional OS YAML files for copy/paste:

RHEL 8.6

```
---
download_images: true
build_name: "rhel-86"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
```

322 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

```
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-rhel-8.6" # change default value with your base template name
  vsphere_guest_os_type: "rhel8_64Guest"
  guest_os_type: "rhel8-64"
  # goss params
  distribution: "RHEL"
  distribution_version: "8.6"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

Ubuntu 20.04

```
---
download_images: true
build_name: "ubuntu-2004"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-ubuntu-20.04" # change default value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "ubuntu2004-64"
  # goss params
  distribution: "ubuntu"
  distribution_version: "20.04"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
```

```
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

Rocky Linux 9.1

```
---
download_images: true
build_name: "rocky-91"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-rocky-9.1" # change default value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "rocky9-64"
  # goss params
  distribution: "rocky"
  distribution_version: "9.1"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

5.12.6.5.3 Next Step:

vSphere FIPS: Cluster Creation (see page 430)

5.12.6.6 vSphere FIPS: Create the Management Cluster

Additional considerations exist when using FIPS. Alter the Create a Kubernetes cluster command by using the FIPS example, or similar, when executing the command to create a cluster.

5.12.6.6.1 Deploying a Cluster in FIPS mode

In order to create a cluster in FIPS mode, we must inform the bootstrap controllers of the appropriate image repository and version tags of the official D2iQ FIPS builds of Kubernetes.

5.12.6.6.1.1 Supported FIPS Builds

Component	Repository	Version
Kubernetes	docker.io/mesosphere[323]	v1.25.4+fips.0
etcd	docker.io/mesosphere[324]	3.5.5+fips.0

5.12.6.6.1.2 Name your Cluster

1. Give your cluster a unique name suitable for your environment.
2. Set the CLUSTER_NAME environment variable with the command:

```
export CLUSTER_NAME=<my-vsphere-cluster>
```

5.12.6.6.1.3 Create a New vSphere Kubernetes Cluster

> ⚠ DKP uses the vsphere CSI driver as the default storage provider (see page 103). Use a Kubernetes CSI[325] compatible storage that is suitable for production.

Follow these steps to create the cluster:

1. Use the following command to set the environment variables for vSphere:

```
export VSPHERE_SERVER=example.vsphere.url
export VSPHERE_USERNAME=user@example.vsphere.url
```

323 https://hub.docker.com/layers/mesosphere/kube-apiserver/v1.25.4_fips.0/images/sha256-cacb721f96ec8764d187ad3c21557630f5f4d96fea4a03ca35d0388b509d970d?context=explore
324 https://hub.docker.com/layers/mesosphere/etcd/v3.5.5_fips.0/images/sha256-ba07521fa1875efa0cc623533eb5557e40a6f8fdc8a71a86751a649ce53de1d0?context=explore
325 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

```
export VSPHERE_PASSWORD=example_password
```

2. Generate the Kubernetes cluster objects by copying and editing this command to include the correct values, including the VM template name you assigned in the previous procedure:

- To increase Dockerhub's rate limit[326] use your Dockerhub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.
- The following example shows a common configuration. See dkp create cluster (see page 1454) reference for the full list of cluster creation options:

```
dkp create cluster vsphere \
  --cluster-name ${CLUSTER_NAME} \
  --network <NETWORK_NAME> \
  --control-plane-endpoint-host <xxx.yyy.zzz.000> \
  --data-center <DATACENTER_NAME> \
  --data-store <DATASTORE_NAME> \
  --folder <FOLDER_NAME> \
  --server <VCENTER_API_SERVER_URL> \
  --ssh-public-key-file <SSH_PUBLIC_KEY_FILE> \
  --resource-pool <RESOURCE_POOL_NAME> \
  --vm-template <TEMPLATE_NAME>
  --virtual-ip-interface <ip_interface_name> \
  --self-managed
  --kubernetes-version=v1.25.4+fips.0 \
  --kubernetes-image-repository=docker.io/mesosphere \
  --etcd-image-repository=docker.io/mesosphere --etcd-version=3.5.5+fips.0
```

A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

326 https://docs.docker.com/docker-hub/download-rate-limit/

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

5.12.6.6.1.4 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1360).

5.12.6.6.1.5 Next Step:

vSphere FIPS: Configure MetalLB (see page 433)

5.12.6.7 vSphere FIPS: Configure MetalLB

If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can proceed to vSphere FIPS: Install Kommander (see page 436) and continue the installation process.

To use MetalLB, create a MetalLB configMap for your vSphere infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

Create a MetalLB configMap for your vSphere infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP)

Select one of the following procedures to create your MetalLB manifest for further editing and deployment.

5.12.6.7.1 Layer 2 configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to Address Resolution Protocol (ARP) requests on your local network directly, to give the machine's MAC address to clients.

- MetalLB IP address ranges/CIDRs should be within the node's primary network subnet (see page 929).
- MetalLB IP address ranges/CIDRs and node subnet should not conflict with the Kubernetes cluster pod and service subnets.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

1. Create a `metallb-conf.yaml` file for editing with the command:

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

2. Edit the file to contain values specific to your environment, then run the following `kubectl` command:

```
kubectl apply -f metallb-conf.yaml
```

5.12.6.7.2 BGP configuration

For a basic configuration featuring one BGP router and one IP address range, you need four pieces of information:

- The router IP address to which MetalLB should connect
- The router's AS number

- The AS number MetalLB should use
- An IP address range expressed as a CIDR prefix

For example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like the code snippet below.

The following example values are generic, enter your specific values into the fields where applicable.

1. Extract the kubeconfig for your cluster and deploy a configMap for MetalLB using the following command:

```
dkp get kubeconfig -c ${DKP_CLUSTER_NAME} > ${DKP_CLUSTER_NAME}.conf
```

2. Create a `metallb-conf.yaml` file for editing with the command:

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

3. Edit the file to contain values specific to your environment, then run the following `kubectl` command:

```
kubectl apply -f metallb-conf.yaml
```

5.12.6.7.3 Next Step:

vSphere FIPS: Install Kommander (see page 436)

5.12.6.8 vSphere FIPS: Install Kommander

You have installed the Konvoy component and created a cluster. Now it is time to Install Kommander which will allow you to access the UI and attach new or existing clusters to monitor.

5.12.6.8.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 119).
- Ensure you have a default StorageClass (see page 1254).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

5.12.6.8.2 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1227) for the deployment:

   ```
   dkp install kommander --init > kommander.yaml
   ```

4. *If required:* Customize your `kommander.yaml`.
   See Kommander customizations (see page 1227) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1271)), etc.

5.12.6.8.3 Enable DKP Catalog Applications and Install Kommander

If you have an Enterprise license and would like to use the DKP Catalog Applications function, follow these steps:

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.5.0
```

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.co
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 99).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

5.12.6.8.4 Next Step:

vSphere FIPS: Verify Install and Log in to the UI (see page 438)

5.12.6.9 vSphere FIPS: Verify Install and Log in to the UI

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander. After the CLI successfully installs the components, it will wait for all applications to be ready by default.

> **NOTE**: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Released helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Released` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
```

```
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

5.12.6.9.1 Failed HelmReleases

If any application fails to successfully deploy, you can check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

5.12.6.9.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 524). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 524):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

5.12.6.9.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 510) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

5.12.6.9.4 Next Step:

vSphere FIPS: Create a Managed Cluster Using the DKP CLI (see page 440)

5.12.6.10 vSphere FIPS: Create a Managed Cluster Using the DKP CLI

Creating a vSphere FIPS Managed cluster with the DKP CLI assumes that you already fulfilled all of the prerequisites and successfully created a vSphere Management cluster. Use this procedure to create a Managed vSphere cluster.

When creating Managed clusters (see page 97), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 97)!

5.12.6.10.1 Choose a Workspace for the New Cluster

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a Workspace[327].

327 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919315/Workspaces#Create-a-Workspace

```
kubectl get workspace -A
```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

```
export WORKSPACE_NAMESPACE=<workspace_namespace>
```

5.12.6.10.2 Name your Cluster

1. Give your cluster a unique name suitable for your environment.
2. Set the CLUSTER_NAME environment variable with the command:

```
export CLUSTER_NAME=<my-vsphere-cluster>
```

5.12.6.10.3 Create a New vSphere Kubernetes Cluster

DKP uses the vsphere CSI driver as the default storage provider (see page 103). Use a Kubernetes CSI[328] compatible storage that is suitable for production.

Follow these steps to create the cluster:

1. Use the following command to set the environment variables for vSphere:

```
export VSPHERE_SERVER=example.vsphere.url
export VSPHERE_USERNAME=user@example.vsphere.url
export VSPHERE_PASSWORD=example_password
```

2. Generate the Kubernetes cluster objects by copying and editing this command to include the correct values, including the VM template name you assigned in the previous procedure:

328 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

- To increase Dockerhub's rate limit[329] use your Dockerhub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.
- The following example shows a common configuration. See dkp create cluster (see page 1454) reference for the full list of cluster creation options:

```
dkp create cluster vsphere \
  --cluster-name ${CLUSTER_NAME} \
  --network <NETWORK_NAME> \
  --control-plane-endpoint-host <xxx.yyy.zzz.000> \
  --data-center <DATACENTER_NAME> \
  --data-store <DATASTORE_NAME> \
  --folder <FOLDER_NAME> \
  --server <VCENTER_API_SERVER_URL> \
  --ssh-public-key-file <SSH_PUBLIC_KEY_FILE> \
  --resource-pool <RESOURCE_POOL_NAME> \
  --vm-template <TEMPLATE_NAME>
  --virtual-ip-interface <ip_interface_name> \
  --kubernetes-version=v1.25.4+fips.0 \
  --kubernetes-image-repository=docker.io/mesosphere \
  --etcd-image-repository=docker.io/mesosphere \
  --etcd-version=3.5.5+fips.0 \
  --namespace=${WORKSPACE_NAMESPACE}
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

5.12.6.10.4 Manually Attach a DKP CLI Cluster to the Management Cluster

1. Find out the `name` of the created `Cluster`, so you can reference it later:

   ```
   kubectl -n <workspace_namespace> get clusters
   ```

2. Attach the cluster by creating a `KommanderCluster`:

329 https://docs.docker.com/docker-hub/download-rate-limit/

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: <cluster_name>
  namespace: <workspace_namespace>
spec:
  kubeconfigRef:
    name: <cluster_name>-kubeconfig
  clusterRef:
    capiCluster:
      name: <cluster_name>
EOF
```

If you have existing clusters or want to create other new clusters to attach, there are many ways to attach a cluster with various requirements and restrictions. To see all the options, visit the section in documentation Day 2 - Attach an Existing Kubernetes Cluster (see page 673).

At this point, you can create more clusters, perform other configuration tasks (see page 919), or proceed to Day 2 Operations (see page 510).

5.12.7 vSphere FIPS Air-gapped Install

This section provides instructions to install DKP in a vSphere with FIPS in an **air-gapped** environment.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

NOTE: Foran air-gapped environment, ensure you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz` , so you can extract the tarball on next page.

5.12.7.1 Further vSphere Prerequisites

- vSphere: VMware Prerequisites (see page 382)
- https://d2iq.atlassian.net/wiki/spaces/DENT/pages/213321127/2.5%2BvSphere%2BStorage%2BOptions (see page 381)
- https://d2iq.atlassian.net/wiki/spaces/DENT/pages/213321150/2.5%2BvSphere%2BMinimum%2BUser%2BPermissions (see page 379)

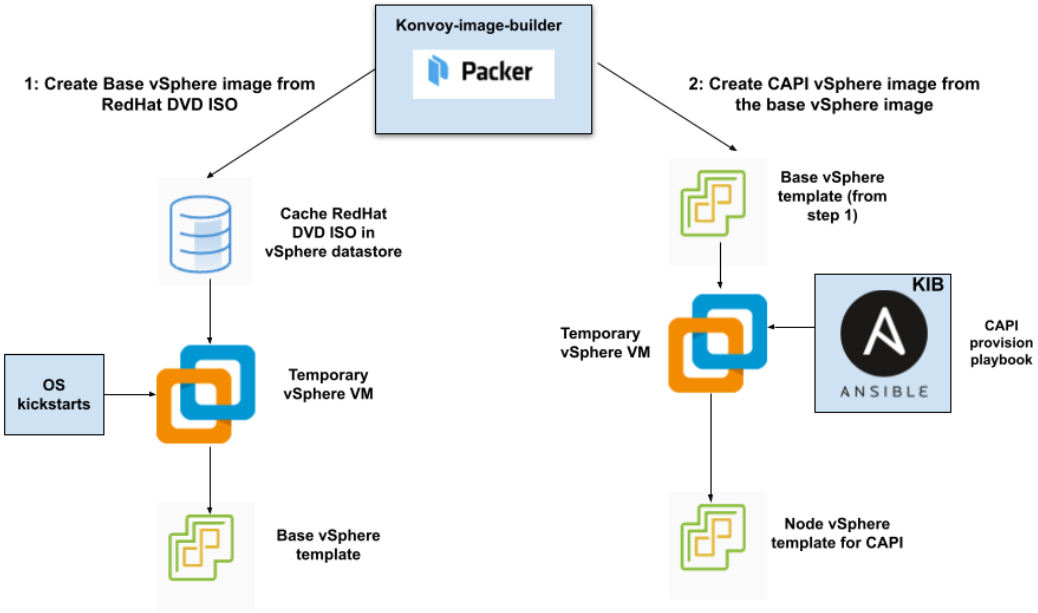
5.12.7.2 Next Step:

vSphere FIPS Air-gapped: Image Creation Overview (see page 444)

5.12.7.3 vSphere FIPS Air-gapped: Image Creation Overview

5.12.7.3.1 Image Creation Process

This diagram illustrates the image creation process:



The workflow on the left shows the creation of a base OS image in the vCenter vSphere client using inputs from Packer. The workflow on the right shows how DKP uses that same base OS image to create CAPI-enabled VM images for your cluster.

After creating the base image, the DKP image builder uses it to create a CAPI-enabled vSphere template that includes the Kubernetes objects for the cluster. You can use that resulting template with the DKP `create cluster` command to create the VM nodes in your cluster directly on a vCenter server. From that point, you can use DKP to provision and manage your cluster.

5.12.7.3.2 Next Step:

vSphere FIPS Air-gapped: Create an Image (see page 445)

5.12.7.4 vSphere FIPS Air-gapped: Create an Image

Creating a base OS image from DVD ISO files is a one-time process. The base OS image file is created in the vSphere Client for use in the vSphere VM template. Therefore, the base OS image is used by Konvoy Image Builder (see page 1282)(KIB) to create a VM template to configure Kubernetes nodes by the DKP vSphere provider.

5.12.7.4.1 Create the Base OS Image

For vSphere, a username is populated by `SSH_USERNAME` and the user can use authorization via `SSH_PASSWORD` or `SSH_PRIVATE_KEY_FILE` environment variables and required by default by packer. This user should have administrator privileges. It is possible to configure a custom user and password when building the OS image, however, that requires the Konvoy Image Builder (KIB) configuration to be overridden (see page 1330).

While creating the base OS image, it is important to take into consideration the following elements:

- Storage configuration: D2iQ recommends customizing disk partitions and not configuring a SWAP partition.
- Network configuration: as KIB must download and install packages, activating the network is required.
- Connect to Red Hat: if using RHEL, registering with Red Hat is required to configure software repositories and install software packages.
- Software selection: D2iQ recommends choosing **Minimal Install**.
- DKP recommends to install with the packages provided by the operating system package managers. Use the version that corresponds to the major version of your operating system.

5.12.7.4.2 Next Step:

vSphere FIPS Air-gapped: Docker Registry (see page 445)

5.12.7.5 vSphere FIPS Air-gapped: Load the Registry

Before creating a Kubernetes cluster, you need the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1162) and either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1349) page for more information.

1. Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz` , extract the tarball to a local directory:

```
tar -xzvf dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz && cd dkp-v2.5.0
```

2. Set an environment variable with your registry address with this command:

```
export REGISTRY_ADDRESS=<registry-address>:<registry-port>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

3. Execute the following command to load the air-gapped image bundle into your private registry:

```
dkp push image-bundle --image-bundle ./container-images/konvoy-image-bundle-
v2.5.0.tar --to-registry $REGISTRY_ADDRESS --to-registry-username
$REGISTRY_USERNAME --to-registry-password $REGISTRY_PASSWORD
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

5.12.7.5.1 Kommander Load Images

If you are operating in an air-gapped environment, a local container registry (see page 1349) containing all the necessary installation images, including the Kommander images is required. See below for how to push the necessary images to this registry.

For the basic air-gapped `kommander` image bundle, run the command below:

1. Run the following command to load the image bundle:

```
dkp push image-bundle --image-bundle ./container-images/kommander-image-bundle-
v2.5.0.tar --to-registry $REGISTRY_ADDRESS
```

For **DKP Catalog Applications**, also perform this image load:

1. Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push image-bundle --image-bundle ./container-images/dkp-catalog-
applications-image-bundle-v2.5.0.tar --to-registry $REGISTRY_ADDRESS
```

5.12.7.5.2 Next Step:

vSphere FIPS Air-gapped: Create a CAPI VM Template Image (see page 447)

5.12.7.6 vSphere FIPS Air-gapped: Create a CAPI VM Template Image

You must have at least one image before creating a new cluster. As long as you have an image, this step in your configuration is not required each time since that image can be used to spin up a new cluster. However, if you need different images for different environments or providers, you will need to create a new custom image.

Create a vSphere template for your cluster from a base OS image

Using KIB (see page 1282), you can create your VM image without requiring access to the internet by providing an additional `--override` flag.

1. Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz`, extract the tarball to a local directory:

```
tar -xzvf dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz && cd dkp-v2.5.0/kib
```

2. Follow the instructions to build a vSphere template below and set the override `--overrides overrides/offline.yaml` flag.

5.12.7.6.1 Create a vSphere Template for Your Cluster from a Base OS Image

Using the base OS image created in a previous procedure, DKP creates the new vSphere template directly on the vCenter server.

1. Set the following vSphere environment variables on the bastion VM host:

```
export VSPHERE_SERVER=your_vCenter_APIserver_URL
export VSPHERE_USERNAME=your_vCenter_user_name
export VSPHERE_PASSWORD=your_vCenter_password
```

2. Copy the base OS image file created in the vSphere Client to your desired location on the bastion VM host, and make a note of the path and file name.

3. Create an `image.yaml` file and add the following variables for vSphere. DKP uses this file and these variables as inputs in the next step.
   NOTE: You will need to replace OS name below based on your OS - EX: “rhel-79” to “rhel-86”. See other YAML examples[330] for copy and paste below last step.

```
---
download_images: true
build_name: "rhel-79"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-
init-vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-rhel-7.9" # change default value with your base template name
  vsphere_guest_os_type: "rhel7_64Guest"
  guest_os_type: "rhel7-64"
  # goss params
  distribution: "RHEL"
  distribution_version: "7.9"
# Use following overrides to select the authentication method that can be used
with base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # if set to true, ssh_password and ssh_private_key
will be ignored
```

4. Create a vSphere VM template with your variation of the following command:

```
konvoy-image build images/ova/<image.yaml>
```

330 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

Any additional configurations can be added to this command using `--overrides` flags as shown below:

a. for FIPS, add this flag - `--overrides overrides/fips.yaml`
b. for air-gapped, add this flag `--overrides overrides/offline-fips.yaml`

5. The Konvoy Image Builder (see page 1305) (KIB) uses the values in `image.yaml` and the input base OS image to create a vSphere template directly on the vCenter server. This template contains the required artifacts needed to create a Kubernetes cluster.
   When KIB provisions the OS image[331] successfully, it creates a manifest file. The `artifact_id` field of this file contains the name of the AMI ID (AWS), template name (vSphere), or image name (GCP/Azure), for example:

```
{
      "name": "vsphere-clone",
      "builder_type": "vsphere-clone",
      "build_time": 1644985039,
      "files": null,
      "artifact_id": "konvoy-ova-vsphere-rhel-84-1.21.6-1644983717",
      "packer_run_uuid": "260e8110-77f8-ca94-e29e-ac7a2ae779c8",
      "custom_data": {
        "build_date": "2022-02-16T03:55:17Z",
        "build_name": "vsphere-rhel-84",
        "build_timestamp": "1644983717",
        [...]
      }
    }
```

6. Next steps are to deploy a DKP cluster using your vSphere template.

5.12.7.6.1.1 Additional OS YAML files for copy/paste:

RHEL 8.6

```
---
download_images: true
build_name: "rhel-86"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
```

331 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

```
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-rhel-8.6" # change default value with your base template name
  vsphere_guest_os_type: "rhel8_64Guest"
  guest_os_type: "rhel8-64"
  # goss params
  distribution: "RHEL"
  distribution_version: "8.6"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

Ubuntu 20.04

```
---
download_images: true
build_name: "ubuntu-2004"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-ubuntu-20.04" # change default value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "ubuntu2004-64"
  # goss params
  distribution: "ubuntu"
  distribution_version: "20.04"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
```

```
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

Rocky Linux 9.1

```
---
download_images: true
build_name: "rocky-91"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-rocky-9.1" # change default value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "rocky9-64"
  # goss params
  distribution: "rocky"
  distribution_version: "9.1"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

5.12.7.6.2 Next Step:

vSphere FIPS Air-gapped: Cluster Creation (see page 452)

5.12.7.7 vSphere FIPS Air-gapped: Create the Management Cluster

5.12.7.7.1 Deploying a Cluster in FIPS mode

In order to create a cluster in FIPS mode, we must inform the bootstrap controllers of the appropriate image repository and version tags of the official D2iQ FIPS builds of Kubernetes.

5.12.7.7.1.1 Supported FIPS Builds

Component	Repository	Version
Kubernetes	docker.io/mesosphere[332]	v1.25.4+fips.0
etcd	docker.io/mesosphere[333]	3.5.5+fips.0

5.12.7.7.1.2 Name your Cluster

1. Give your cluster a unique name suitable for your environment.
2. Set the CLUSTER_NAME environment variable with the command:

```
export CLUSTER_NAME=<my-vsphere-cluster>
```

DKP uses the vsphere CSI driver as the default storage provider (see page 103). Use a Kubernetes CSI[334] compatible storage that is suitable for production.

5.12.7.7.1.3 Create a New vSphere Kubernetes Cluster

Use the following steps to create a new, air-gapped vSphere cluster.

332 https://hub.docker.com/layers/mesosphere/kube-apiserver/v1.25.4_fips.0/images/sha256-cacb721f96ec8764d187ad3c21557630f5f4d96fea4a03ca35d0388b509d970d?context=explore
333 https://hub.docker.com/layers/mesosphere/etcd/v3.5.5_fips.0/images/sha256-ba07521fa1875efa0cc623533eb5557e40a6f8fdc8a71a86751a649ce53de1d0?context=explore
334 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

1. Configure your cluster to use an existing registry as a mirror when attempting to pull images:
   IMPORTANT: The image must be created by Konvoy Image Builder (see page 1305) in order to use the registry mirror feature.

   ```
   export REGISTRY_URL=<https/http>://<registry-address>:<registry-port>
   export REGISTRY_CA=<path to the CA on the bastion>
   export REGISTRY_USERNAME=<username>
   export REGISTRY_PASSWORD=<password>
   ```

   - `REGISTRY_URL` : the address of an existing registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
   - `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
   - `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
   - `REGISTRY_PASSWORD` : optional if username is not set.

2. Load the container image, using either the `docker` or `podman` command:

   ```
   docker load -i konvoy-bootstrap-image-v2.5.2.tar
   ```

   OR

   ```
   podman load -i konvoy-bootstrap-image-v2.5.2.tar
   ```

3. Create a Kubernetes cluster by copying the following command and substituting the valid values for your environment:

```
dkp create cluster vsphere
  --cluster-name ${CLUSTER_NAME} \
  --network <NETWORK_NAME> \
  --control-plane-endpoint-host <CONTROL_PLANE_IP> \
  --data-center <DATACENTER_NAME> \
  --data-store <DATASTORE_NAME> \
  --folder <FOLDER_NAME> \
  --server <VCENTER_API_SERVER_URL> \
  --ssh-public-key-file </path/to/key.pub> \
  --resource-pool <RESOURCE_POOL_NAME> \
  --vm-template konvoy-ova-vsphere-os-release-k8s_release-vsphere-timestamp \
  --virtual-ip-interface <ip_interface_name> \
  --extra-sans "127.0.0.1" \
  --registry-mirror-url=${DOCKER_REGISTRY_URL} \
  --registry-mirror-cacert=${DOCKER_REGISTRY_CA} \
  --registry-mirror-username=${DOCKER_REGISTRY_USERNAME} \
```

```
--registry-mirror-password=${DOCKER_REGISTRY_PASSWORD} \
--kubernetes-version=v1.25.4+fips.0 \
--kubernetes-image-repository=docker.io/mesosphere \
--etcd-image-repository=docker.io/mesosphere --etcd-version=3.5.5+fips.0 \
--self-managed
```

A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

5.12.7.7.1.4 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1360).

As they progress, the controllers create Events, which you can also monitor using the command:

```
kubectl get events | grep ${CLUSTER_NAME}
```

For brevity, this example uses `grep`. You can also use separate commands to get Events for specific objects, such as `kubectl get events --field-selector involvedObject.kind="VSphereCluster"` and `kubectl get events --field-selector involvedObject.kind="VSphereMachine"`.

5.12.7.7.1.5 Next Step:

vSphere FIPS Air-gapped: Configure MetalLB (see page 454)

5.12.7.8 vSphere FIPS Air-gapped: Configure MetalLB

If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can proceed to vSphere FIPS Air-gapped: Install Kommander (see page 457) and continue the installation process.

To use MetalLB, create a MetalLB configMap for your vSphere infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

Create a MetalLB configMap for your vSphere infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP)

Select one of the following procedures to create your MetalLB manifest for further editing and deployment.

5.12.7.8.1 Layer 2 configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to Address Resolution Protocol (ARP) requests on your local network directly, to give the machine's MAC address to clients.

- MetalLB IP address ranges/CIDRs should be within the node's primary network subnet (see page 929).
- MetalLB IP address ranges/CIDRs and node subnet should not conflict with the Kubernetes cluster pod and service subnets.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

1. Create a `metallb-conf.yaml` file for editing with the command:

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
```

```
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

2. Edit the file to contain values specific to your environment, then run the following `kubectl` command:

```
kubectl apply -f metallb-conf.yaml
```

5.12.7.8.2 BGP configuration

For a basic configuration featuring one BGP router and one IP address range, you need four pieces of information:

- The router IP address to which MetalLB should connect
- The router's AS number
- The AS number MetalLB should use
- An IP address range expressed as a CIDR prefix

For example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like the code snippet below.

The following example values are generic, enter your specific values into the fields where applicable.

1. Extract the kubeconfig for your cluster and deploy a configMap for MetalLB using the following command:

```
dkp get kubeconfig -c ${DKP_CLUSTER_NAME} > ${DKP_CLUSTER_NAME}.conf
```

2. Create a `metallb-conf.yaml` file for editing with the command:

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
```

```
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

3. Edit the file to contain values specific to your environment, then run the following `kubectl` command:

```
kubectl apply -f metallb-conf.yaml
```

5.12.7.8.3 Next Step:

vSphere FIPS Air-gapped: Install Kommander (see page 457)

5.12.7.9 vSphere FIPS Air-gapped: Install Kommander

5.12.7.9.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 119).
- Ensure you have a default StorageClass (see page 1254).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

5.12.7.9.2 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

```
export CLUSTER_NAME=<your-management-cluster-name>
```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

```
dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
```

3. Create a configuration file (see page 1227) for the deployment:

```
dkp install kommander --init --airgapped > kommander.yaml
```

4. *If required:* Customize your `kommander.yaml`.
   See Kommander Additional Install Configurations (see page 1227) for customization options. Some of them include:
   Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1271)), etc.

5.12.7.9.3 Enable DKP Catalog Applications and Install Kommander

If you have an Enterprise license and would like to use the DKP Catalog Applications function, follow these steps:

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.5.0
```

   ⚠ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.5.0.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.5.0.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.5.0.tar.gz
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 99) .
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

5.12.7.9.4 Next Step:

vSphere FIPS Air-gapped: Verify Install and Log in to the UI (see page 459)

5.12.7.10 vSphere FIPS Air-gapped: Verify Install and Log in to the UI

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander. After the CLI successfully installs the components, it will wait for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Released helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Released` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

5.12.7.10.1 Failed HelmReleases

If any application fails to successfully deploy, you can check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

5.12.7.10.2 Log in to the UI with Kommander

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 524). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 524):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

5.12.7.10.2.1 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 510) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

5.12.7.10.2.2 Next Step:

vSphere FIPS Air-gapped: Create a Managed Cluster Using the DKP CLI (see page 462)

5.12.7.11 vSphere FIPS Air-gapped: Create a Managed Cluster Using the DKP CLI

Creating an air-gapped vSphere FIPS Managed cluster with the DKP CLI assumes that you already fulfilled all of the prerequisites and successfully created a vSphere Management cluster. Use this procedure to create a Managed vSphere cluster.

> ⚠ When creating Managed clusters (see page 97), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 97)!

5.12.7.11.1 Choose a Workspace for the New Cluster

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a Workspace[335].

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

5.12.7.11.2 Name your cluster

1. Give your cluster a unique name suitable for your environment.
2. Set the CLUSTER_NAME environment variable with the command:

   ```
   export CLUSTER_NAME=<my-managed-vsphere-cluster>
   ```

> ⚠ DKP uses local static provisioner as the default storage provider (see page 103). However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[336] compatible storage that is suitable for production.

335 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919315/Workspaces#Create-a-Workspace
336 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

- You can choose from any of the storage options[337] available for Kubernetes. To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your newly created StorageClass to be the default by following the commands in the Kubernetes documentation called Changing the Default Storage Class[338].

5.12.7.11.3 Create a new vSphere Kubernetes cluster

Use the following steps to create a new, air-gapped vSphere cluster.

1. Configure your cluster to use an existing Docker registry as a mirror when attempting to pull images: **IMPORTANT:** The image must be created by Konvoy Image Builder (see page 1282) so that it uses the registry mirror feature.

   ```
   export REGISTRY_URL=<https/http>://<registry-address>:<registry-port>
   export REGISTRY_CA=<path to the CA on the bastion>
   export REGISTRY_USERNAME=<username>
   export REGISTRY_PASSWORD=<password>
   ```

   - `REGISTRY_URL` : the address of an existing Docker registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
   - `REGISTRY_CA` : (optional) the path on the bastion machine to the Docker registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
   - `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
   - `REGISTRY_PASSWORD` : optional if username is not set.

2. Create a Kubernetes cluster by copying the following command and substituting the valid values for your environment:

```
dkp create cluster vsphere \
  --cluster-name ${CLUSTER_NAME} \
  --network <NETWORK_NAME> \
  --control-plane-endpoint-host <CONTROL_PLANE_IP> \
  --data-center <DATACENTER_NAME> \
  --data-store <DATASTORE_NAME> \
  --folder <FOLDER_NAME> \
  --server <VCENTER_API_SERVER_URL> \
  --ssh-public-key-file </path/to/key.pub> \
  --resource-pool <RESOURCE_POOL_NAME> \
  --vm-template konvoy-ova-vsphere-os-release-k8s_release-vsphere-timestamp \
  --virtual-ip-interface <ip_interface_name> \
  --extra-sans "127.0.0.1" \
```

337 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
338 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

```
--registry-mirror-url=${REGISTRY_URL} \
--registry-mirror-cacert=${REGISTRY_CA} \
--registry-mirror-username=${REGISTRY_USERNAME} \
--registry-mirror-password=${REGISTRY_PASSWORD} \
--kubernetes-version=v1.25.4+fips.0 \
--kubernetes-image-repository=docker.io/mesosphere \
--etcd-image-repository=docker.io/mesosphere \
--etcd-version=3.5.5+fips.0 \
--namespace=${WORKSPACE_NAMESPACE}
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

5.12.7.11.4 Manually Attach a DKP CLI Cluster to the Management Cluster

1. Find out the `name` of the created `Cluster`, so you can reference it later:

   ```
   kubectl -n <workspace_namespace> get clusters
   ```

2. Attach the cluster by creating a `KommanderCluster`:

   ```
   cat << EOF | kubectl apply -f -
   apiVersion: kommander.mesosphere.io/v1beta1
   kind: KommanderCluster
   metadata:
     name: <cluster_name>
     namespace: <workspace_namespace>
   spec:
     kubeconfigRef:
       name: <cluster_name>-kubeconfig
     clusterRef:
       capiCluster:
         name: <cluster_name>
   EOF
   ```

If you have existing clusters or want to create other new clusters to attach, there are many ways to attach a cluster with various requirements and restrictions. To see all the options, visit the section in documentation Day 2 - Attach an Existing Kubernetes Cluster (see page 673).

At this point, you can create more clusters, perform other configuration tasks (see page 919), or proceed to Day 2 Operations (see page 510).

5.13 Azure Install Options

For an environment that is Azure, an install is provided for you in this one location. Remember, there are always more options in the Additional Infrastructure Configurations (see page 1033) sections, but this will get you operative in the most common scenario.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

Below are the supported environment variable combinations: (Refer to this page to check if your OS is a Supported Operating System (see page 62).)

- Azure Install (see page 465)

Additional Resource Information specific to Azure is below.

- **Control plane nodes** - DKP on Azure defaults to deploying a `Standard_D4s_v3` virtual machine with an 128 GiB volume for the OS and an 80GiB volume for etcd storage, which meets the above resource requirements (see page 110).
- **Worker nodes** - DKP on Azure defaults to deploying a `Standard_D8s_v3` virtual machine with an 80 GiB volume for the OS, which meets the above resource requirements (see page 110).

5.13.1 Azure Install

For an environment that is on the Azure Infrastructure, install options are provided for you in this one location. Remember, there are always more options in the Additional Infrastructure Configuration (see page 1282) sections, but this will get you up and running in the most common scenario.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

5.13.1.1 Azure Prerequisites

1. Sign in to Azure:

```
az login
```


```
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "id": "b1234567-abcd-11a1-a0a0-1234a5678b90",
    "isDefault": true,
    "managedByTenants": [],
    "name": "D2iQ Developer Subscription",
    "state": "Enabled",
    "tenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "user": {
      "name": "user@azuremesosphere.onmicrosoft.com",
      "type": "user"
    }
  }
]
```


2. Create an Azure Service Principal (SP) by running the following commands:
   a. If you have more than one Azure account, run this command to identify your account:

   ```
   echo $(az account show --query id -o tsv)
   ```

   b. Run this command to ensure you are pointing to the correct Azure subscription ID:

   ```
   az account set --subscription "D2iQ Developer Subscription"
   ```

   c. If an SP with the name exists, this command rotates the password.

   ```
   az ad sp create-for-rbac --role contributor --name "$(whoami)-konvoy" --
   scopes=/subscriptions/$(az account show --query id -o tsv) --query
   "{ client_id: appId, client_secret: password, tenant_id: tenant }"
   ```


   ```
   {
     "client_id": "7654321a-1a23-567b-b789-0987b6543a21",
     "client_secret": "Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C",
     "tenant_id": "a1234567-b132-1234-1a11-1234a5678b90"
   }
   ```


3. Set the `AZURE_CLIENT_SECRET` environment variable:

```
export AZURE_CLIENT_SECRET="<azure_client_secret>" #
Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C
export AZURE_CLIENT_ID="<client_id>"               # 7654321a-1a23-567b-
b789-0987b6543a21
export AZURE_TENANT_ID="<tenant_id>"               # a1234567-b132-1234-1a11-12
34a5678b90
export AZURE_SUBSCRIPTION_ID="<subscription_id>"   # b1234567-abcd-11a1-
a0a0-1234a5678b90
```

4. Ensure you have an override file (see page 1330) to configure specific attributes of your Azure image.

5.13.1.2 Next Step:

Azure: Create Image (see page 467)

5.13.1.3 Azure: Create Image

This procedure describes how to use the Konvoy Image Builder (see page 1282) (KIB) to create a Cluster API[339] compliant Azure Virtual Machine (VM) Image. The VM Image contains the base operating system you specify and all the necessary Kubernetes components. The Konvoy Image Builder uses variable `overrides` to specify the base image and container images to use in your new Azure VM image.

The default Azure image is not recommended for use in production. We suggest using KIB for Azure to build the image in order to take advantage of enhanced cluster operations. To explore more information on this topic refer to the Azure Infrastructure (see page 1033).

5.13.1.3.1 Prerequisites

Before you begin, you must:

- Check the DKP 2.5.0 Supported Kubernetes Versions (see page 1543)
- Check the Supported Infrastructure Operating Systems (see page 62)
- Download the Konvoy Image Builder (see page 1282) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
- Create a working `Docker` setup

339 https://cluster-api.sigs.k8s.io/

5.13.1.3.2 Extract the KIB Bundle

Extract the bundle and `cd` into the extracted `konvoy-image-bundle-$VERSION_$OS` folder. The bundled version of `konvoy-image` contains an embedded `docker` image that contains all the requirements for building.

> The `konvoy-image` binary and all supporting folders are also extracted. When extracted, `konvoy-image` bind mounts the current working directory (`${PWD}`) into the container to be used.

5.13.1.3.3 Build the Image

Run the `konvoy-image` command to build and validate the image.

```
konvoy-image build azure --client-id ${AZURE_CLIENT_ID} --tenant-id $
{AZURE_TENANT_ID} --overrides override-source-image.yaml images/azure/ubuntu-2004.yam
l
```

By default, the image builder builds in the `westus2` location. To specify another location set the `--location` flag (shown in example below is how to change the location to `eastus`):

```
konvoy-image build azure --client-id ${AZURE_CLIENT_ID} --tenant-id $
{AZURE_TENANT_ID} --location eastus --overrides override-source-image.yaml images/
azure/centos-7.yaml
```

When the command is complete, the image id is printed and written to the `./manifest.json` file. This file has an `artifact_id` field whose value provides the name of the image. You should then specify this image id when creating the cluster.

5.13.1.3.4 Image Gallery

By default Konvoy Image Builder will create a Resource Group, Gallery, and Image Name to store the resulting image in. To specify a specific Resource Group, Gallery, or Image Name flags may be specified:

```
--gallery-image-locations string     a list of locations to publish the image
(default same as location)
--gallery-image-name string          the gallery image name to publish the image to
```

```
--gallery-image-offer string          the gallery image offer to set (default "dkp")
--gallery-image-publisher string      the gallery image publisher to set (default
"dkp")
--gallery-image-sku string            the gallery image sku to set
--gallery-name string                 the gallery name to publish the image in
(default "dkp")
--resource-group string               the resource group to create the image in
(default "dkp")
```

When creating your cluster, you will then add this flag during the create process for your custom image: `--compute-gallery-id "<Managed Image Shared Image Gallery Id>"`. The SKU and Image Name will default to the values found in the image YAML.

5.13.1.3.4.1 More customization options:

- See Create a New Azure Cluster (see page 467) for specific consumption of image commands.
- See KIB for Azure (see page 1295) for more flags and Rocky Linux Marketplace.

5.13.1.3.5 Next Step:

Azure: Cluster Creation (see page 469)

5.13.1.4 Azure: Create the Management Cluster

5.13.1.4.1 Name your cluster

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

   ```
   export CLUSTER_NAME=<azure-example>
   ```

5.13.1.4.2 Encode your Azure Credential Variables:

Base64 encode the Azure environment variables set in the Azure install prerequisites (see page 465) step:

```
export AZURE_SUBSCRIPTION_ID_B64="$(echo -n "${AZURE_SUBSCRIPTION_ID}" | base64 | tr
-d '\n')"
export AZURE_TENANT_ID_B64="$(echo -n "${AZURE_TENANT_ID}" | base64 | tr -d '\n')"
export AZURE_CLIENT_ID_B64="$(echo -n "${AZURE_CLIENT_ID}" | base64 | tr -d '\n')"
export AZURE_CLIENT_SECRET_B64="$(echo -n "${AZURE_CLIENT_SECRET}" | base64 | tr -d
'\n')"
```

DKP uses the Azure CSI driver as the default storage provider (see page 103). Use a Kubernetes CSI[340] compatible storage that is suitable for production.

5.13.1.4.3 Create an Azure Kubernetes Cluster

If you use thee instructions to create an Azure cluster using the DKP default settings, your cluster is deployed with 3 control plane nodes and 4 worker nodes.

Availability zones (AZs) are isolated locations within data center regions from which public cloud services originate and operate. Because all the nodes in a node pool are deployed in a single Availability Zone, you may wish to create additional node pools to ensure your cluster has nodes deployed in multiple Availability Zones.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

1. Generate the Kubernetes cluster objects. The following example shows a common configuration. See dkp create cluster azure (see page 1454) reference for the full list of cluster creation options.

   ```
   dkp create cluster azure \
   --cluster-name=${CLUSTER_NAME} \
   --self-managed
   ```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

   Output is similar to below:

   ```
   Generating cluster resources
   ```

   A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing. As part of the underlying processing using the `--self-managed` flag, the DKP CLI:

   - creates a bootstrap cluster
   - creates a workload cluster

340 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

- moves CAPI controllers from the bootstrap cluster to the workload cluster, making it self-managed
- deletes the bootstrap cluster

To understand how this process works step by step, you can find a customizable Create a Custom Azure Cluster (see page 1041) under Additional Infrastructure Configuration.

5.13.1.4.4 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1360).

5.13.1.4.5 Known Limitations

The Konvoy version used to create a bootstrap cluster must match the Konvoy version used to create a workload cluster.

- Konvoy supports deploying one workload cluster.
- Konvoy generates a set of objects for one Node Pool.
- Konvoy does not validate edits to cluster objects.

5.13.1.4.6 Next Step:

Azure: Install Kommander (see page 471)

5.13.1.5 Azure: Install Kommander

You have installed the Konvoy component and created a cluster. Now it is time to Install Kommander which will allow you to access the UI and attach new or existing clusters to monitor.

5.13.1.5.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 119).
- Ensure you have a default StorageClass (see page 1254).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

5.13.1.5.2 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

```
export CLUSTER_NAME=<your-management-cluster-name>
```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

```
dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
```

3. Create a configuration file (see page 1227) for the deployment:

```
dkp install kommander --init > kommander.yaml
```

4. *If required:* Customize your `kommander.yaml`.
   See Kommander customizations (see page 1227) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1271)), etc.

5.13.1.5.3 Enable DKP Catalog Applications and Install Kommander

If you have an Enterprise license and would like to use the DKP Catalog Applications function, follow these steps:

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.5.0
```

   ⚠ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.co
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-`

`proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 99) .
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

5.13.1.5.4 Next Step:

Azure: Verify Install and Log in to the UI (see page 473)

5.13.1.6 Azure: Verify Install and Log in to the UI

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander. After the CLI successfully installs the components, it will wait for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Released helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Released` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

5.13.1.6.1 Failed HelmReleases

If any application fails to successfully deploy, you can check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 524). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 524):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

5.13.1.6.2 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 510) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

Now you can create additional new clusters (see page 476), or proceed to Day 2 Operations in the link.

5.13.1.6.3 Next Step:

Azure: Subsequent New Clusters (see page 476)

5.13.1.7 Azure: Create a Managed Cluster Using the DKP CLI

In the previous step, the new cluster was created as Self-managed which allows it to be a Management cluster. Subsequent new clusters are not self-managed as they will likely be Managed or Attached clusters to this Management Cluster.

When creating Managed clusters (see page 97), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 97)!

5.13.1.7.1 Choose a Workspace for the New Cluster

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a Workspace[341].

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

5.13.1.7.2 Name your cluster

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

   ```
   export CLUSTER_NAME=<azure-additional>
   ```

5.13.1.7.3 Use DKP CLI

Execute this command to create an additional cluster without `self-managed` flag:

341 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919315/Workspaces#Create-a-Workspace

```
dkp create cluster azure --cluster-name=${CLUSTER_NAME} --namespace=$
{WORKSPACE_NAMESPACE}
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

5.13.1.7.4 Manually Attach a DKP CLI Cluster to the Management Cluster

1. Find out the `name` of the created `Cluster`, so you can reference it later:

   ```
   kubectl -n <workspace_namespace> get clusters
   ```

2. Attach the cluster by creating a `KommanderCluster`:

   ```
   cat << EOF | kubectl apply -f -
   apiVersion: kommander.mesosphere.io/v1beta1
   kind: KommanderCluster
   metadata:
     name: <cluster_name>
     namespace: <workspace_namespace>
   spec:
     kubeconfigRef:
       name: <cluster_name>-kubeconfig
     clusterRef:
       capiCluster:
         name: <cluster_name>
   EOF
   ```

You can also Create a new Azure Cluster Using the DKP UI (see page 1066).

5.13.1.7.5 Next Step:

Day 2 - Cluster Operations Management (see page 510)

5.14 AKS Install Options

Enterprise Gov Advanced

For an environment that is AKS on the Azure Infrastructure, install options are provided for you in this one location. Remember, there are always more options in the Additional Infrastructure Configurations (see page 1068) sections, but this will get you operative in the most common scenario.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

Below are the supported environment variable combinations: (Refer to this page to check if your OS is a Supported Operating System (see page 62).)

- AKS Install (see page 478)

> In order to install Kommander, you need to have CAPI components, cert-manager, etc on a self-managed cluster. The CAPI components mean you can control the lifecycle of the cluster, and other clusters. However, because AKS is semi-managed by Azure, the AKS clusters are under Azure control and don't have those components. Therefore, Kommander will not be able to be installed and these clusters will be attached to the management cluster.

5.14.1 AKS Install

Enterprise Gov Advanced

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

5.14.1.1 DKP Prerequisites

Before starting the DKP installation, verify that you have:

- An x86_64-based Linux or macOS machine with a supported version of the operating system (see page 62).
- A Self-managed Azure cluster (see page 1054), if used Day 1-Basic Install for Azure instructions, your cluster was created using `--self-managed` flag and therefore is already a self-managed cluster.
- Download the `dkp` binary (see page 71) for Linux, or macOS. To check which version of DKP you installed for compatibility reasons, run the `dkp version -h` command (dkp version (see page 1534)).
- Docker[342] version 18.09.2 or later.
- kubectl[343] for interacting with the running cluster.
- The Azure CLI[344].
- A valid Azure account used to sign in to the Azure CLI[345].
- All Resource requirements (see page 0)

5.14.1.2 AKS Prerequisites

Follow these steps:

1. Log in to Azure:

```
az login
```

```
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "id": "b1234567-abcd-11a1-a0a0-1234a5678b90",
    "isDefault": true,
    "managedByTenants": [],
    "name": "Mesosphere Developer Subscription",
    "state": "Enabled",
    "tenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "user": {
      "name": "user@azuremesosphere.onmicrosoft.com",
      "type": "user"
    }
  }
]
```

2. Create an Azure Service Principal (SP) by running the following command:

   NOTE: If an SP with the name exists, this command will rotate the password.

342 https://docs.docker.com/get-docker/
343 https://kubernetes.io/docs/tasks/tools/#kubectl
344 https://docs.microsoft.com/en-us/cli/azure/install-azure-cli
345 https://docs.microsoft.com/en-us/cli/azure/authenticate-azure-cli?view=azure-cli-latest

```
az ad sp create-for-rbac --role contributor --name "$(whoami)-konvoy" --
scopes=/subscriptions/$(az account show --query id -o tsv)
```

```
{
  "appId": "7654321a-1a23-567b-b789-0987b6543a21",
  "displayName": "azure-cli-2021-03-09-23-17-06",
  "password": "Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C",
  "tenant": "a1234567-b132-1234-1a11-1234a5678b90"
}
```

3. Set the required environment variables:

```
export AZURE_SUBSCRIPTION_ID="<id>"       # b1234567-abcd-11a1-
a0a0-1234a5678b90
export AZURE_TENANT_ID="<tenant>"         # a1234567-b132-1234-1a11-1234a5678b9
0
export AZURE_CLIENT_ID="<appId>"          # 7654321a-1a23-567b-
b789-0987b6543a21
export AZURE_CLIENT_SECRET="<password>"   # Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C
```

4. Base64 encode the same environment variables:

```
export AZURE_SUBSCRIPTION_ID_B64="$(echo -n "${AZURE_SUBSCRIPTION_ID}" | base64
| tr -d '\n')"
export AZURE_TENANT_ID_B64="$(echo -n "${AZURE_TENANT_ID}" | base64 | tr -d
'\n')"
export AZURE_CLIENT_ID_B64="$(echo -n "${AZURE_CLIENT_ID}" | base64 | tr -d
'\n')"
export AZURE_CLIENT_SECRET_B64="$(echo -n "${AZURE_CLIENT_SECRET}" | base64 |
tr -d '\n')"
```

5. Check to see what version of Kubernetes is available in your region. When deploying with AKS, you must pick a version of Kubernetes that is available in AKS and use that version for subsequent steps. To find out the list of available Kubernetes versions in the Azure Region you are using, run the following command, substituting `<your-location>` for the Azure region you're deploying to:

```
az aks get-versions -o table --location <your-location>
```

The output from this command will resemble the following:

```
$ az aks get-versions -o table --location westus
KubernetesVersion    Upgrades
```

```
-------------------  -----------------------
1.27.1(preview)      None available
1.26.3               1.27.1(preview)
1.26.0               1.26.3, 1.27.1(preview)
1.25.6               1.26.0, 1.26.3
1.25.5               1.25.6, 1.26.0, 1.26.3
1.24.10              1.25.5, 1.25.6
1.24.9               1.24.10, 1.25.5, 1.25.6
```

6. Choose a version of Kubernetes to install from the list of `KubernetesVersion`, choosing a compatible version as documented in the Supported Kubernetes Versions (see page 1543) for this version of DKP. The version listed in the command is an example:

```
export KUBERNETES_VERSION=1.24.6
```

5.14.1.3 Next Step:

AKS: Create Image (see page 481)

5.14.1.4 AKS: Create Image

Enterprise Gov Advanced

AKS best practices discourage building custom images. If the image is customized, it breaks some of the autoscaling and security capabilities of AKS. Since custom virtual machine images are discouraged in AKS, Konvoy Image Builder (KIB) does not include any support for building custom machine images for AKS.

5.14.1.4.1 Next Step:

AKS: Cluster Creation (see page 481)

5.14.1.5 AKS: Create an AKS Cluster

Enterprise Gov Advanced

When installing DKP on your AKS infrastructure, you can choose from multiple configuration types. Create a new Kubernetes cluster in a non-air-gapped environment with the steps below.

5.14.1.5.1 Use DKP to create a new AKS cluster

Ensure that the `KUBECONFIG` environment variable is set to the self-managed cluster by running `export KUBECONFIG=${SELF_MANAGED_AZURE_CLUSTER}.conf`

5.14.1.5.2 Name Your Cluster

Give your cluster a unique name suitable for your environment.

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[346] for more naming information.

5.14.1.5.3 Create a New AKS Kubernetes Cluster

1. Set the environment variable to a name for this cluster.

```
export CLUSTER_NAME=<aks-example>
```

2. Check to see what version of Kubernetes is available in your region. When deploying with AKS, you need to declare the version of Kubernetes you wish to use by running the following command, substituting `<your-location>` for the Azure region you're deploying to:

```
az aks get-versions -o table --location <your-location>
```

3. Set the version of Kubernetes you've chosen:
NOTE: Using Kubernetes v1.25.4 is recommended, but if it is not available, choose an available 1.25.x version. The version listed in the command is an example.

```
export KUBERNETES_VERSION=1.25.4
```

4. Create the cluster:

346 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

```
dkp create cluster aks --cluster-name=${CLUSTER_NAME} --additional-tags=owner=$
(whoami) --kubernetes-version=${KUBERNETES_VERSION}
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).
The output displays similar to this:

```
Generating cluster resources
cluster.cluster.x-k8s.io/aks-example created
azuremanagedcontrolplane.infrastructure.cluster.x-k8s.io/aks-example created
azuremanagedcluster.infrastructure.cluster.x-k8s.io/aks-example created
machinepool.cluster.x-k8s.io/aks-example created
azuremanagedmachinepool.infrastructure.cluster.x-k8s.io/cp6dsz8 created
machinepool.cluster.x-k8s.io/aks-example-md-0 created
azuremanagedmachinepool.infrastructure.cluster.x-k8s.io/mp6gglj created
clusterresourceset.addons.cluster.x-k8s.io/cluster-autoscaler-aks-example
created
configmap/cluster-autoscaler-aks-example created
clusterresourceset.addons.cluster.x-k8s.io/node-feature-discovery-aks-example
created
configmap/node-feature-discovery-aks-example created
clusterresourceset.addons.cluster.x-k8s.io/nvidia-feature-discovery-aks-example
created
configmap/nvidia-feature-discovery-aks-example created
```

5.14.1.5.4 Inspecting or Editing the Cluster Objects

Use your favorite editor.

> **NOTE:** Editing the cluster objects requires some understanding of Cluster API. Edits can prevent the cluster from deploying successfully.

The objects are Custom Resources[347] defined by Cluster API components, and they belong in three different categories:

- Cluster
  A *Cluster* object has references to the infrastructure-specific and control plane objects.
- Control Plane
- Node Pool

347 https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/

A Node Pool is a collection of machines with identical properties. For example, a cluster might have one Node Pool with large memory capacity, another Node Pool with GPU support. Each Node Pool is described by three objects: The MachinePool references an object that describes the configuration of Kubernetes components (for example, kubelet) deployed on each node pool machine, and an infrastructure-specific object that describes the properties of all node pool machines. Here, it references a *KubeadmConfigTemplate*.

For in-depth documentation about the objects, read Concepts[348] in the Cluster API Book.

1. Wait for the cluster control-plane to be ready:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=20m
```

```
cluster.cluster.x-k8s.io/aks-example condition met
```

The `READY` status will become `True` after the cluster control-plane becomes ready.

2. Once the objects are created on the API server, the Cluster API controllers reconcile them. They create infrastructure and machines. As they progress, they update the Status of each object. DKP provides a command to describe the current status of the cluster:

```
dkp describe cluster -c ${CLUSTER_NAME}
```

```
NAME                                                          READY  SEVERITY
REASON  SINCE  MESSAGE
Cluster/aks-example                                           True
48m
 ├──ClusterInfrastructure - AzureManagedCluster/aks-example
 └──ControlPlane - AzureManagedControlPlane/aks-example
```

3. As they progress, the controllers also create Events. List the Events using this command:

```
kubectl get events | grep ${CLUSTER_NAME}
```

For brevity, the example uses `grep`. It is also possible to use separate commands to get Events for specific objects. For example, `kubectl get events --field-selector`

348 https://cluster-api.sigs.k8s.io/user/concepts.html

`involvedObject.kind="AKSCluster"` and `kubectl get events --field-selector involvedObject.kind="AKSMachine"`.

```
48m         Normal    SuccessfulSetNodeRefs               machinepool/aks-
example-md-0                  [{Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000000 UID:e3c30389-660d-46f5-b9d7-219f80b5674d APIVersion:
ResourceVersion: FieldPath:} {Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000001 UID:300d71a0-f3a7-4c29-9ff1-1995ffb9cfd3 APIVersion:
ResourceVersion: FieldPath:} {Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000002 UID:8eae2b39-a415-425d-8417-d915a0b2fa52 APIVersion:
ResourceVersion: FieldPath:} {Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000003 UID:3e860b88-f1a4-44d1-b674-a54fad599a9d APIVersion:
ResourceVersion: FieldPath:}]
6m4s        Normal    AzureManagedControlPlane available
azuremanagedcontrolplane/aks-example          successfully reconciled
48m         Normal    SuccessfulSetNodeRefs               machinepool/aks-
example                       [{Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000000 UID:e3c30389-660d-46f5-b9d7-219f80b5674d APIVersion:
ResourceVersion: FieldPath:} {Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000001 UID:300d71a0-f3a7-4c29-9ff1-1995ffb9cfd3 APIVersion:
ResourceVersion: FieldPath:} {Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000002 UID:8eae2b39-a415-425d-8417-d915a0b2fa52 APIVersion:
ResourceVersion: FieldPath:}]
```

5.14.1.5.5 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1360).

5.14.1.5.6 Next Steps:

AKS: Retrieve kubeconfig (see page 485)

5.14.1.6 AKS: Retrieve kubeconfig for AKS Cluster

Enterprise Gov Advanced

5.14.1.6.1 Learn to interact with your AKS Kubernetes cluster

This guide explains how to use the command line to interact with your newly deployed Kubernetes cluster.

Before you start, make sure you have created a workload cluster, as described in Create an AKS Cluster (see page 481) .

5.14.1.6.2 Explore the New AKS Cluster

1. Get a kubeconfig file for the workload cluster:

   When the workload cluster is created, the cluster lifecycle services generate a kubeconfig file for the workload cluster, and write it to a *Secret*. The kubeconfig file is scoped to the cluster administrator.

   Get the kubeconfig from the *Secret*, and write it to a file, using this command:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

2. List the Nodes using this command:

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf get nodes
   ```

   ```
   NAME                                STATUS   ROLES   AGE   VERSION
   aks-cp6dsz8-41174201-vmss000000   Ready    agent   56m   v1.25.4
   aks-cp6dsz8-41174201-vmss000001   Ready    agent   55m   v1.25.4
   aks-cp6dsz8-41174201-vmss000002   Ready    agent   56m   v1.25.4
   aks-mp6gglj-41174201-vmss000000   Ready    agent   55m   v1.25.4
   aks-mp6gglj-41174201-vmss000001   Ready    agent   55m   v1.25.4
   aks-mp6gglj-41174201-vmss000002   Ready    agent   55m   v1.25.4
   aks-mp6gglj-41174201-vmss000003   Ready    agent   56m   v1.25.4
   ```

NOTE: It may take a few minutes for the Status to move to `Ready` while the Pod network is deployed. The Nodes' Status should change to Ready soon after the `calico-node` DaemonSet Pods are Ready.

3. List the Pods using this command:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf get --all-namespaces pods
```

```
NAMESPACE                NAME                                        READY
STATUS     RESTARTS   AGE
calico-system            calico-kube-controllers-5dcd4b47b5-tgslm    1/1
Running    0          3m58s
calico-system            calico-node-46dj9                           1/1
Running    0          3m58s
calico-system            calico-node-crdgc                           1/1
Running    0          3m58s
calico-system            calico-node-m7s7x                           1/1
Running    0          3m58s
calico-system            calico-node-qfkqc                           1/1
Running    0          3m57s
calico-system            calico-node-sfqfm                           1/1
Running    0          3m57s
calico-system            calico-node-sn67x                           1/1
Running    0          3m53s
calico-system            calico-node-w2pvt                           1/1
Running    0          3m58s
calico-system            calico-typha-6f7f59969c-5z4t5               1/1
Running    0          3m51s
calico-system            calico-typha-6f7f59969c-ddzqb               1/1
Running    0          3m58s
calico-system            calico-typha-6f7f59969c-rr4lj               1/1
Running    0          3m51s
kube-system              azure-ip-masq-agent-4f4v6                   1/1
Running    0          4m11s
kube-system              azure-ip-masq-agent-5xfh2                   1/1
Running    0          4m11s
kube-system              azure-ip-masq-agent-9hlk8                   1/1
Running    0          4m8s
kube-system              azure-ip-masq-agent-9vsgg                   1/1
Running    0          4m16s
kube-system              azure-ip-masq-agent-b9wjj                   1/1
Running    0          3m57s
kube-system              azure-ip-masq-agent-kpjtl                   1/1
Running    0          3m53s
kube-system              azure-ip-masq-agent-vr7hd                   1/1
Running    0          3m57s
kube-system              cluster-autoscaler-b4789f4bf-qkfk2          0/1
Init:0/1   0          3m28s
kube-system              coredns-845757d86-9jf8b                     1/1
Running    0          5m29s
kube-system              coredns-845757d86-h4xfs                     1/1
Running    0          4m
kube-system              coredns-autoscaler-5f85dc856b-xjb5z         1/1
Running    0          5m23s
kube-system              csi-azuredisk-node-4n4fx                    3/3
Running    0          3m53s
kube-system              csi-azuredisk-node-8pnjj                    3/3
Running    0          3m57s
```

```
kube-system              csi-azuredisk-node-sbt6r                          3/3
Running    0          3m57s
kube-system              csi-azuredisk-node-v25wc                          3/3
Running    0          4m16s
kube-system              csi-azuredisk-node-vfbxg                          3/3
Running    0          4m11s
kube-system              csi-azuredisk-node-w5ff5                          3/3
Running    0          4m11s
kube-system              csi-azuredisk-node-zzgqx                          3/3
Running    0          4m8s
kube-system              csi-azurefile-node-2rpcc                          3/3
Running    0          3m57s
kube-system              csi-azurefile-node-4gqkf                          3/3
Running    0          4m11s
kube-system              csi-azurefile-node-f6k8m                          3/3
Running    0          4m16s
kube-system              csi-azurefile-node-k72xq                          3/3
Running    0          4m8s
kube-system              csi-azurefile-node-vx7r4                          3/3
Running    0          3m53s
kube-system              csi-azurefile-node-zc8kr                          3/3
Running    0          4m11s
kube-system              csi-azurefile-node-zkl6b                          3/3
Running    0          3m57s
kube-system              kube-proxy-4fpb6                                  1/1
Running    0          3m53s
kube-system              kube-proxy-6qfbf                                  1/1
Running    0          4m16s
kube-system              kube-proxy-6wnt2                                  1/1
Running    0          4m8s
kube-system              kube-proxy-cspd5                                  1/1
Running    0          3m57s
kube-system              kube-proxy-nsgq6                                  1/1
Running    0          4m11s
kube-system              kube-proxy-qz2st                                  1/1
Running    0          4m11s
kube-system              kube-proxy-zvh9k                                  1/1
Running    0          3m57s
kube-system              metrics-server-6bc97b47f7-ltkkj                   1/1
Running    0          5m28s
kube-system              tunnelfront-77d68f78bf-t78ck                      1/1
Running    0          5m23s
node-feature-discovery   node-feature-discovery-master-65dc499cd-fxwb5     1/1
Running    0          3m28s
node-feature-discovery   node-feature-discovery-worker-277xc               1/1
Running    0          3m28s
node-feature-discovery   node-feature-discovery-worker-4dq5k               1/1
Running    0          3m28s
node-feature-discovery   node-feature-discovery-worker-57nb8               1/1
Running    0          3m28s
node-feature-discovery   node-feature-discovery-worker-b4lkl               1/1
Running    0          3m28s
```

```
node-feature-discovery   node-feature-discovery-worker-kslst            1/1
Running    0          3m28s
node-feature-discovery   node-feature-discovery-worker-ppjtm            1/1
Running    0          3m28s
node-feature-discovery   node-feature-discovery-worker-x5bgf            1/1
Running    0          3m28s
tigera-operator          tigera-operator-74c4d9cf84-k7css               1/1
Running    0          5m25s
```

5.14.1.6.3 Next Step:

AKS: Attach Cluster (see page 489)

5.14.1.7 AKS: Attach Cluster

ENTERPRISE

5.14.1.7.0.1 Attach an existing AKS cluster

You can attach existing Kubernetes clusters to the Management Cluster. After attaching the cluster, you can use the UI to examine and manage (see page 669) this cluster. The following procedure shows how to attach an existing Azure Kubernetes Service (AKS) cluster.

5.14.1.7.1 Before you Begin

This procedure requires the following items and configurations:

- A fully configured and running Azure AKS[349]cluster with administrative privileges.
- The current version DKP Enterprise is installed (see page 1269)on your cluster.
- Ensure you have installed `kubectl` in your Management cluster.

This procedure assumes you have an existing and spun up Azure AKS cluster(s) with administrative privileges. Refer to the Azure AKS[350]for setup and configuration information.

349 https://azure.microsoft.com/en-us/products/kubernetes-service/
350 https://azure.microsoft.com/en-us/products/kubernetes-service/

5.14.1.7.2 Attach AKS Clusters

Ensure that the `KUBECONFIG` environment variable is set to the Management cluster before attaching by running:

```
export KUBECONFIG=<Management_cluster_kubeconfig>.conf
```

5.14.1.7.2.1 Ensure you have access to your AKS clusters

1. Ensure you are connected to your AKS clusters. Enter the following commands for each of your clusters:

   ```
   kubectl config get-contexts
   kubectl config use-context <context for first AKS cluster>
   ```

2. Confirm `kubectl` can access the AKS cluster:

   ```
   kubectl get nodes
   ```

5.14.1.7.2.2 Create a kubeconfig file for your AKS cluster

To get started, ensure you have kubectl[351] set up and configured with ClusterAdmin[352] for the cluster you want to connect to Kommander.

1. Create the necessary service account:

   ```
   kubectl -n kube-system create serviceaccount kommander-cluster-admin
   ```

2. Create a token secret for the `serviceaccount`:

   ```
   kubectl -n kube-system create  -f - <<EOF
   apiVersion: v1
   kind: Secret
   metadata:
     name: kommander-cluster-admin-sa-token
     annotations:
       kubernetes.io/service-account.name: kommander-cluster-admin
   type: kubernetes.io/service-account-token
   ```

351 https://kubernetes.io/docs/tasks/tools/#kubectl
352 https://kubernetes.io/docs/concepts/cluster-administration/

```
EOF
```

3. Verify that the `serviceaccount` token is ready by running this command:

   ```
   kubectl -n kube-system get secret kommander-cluster-admin-sa-token -oyaml
   ```

   Verify that the `data.token` field is populated. The output should be similar to this:

   ```
   apiVersion: v1
   data:
     ca.crt: LS0tLS1CRUdJTiBDR...
     namespace: ZGVmYXVsdA==
     token: ZXlKaGJHY2lPaUpTVX...
   kind: Secret
   metadata:
     annotations:
       kubernetes.io/service-account.name: kommander-cluster-admin
       kubernetes.io/service-account.uid: b62bc32e-b502-4654-921d-94a742e273a8
     creationTimestamp: "2022-08-19T13:36:42Z"
     name: kommander-cluster-admin-sa-token
     namespace: default
     resourceVersion: "8554"
     uid: 72c2a4f0-636d-4a70-9f1c-55a75f15e520
   type: kubernetes.io/service-account-token
   ```

4. Configure the new service account for `cluster-admin` permissions:

   ```
   cat << EOF | kubectl apply -f -
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRoleBinding
   metadata:
     name: kommander-cluster-admin
   roleRef:
     apiGroup: rbac.authorization.k8s.io
     kind: ClusterRole
     name: cluster-admin
   subjects:
   - kind: ServiceAccount
     name: kommander-cluster-admin
     namespace: kube-system
   EOF
   ```

5. Set up the following environment variables with the access data that is needed for producing a new `kubeconfig` file:

```
export USER_TOKEN_VALUE=$(kubectl -n kube-system get secret/kommander-cluster-
admin-sa-token -o=go-template='{{.data.token}}' | base64 --decode)
export CURRENT_CONTEXT=$(kubectl config current-context)
export CURRENT_CLUSTER=$(kubectl config view --raw -o=go-
template='{{range .contexts}}{{if eq .name "'''${CURRENT_CONTEXT}'''"}}
{{ index .context "cluster" }}{{end}}{{end}}')
export CLUSTER_CA=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}"{{with
index .cluster "certificate-authority-data" }}{{.}}{{end}}"{{ end }}{{ end }}')
export CLUSTER_SERVER=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}
{{ .cluster.server }}{{end}}{{ end }}')
```

6. Confirm these variables have been set correctly:

```
export -p | grep -E 'USER_TOKEN_VALUE|CURRENT_CONTEXT|CURRENT_CLUSTER|
CLUSTER_CA|CLUSTER_SERVER'
```

7. Generate a `kubeconfig` file that uses the environment variable values from the previous step:

```
cat << EOF > kommander-cluster-admin-config
apiVersion: v1
kind: Config
current-context: ${CURRENT_CONTEXT}
contexts:
- name: ${CURRENT_CONTEXT}
  context:
    cluster: ${CURRENT_CONTEXT}
    user: kommander-cluster-admin
    namespace: kube-system
clusters:
- name: ${CURRENT_CONTEXT}
  cluster:
    certificate-authority-data: ${CLUSTER_CA}
    server: ${CLUSTER_SERVER}
users:
- name: kommander-cluster-admin
  user:
    token: ${USER_TOKEN_VALUE}
EOF
```

8. This process produces a file in your current working directory called `kommander-cluster-admin-config`. The contents of this file are used in Kommander to attach the cluster.
   Before importing this configuration, verify the `kubeconfig` file can access the cluster:

```
kubectl --kubeconfig $(pwd)/kommander-cluster-admin-config get all --all-
namespaces
```

5.14.1.7.2.3 Finalize attaching your cluster from the UI

Now that you have `kubeconfig`, go to the DKP UI and follow these steps below:

1. From the top menu bar, select your target workspace.
2. On the Dashboard page, select the **Add Cluster** option in the **Actions** dropdown menu at the top right.
3. Select **Attach Cluster**.
4. Select the **No additional networking restrictions** card. Alternatively, if you must use network restrictions, stop following the steps below, and see the instructions on the page Attach a cluster WITH network restrictions (see page 693).
5. Upload the kubeconfig file you created in the previous section (or copy its contents) into the **Cluster Configuration** section.
6. The **Cluster Name** field automatically populates with the name of the cluster in the kubeconfig. You can edit this field with the name you want for your cluster.
7. Add labels to classify your cluster as needed.
8. Select **Create** to attach your cluster.

If a cluster has limited resources to deploy all the federated platform services, it will fail to stay attached in the DKP UI. If this happens, ensure your system has sufficient resources for all pods.

Next Steps:
Day 2 - Cluster Operations Management (see page 510)

5.15 GCP Install Options

For an environment that is Google Cloud Platform, an install is provided for you in this one location. Remember, there are always more options in the Additional Infrastructure Configuration (see page 1199) section for Google Cloud, but this will get you operative in the most common scenario.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

Below are the supported environment variable combinations: (Refer to this page to check if your OS is a Supported Operating System (see page 62).)

- GCP Install (see page 494)

Additional Resource Information specific to GCP is below.

- **Control Plane Nodes** - DKP on GCP defaults to deploying an `n2-standard-4` instance with an 80GiB root volume for control plane nodes, which meets the above requirements.
- **Worker Nodes** - DKP on GCP defaults to deploying a `n2-standard-8` instance with an 80GiB root volume for worker nodes, which meets the above requirements.

5.15.1 GCP Install

For an environment that is on the GCP Infrastructure, install options are provided for you in this one location. Remember, there are always more options in the Additional Infrastructure Configuration (see page 919) sections, but this will get you up and running in the most common scenario.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

5.15.1.1 GCP Prerequisites:

- Verify that your Google Cloud project does not have the Enable OS Login feature enabled.

The Enable OS Login feature is sometimes enabled by default in GCP projects. If the OS login feature is enabled, KIB will not be able to `ssh` to the VM instances it creates and will not be able to successfully create an image.
To check if it is enabled, use the commands on this page https://cloud.google.com/compute/docs/metadata/setting-custom-metadata#console_2 to inspect the metadata configured in in your project. If you find the the `enable-oslogin` flag set to TRUE, you must remove (or set it to FALSE) to use KIB.

- The user creating the Service Accounts needs additional privileges in addition to the Editor role.
  - See GCP Prerequisite Roles (see page 0)

5.15.1.2 Next Step:

GCP: Create Image (see page 495)

5.15.1.3 GCP: Create Image

This procedure describes how to use the Konvoy Image Builder (see page 1282) (KIB) to create a Cluster API[353] compliant GCP image. GCP images contain configuration information and software to create a specific, pre-configured, operating environment. For example, you can create a GCP image of your current computer system settings and software. The GCP image can then be replicated and distributed, creating your computer system for other users. KIB uses variable overrides (see page 1330) to specify base image and container images to use in your new GCP image.

Google Cloud Platform does not publish images. You must first build the image using Konvoy Image Builder. For more information regarding images and clusters, refer to the GCP Infrastructure (see page 1199) section of the documentation.

5.15.1.3.1 Prerequisites

Before you begin, you must:

- Check the supported DKP version (see page 1543).
- Check the Supported Infrastructure Operating Systems (see page 62)
- Download the KIB (see page 1283) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
- Create a working `Docker` setup.
- On Debian-based Linux distributions, install a version of the cri-tools[354] package known to be compatible with both the Kubernetes and container runtime versions.
- Verify that your Google Cloud project does not have the Enable OS Login feature enabled. See below for more information:

The Enable OS Login feature is sometimes enabled by default in GCP projects. If the OS login feature is enabled, KIB will not be able to `ssh` to the VM instances it creates and will not be able to successfully create an image.

353 https://cluster-api.sigs.k8s.io/
354 https://github.com/kubernetes-sigs/cri-tools

To check if it is enabled, use the commands on this page https://cloud.google.com/compute/docs/metadata/setting-custom-metadata#console_2 to inspect the metadata configured in in your project. If you find the the `enable-oslogin` flag set to TRUE, you must remove (or set it to FALSE) to successfully use KIB.

5.15.1.3.2 GCP Prerequisite Roles

- If you are creating your image on either a non-GCP instance or one that does not have the required roles (see page 1199):
  - (option 1) Create a service account using the following `gcloud` commands:

```
export GCP_PROJECT=<your GCP project ID>
export GCP_SERVICE_ACCOUNT_USER=<some new service account user>
export GOOGLE_APPLICATION_CREDENTIALS="$HOME/.gcloud/credentials.json"

gcloud iam service-accounts create "${GCP_SERVICE_ACCOUNT_USER}" --
project=${GCP_PROJECT}
gcloud projects add-iam-policy-binding ${GCP_PROJECT} --
member="serviceAccount:${GCP_SERVICE_ACCOUNT_USER}@$
{GCP_PROJECT}.iam.gserviceaccount.com" --role=roles/
compute.instanceAdmin.v1
gcloud projects add-iam-policy-binding ${GCP_PROJECT} --
member="serviceAccount:${GCP_SERVICE_ACCOUNT_USER}@$
{GCP_PROJECT}.iam.gserviceaccount.com" --role=roles/
iam.serviceAccountUser
gcloud iam service-accounts keys create $
{GOOGLE_APPLICATION_CREDENTIALS} --iam-account="$
{GCP_SERVICE_ACCOUNT_USER}@${GCP_PROJECT}.iam.gserviceaccount.com"
```

  - (option 2) If you have already created a service account, retrieve the credentials for an existing service account using the following `gcloud` commands:

```
export GCP_PROJECT=<your GCP project ID>
export GCP_SERVICE_ACCOUNT_USER=<existing service account user>
export GOOGLE_APPLICATION_CREDENTIALS="$HOME/.gcloud/credentials.json"

gcloud projects add-iam-policy-binding ${GCP_PROJECT} --
member="serviceAccount:${GCP_SERVICE_ACCOUNT_USER}@$
{GCP_PROJECT}.iam.gserviceaccount.com" --role=roles/
compute.instanceAdmin.v1
gcloud projects add-iam-policy-binding ${GCP_PROJECT} --
member="serviceAccount:${GCP_SERVICE_ACCOUNT_USER}@$
{GCP_PROJECT}.iam.gserviceaccount.com" --role=roles/
iam.serviceAccountUser
```

```
gcloud iam service-accounts keys create $
{GOOGLE_APPLICATION_CREDENTIALS} --iam-account="$
{GCP_SERVICE_ACCOUNT_USER}@${GCP_PROJECT}.iam.gserviceaccount.com"
```

- Export the static credentials that will be used to create the cluster:

```
export GCP_B64ENCODED_CREDENTIALS=$(base64 < "$
{GOOGLE_APPLICATION_CREDENTIALS}" | tr -d '\n')
```

5.15.1.3.3 Build the GCP image

1. Run the `konvoy-image` command to build and validate the image:

```
./konvoy-image build gcp --project-id ${GCP_PROJECT} --network ${NETWORK_NAME}
images/gcp/ubuntu-2004.yaml
```

2. KIB will run and print out the name of the created image, you will use this name when creating a Kubernetes cluster. See sample output below:

```
...
==> ubuntu-2004-focal-v20220419: Deleting instance...
    ubuntu-2004-focal-v20220419: Instance has been deleted!
==> ubuntu-2004-focal-v20220419: Creating image...
==> ubuntu-2004-focal-v20220419: Deleting disk...
    ubuntu-2004-focal-v20220419: Disk has been deleted!
==> ubuntu-2004-focal-v20220419: Running post-processor: manifest
Build 'ubuntu-2004-focal-v20220419' finished after 7 minutes 46 seconds.

==> Wait completed after 7 minutes 46 seconds

==> Builds finished. The artifacts of successful builds are:
--> ubuntu-2004-focal-v20220419: A disk image was created: konvoy-ubuntu-2004-1-2
3-7-1658523168
--> ubuntu-2004-focal-v20220419: A disk image was created: konvoy-ubuntu-2004-1-2
3-7-1658523168
```

3. To find a list of images you have created in your account, run the following command:

```
gcloud compute images list --no-standard-images
```

5.15.1.3.4 Next Step:

GCP: Cluster Creation (see page 498)

5.15.1.4 GCP: Create the Management Cluster

Create a new Google Cloud Platform Kubernetes cluster in a non-air-gapped environment with the steps below.

DKP uses the GCP CSI driver as the default storage provider (see page 103). Use a Kubernetes CSI[355] compatible storage that is suitable for production.

5.15.1.4.1 Name your Cluster

1. Give your cluster a unique name suitable for your environment.

   In GCP it is critical that the name is unique, as no two clusters in the same GCP account can have the same name.

2. Set the environment variable:

   ```
   export CLUSTER_NAME=<gcp-example>
   ```

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[356] for more naming information.

To increase Docker Hub's rate limit[357] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username=<username> --registry-mirror-password=<password>` on the `dkp create cluster` command.

355 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
356 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/
357 https://docs.docker.com/docker-hub/download-rate-limit/

5.15.1.4.2 Create a New GCP Cluster

Availability zones (AZs) are isolated locations within data center regions from which public cloud services originate and operate. Because all the nodes in a node pool are deployed in a single Availability Zone, you may wish to create additional node pools is to ensure your cluster has nodes deployed in multiple Availability Zones.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single zone. You may create additional node pools in other zones with the `dkp create nodepool` command. The default region for the availability zones is `us-west1`.

Google Cloud Platform does not publish images. You must first build the image using Konvoy Image Builder (see page 1282).

1. Create an image using Konvoy Image Builder (KIB) (see page 1282) and then export the image name:

   ```
   export IMAGE_NAME=projects/${GCP_PROJECT}/global/images/<image_name_from_kib>
   ```

2. (Optional) You can modify Control Plane Audit logs settings using the information contained in the page Configuring the Control Plane (see page 1351).
3. (Optional) Determine what VPC Network to use. All GCP accounts come with a preconfigured VPC Network named `default`, which will be used if you do not specify a different network.
   To use a different VPC network for your cluster, create one by following these instructions for Create and Manage VPC Networks[358]. Then specify the `--network <new_vpc_network_name>` option on the create cluster command below. Follow the link for more information on GCP Cloud Nat[359] and network flag.

358 https://cloud.google.com/vpc/docs/create-modify-vpc-networks#create-auto-network

359 https://github.com/kubernetes-sigs/cluster-api-provider-gcp/blob/3406adaae0f8f65a615844e55d856d306eddacc1/docs/book/src/topics/prerequisites.md#cloud-nat

- Ensure your subnets[360][361] do not overlap with your host subnet because they cannot be changed after cluster creation. If you need to change the kubernetes subnets, you must do this at cluster creation. The default subnets used in DKP are:

```
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 192.168.0.0/16
    services:
      cidrBlocks:
      - 10.96.0.0/12
```

4. Create a Kubernetes cluster. The following example shows a common configuration. See dkp create cluster gcp (see page 1469) reference for the full list of cluster creation options:

```
dkp create cluster gcp \
--cluster-name=${CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--with-gcp-bootstrap-credentials=true
--project=${GCP_PROJECT} \
--image=${IMAGE_NAME}
--self-managed
```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

5. Wait for the cluster control-plane to be ready:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=20m
```

6. After the objects are created on the API server, the Cluster API controllers reconcile them. They create infrastructure and machines. As they progress, they update the Status of each object. Konvoy provides a command to describe the current status of the cluster:

```
dkp describe cluster -c ${CLUSTER_NAME}
```

360 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/296452108
361 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/296452108

```
NAME                                                                READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/gcp-example                                                 True
52s
├──ClusterInfrastructure - GCPCluster/gcp-example
├──ControlPlane - KubeadmControlPlane/gcp-example-control-plane
True                     52s
│ ├──Machine/gcp-example-control-plane-6fbzn
True                     2m32s
│ │ └──MachineInfrastructure - GCPMachine/gcp-example-control-plane-62g6s
│ ├──Machine/gcp-example-control-plane-jf6s2
True                     7m36s
│ │ └──MachineInfrastructure - GCPMachine/gcp-example-control-plane-bsr2z
│ └──Machine/gcp-example-control-plane-mnbfs
True                     54s
│   └──MachineInfrastructure - GCPMachine/gcp-example-control-plane-s8xsx
└──Workers
   └──MachineDeployment/gcp-example-md-0
True                     78s
     ├──Machine/gcp-example-md-0-68b86fddb8-8glsw
True                     2m49s
     │ └──MachineInfrastructure - GCPMachine/gcp-example-md-0-zls8d
     ├──Machine/gcp-example-md-0-68b86fddb8-bvbm7
True                     2m48s
     │ └──MachineInfrastructure - GCPMachine/gcp-example-md-0-5zcvc
     ├──Machine/gcp-example-md-0-68b86fddb8-k9499
True                     2m49s
     │ └──MachineInfrastructure - GCPMachine/gcp-example-md-0-k8h5p
     └──Machine/gcp-example-md-0-68b86fddb8-l6vfb
True                     2m49s
       └──MachineInfrastructure - GCPMachine/gcp-example-md-0-9h5vn
```

- A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing. As part of the underlying processing using the `--self-managed` flag, the DKP CLI:
    - creates a bootstrap cluster
    - creates a workload cluster
    - moves CAPI controllers from the bootstrap cluster to the workload cluster, making it self-managed
    - deletes the bootstrap cluster

To understand how this process works step by step, you can find customizable steps in GCP Infrastructure (see page 1199) under Additional Infrastructure Configuration.

5.15.1.4.2.1 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1360).

5.15.1.4.2.2 Next Steps:

GCP: Install Kommander (see page 502)

5.15.1.5 GCP: Install Kommander

You have installed the Konvoy component and created a cluster. Now it is time to Install Kommander which will allow you to access the UI and attach new or existing clusters to monitor.

5.15.1.5.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 119).
- Ensure you have a default StorageClass (see page 1254).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

5.15.1.5.2 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1227) for the deployment:

   ```
   dkp install kommander --init > kommander.yaml
   ```

4. *If required:* Customize your `kommander.yaml`.
   See Kommander customizations (see page 1227) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1271)), etc.

5.15.1.5.3 Enable DKP Catalog Applications and Install Kommander

If you have an Enterprise license and would like to use the DKP Catalog Applications function, follow these steps:

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.5.0
```

⚠ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.co
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 99).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

5.15.1.5.4 Next Step:

GCP: Verify Install and Log in the UI (see page 504)

5.15.1.6 GCP: Verify Install and Log in the UI

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander. After the CLI successfully installs the components, it will wait for all applications to be ready by default.

> **NOTE**: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Released helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Released` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
```

```
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

5.15.1.6.1 Failed HelmReleases

If any application fails to successfully deploy, you can check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a "broken" release state such as "exhausted" or "another rollback/release in progress", you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 524). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 524):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

5.15.1.6.2 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 510) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

5.15.1.6.3 Next Step:

GCP: Subsequent New Clusters (see page 506)

5.15.1.7 GCP: Create a Managed Cluster Using the DKP CLI

Use this procedure to create GCP Managed clusters using the DKP command line interface (CLI)

> ⚠ When creating Managed clusters (see page 97), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 97)!

5.15.1.7.1 Choose a Workspace for the New Cluster

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a Workspace[362].

   ```
   kubectl get workspace -A
   ```

362 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919315/Workspaces#Create-a-Workspace

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

```
export WORKSPACE_NAMESPACE=<workspace_namespace>
```

5.15.1.7.2 Name your Cluster

1. Give your cluster a unique name suitable for your environment.

   In GCP it is critical that the name is unique, as no two clusters in the same GCP account can have the same name.

2. Set the environment variable:

```
export CLUSTER_NAME=<gcp-additional>
```

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[363] for more naming information.

To increase Docker Hub's rate limit[364], use your Docker Hub credentials when creating the cluster by setting the following flag, `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username=<username> --registry-mirror-password=<password>` on the `dkp create cluster` command.

5.15.1.7.3 Create a New GCP Cluster

Availability zones (AZs) are isolated locations within data center regions from which public cloud services originate and operate. Because all the nodes in a node pool are deployed in a single Availability Zone, you may wish to create additional node pools is to ensure your cluster has nodes deployed in multiple Availability Zones.

363 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/
364 https://docs.docker.com/docker-hub/download-rate-limit/

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single zone. You may create additional node pools in other zones with the `dkp create nodepool` command. The default region for the availability zones is `us-west1`.

Google Cloud Platform does not publish images. You must first build the image using Konvoy Image Builder (see page 1282).

1. Create an image using Konvoy Image Builder (KIB) (see page 1282) and then export the image name:

   ```
   export IMAGE_NAME=projects/${GCP_PROJECT}/global/images/<image_name_from_kib>
   ```

2. Create a Kubernetes cluster. The following example shows a common configuration. See dkp create cluster gcp (see page 1469) reference for the full list of cluster creation options:

   ```
   dkp create cluster gcp \
   --cluster-name=${CLUSTER_NAME} \
   --additional-tags=owner=$(whoami) \
   --project=${GCP_PROJECT} \
   --image=${IMAGE_NAME} \
   --namespace=${WORKSPACE_NAMESPACE}
   ```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

5.15.1.7.4 Manually Attach a DKP CLI Cluster to the Management Cluster

1. Find out the `name` of the created `Cluster`, so you can reference it later:

   ```
   kubectl -n <workspace_namespace> get clusters
   ```

2. Attach the cluster by creating a `KommanderCluster`:

   ```
   cat << EOF | kubectl apply -f -
   apiVersion: kommander.mesosphere.io/v1beta1
   kind: KommanderCluster
   ```

```
metadata:
  name: <cluster_name>
  namespace: <workspace_namespace>
spec:
  kubeconfigRef:
    name: <cluster_name>-kubeconfig
  clusterRef:
    capiCluster:
      name: <cluster_name>
EOF
```

5.15.1.7.5 Next Step:

Day 2 - Cluster Operations Management (see page 510)

6 Day 2 - Cluster Operations Management

Use these sections to manage your DKP environment.

6.1 Deploy a Sample Application

6.1.1 Learn how to deploy a sample application on a DKP cluster

After you have a basic DKP cluster installed and ready to use, you might want to test operations by deploying a simple, sample application. This task is **optional** and is only intended to demonstrate the basic steps for deploying applications in a production environment. If you are configuring the DKP cluster for a production deployment, you can use this section to learn the deployment process. However, deploying applications on a production cluster typically involves more planning and custom configuration than covered in this example.

This tutorial shows how to deploy a simple application that connects to the `redis` service. The sample application used in this tutorial is a condensed form of the Kubernetes sample guestbook[365] application.

6.1.2 Before You Begin

You must have a DKP cluster running (see page 1282).

Before running the commands below, ensure that your `kubectl` configuration references the DKP cluster on which you want to install the application. You can do this by setting the `KUBECONFIG` environment variable to the appropriate kubeconfig file's location.

365 https://kubernetes.io/docs/tutorials/stateless-application/guestbook/

6.1.3 Deploy a Sample Application

1. Deploy the Redis pods and service by running the following commands:

```
kubectl apply -f https://k8s.io/examples/application/guestbook/redis-leader-deployment.yaml
kubectl apply -f https://k8s.io/examples/application/guestbook/redis-leader-service.yaml
```

2. Deploy Redis followers. The leader deployment created above is a single pod. Adding followers (or replicas) makes it highly available to meet greater traffic demands. You must then setup the guestbook application to communicate with the Redis followers to read the data. To do this, set up another service (the `redis-follower-service.yaml` below). Do this by running the following commands:

```
kubectl apply -f https://k8s.io/examples/application/guestbook/redis-follower-deployment.yaml
kubectl apply -f https://k8s.io/examples/application/guestbook/redis-follower-service.yaml
```

3. Deploy the web app frontend by running the following command:

```
kubectl apply -f https://k8s.io/examples/application/guestbook/frontend-deployment.yaml
```

4. Confirm that there are three frontend replicas running:

```
kubectl get pods -l app=guestbook -l tier=frontend
```

5. Apply the frontend Service:

```
kubectl apply -f https://k8s.io/examples/application/guestbook/frontend-service.yaml
```

6. Configure the frontend Service to use a cloud load balancer:

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
```

```
    app: guestbook
    tier: frontend
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: guestbook
    tier: frontend
EOF
```

7. View the frontend service via the LoadBalancer by running the following command to get the IP address for the frontend Service:

```
kubectl get service frontend
```

8. the external IP address, and load the page in your browser to view your guestbook.

   The service properties provide the name of the load balancer. You can connect to the application by accessing that load balancer address in your web browser. Because this sample deployment creates a **cloud load balancer**, you should keep in mind that creating the load balancer can take up to a few minutes. You also might experience a slight delay before it is running properly due to DNS propagation and synchronization.

9. Remove the sample application by running the following commands:

```
kubectl delete deployment -l app=redis
kubectl delete service -l app=redis
kubectl delete deployment frontend
kubectl delete service frontend
```

WARNING: This step is **required** because the sample deployment attaches a **cloud provider load balancer** to the DKP cluster. Therefore, you **must delete** the sample application before tearing down the cluster.

6.1.4 Related Information

- Example: Deploying PHP Guestbook application with Redis[366]

366 https://kubernetes.io/docs/tutorials/stateless-application/guestbook/

6.2 Operations

Manage your cluster and deployed applications using platform applications

After you deploy a DKP cluster and the platform applications you want to use, you are ready to begin managing cluster operations and their application workloads to optimize your organization's productivity.

In most cases, a production cluster requires additional advanced configuration tailored for your environment, ongoing maintenance, authentication and authorization, and other common activities. For example, it is important to monitor cluster activity and collect metrics to ensure application performance and response time, evaluate network traffic patterns, manage user access to services, and verify workload distribution and efficiency.

- Access Control (see page 513)
- Identity Providers (see page 524)
- Infrastructure Providers (see page 530)

6.2.1 Access Control

6.2.1.1 Centrally Manage Access Across Clusters

You can centrally define role-based authorization within DKP UI to control resource access on the management cluster and a set, or all, of the target clusters. These resources are similar to Kubernetes RBAC but with crucial differences, and they make it possible to define the roles and role bindings once, and have them federated to clusters within a given scope.

DKP UI has two conceptual groups of resources that are used to manage access control:

- Kommander Roles: control access to resources on the management cluster.
- Cluster Roles: control access to resources on all target clusters in scope.

Use these two groups of resources to manage access control within 3 levels of scope:

Context	Kommander Roles	Cluster Roles
Global	Create ClusterRoles on the management cluster.	Federates ClusterRoles on all target clusters across all workspaces.
Workspace	Create namespaced Roles on the management cluster in the workspace namespace.	Federates ClusterRoles on all target clusters in the workspace.
Project	Create namespaced Roles on the management cluster in the project namespace.	Federates namespaced Roles on all target clusters in the project in the project namespace.

The role bindings (see page 516) for each level and type create `RoleBindings` or `ClusterRoleBindings` on the clusters that apply to each category.

This approach gives you maximum flexibility over who has access to what resources, conveniently mapped to your existing identity providers' claims.

6.2.1.2 Special Limitation for Kommander Roles

In addition to granting a Kommander Role, you must also grant the appropriate DKP role to allow external users and groups into the UI. See RBAC - DKP UI Authorization (see page 517) for details about the built-in DKP roles. Here are examples of `ClusterRoleBindings` that grant an IDP group admin access to the Kommander routes:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eng-kommander-dashboard
  labels:
    "workspaces.kommander.mesosphere.io/rbac": ""
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: dkp-kommander-admin
subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: Group
    name: oidc:engineering
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eng-dkp-routes
  labels:
    "workspaces.kommander.mesosphere.io/rbac": ""
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: dkp-admin
subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: Group
    name: oidc:engineering
```

6.2.1.3 Types of Access Control Objects

Kubernetes role-based access control can be controlled with three different object categories: Groups, Roles and Policies, as explained in more detail below.

6.2.1.3.1 Groups

You can map group and user claims made by your configured identity providers to Kommander groups by selecting Administration / Identity providers in the left sidebar in the global workspace level, and then selecting the **Groups** tab.

6.2.1.3.2 Roles

`ClusterRoles` are named collections of rules defining which verbs can be applied to which resources.

- Kommander Roles apply specifically to resources on the management cluster.
- Cluster Roles apply to target clusters within their scope at these levels:
    - Global level - this is all target clusters in all workspaces,
    - Workspace level - this is all target clusters in the workspace,
    - Project level - this is all target clusters that have been added to the project.

6.2.1.3.3 Propagate Workspace Roles to Projects

By default, users granted the Kommander Workspace Admin, Edit, or View roles will also be granted the equivalent Kommander Project Admin, Edit, or View role for any project created in the workspace. Other workspace roles are not automatically propagated to the equivalent role for a project in the workspace.

Each workspace has roles defined using `KommanderWorkspaceRole` resources. Automatic propagation is controlled using the annotation `"workspace.kommander.mesosphere.io/sync-to-project": "true"` on a `KommanderWorkspaceRole` resource. You can manage this only by using the CLI.

```
kubectl get kommanderworkspaceroles -n test-qznrn-6sz52
```

```
NAME                         DISPLAY NAME                      AGE
kommander-workspace-admin    Kommander Workspace Admin Role    2m18s
kommander-workspace-edit     Kommander Workspace Edit Role     2m18s
kommander-workspace-view     Kommander Workspace View Role     2m18s
```

To prevent propagation of the `kommander-workspace-view` role, remove this annotation from the `KommanderWorkspaceRole` resource.

```
kubectl annotate kommanderworkspacerole -n test-qznrn-6sz52 kommander-workspace-view
workspace.kommander.mesosphere.io/sync-to-project-
```

To enable propagation of the role, add this annotation to the relevant `KommanderWorkspaceRole` resource.

```
kubectl annotate kommanderworkspacerole -n test-qznrn-6sz52 kommander-workspace-view
workspace.kommander.mesosphere.io/sync-to-project=true
```

6.2.1.3.4 Special Limitation for Workspace > Project Role Inheritance

When granting users access to a workspace, you must manually grant access to the projects within that workspace. Each project is created with a set of admin/edit/view roles, and you can choose to add an additional `RoleBinding` to each group or user of the workspace for one of these project roles. Usually these are prefixed `kommander-project-(admin/edit/view)`. Here is an example `RoleBinding` that grants the Kommander Project Admin role access for the project namespace to the engineering group:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: workspace-admin-project1-admin
  namespace: my-project-namespace-xxxxx
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: kommander-project-admin-xxxxx
subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: Group
    name: oidc:engineering
```

6.2.1.3.5 Role Bindings

Kommander role bindings, cluster role bindings, and project role bindings bind a Kommander group to any number of roles. All groups defined in the **Groups** tab will be present at the global, workspace, or project level, and are ready for you to assign roles to them.

6.2.1.4 Related Information

- Project Role Bindings (see page 651)
- Workspace Role Bindings (see page 608)
- Kommander RBAC Tutorial (see page 517)
- RBAC - DKP UI Authorization (see page 517)
- Kubernetes RBAC Authorization[367]

367 https://kubernetes.io/docs/reference/access-authn-authz/rbac/

6.2.1.5 Granting Access to Kubernetes and Kommander Resources

6.2.1.5.1 Grant access to Kommander and Kubernetes resources using RBAC

6.2.1.5.2 Granting Access to External Users

Users and groups from an external identity provider initially have no access to kubernetes resources. Privileges must be granted explicitly by interacting with the RBAC API. This section provides some basic examples for general usage. More information about the RBAC API can be found in the Kubernetes documentation[368].

6.2.1.5.2.1 The Basics

Kubernetes does not provide an identity database for standard users. Users and group membership must be provided by a trusted identity provider. In Kubernetes, RBAC policies are additive, which means that a subject (user, group, or service account) is denied access to a resource unless explicitly granted access by a cluster administrator. You can grant access by binding a subject to a role, which grants some level of access to one or more resources. Kubernetes ships with some default roles[369], which aid in creating broad access control policies.

For example, if you want to make `mary@example.com` a cluster administrator, bind her username to the `cluster-admin` default role as follows:

```
cat << EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: mary-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: mary@example.com
EOF
```

This user now has the highest level of access which can be achieved. Use the `cluster-admin` role and `system:masters` group sparingly.

368 https://kubernetes.io/docs/reference/access-authn-authz/rbac/
369 https://kubernetes.io/docs/reference/access-authn-authz/rbac/#default-roles-and-role-bindings

6.2.1.5.2.2 Restricting a User to Namespace

A more common example would be to grant a user access to a specific namespace, by creating a RoleBinding (RoleBindings are namespaced scoped). For example, to make the user `bob@example.com` a *reader* of the `baz` namespace, bind the user to the `view` role:

```
cat << EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: bob-view
  namespace: baz
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: view
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: bob@example.com
EOF
```

The user can now perform non-destructive operations targeting resources in the `baz` namespace only.

6.2.1.5.2.3 Groups

If your external identity provider supports group claims, you can also bind groups to roles. To make the `devops` LDAP group administrators of the `production` namespace bind the group to the `admin` role:

```
cat << EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: devops-admin
  namespace: production
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: oidc:devops
EOF
```

One important distinction from adding users is that all external groups are prefixed with `oidc:`, so a group name becomes `oidc:devops`. This prevents collision with locally defined groups.

6.2.1.5.3 DKP UI Authorization

The DKP UI, and other HTTP applications protected by Kommander forward authentication, are also authorized by the Kubernetes RBAC API. In addition to Kubernetes API resources, it is possible to define rules which map to HTTP URIs and HTTP verbs. Kubernetes RBAC calls these `nonResourceURLs`, Kommander forward authentication uses these rules to grant or deny access to HTTP endpoints.

6.2.1.5.3.1 Default Roles

Roles have been created for granting access to the dashboard and select applications which expose an HTTP server through the ingress controller. The `cluster-admin` role is actually a system role that defines grants permission to all actions (verbs) on any resource; including non-resource URLs. The default dashboard user is bound to this role.

> Granting user `admin` privileges on `/dkp/*` grants `admin` privileges to all sub-resources, even if bindings exist for sub-resources with less privileges.

Dashboard	Role	Path	access
*	cluster-admin	*	read, write, delete
kommander	dkp-view	/dkp/*	read
kommander	dkp-edit	/dkp/*	read, write
kommander	dkp-admin	/dkp/*	read, write, delete
kommander-dashboard	dkp-kommander-view	/dkp/kommander/dashboard/*	read
kommander-dashboard	dkp-kommander-edit	/dkp/kommander/dashboard/*	read, write
kommander-dashboard	dkp-kommander-admin	/dkp/kommander/dashboard/*	read, write, delete

Dashboard	Role	Path	access
alertmanager	dkp-kube-prometheus-stack-alertmanager-view	/dkp/alertmanager/*	read
alertmanager	dkp-kube-prometheus-stack-alertmanager-edit	/dkp/alertmanager/*	read, write
alertmanager	dkp-kube-prometheus-stack-alertmanager-admin	/dkp/alertmanager/*	read, write, delete
centralized-grafana	dkp-centralized-grafana-grafana-view	/dkp/kommander/monitoring/grafana/*	read
centralized-grafana	dkp-centralized-grafana-grafana-edit	/dkp/kommander/monitoring/grafana/*	read, write
centralized-grafana	dkp-centralized-grafana-grafana-admin	/dkp/kommander/monitoring/grafana/*	read, write, delete
centralized-kubecost	dkp-centralized-kubecost-view	/dkp/kommander/kubecost/*	read
centralized-kubecost	dkp-centralized-kubecost-edit	/dkp/kommander/kubecost/*	read, write
centralized-kubecost	dkp-centralized-kubecost-admin	/dkp/kommander/kubecost/*	read, write, delete
grafana	dkp-kube-prometheus-stack-grafana-view	/dkp/grafana/*	read
grafana	dkp-kube-prometheus-stack-grafana-edit	/dkp/grafana/*	read, write
grafana	dkp-kube-prometheus-stack-grafana-admin	/dkp/grafana/*	read, write, delete
grafana-logging	dkp-grafana-logging-view	/dkp/logging/grafana/*	read

Dashboard	Role	Path	access
grafana-logging	dkp-grafana-logging-edit	/dkp/logging/grafana/*	read, write
grafana-logging	dkp-grafana-logging-admin	/dkp/logging/grafana/*	read, write, delete
karma	dkp-karma-view	/dkp/kommander/monitoring/karma/*	read
karma	dkp-karma-edit	/dkp/kommander/monitoring/karma/*	read, write
karma	dkp-karma-admin	/dkp/kommander/monitoring/karma/*	read, write, delete
kubernetes-dashboard	dkp-kubernetes-dashboard-view	/dkp/kubernetes/*	read
kubernetes-dashboard	dkp-kubernetes-dashboard-edit	/dkp/kubernetes/*	read, write
kubernetes-dashboard	dkp-kubernetes-dashboard-admin	/dkp/kubernetes/*	read, write, delete
prometheus	dkp-kube-prometheus-stack-prometheus-view	/dkp/prometheus/*	read
prometheus	dkp-kube-prometheus-stack-prometheus-edit	/dkp/prometheus/*	read, write
prometheus	dkp-kube-prometheus-stack-prometheus-admin	/dkp/prometheus/*	read, write, edit
traefik	dkp-traefik-view	/dkp/traefik/*	read
traefik	dkp-traefik-edit	/dkp/traefik/*	read, edit
traefik	dkp-traefik-admin	/dkp/traefik/*	read, edit, delete

Dashboard	Role	Path	access
thanos	dkp-thanos-query-view	/dkp/kommander/monitoring/query/*	read
thanos	dkp-thanos-query-edit	/dkp/kommander/monitoring/query/*	read, write
thanos	dkp-thanos-query-admin	/dkp/kommander/monitoring/query/*	read, write, delete

This section provides a few examples of binding subjects to the default roles defined for the DKP UI endpoints.

6.2.1.5.3.2 Examples

User

To grant the user `mary@example.com` administrative access to all Kommander resources, bind the user to the `dkp-admin` role:

```
cat << EOF | kubectl apply -f -
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: dkp-admin-mary
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: dkp-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: mary@example.com
EOF
```

If you inspect the role, you see what access is now granted:

```
kubectl describe clusterroles dkp-admin
```

```
Name:         dkp-admin
```

```
Labels:       app.kubernetes.io/instance=kommander
              app.kubernetes.io/managed-by=Helm
              app.kubernetes.io/version=v2.0.0
              helm.toolkit.fluxcd.io/name=kommander
              helm.toolkit.fluxcd.io/namespace=kommander
              rbac.authorization.k8s.io/aggregate-to-admin=true
Annotations:  meta.helm.sh/release-name: kommander
              meta.helm.sh/release-namespace: kommander
PolicyRule:
  Resources  Non-Resource URLs  Resource Names  Verbs
  ---------  -----------------  --------------  -----
             [/dkp/*]           []              [delete]
             [/dkp]             []              [delete]
             [/dkp/*]           []              [get]
             [/dkp]             []              [get]
             [/dkp/*]           []              [head]
             [/dkp]             []              [head]
             [/dkp/*]           []              [post]
             [/dkp]             []              [post]
             [/dkp/*]           []              [put]
             [/dkp]             []              [put]
```

The user can now use the HTTP verbs HEAD, GET, DELETE, POST, and PUT when accessing any URL at or under `/dkp`. Provided the downstream application follows REST conventions, this effectively allows read, edit, and delete privileges.

NOTE: To allow users to access the DKP UI, ensure they have the appropriate `dkp-kommander-` role in addition to the Kommander roles granted in the DKP UI. For more information, see the Access Control section of the Kommander documentation (see page 513).

Group

In order to grant view access to the `/dkp/*` endpoints and edit access to the grafana logging endpoint to group `logging-ops`, create the following ClusterRoleBindings:

```
cat << EOF | kubectl apply -f -
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: dkp-view-logging-ops
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: dkp-view
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: oidc:logging-ops
```

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: dkp-logging-edit-logging-ops
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: dkp-logging-edit
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: oidc:logging-ops
EOF
```

Note: external groups must be prefixed by `oidc:`

Members of `logging-ops` are now able to `view` all resources under `/dkp` and edit all resources under `/dkp/logging/grafana`.

6.2.1.5.4 Accessing the Kubernetes Dashboard

The Kubernetes dashboard offloads authorization directly to the Kubernetes API server. Once authenticated, all users may access the dashboard at `/dkp/kubernetes/` without needing a `dkp` role. However, access to the underlying kubernetes resources exposed by the dashboard are protected by the cluster RBAC policy.

6.2.1.5.5 Further Reading

This page has provides some basic examples of operations which provide the building blocks of creating an access control policy. For more information about creating your own roles and advanced policies, we highly recommend reading the Kubernetes RBAC documentation[370].

6.2.2 Identity Providers

6.2.2.1 Grant access to users in your organization

DKP supports GitHub, LDAP (see page 527), SAML and standard OIDC identity providers such as Google. These identity management providers support the login and authentication process for DKP and your Kubernetes clusters. You can configure as many identity providers as you want and users can select from any method when logging in.

6.2.2.2 Prerequisites

To get started with DKP, you must:

370 https://kubernetes.io/docs/reference/access-authn-authz/rbac/

- Log in to the UI (see page 1280)

6.2.2.2.1 Limit Access

- The GitHub (see page 525) provider allows you to specify any of the organizations and teams are eligible for access.
- The LDAP (see page 527) provider allows you to configure search filters for either users or groups.
- The OIDC provider cannot limit users based on identity.
- The SAML provider allows users to log in using a single sign-on (SSO) profile.

6.2.2.2.2 Configure an Identity Provider via the UI

1. From the drop down, select the **Global** workspace.
2. Select **Administration > Identity Providers**.
3. Select the **Identity Providers** tab.
4. Select **+ Add Identity Provider**.
5. Select an identity provider and complete the form field with the relevant details.
6. Select **Save** to create your Identity Provider.

6.2.2.2.3 Temporarily Disabling a Provider

Select the three dot button on the Identity Providers table and select **Disable** from the drop-down menu. The provider option no longer appears on the login screen.

6.2.2.3 Groups

Access control groups are configured in the Groups tab of the Identity Providers page. See Access Control (see page 513) for an overview of groups in DKP.

6.2.2.4 Authorize a Group in Github

Enterprise Gov Advanced

Install GitHub as an identity provider and grant access to all developers.

6.2.2.4.1 Authorize Access to Your Clusters

Ensure every developer in your GitHub organization has access to your Kubernetes clusters. To authorize all developers to have read access to your clusters:

1. Set up GitHub as an identity provider. Start by creating a new OAuth Application in our GitHub Organization by filling out this form[371].

   Important: Use your cluster URL followed by `/dex/callback` as the Authorization callback URL.

2. After you create the application, you will be taken to a settings page. You will need the **Client ID** and **Client Secret** from this page for the DKP UI. Select the **Generate a new client secret** button if you do not already have a Client Secret for the application.
3. From the top menu bar in the DKP UI, select the **Global** workspace.
4. Select **Identity Providers** in the **Administration** section of the sidebar menu.
5. Select the **Identity Providers** tab, and then select the **+ Add Identity Provider** button.
6. Ensure GitHub is selected as the identity provider type, and copy the **Client ID** and **Client Secret** values into the form.
7. Select **Save** to create your Identity Provider.

D2iQ configured the identity provider to load all groups, so you must map these groups to the Kubernetes groups.

6.2.2.4.1.1 Map the Identity Provider Groups to the Kubernetes Groups

1. Select the **Groups** tab, and then select the **Create Group** button.
2. Give your group a descriptive name and add the groups from your GitHub provider under **Identity Provider Groups**.
3. Click **Save** to create the group, which creates it on the management cluster and federated to all target clusters, and also describes the developers for your organization.

To enable this group, you need to first create a role which allows you to view every resource.

6.2.2.4.1.2 Create a “Read Everything” Role

1. Select **Access Control** in the **Administration** section of the sidebar menu.
2. Select the **Cluster Roles** tab, and then select the **+ Create Role** button.
3. Give the role a descriptive name, and ensure that **Cluster Role** is selected as the type.
4. For a read-only role, select **+ Add Rule**, then select **All Resource Types** in the **Resources** input, and select the **get**, **list**, and **watch** verbs.

Now you can assign the “Read Everything” role to the developers group.

6.2.2.4.1.3 Assign the Role to the Developers Group

1. Select the **Cluster Role Bindings** tab, and then select the **Add roles** button for your group.

---

371 https://github.com/settings/applications/new

2. Select “Read Everything” role from the **Roles** drop-down.

Lastly, follow the example in the Access Control documentation (see page 513) to grant users access to Kommander routes on your cluster.

When you check your attached clusters and login as a user from your matched groups, you can see every resource, but neither delete or edit them, as intended.

6.2.2.5 External LDAP directory

6.2.2.5.1 How to connect your cluster to an external LDAP directory

This guide shows you how to configure your DKP cluster so that users can log in with the credentials stored in an external LDAP directory service.

6.2.2.5.2 Step 1: Add LDAP connector

Each LDAP directory is set up in its own specific manner, so these steps are important. The LDAP authentication mechanism can be added using the CLI or the UI in Kommander.

The following example does not cover all possible configurations. Refer to the Dex LDAP connector reference documentation[372] for more details.

The example below configures a DKP cluster to connect to the Online LDAP Test Server[373] and for demonstration purposes, the configuration shown uses `insecureNoSSL: true`. In production, you should protect LDAP communication with a properly-configured transport layer security (TLS). When using TLS, the admin can add `insecureSkipVerify: true` to `spec.ldap` to skip server certificate verification, if needed.

Below are steps for the CLI followed by UI in the event you prefer to use it.

1. Create a YAML file (`ldap.yaml`) similar to the following:

```
apiVersion: v1
kind: Secret
metadata:
  name: ldap-password
  namespace: kommander
type: Opaque
stringData:
  password: password
```

372 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/ldap.md
373 https://www.forumsys.com/tutorials/integration-how-to/ldap/online-ldap-test-server/

```
---
apiVersion: dex.mesosphere.io/v1alpha1
kind: Connector
metadata:
  name: ldap
  namespace: kommander
spec:
  enabled: true
  type: ldap
  displayName: LDAP Test
  ldap:
    host: ldap.forumsys.com:389
    insecureNoSSL: true
    bindDN: cn=read-only-admin,dc=example,dc=com
    bindSecretRef:
      name: ldap-password
    userSearch:
      baseDN: dc=example,dc=com
      filter: "(objectClass=inetOrgPerson)"
      username: uid
      idAttr: uid
      emailAttr: mail
    groupSearch:
      baseDN: dc=example,dc=com
      filter: "(objectClass=groupOfUniqueNames)"
      userMatchers:
      - userAttr: DN
        groupAttr: uniqueMember
      nameAttr: ou
```

NOTE: The value for the LDAP connector `name` parameter (here: `LDAP Test`) appears on one of the login buttons in the DKP UI. You should choose an expressive name.

2. Run the following command to deploy the LDAP connector.

```
kubectl apply -f ldap.yaml
```

NOTE: UI - Deploy LDAP through **Add Identity Provider** screen within the UI.

3. Retrieve a list of connectors:

```
kubectl get connector.dex.mesosphere.io -A
```

4. Run the following command to verify that the LDAP connector was created successfully:

```
kubectl get Connector.dex.mesosphere.io -n kommander <LDAP-CONNECTOR-NAME> -o
yaml
```

6.2.2.5.3 Step 2: Log in

1. Visit `https://<YOUR-CLUSTER-HOST>/token` and initiate a login flow.
2. On the login page choose the `Log in with <ldap-name>` button.
3. Enter the LDAP credentials, and log in.

> UI - While LDAP authentication has been enabled, additional access rights will need to be configured through the Add Identity Provider screen in the UI using the documentation for Granting Access to Kubernetes and Kommander Resources (see page 517).

6.2.2.5.4 Troubleshooting

It is likely that the Dex LDAP connector configuration is not quite right from the start. In that case, you need to be able to debug the problem and iterate on it. The Dex log output contains helpful error messages as indicated by the following examples.

6.2.2.5.4.1 Errors during Dex startup

If the Dex configuration fragment provided results in an invalid Dex config, Dex does not properly start up. In that case, reviewing the Dex logs will provide error details. Use the following command to retrieve the Dex logs:

```
kubectl logs -f dex-66675fcb7c-snxb8  -n kommander
```

You may see an error similar to the following:

```
error parse config file /etc/dex/cfg/config.yaml: error unmarshaling JSON: parse
connector config: illegal base64 data at input byte 0
```

Another reason for Dex not starting up correctly is that `https://<YOUR-CLUSTER-HOST>/token` throws a 5xx HTTP error response after timing out.

6.2.2.5.4.2 Errors upon login

Most problems with the Dex LDAP connector configuration become apparent only after a login attempt. A login failing from misconfiguration will result in an error page showing only `Internal Server Error` and `Login error`. You can then usually find the root cause by reading the Dex log, as shown in the following example:

```
kubectl logs -f dex-5d55b6b94b-9pm2d -n kommander
```

You can look for output similar to this example:

```
[...]
time="2019-07-29T13:03:57Z" level=error msg="Failed to login user: failed to connect:
LDAP Result Code 200 \"Network Error\": dial tcp: lookup freeipa.example.com on
10.255.0.10:53: no such host"
```

Here, the directory's DNS name was misconfigured, which should be easy to address.

A more difficult problem occurs when a login through Dex through LDAP fails because Dex was not able to find the specified user unambiguously in the directory. That could be the result of an invalid LDAP user search configuration. Here's an example error message from the Dex log:

```
time="2019-07-29T14:21:27Z" level=info msg="performing ldap search
cn=users,cn=compat,dc=demo1,dc=freeipa,dc=org sub (&(objectClass=posixAccount)
(uid=employee))"
time="2019-07-29T14:21:27Z" level=error msg="Failed to login user: ldap: filter
returned multiple (2) results: \"(&(objectClass=posixAccount)(uid=employee))\""
```

Solving problems like this requires you to review the directory structure carefully. (Directory structures can be very different between different LDAP setups.) Then you must carefully assemble a user search configuration matching the directory structure.

Notably, with some directories, it can be hard to distinguish between the cases "properly configured, and user not found" (login fails in an expected way) and "not properly configured, and therefore user not found" (login fails in an unexpected way).

6.2.2.5.4.3 Example for successful login

For comparison, here are some sample log lines issued by Dex after successful login:

```
time="2019-07-29T15:35:51Z" level=info msg="performing ldap search
cn=accounts,dc=demo1,dc=freeipa,dc=org sub (&(objectClass=posixAccount)
(uid=employee))"
time="2019-07-29T15:35:52Z" level=info msg="username \"employee\" mapped to entry
uid=employee,cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org"
time="2019-07-29T15:35:52Z" level=info msg="login successful: connector \"ldap\",
username=\"\", email=\"employee@demo1.freeipa.org\", groups=[]"
```

6.2.3 Infrastructure Providers

Enterprise Gov Advanced

Managing infrastructure providers used by Kommander

Infrastructure providers, like AWS, provide the infrastructure for your Management clusters. To automate their provisioning, Kommander needs authentication keys for your preferred infrastructure provider. You may have many accounts for a single infrastructure provider. Kommander needs authentication keys for your preferred infrastructure provider.

To provision new clusters and manage them, Kommander needs infrastructure provider credentials. Currently, AWS is supported.

Infrastructure provider credentials are configured in each workspace. The name you assign must be unique across all namespaces in your cluster.

6.2.3.1 View and Modify Infrastructure Providers

1. From the top menu bar, select your target workspace.
2. Select **Infrastructure Providers** in the **Administration** section of the sidebar menu.

6.2.3.2 AWS

- Configure AWS Provider with Role Credentials (see page 531) (Recommended if using AWS)
- Configure AWS Provider with Static Credentials (see page 538)

6.2.3.3 Delete an infrastructure provider

Before deleting an infrastructure provider, Kommander verifies if any existing managed clusters were created using this provider. The infrastructure provider cannot be deleted until all clusters created with the infrastructure provider have been deleted. This ensures Kommander has access to your infrastructure provider to remove all resources created for a managed cluster.

6.2.3.4 Configure an AWS Provider with a User Role

Enterprise Gov Advanced

Configure your provider to add resources to your AWS account

We highly recommend using the role-based method as this is more secure.

The role authentication method can only be used if your management cluster is running in AWS.

For more flexible credential configuration, we offer a role-based authentication[374] method with an optional External ID for third party access.

6.2.3.4.1 Create a Role Manually

The role should grant permissions to create the following resources in the AWS account:

- EC2 Instances
- VPC
- Subnets
- Elastic Load Balancer (ELB)
- Internet Gateway
- NAT Gateway
- Elastic Block Storage (EBS) Volumes
- Security Groups
- Route Tables
- IAM Roles

The user you delegate from your role must have a minimum set of permissions. Below is the minimal IAM policy required:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:AllocateAddress",
        "ec2:AssociateRouteTable",
        "ec2:AttachInternetGateway",
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:CreateInternetGateway",
        "ec2:CreateNatGateway",
        "ec2:CreateRoute",
        "ec2:CreateRouteTable",
        "ec2:CreateSecurityGroup",
        "ec2:CreateSubnet",
        "ec2:CreateTags",
        "ec2:CreateVpc",
        "ec2:ModifyVpcAttribute",
        "ec2:DeleteInternetGateway",
        "ec2:DeleteNatGateway",
```

374 https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html

```
        "ec2:DeleteRouteTable",
        "ec2:DeleteSecurityGroup",
        "ec2:DeleteSubnet",
        "ec2:DeleteTags",
        "ec2:DeleteVpc",
        "ec2:DescribeAccountAttributes",
        "ec2:DescribeAddresses",
        "ec2:DescribeAvailabilityZones",
        "ec2:DescribeInstances",
        "ec2:DescribeInternetGateways",
        "ec2:DescribeImages",
        "ec2:DescribeNatGateways",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeNetworkInterfaceAttribute",
        "ec2:DescribeRouteTables",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "ec2:DescribeVpcAttribute",
        "ec2:DescribeVolumes",
        "ec2:DetachInternetGateway",
        "ec2:DisassociateRouteTable",
        "ec2:DisassociateAddress",
        "ec2:ModifyInstanceAttribute",
        "ec2:ModifyNetworkInterfaceAttribute",
        "ec2:ModifySubnetAttribute",
        "ec2:ReleaseAddress",
        "ec2:RevokeSecurityGroupIngress",
        "ec2:RunInstances",
        "ec2:TerminateInstances",
        "tag:GetResources",
        "elasticloadbalancing:AddTags",
        "elasticloadbalancing:CreateLoadBalancer",
        "elasticloadbalancing:ConfigureHealthCheck",
        "elasticloadbalancing:DeleteLoadBalancer",
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeLoadBalancerAttributes",
        "elasticloadbalancing:ApplySecurityGroupsToLoadBalancer",
        "elasticloadbalancing:DescribeTags",
        "elasticloadbalancing:ModifyLoadBalancerAttributes",
        "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
        "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
        "elasticloadbalancing:RemoveTags",
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeInstanceRefreshes",
        "ec2:CreateLaunchTemplate",
        "ec2:CreateLaunchTemplateVersion",
        "ec2:DescribeLaunchTemplates",
        "ec2:DescribeLaunchTemplateVersions",
        "ec2:DeleteLaunchTemplate",
        "ec2:DeleteLaunchTemplateVersions",
        "ec2:DescribeKeyPairs"
```

```
      ],
      "Resource": ["*"]
    },
    {
      "Effect": "Allow",
      "Action": [
        "autoscaling:CreateAutoScalingGroup",
        "autoscaling:UpdateAutoScalingGroup",
        "autoscaling:CreateOrUpdateTags",
        "autoscaling:StartInstanceRefresh",
        "autoscaling:DeleteAutoScalingGroup",
        "autoscaling:DeleteTags"
      ],
      "Resource": [
        "arn:*:autoscaling:*:*:autoScalingGroup:*:autoScalingGroupName/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:*:iam::*:role/aws-service-role/autoscaling.amazonaws.com/
AWSServiceRoleForAutoScaling"
      ],
      "Condition": {
        "StringLike": { "iam:AWSServiceName": "autoscaling.amazonaws.com" }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:*:iam::*:role/aws-service-role/elasticloadbalancing.amazonaws.com/
AWSServiceRoleForElasticLoadBalancing"
      ],
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "elasticloadbalancing.amazonaws.com"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:*:iam::*:role/aws-service-role/spot.amazonaws.com/
AWSServiceRoleForEC2Spot"
      ],
      "Condition": {
        "StringLike": { "iam:AWSServiceName": "spot.amazonaws.com" }
      }
    },
```

```
    {
      "Effect": "Allow",
      "Action": ["iam:PassRole"],
      "Resource": ["arn:*:iam::*:role/*.cluster-api-provider-aws.sigs.k8s.io"]
    },
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:CreateSecret",
        "secretsmanager:DeleteSecret",
        "secretsmanager:TagResource"
      ],
      "Resource": ["arn:*:secretsmanager:*:*:secret:aws.cluster.x-k8s.io/*"]
    },
    {
      "Effect": "Allow",
      "Action": ["ssm:GetParameter"],
      "Resource": ["arn:*:ssm:*:*:parameter/aws/service/eks/optimized-ami/*"]
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:*:iam::*:role/aws-service-role/eks.amazonaws.com/
AWSServiceRoleForAmazonEKS"
      ],
      "Condition": {
        "StringLike": { "iam:AWSServiceName": "eks.amazonaws.com" }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:*:iam::*:role/aws-service-role/eks-nodegroup.amazonaws.com/
AWSServiceRoleForAmazonEKSNodegroup"
      ],
      "Condition": {
        "StringLike": { "iam:AWSServiceName": "eks-nodegroup.amazonaws.com" }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:aws:iam::*:role/aws-service-role/eks-fargate-pods.amazonaws.com/
AWSServiceRoleForAmazonEKSForFargate"
      ],
      "Condition": {
        "StringLike": { "iam:AWSServiceName": "eks-fargate.amazonaws.com" }
      }
    },
```

```
{
  "Effect": "Allow",
  "Action": ["iam:GetRole", "iam:ListAttachedRolePolicies"],
  "Resource": ["arn:*:iam::*:role/*"]
},
{
  "Effect": "Allow",
  "Action": ["iam:GetPolicy"],
  "Resource": ["arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"]
},
{
  "Effect": "Allow",
  "Action": [
    "eks:DescribeCluster",
    "eks:ListClusters",
    "eks:CreateCluster",
    "eks:TagResource",
    "eks:UpdateClusterVersion",
    "eks:DeleteCluster",
    "eks:UpdateClusterConfig",
    "eks:UntagResource",
    "eks:UpdateNodegroupVersion",
    "eks:DescribeNodegroup",
    "eks:DeleteNodegroup",
    "eks:UpdateNodegroupConfig",
    "eks:CreateNodegroup",
    "eks:AssociateEncryptionConfig"
  ],
  "Resource": ["arn:*:eks:*:*:cluster/*", "arn:*:eks:*:*:nodegroup/*/*/*"]
},
{
  "Effect": "Allow",
  "Action": [
    "eks:ListAddons",
    "eks:CreateAddon",
    "eks:DescribeAddonVersions",
    "eks:DescribeAddon",
    "eks:DeleteAddon",
    "eks:UpdateAddon",
    "eks:TagResource",
    "eks:DescribeFargateProfile",
    "eks:CreateFargateProfile",
    "eks:DeleteFargateProfile"
  ],
  "Resource": ["*"]
},
{
  "Effect": "Allow",
  "Action": ["iam:PassRole"],
  "Resource": ["*"],
  "Condition": {
    "StringEquals": { "iam:PassedToService": "eks.amazonaws.com" }
```

```
      }
    },
    {
      "Effect": "Allow",
      "Action": ["kms:CreateGrant", "kms:DescribeKey"],
      "Resource": ["*"],
      "Condition": {
        "ForAnyValue:StringLike": {
          "kms:ResourceAliases": "alias/cluster-api-provider-aws-*"
        }
      }
    }
  ]
}
```

Make sure to also add a correct trust relationship[375] to the created role. This example allows everyone within the same account to `AssumeRole` with the created role. `YOURACCOUNTRESTRICTION` must be replaced with the AWS Account ID you would like to `AssumeRole` from.

IMPORTANT: Never add a `*/` wildcard. This opens your account to the whole world.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com",
        "AWS": "arn:aws:iam::YOURACCOUNTRESTRICTION:root"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

To use the role created, attach the following policy to the role which is already attached to your Managed or Attached cluster. Replace `YOURACCOUNTRESTRICTION` with the AWS Account ID where the role you like to `AssumeRole` is saved. Also, replace `THEROLEYOUCREATED` with the AWS Role name.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

375 https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html

```
      "Sid": "AssumeRoleKommander",
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::YOURACCOUNTRESTRICTION:role/THEROLEYOUCREATED"
    }
  ]
}
```

6.2.3.4.2 Create Infrastructure Provider in the UI

1. From the top menu bar, select your target workspace.
2. Select **Infrastructure Providers** in the **Administration** section of the sidebar menu.
3. Select the **Add Infrastructure Provider** button.
4. Select the **Amazon Web Services (AWS)** option.
5. Ensure **Role** is selected as the **Authentication Method**.
6. Enter a name for your infrastructure provider. Select a name that matches the AWS user.
7. Enter the **Role ARN**.
8. You can add an **External ID** if you share the Role with a 3rd party. External IDs secure your environment from accidentally used roles. Read more about External IDs[376].
9. Select **Save** to save your provider.

6.2.3.5 AWS Static Credentials

Enterprise Gov Advanced

Configuring an AWS Infrastructure Provider with static credentials

6.2.3.5.1 Configure an AWS Infrastructure Provider with Static Credentials

When configuring an infrastructure provider with static credentials, you need an access id and secret key for a user with a set of minimum capabilities.

6.2.3.5.2 Create a New User via CLI Commands

You will need to have the AWS CLI utility installed[377]. Create a new user via the AWS CLI commands below:

376 https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create_for-user_externalid.html
377 https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html

```
aws iam create-user --user-name Kommander
```

```
aws iam create-policy --policy-name kommander-policy --policy-document
'{"Version":"2012-10-17","Statement":[{"Effect":"Allow","Action":
["ec2:AllocateAddress","ec2:AssociateRouteTable","ec2:AttachInternetGateway","ec2:AuthorizeSecurityG
roupIngress","ec2:CreateInternetGateway","ec2:CreateNatGateway","ec2:CreateRoute","ec2:CreateRouteTa
ble","ec2:CreateSecurityGroup","ec2:CreateSubnet","ec2:CreateTags","ec2:CreateVpc","ec2:ModifyVpcAtt
ribute","ec2:DeleteInternetGateway","ec2:DeleteNatGateway","ec2:DeleteRouteTable","ec2:DeleteSecurit
yGroup","ec2:DeleteSubnet","ec2:DeleteTags","ec2:DeleteVpc","ec2:DescribeAccountAttributes","ec2:Des
cribeAddresses","ec2:DescribeAvailabilityZones","ec2:DescribeInstances","ec2:DescribeInternetGateway
s","ec2:DescribeImages","ec2:DescribeNatGateways","ec2:DescribeNetworkInterfaces","ec2:DescribeNetwo
rkInterfaceAttribute","ec2:DescribeRouteTables","ec2:DescribeSecurityGroups","ec2:DescribeSubnets","
ec2:DescribeVpcs","ec2:DescribeVpcAttribute","ec2:DescribeVolumes","ec2:DetachInternetGateway","ec2:
DisassociateRouteTable","ec2:DisassociateAddress","ec2:ModifyInstanceAttribute","ec2:ModifyNetworkIn
terfaceAttribute","ec2:ModifySubnetAttribute","ec2:ReleaseAddress","ec2:RevokeSecurityGroupIngress",
"ec2:RunInstances","ec2:TerminateInstances","tag:GetResources","elasticloadbalancing:AddTags","elast
icloadbalancing:CreateLoadBalancer","elasticloadbalancing:ConfigureHealthCheck","elasticloadbalancin
g:DeleteLoadBalancer","elasticloadbalancing:DescribeLoadBalancers","elasticloadbalancing:DescribeLoa
dBalancerAttributes","elasticloadbalancing:ApplySecurityGroupsToLoadBalancer","elasticloadbalancing:
DescribeTags","elasticloadbalancing:ModifyLoadBalancerAttributes","elasticloadbalancing:RegisterInst
ancesWithLoadBalancer","elasticloadbalancing:DeregisterInstancesFromLoadBalancer","elasticloadbalanc
ing:RemoveTags","autoscaling:DescribeAutoScalingGroups","autoscaling:DescribeInstanceRefreshes","ec2
:CreateLaunchTemplate","ec2:CreateLaunchTemplateVersion","ec2:DescribeLaunchTemplates","ec2:Describe
LaunchTemplateVersions","ec2:DeleteLaunchTemplate","ec2:DeleteLaunchTemplateVersions","ec2:DescribeK
eyPairs"],"Resource":["*"]},{"Effect":"Allow","Action":
["autoscaling:CreateAutoScalingGroup","autoscaling:UpdateAutoScalingGroup","autoscaling:CreateOrUpda
teTags","autoscaling:StartInstanceRefresh","autoscaling:DeleteAutoScalingGroup","autoscaling:DeleteT
ags"],"Resource":["arn:*:autoscaling:*:*:autoScalingGroup:*:autoScalingGroupName/*"]},
{"Effect":"Allow","Action":["iam:CreateServiceLinkedRole"],"Resource":["arn:*:iam::*:role/aws-
service-role/autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling"],"Condition":{"StringLike":
{"iam:AWSServiceName":"autoscaling.amazonaws.com"}}},{"Effect":"Allow","Action":
["iam:CreateServiceLinkedRole"],"Resource":["arn:*:iam::*:role/aws-service-role/
elasticloadbalancing.amazonaws.com/AWSServiceRoleForElasticLoadBalancing"],"Condition":
{"StringLike":{"iam:AWSServiceName":"elasticloadbalancing.amazonaws.com"}}},
{"Effect":"Allow","Action":["iam:CreateServiceLinkedRole"],"Resource":["arn:*:iam::*:role/aws-
service-role/spot.amazonaws.com/AWSServiceRoleForEC2Spot"],"Condition":{"StringLike":
{"iam:AWSServiceName":"spot.amazonaws.com"}}},{"Effect":"Allow","Action":
["iam:PassRole"],"Resource":["arn:*:iam::*:role/*.cluster-api-provider-aws.sigs.k8s.io"]},
{"Effect":"Allow","Action":
["secretsmanager:CreateSecret","secretsmanager:DeleteSecret","secretsmanager:TagResource"],"Resource
":["arn:*:secretsmanager:*:*:secret:aws.cluster.x-k8s.io/*"]},{"Effect":"Allow","Action":
["ssm:GetParameter"],"Resource":["arn:*:ssm:*:*:parameter/aws/service/eks/optimized-ami/*"]},
{"Effect":"Allow","Action":["iam:CreateServiceLinkedRole"],"Resource":["arn:*:iam::*:role/aws-
service-role/eks.amazonaws.com/AWSServiceRoleForAmazonEKS"],"Condition":{"StringLike":
{"iam:AWSServiceName":"eks.amazonaws.com"}}},{"Effect":"Allow","Action":
["iam:CreateServiceLinkedRole"],"Resource":["arn:*:iam::*:role/aws-service-role/eks-
nodegroup.amazonaws.com/AWSServiceRoleForAmazonEKSNodegroup"],"Condition":{"StringLike":
{"iam:AWSServiceName":"eks-nodegroup.amazonaws.com"}}},{"Effect":"Allow","Action":
["iam:CreateServiceLinkedRole"],"Resource":["arn:aws:iam::*:role/aws-service-role/eks-fargate-
pods.amazonaws.com/AWSServiceRoleForAmazonEKSForFargate"],"Condition":{"StringLike":
{"iam:AWSServiceName":"eks-fargate.amazonaws.com"}}},{"Effect":"Allow","Action":
["iam:GetRole","iam:ListAttachedRolePolicies"],"Resource":["arn:*:iam::*:role/*"]},
{"Effect":"Allow","Action":["iam:GetPolicy"],"Resource":["arn:aws:iam::aws:policy/
AmazonEKSClusterPolicy"]},{"Effect":"Allow","Action":
["eks:DescribeCluster","eks:ListClusters","eks:CreateCluster","eks:TagResource","eks:UpdateClusterVe
rsion","eks:DeleteCluster","eks:UpdateClusterConfig","eks:UntagResource","eks:UpdateNodegroupVersion
","eks:DescribeNodegroup","eks:DeleteNodegroup","eks:UpdateNodegroupConfig","eks:CreateNodegroup","e
ks:AssociateEncryptionConfig"],"Resource":["arn:*:eks:*:*:cluster/*","arn:*:eks:*:*:nodegroup/*/*/
*"]},{"Effect":"Allow","Action":
["eks:ListAddons","eks:CreateAddon","eks:DescribeAddonVersions","eks:DescribeAddon","eks:DeleteAddon
","eks:UpdateAddon","eks:TagResource","eks:DescribeFargateProfile","eks:CreateFargateProfile","eks:D
eleteFargateProfile"],"Resource":["*"]},{"Effect":"Allow","Action":["iam:PassRole"],"Resource":
["*"],"Condition":{"StringEquals":{"iam:PassedToService":"eks.amazonaws.com"}}},
```

```
{"Effect":"Allow","Action":["kms:CreateGrant","kms:DescribeKey"],"Resource":["*"],"Condition":
{"ForAnyValue:StringLike":{"kms:ResourceAliases":"alias/cluster-api-provider-aws-*"}}}]}'
```

```
aws iam attach-user-policy --user-name Kommander --policy-arn $(aws iam list-policies
--query 'Policies[?PolicyName==`kommander-policy`].Arn' | grep -o '".*"' | tr -d '"')
```

```
aws iam create-access-key --user-name Kommander
```

6.2.3.5.3 Using an Existing User

You can use an existing AWS user with credentials configured[378]. The user must be authorized to create the following resources in the AWS account:

- EC2 Instances
- VPC
- Subnets
- Elastic Load Balancer (ELB)
- Internet Gateway
- NAT Gateway
- Elastic Block Storage (EBS) Volumes
- Security Groups
- Route Tables
- IAM Roles

Below is the minimal IAM policy required:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:AllocateAddress",
        "ec2:AssociateRouteTable",
        "ec2:AttachInternetGateway",
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:CreateInternetGateway",
        "ec2:CreateNatGateway",
        "ec2:CreateRoute",
        "ec2:CreateRouteTable",
        "ec2:CreateSecurityGroup",
        "ec2:CreateSubnet",
        "ec2:CreateTags",
        "ec2:CreateVpc",
        "ec2:ModifyVpcAttribute",
```

378 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html

```
        "ec2:DeleteInternetGateway",
        "ec2:DeleteNatGateway",
        "ec2:DeleteRouteTable",
        "ec2:DeleteSecurityGroup",
        "ec2:DeleteSubnet",
        "ec2:DeleteTags",
        "ec2:DeleteVpc",
        "ec2:DescribeAccountAttributes",
        "ec2:DescribeAddresses",
        "ec2:DescribeAvailabilityZones",
        "ec2:DescribeInstances",
        "ec2:DescribeInternetGateways",
        "ec2:DescribeImages",
        "ec2:DescribeNatGateways",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeNetworkInterfaceAttribute",
        "ec2:DescribeRouteTables",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "ec2:DescribeVpcAttribute",
        "ec2:DescribeVolumes",
        "ec2:DetachInternetGateway",
        "ec2:DisassociateRouteTable",
        "ec2:DisassociateAddress",
        "ec2:ModifyInstanceAttribute",
        "ec2:ModifyNetworkInterfaceAttribute",
        "ec2:ModifySubnetAttribute",
        "ec2:ReleaseAddress",
        "ec2:RevokeSecurityGroupIngress",
        "ec2:RunInstances",
        "ec2:TerminateInstances",
        "tag:GetResources",
        "elasticloadbalancing:AddTags",
        "elasticloadbalancing:CreateLoadBalancer",
        "elasticloadbalancing:ConfigureHealthCheck",
        "elasticloadbalancing:DeleteLoadBalancer",
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeLoadBalancerAttributes",
        "elasticloadbalancing:ApplySecurityGroupsToLoadBalancer",
        "elasticloadbalancing:DescribeTags",
        "elasticloadbalancing:ModifyLoadBalancerAttributes",
        "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
        "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
        "elasticloadbalancing:RemoveTags",
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeInstanceRefreshes",
        "ec2:CreateLaunchTemplate",
        "ec2:CreateLaunchTemplateVersion",
        "ec2:DescribeLaunchTemplates",
        "ec2:DescribeLaunchTemplateVersions",
        "ec2:DeleteLaunchTemplate",
```

```
        "ec2:DeleteLaunchTemplateVersions",
        "ec2:DescribeKeyPairs"
      ],
      "Resource": ["*"]
    },
    {
      "Effect": "Allow",
      "Action": [
        "autoscaling:CreateAutoScalingGroup",
        "autoscaling:UpdateAutoScalingGroup",
        "autoscaling:CreateOrUpdateTags",
        "autoscaling:StartInstanceRefresh",
        "autoscaling:DeleteAutoScalingGroup",
        "autoscaling:DeleteTags"
      ],
      "Resource": [
        "arn:*:autoscaling:*:*:autoScalingGroup:*:autoScalingGroupName/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:*:iam::*:role/aws-service-role/autoscaling.amazonaws.com/
AWSServiceRoleForAutoScaling"
      ],
      "Condition": {
        "StringLike": { "iam:AWSServiceName": "autoscaling.amazonaws.com" }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:*:iam::*:role/aws-service-role/elasticloadbalancing.amazonaws.com/
AWSServiceRoleForElasticLoadBalancing"
      ],
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "elasticloadbalancing.amazonaws.com"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:*:iam::*:role/aws-service-role/spot.amazonaws.com/
AWSServiceRoleForEC2Spot"
      ],
      "Condition": {
        "StringLike": { "iam:AWSServiceName": "spot.amazonaws.com" }
```

```
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:PassRole"],
      "Resource": ["arn:*:iam::*:role/*.cluster-api-provider-aws.sigs.k8s.io"]
    },
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:CreateSecret",
        "secretsmanager:DeleteSecret",
        "secretsmanager:TagResource"
      ],
      "Resource": ["arn:*:secretsmanager:*:*:secret:aws.cluster.x-k8s.io/*"]
    },
    {
      "Effect": "Allow",
      "Action": ["ssm:GetParameter"],
      "Resource": ["arn:*:ssm:*:*:parameter/aws/service/eks/optimized-ami/*"]
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:*:iam::*:role/aws-service-role/eks.amazonaws.com/
AWSServiceRoleForAmazonEKS"
      ],
      "Condition": {
        "StringLike": { "iam:AWSServiceName": "eks.amazonaws.com" }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:*:iam::*:role/aws-service-role/eks-nodegroup.amazonaws.com/
AWSServiceRoleForAmazonEKSNodegroup"
      ],
      "Condition": {
        "StringLike": { "iam:AWSServiceName": "eks-nodegroup.amazonaws.com" }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:aws:iam::*:role/aws-service-role/eks-fargate-pods.amazonaws.com/
AWSServiceRoleForAmazonEKSForFargate"
      ],
      "Condition": {
        "StringLike": { "iam:AWSServiceName": "eks-fargate.amazonaws.com" }
```

```
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:GetRole", "iam:ListAttachedRolePolicies"],
      "Resource": ["arn:*:iam::*:role/*"]
    },
    {
      "Effect": "Allow",
      "Action": ["iam:GetPolicy"],
      "Resource": ["arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"]
    },
    {
      "Effect": "Allow",
      "Action": [
        "eks:DescribeCluster",
        "eks:ListClusters",
        "eks:CreateCluster",
        "eks:TagResource",
        "eks:UpdateClusterVersion",
        "eks:DeleteCluster",
        "eks:UpdateClusterConfig",
        "eks:UntagResource",
        "eks:UpdateNodegroupVersion",
        "eks:DescribeNodegroup",
        "eks:DeleteNodegroup",
        "eks:UpdateNodegroupConfig",
        "eks:CreateNodegroup",
        "eks:AssociateEncryptionConfig"
      ],
      "Resource": ["arn:*:eks:*:*:cluster/*", "arn:*:eks:*:*:nodegroup/*/*/*"]
    },
    {
      "Effect": "Allow",
      "Action": [
        "eks:ListAddons",
        "eks:CreateAddon",
        "eks:DescribeAddonVersions",
        "eks:DescribeAddon",
        "eks:DeleteAddon",
        "eks:UpdateAddon",
        "eks:TagResource",
        "eks:DescribeFargateProfile",
        "eks:CreateFargateProfile",
        "eks:DeleteFargateProfile"
      ],
      "Resource": ["*"]
    },
    {
      "Effect": "Allow",
      "Action": ["iam:PassRole"],
      "Resource": ["*"],
```

```
      "Condition": {
        "StringEquals": { "iam:PassedToService": "eks.amazonaws.com" }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["kms:CreateGrant", "kms:DescribeKey"],
      "Resource": ["*"],
      "Condition": {
        "ForAnyValue:StringLike": {
          "kms:ResourceAliases": "alias/cluster-api-provider-aws-*"
        }
      }
    }
  ]
}
```

6.2.3.5.4 Fill out the Add Infrastructure Provider Form in the UI

1. In Kommander, select the Workspace associated with the credentials you are adding.
2. Navigate to **Administration > Infrastructure Providers** and click the **Add Infrastructure Provider** button.
3. Select the Amazon Web Services (AWS) option.
4. Ensure **Static** is selected as the Authentication Method.
5. Enter a name for your infrastructure provider for later reference. Consider choosing a name that matches the AWS user.
6. Fill out the access and secret keys using the keys generated above.
7. Select **Save** to save your provider.

6.3 Applications

This section includes information on the applications you can deploy in DKP.

6.3.1 AppDeployment resources

Use AppDeployments to deploy, and customize platform, DKP catalog, and custom applications in your environment

An `AppDeployment` is a Custom Resource[379] created by DKP with the purpose of deploying applications (platform, DKP catalog and custom applications) in the management cluster, managed clusters, or both. Customers of both Essential and Enterprise products use `AppDeployments`, regardless of their setup (air-gapped, non-air-gapped, etc.), and their infrastructure provider.

When installing DKP, an `AppDeployment` resource is created for each enabled **Platform Application**. This `AppDeployment` resource references a `ClusterApp`, which then references the repository that contains a concrete declarative and preconfigured setup of an application, usually in the form of a `HelmRelease`. `ClusterApps` are cluster-scoped so that these platform applications are deployable to all workspaces or projects.

In the case of **DKP catalog** and **custom applications**, the `AppDeployment` references an `App` instead of a `ClusterApp`, which also references the repository containing the installation and deployment information. `Apps` are namespace-scoped and are meant to only be deployable to the workspace or project in which they have been created.

For example, this is the default `AppDeployment` for the Kube Prometheus Stack platform application:

```
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: kube-prometheus-stack
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: kube-prometheus-stack-44.2.1
    kind: ClusterApp
```

6.3.1.1 Customization

6.3.1.2 Prerequisites

Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace where the cluster is attached:

```
export WORKSPACE_NAMESPACE=<your_workspace_namespace>
```

You are now able to copy the following commands without having to replace the placeholder with your workspace namespace every time you run a command.

379 https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/

6.3.1.3 Customize Your Application

If you want to customize an application, or change how a specific app is deployed, you can create a `ConfigMap` to change or add values to the information that is stored in the `HelmRelease`. Override the default configuration of an application by setting the `configOverrides` field on the `AppDeployment` to that `ConfigMap`. This overrides the configuration of the app for all clusters within the workspace.

For workspace applications, you can also enable and customize them on a per-cluster basis. Refer to the cluster-scoped configuration (see page 577) page for instructions on how to enable and customize an application per cluster in a given workspace.

This is an example, of how to customize the `AppDeployment` of Kube Prometheus Stack:

1. Provide the name of a `ConfigMap` with the custom configuration in the `AppDeployment`:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: kube-prometheus-stack
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: kube-prometheus-stack-44.2.1
    kind: ClusterApp
  configOverrides:
    name: kube-prometheus-stack-overrides-attached
EOF
```

2. Create the `ConfigMap` with the name provided in the previous step, which provides the custom configuration on top of the default configuration:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: kube-prometheus-stack-overrides-attached
data:
  values.yaml: |
    prometheus:
      prometheusSpec:
        storageSpec:
          volumeClaimTemplate:
            spec:
              resources:
                requests:
```

```
                storage: 150Gi
EOF
```

6.3.1.4 Print and Review the Current State of an `AppDeployment` Resource

If you want to know how the `AppDeployment` resource is currently configured, use the commands below to print a table of the declared information. If the `AppDeployment` is configured for several clusters in a workspace, a column will display a list of the clusters.

6.3.1.4.1 Review all `AppDeployments` in a workspace

To review the state of the `AppDeployment` resource for a specific workspace, run the `get` command with the name of your workspace, as in this example:

```
dkp get appdeployments -w kommander-workspace
```

The output should contain a list of all your applications, here is an example:

```
NAME APP CLUSTERS
[...]
kube-oidc-proxy                 kube-oidc-proxy-0.3.2           host-cluster
kube-prometheus-stack           kube-prometheus-stack-44.2.1    host-cluster
kubecost                        kubecost-0.33.1                 host-cluster
[...]
```

6.3.1.4.2 Review a specific `AppDeployment` of an application in a workspace

To review the state of a specific `AppDeployment` of an application, run the `get` command with the name of the application and your workspace, as in this example:

```
dkp get appdeployment kube-prometheus-stack -w kommander-workspace
```

The output should look similar to this:

```
NAME APP CLUSTERS
kube-prometheus-stack kube-prometheus-stack-44.2.1 host-cluster
```

6.3.1.5 Deployment Scope

In a single-cluster environment with an **Essential license**, `AppDeployments` enable customizing any platform application.

In a multi-cluster environment with an **Enterprise license**, `AppDeployments` enable workspace-level (see page 114), project-level (see page 613), and per-cluster deployment and customization of workspace applications (see page 577).

6.3.1.6 More Information

Refer to the CLI documentation (see page 1436) for more information on how to `create`, or `get` an `AppDeployment`.

6.3.2 Logging Stack Application Sizing Recommendations

# of worker nodes	Log Generating Load	Application	Suggested Configuration

<table>
<tr>
<td>50</td>
<td>1.4 MB/s</td>
<td>Logging Operator Logging Override Config</td>
<td>

```
values.yaml: |-
  clusterOutputs:
    - name: loki
      spec:
        loki:
          # change ${WORKSPACE_NAMESPACE} to the actual value of your workspace namespace
          url: http://grafana-loki-loki-distributed-gateway.${WORKSPACE_NAMESPACE}.svc.cluster.local:80

extract_kubernetes_labels: true

configure_kubernetes_labels: true
          buffer:
            disabled: true

retry_forever: false

retry_max_times: 5
            flush_mode: interval

flush_interval: 10s

flush_thread_count: 8
          extra_labels:
            log_source: kubernetes_container
  fluentbit:
    inputTail:
      Mem_Buf_Limit: 512MB

  fluentd:
    bufferStorageVolume:
      emptyDir:
        medium: Memory
    disablePvc: true
    scaling:
      replicas: 10
    resources:
```

</td>
</tr>
</table>

<table>
<tr><td></td><td></td><td></td><td><pre>
requests:
  memory: 1000Mi
  cpu: 1000m
limits:
  memory: 2000Mi
  cpu: 1000m
</pre></td></tr>
<tr><td></td><td></td><td>Loki</td><td><pre>
ingester:
  replicas: 10
distributor:
  replicas: 2
</pre></td></tr>
</table>

100	8.5 MB/s	Logging Operator Logging Override Config	

```
values.yaml: |-
  clusterOutputs:
    - name: loki
      spec:
        loki:
          # change ${WORKSPACE_NAMESPACE} to the actual value of your workspace namespace
          url: http://grafana-loki-loki-distributed-gateway.${WORKSPACE_NAMESPACE}.svc.cluster.local:80

          extract_kubernetes_labels: true

          configure_kubernetes_labels: true
          buffer:
            disabled: true

          retry_forever: false

          retry_max_times: 5
            flush_mode: interval

          flush_interval: 10s

          flush_thread_count: 8
          extra_labels:
            log_source: kubernetes_container
  fluentbit:
    inputTail:
      Mem_Buf_Limit: 512MB

  fluentd:
    bufferStorageVolume:
      emptyDir:
        medium: Memory
    disablePvc: true
    scaling:
      replicas: 15
    resources:
```

			<pre>requests:
			  memory: 1000Mi
			  cpu: 1000m
			limits:
			  memory: 2000Mi
			  cpu: 1000m</pre>

		Loki Override Config	<pre>ingester:
			  replicas: 12
			distributor:
			  replicas: 3
			loki:
			  structuredConfig:
			  limits_config:
			  ingestion_rate_mb:
			1024
			
			ingestion_burst_size_mb:
			1024
			
			per_stream_rate_limit:
			30MB
			
			per_stream_rate_limit_bu
			rst: 60MB
			
			max_streams_per_user: 0
			
			max_global_streams_per_u
			ser: 0
			
			max_label_names_per_seri
			es: 40
			gateway:
			  nginxConfig:
			  httpSnippet: |-
			
			client_max_body_size
			50M;
			  serverSnippet: |-
			
			client_max_body_size
			50M;</pre>

Refer to AppDeployment resources (see page 545) for information on how you customize your AppDeployments.

When configuring storage for `logging-operator-logging-overrides`, ensure that you create a ConfigMap in your workspace namespace for every cluster in that workspace.
Keep in mind that `logging-operator-logging-overrides` can only be configured via the CLI.

6.3.3 Rook Ceph Cluster Sizing Recommendations

# of worker nodes	Application	Suggested Configuration

50	Rook Ceph Cluster	

```
cephClusterSpec:
  labels:
    monitoring:

prometheus.kommander.d2iq.io/
select: "true"
  storage:
    storageClassDeviceSets:
      - name: rook-ceph-osd-
set1
        count: 4
        portable: true
        encrypted: false
        placement:

topologySpreadConstraints:
          - maxSkew: 1
            topologyKey:
topology.kubernetes.io/zone #
The nodes in the same rack have
the same
topology.kubernetes.io/zone
label.
            whenUnsatisfiable:
ScheduleAnyway
            labelSelector:
              matchExpressions:
                - key: app
                  operator: In
                  values:
                    - rook-
ceph-osd
                    - rook-
ceph-osd-prepare
          - maxSkew: 1
            topologyKey:
kubernetes.io/hostname
            whenUnsatisfiable:
ScheduleAnyway
            labelSelector:
              matchExpressions:
                - key: app
                  operator: In
                  values:
                    - rook-
ceph-osd
                    - rook-
ceph-osd-prepare
        volumeClaimTemplates:
```

```
          # If there are some
faster devices and some slower
devices, it is more efficient
to use
          # separate metadata,
wal, and data devices.
          # Refer https://
rook.io/docs/rook/v1.10/CRDs/
Cluster/pvc-cluster/#dedicated-
metadata-and-wal-device-for-
osd-on-pvc
          - metadata:
              name: data
            spec:
              resources:
                requests:
                  storage:
120Gi
              volumeMode: Block
              accessModes:
                - ReadWriteOnce
```

100

```
dkp:
  grafana-loki:
    additionalConfig:
      maxSize: "1000G"

cephClusterSpec:
  labels:
    monitoring:

prometheus.kommander.d2iq.io/
select: "true"
  storage:
    storageClassDeviceSets:
      - name: rook-ceph-osd-
set1
        count: 8
        portable: true
        encrypted: false
        placement:

topologySpreadConstraints:
          - maxSkew: 1
            topologyKey:
topology.kubernetes.io/zone #
The nodes in the same rack have
the same
topology.kubernetes.io/zone
label.
            whenUnsatisfiable:
ScheduleAnyway
            labelSelector:
              matchExpressions:
                - key: app
                  operator: In
                  values:
                    - rook-
ceph-osd
                    - rook-
ceph-osd-prepare
          - maxSkew: 1
            topologyKey:
kubernetes.io/hostname
            whenUnsatisfiable:
ScheduleAnyway
            labelSelector:
              matchExpressions:
                - key: app
                  operator: In
                  values:
```

```
                    - rook-
ceph-osd
                    - rook-
ceph-osd-prepare
        volumeClaimTemplates:
            - metadata:
              name: data
              spec:
                resources:
                  requests:
                    storage:
200Gi
                volumeMode:
Block
                accessModes:
                  -
ReadWriteOnce
```

Refer to AppDeployment resources (see page 545) for information on how you customize your AppDeployments.

To add more storage to `rook-ceph-cluster`, copy and paste the `storageClassDeviceSets` list from the `rook-ceph-cluster-1.10.11-d2iq-defaults` ConfigMap into your workspace where `rook-ceph-cluster` is present and then modify `count` and `volumeClaimTemplates.spec.resource.requests.storage`.

6.3.4 Manage an Application using the UI

Choose your license type for instructions on how to enable, and customize an application and then verify it has been deployed correctly.

6.3.4.1 Enterprise - Manage an Application using the UI

Enterprise Gov Advanced

To use the CLI to deploy or uninstall applications, see Application Deployment (see page 564).

6.3.4.1.1 Enable an Application using the UI

Enterprise Gov Advanced

Follow these steps to enable your platform applications from the UI:

1. From the top menu bar, select your target workspace.
2. Select **Applications** from the sidebar to browse through the available applications from your configured repositories.
3. Select the three-dot button of the desired application card > **Enable**.
4. If available, select a version from the drop-down menu. This drop-down menu is only visible if there is more than one version to choose from.
5. Select the clusters where you want to deploy the application.
6. **For customizations only:** to override the default configuration values, select **Configuration** in the sidebar.
   Note: If there are customization *Overrides* at the workspace and cluster level, they are combined for implementation. *Cluster-level Overrides* take precedence over *Workspace Overrides*.
   a. To customize an application for all clusters in a workspace, copy your customized values into the text editor under **Workspace Application Configuration** or upload your YAML file that contains the values:

```
someField: someValue
```

b. To add a customization per cluster, copy the customized values into the text editor of each cluster under **Cluster Application Configuration Override** or upload your YAML file that contains the values:

```
someField: someValue
```

7. Confirm the details are correct, and then select the **Enable** button.

There may be dependencies between the applications, which are listed in the Workspace Platform Application Dependencies (see page 568) documentation. Review them carefully prior to customizing to ensure that the applications are deployed successfully.

6.3.4.1.1.1 Next topic:

Customize an App using the UI (see page 560)

6.3.4.1.2 Customize an Application using the UI

If you want to enable an application for the first time and customize it, refer to Enable an Application using the UI (see page 559).

You can customize the applications that are deployed to a workspace's cluster using the UI:

1. From the top menu bar, select your target workspace.
2. Select **Applications** from the sidebar menu to browse all applications.
3. In the **Application** card you want to customize, select the three dot menu and **Edit**.
4. To override the default configuration values, select **Configuration** in the sidebar.
   Note: If there are customization *Overrides* at the workspace and cluster level, they are combined for implementation. *Cluster-level Overrides* take precedence over *Workspace Overrides*.
   a. To customize an application for all clusters in a workspace, copy your customized values into the text editor under **Workspace Application Configuration** or upload your YAML file that contains the values:

```
someField: someValue
```

b. To add a customization per cluster, copy the customized values into the text editor of each cluster under **Cluster Application Configuration Override** or upload your YAML file that contains the values:

```
someField: someValue
```

5. Confirm the details are correct, and select **Save**.

6.3.4.1.2.1 Customize an App for a Specific Cluster from the **Clusters** page

You can also customize an application *for a specific cluster* from the **Clusters** view:

1. Select **Clusters** from the sidebar menu.
2. Selecting the target cluster.
3. Select the **Applications** tab.
4. Navigate to the target **Application** card.
5. Select the three-dot menu > **Edit**.

6.3.4.1.2.2 Next topic:

Verify an App using the UI (see page 561)

6.3.4.1.3 Verify an Application using the UI

Enterprise Gov Advanced

The application has now been enabled. Follow these steps to ensure the application was deployed correctly:

1. Select your target workspace from the top menu bar.
2. Select the cluster you want to verify from the left sidebar menu:
   a. Select **Management Cluster** if your target cluster is the Management Cluster Workspace.
   b. Otherwise, select **Clusters**, and choose your target cluster.
3. Select the **Applications** tab and navigate to the application you want to verify.
4. If the application was deployed successfully, the status **Deployed** appears in the application card. Otherwise, hover over the failed status to obtain more information on why the application failed to deploy.

It can take several minutes for the application to deploy completely. If the **Deployed** or **Failed** status is not displayed, the deployment process is not finished.

6.3.4.2 Essential - Manage an Application using the UI

To use the CLI to deploy or uninstall applications, see Application Deployment (see page 564).

6.3.4.2.1 Enable an Application using the UI - Essential

Follow these steps to enable your platform applications from the UI in Kommander:

1. Select **Applications** from the sidebar to browse through the available applications from your configured repositories.
2. Select the three-dot button of the desired application card > **Enable**.
3. If available, select a version from the drop-down menu. This drop-down menu is only visible if there is more than one version to choose from.
4. **For customizations only:**
   - To override the default configuration values, select **Configuration** in the sidebar.
   - Copy your customized values into the text editor under **Workspace Application Configuration** or upload your YAML file that contains the values:

     ```
     someField: someValue
     ```
5. Confirm the details are correct, and then select the **Enable** button.

There may be dependencies between the applications, which are listed in the Workspace Platform Application Dependencies (see page 568) documentation. Review them carefully prior to customizing to ensure that the applications are deployed successfully.

6.3.4.2.1.1 Next topic:

Customize an App using the UI - Essential (see page 563)

6.3.4.2.2 Customize an Application using the UI - Essential

> If you want to enable an application for the first time and customize it, refer to Enable an Application using the UI - Essential (see page 562).

You can customize the applications that are deployed to your Management Cluster using the UI:

1. Select **Applications** from the sidebar to browse all applications.
2. In the **Application** card you want to customize, select the three dot menu and **Edit**.
3. To override the default configuration values, select **Configuration** in the sidebar.
    a. To customize an application, copy your customized values into the text editor under **Workspace Application Configuration** or upload your YAML file that contains the values:

    ```
    someField: someValue
    ```

4. Confirm the details are correct, and select **Save**.

6.3.4.2.2.1 Next topic:

Verify an App via UI - Essential (see page 563)

6.3.4.2.3 Verify an Application via UI - Essential

The application has now been enabled. Follow these steps to ensure the application was deployed correctly:

1. Select **Management Cluster** from the sidebar.
2. Select the **Applications** tab and navigate to the application you want to verify.
3. If the application was deployed successfully, the status **Deployed** appears in the application card. Otherwise, hover over the failed status to obtain more information on why the application failed to deploy.

It can take several minutes for the application to deploy completely. If the **Deployed** or **Failed** status is not displayed, the deployment process is not finished.

6.3.5 Platform Applications

How platform applications work

When attaching a cluster, DKP deploys certain platform applications on the newly attached cluster. Operators can use the DKP UI to customize which platform applications to deploy to the attached clusters in a given workspace.

Currently, the monitoring stack is deployed by default. The logging stack is not.

Review the Workspace Platform Application Defaults and Resource Requirements (see page 114) to ensure that the attached clusters have sufficient resources.

When deploying and upgrading applications, platform applications come as a bundle; they are tested as a single unit, and you must deploy or upgrade them in a single process, for each workspace. This means all clusters in a workspace have the same set and versions of platform applications deployed.

6.3.5.1 Related Topics

- Deploy Platform Applications via CLI (see page 564)
- Cluster-scoped Configuration for Existing AppDeployments (see page 577)
- Manage an Application using the UI (see page 558)
- Cluster-scoped Application Configuration via the UI (see page 575)
- Platform Application Dependencies (see page 568)
- Workspace Platform Application Defaults and Resource Requirements (see page 114)

6.3.5.2 Deploy Platform Applications via CLI

This topic describes how to use the CLI to enable an application to deploy to managed and attached clusters in a workspace.

6.3.5.2.1 Related Topics

- **Available Workspace Platform Applications** (see page 114)
- **Customize** an existing AppDeployment (see page 546)
- **Cluster-scoped** configuration via CLI (see page 577)
- **Cluster-scoped** configuration via UI (see page 575)

6.3.5.2.2 Prerequisites

Before you begin, you must have:

- A running cluster with Kommander installed (see page 71).
- An existing Kubernetes cluster attached to Kommander (see page 673).
- Determine the name of the workspace where you wish to perform the deployments. You can use the `dkp get workspaces` command to see the list of workspace names and their corresponding namespaces.
- Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace where the cluster is attached:

  ```
  export WORKSPACE_NAMESPACE=<workspace_namespace>
  ```

- Set the `WORKSPACE_NAME` environment variable to the name of the workspace where the cluster is attached:

  ```
  export WORKSPACE_NAME=<workspace_name>
  ```

From the CLI, you can enable applications to deploy in the workspace. Verify that an application has successfully deployed via the CLI (see page 566).

6.3.5.2.3 Create the `AppDeployment` to Enable the Application

Enable a supported application to deploy to your existing attached or managed cluster with an AppDeployment resource (see page 545).

1. Obtain the **APP ID** and **Version** of the application from the applications table in the latest Release Notes (see page 1544). You will need them it in the `<APP-ID>-<Version>` format, for example, `istio-1.16.2`.
2. Run the following command and define the `--app` flag to specify which platform application and version will be enabled:

   ```
   dkp create appdeployment istio --app istio-1.16.2 --workspace ${WORKSPACE_NAME}
   ```

- The `--app` flag must match the `APP NAME` from the list of available platform applications.

- Observe that the `dkp create` command must be run with the `WORKSPACE_NAME` instead of the `WORKSPACE_NAMESPACE` flag.

This instructs Kommander to create and deploy the `AppDeployment` to the `KommanderClusters` in the specified `WORKSPACE_NAME`.

6.3.5.2.4 Verify Applications

The applications are now enabled. Connect to the attached cluster and watch the `HelmReleases` to verify the deployment. In this example, we are checking if `istio` got deployed correctly:

```
kubectl get helmreleases istio -n ${WORKSPACE_NAMESPACE} -w
```

You should eventually see the `HelmRelease` marked as `Ready`:

```
NAMESPACE               NAME        READY   STATUS                             AGE
workspace-test-vjsfq    istio       True    Release reconciliation succeeded   7m3s
```

Some supported applications have dependencies on other applications. See Workspace Platform Application Dependencies (see page 568) for that table.

6.3.5.3 Upgrade Platform Applications

6.3.5.3.1 Prerequisites

Before you begin, you must:

- Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace where the cluster is attached:

  ```
  export WORKSPACE_NAMESPACE=<workspace_namespace>
  ```

- Set the `WORKSPACE_NAME` environment variable to the name of the workspace where the cluster is attached:

```
export WORKSPACE_NAME=<workspace_name>
```

6.3.5.3.2 Upgrade Platform Applications from the CLI

The DKP upgrade (see page 1365) process deploys and upgrades Platform applications as a bundle for each cluster or workspace. For the Management Cluster or workspace, DKP upgrade handles all Platform applications; no other steps are necessary to upgrade the Platform application bundle. However, for managed or attached clusters or workspaces, you MUST manually upgrade the Platform applications bundle with the following command.

If you are upgrading your Platform applications as part of the DKP upgrade (see page 1365), upgrade your Platform applications on any additional Workspaces before proceeding with the Konvoy upgrade. Some applications in the previous release are not compatible with the Kubernetes version (see page 1543) of this release, and upgrading Kubernetes is part of the DKP Konvoy upgrade process.

6.3.5.3.2.1 Execute the DKP Upgrade Command

Use this command to upgrade all platform applications in the given workspace and its projects to the same version as the platform applications running on the management cluster:

```
dkp upgrade workspace ${WORKSPACE_NAME}
```

An output similar to this appears:

```
✓ Ensuring HelmReleases are upgraded on clusters in namespace "${WORKSPACE_NAME}"
...
```

If the upgrade fails or times out, retry the command with higher verbosity to get more information on the upgrade process:

```
dkp upgrade workspace ${WORKSPACE_NAME} -v 4
```

If you find any `HelmReleases` in a “broken” release state such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n ${WORKSPACE_NAMESPACE} patch helmrelease <HELMRELEASE_NAME> --type='json'
 -p='[{"op": "replace", "path": "/spec/suspend", "value": true}]'
kubectl -n 4{WORKSPACE_NAMESPACE} patch helmrelease <HELMRELEASE_NAME> --type='json'
 -p='[{"op": "replace", "path": "/spec/suspend", "value": false}]'
```

6.3.5.4 Platform Application Dependencies

6.3.5.4.1 Dependencies between workspace applications

Platform applications that are deployed to a workspace's attached clusters can depend on each other. It is important to note these dependencies when customizing the workspace platform applications to ensure that your applications are properly deployed to the clusters. For more information on how to customize workspace platform applications, see Workspace Platform Applications (see page 564).

When deploying or troubleshooting platform applications, it helps to understand how platform applications interact and may require other platform applications as dependencies.

If a platform application's dependency does not successfully deploy, the platform application requiring that dependency does not successfully deploy.

The following sections detail information about the workspace platform application.

6.3.5.4.2 Foundational Applications

Provides the foundation for all platform application capabilities and deployments on managed clusters. These applications must be enabled for any platform applications to work properly.

The foundational applications are comprised of the following platform application:

- cert-manager[380]: Automates TLS certificate management and issuance.
- reloader[381]: A controller that watches changes on ConfigMaps and Secrets, and automatically triggers updates on the dependent applications.
- traefik[382]: Provides an HTTP reverse proxy and load balancer. Requires cert-manager and reloader.

Platform Application	Required Dependencies
cert-manager	
reloader	
traefik	cert-manager, reloader

380 https://cert-manager.io/docs
381 https://github.com/stakater/Reloader
382 https://traefik.io/

6.3.5.4.3 Logging

Collects logs over time from Kubernetes and applications deployed on managed clusters. Also provides the ability to visualize and query the aggregated logs.

- fluent-bit[383]: Open source and multi-platform log processor tool which aims to be a generic Swiss knife for logs processing and distribution.
- grafana-logging[384]: Logging dashboard used to view logs aggregated to Grafana Loki.
- grafana-loki[385]: A horizontally-scalable, highly-available, multi-tenant log aggregation system inspired by Prometheus.
- logging-operator[386]: Automates the deployment and configuration of a Kubernetes logging pipeline.
- rook-ceph[387] and rook-ceph-cluster[388]: A Kubernetes-native high performance object store with an S3-compatible API that supports deploying into private and public cloud infrastructures.

Platform Application	Required Dependencies	Optional Dependencies
fluent-bit		
grafana-logging	grafana-loki	
grafana-loki	rook-ceph-cluster	
logging-operator		
rook-ceph		
rook-ceph-cluster	rook-ceph	kube-prometheus-stack Users can override the config to remove the dependency, as needed.

383 https://docs.fluentbit.io/manual/
384 https://grafana.com/oss/grafana/
385 https://grafana.com/oss/loki/
386 https://banzaicloud.com/docs/one-eye/logging-operator/
387 https://rook.io/docs/rook/v1.10/Helm-Charts/operator-chart/
388 https://rook.io/docs/rook/v1.10/Helm-Charts/ceph-cluster-chart/

6.3.5.4.4 Monitoring

Provides monitoring capabilities by collecting metrics, including cost metrics, for Kubernetes and applications deployed on managed clusters. Also provides visualization of metrics and evaluates rule expressions to trigger alerts when specific conditions are observed.

- kubecost[389]: provides real-time cost visibility and insights for teams using Kubernetes, helping you continuously reduce your cloud costs.
- kubernetes-dashboard[390]: A general purpose, web-based UI for Kubernetes clusters. It allows users to manage applications running in the cluster, troubleshoot them and manage the cluster itself.
- kube-prometheus-stack[391]: A stack of applications that collect metrics and provide visualization and alerting capabilities.
  NOTE: Prometheus[392], Prometheus Alertmanager[393] and Grafana[394] are included in the bundled installation.
- nvidia-gpu-operator[395]: The NVIDIA GPU Operator manages NVIDIA GPU resources in a Kubernetes cluster and automates tasks related to bootstrapping GPU nodes.
- prometheus-adapter[396]: Provides cluster metrics from Prometheus.

Platform Application	Required Dependencies
kubecost	traefik
kubernetes-dashboard	traefik
kube-prometheus-stack	traefik
prometheus-adapter	kube-prometheus-stack
nvidia-gpu-operator	

6.3.5.4.5 Security

Allows management of security constraints and capabilities for the clusters and users.

---

389 https://kubecost.com/
390 https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/
391 https://github.com/prometheus-community/helm-charts/tree/main/charts/kube-prometheus-stack
392 https://prometheus.io/
393 https://prometheus.io/docs/alerting/latest/alertmanager
394 https://grafana.com/
395 https://catalog.ngc.nvidia.com/orgs/nvidia/containers/gpu-operator
396 https://github.com/DirectXMan12/k8s-prometheus-adapter

- gatekeeper[397]: A policy Controller for Kubernetes.

Platform Application	Required Dependencies
gatekeeper	

6.3.5.4.6 Single Sign On (SSO)

Group of platform applications that allow enabling SSO on attached clusters. SSO is a centralized system for connecting attached clusters to the centralized authority on the management cluster.

- kube-oidc-proxy[398]: A reverse proxy server that authenticates users using OIDC to Kubernetes API servers where OIDC authentication is not available.
- traefik-forward-auth[399]: Installs a forward authentication application providing Google OAuth based authentication for Traefik.

Platform Application	Required Dependencies
kube-oidc-proxy	cert-manager, traefik
traefik-forward-auth	traefik

6.3.5.4.7 Backup

This platform application assists you with backing up and restoring your environment.

- velero[400]: An open source tool for safely backing up and restoring resources in a Kubernetes cluster, performing disaster recovery, and migrating resources and persistent volumes to another Kubernetes cluster.

397 https://github.com/open-policy-agent/gatekeeper
398 https://github.com/jetstack/kube-oidc-proxy
399 https://github.com/thomseddon/traefik-forward-auth
400 https://velero.io/

Platform Application	Dependencies
velero	rook-ceph-cluster This is an optional dependency. Users can override the config to remove the dependency, as needed.

6.3.5.4.8 Service Mesh

Allows deploying service mesh on clusters, enabling the management of microservices in cloud-native applications. Service mesh can provide a number of benefits, such as providing observability into communications, providing secure connections, or automating retries and backoff for failed requests.

- istio[401]: Addresses the challenges developers and operators face with a distributed or microservices architecture.
- jaeger[402]: A distributed tracing system used for monitoring and troubleshooting microservices-based distributed systems.
- kiali[403]: A management console for an Istio-based service mesh. It provides dashboards, observability, and lets you operate your mesh with robust configuration and validation capabilities.

Platform Application	Required Dependencies	Optional Dependencies
istio	kube-prometheus-stack	
jaeger	istio	
kiali	istio	jaeger (optional for monitoring purposes)
knative	istio	

401 https://istio.io/latest/about/service-mesh/
402 https://www.jaegertracing.io/
403 https://kiali.io/

6.3.5.4.9 Related Information

- Kommander security architecture (see page 829)
- Traefik Ingress controller (see page 829)
- Logging and audits (see page 796)

6.3.5.5 Workspace Platform Application Resource Requirements

Refer to Workspace Platform Application Defaults and Resource Requirements (see page 114) for a list of all platform applications, their default deployment configuration, required resources, and storage minimums.

6.4 Workspaces

Enterprise Gov Advanced

Allow teams to manage their own clusters using workspaces

Workspaces give you the flexibility to represent your organization in a way that makes sense for your team and configuration. For example, you can create separate workspaces for each department, product, or business function. Each workspace manages their own clusters and role-based permissions, while retaining an overall organization-level view of all clusters in operation.

6.4.1 Global / Workspace UI

The UI is designed to be accessible for different roles at different levels:

- **Global**: At the top level, IT administrators manage all clusters across all workspaces.
- **Workspace**: DevOps administrators manage multiple clusters within a workspace.
- **Projects**: DevOps administrators or developers manage configuration and services across multiple clusters.

6.4.2 Default Workspace

To get started immediately, you can use the default workspace deployed in DKP. Your workspace delegation can be done at a later time.

6.4.3 Create a Workspace

In DKP you can create your own Workspaces. The following steps describe this procedure.

1. From the workspace selection dropdown in the top menu bar, select **Create Workspace**.

2. Type a name and description and select **Save**. The workspace is now accessible from the workspace selection drop down.

6.4.4 Add, Edit, and Delete Workspace Annotations and Labels

When creating or editing a workspace, you can use the **Advanced Options** to add, edit, or delete annotations and labels to your workspace. Both the annotations and labels are applied to the workspace namespace.

1. From the top menu bar, select your target workspace.
2. Select the **Actions** dropdown button in the top-right, and select **Edit Workspace**.
3. Type in new **Key** and **Value** labels for your workspace or edit existing **Key** and **Value** labels.

> Labels that are added to a workspace, are also applied to the `kommanderclusters` object as well as to all of the clusters in the workspace.

6.4.5 Delete a Workspace

In DKP you can delete existing Workspaces. The following steps describe this procedure.

> Workspaces can only be deleted if all the clusters in the workspace have been deleted or detached.

1. From the top menu bar, select **Global**.
2. From the sidebar menu, select **Workspaces**.
3. Select the three dot button to the right of the workspace you want to delete and then select **Delete**.
4. Confirm deleting the Workspace in the **Delete Workspace** dialog box.

The following procedures are supported for workspaces:

- Application Deployment using the CLI (see page 564)
- Platform Applications (see page 564)
- Platform Application Dependencies (see page 568)
- Workspace Platform Application Defaults and Resource Requirements (see page 114)

6.4.6 Workspace Applications

Enterprise Gov Advanced

This section documents the applications and application types that you can use with DKP.

Application types are:

- Catalog Applications (see page 586) are either pre-packaged applications from the D2iQ Application Catalog, or custom applications that you maintain for your teams or organization.
- Platform Applications (see page 564) are applications integrated into DKP.

- Cluster-scoped Application Configuration via the UI (see page 575)
- Cluster-scoped Configuration for Existing AppDeployments (see page 577)

6.4.6.1 Cluster-scoped Application Configuration via the UI

Enterprise Gov Advanced

When you enable an application for a workspace, you deploy this application to all clusters within that workspace. You can also choose to enable or customize an application on certain clusters within a workspace.

This functionality allows you to use DKP in a multi-cluster scenario without restricting the management of multiple clusters from a single workspace.

NOTE: DKP Essential users are only be able to configure and deploy applications to a single cluster within a workspace. Selecting an application to deploy to a cluster skips cluster selection and takes you directly to the workspace configuration overrides page.

6.4.6.1.1 Prerequisites

Ensure that you've provisioned or attached clusters in one of the following environments:

- Amazon Web Services (AWS) (see page 980)
- Amazon Elastic Kubernetes Service (EKS) (see page 1022)
- Microsoft Azure (see page 1066)

Refer to the current list of Catalog and Platform Applications:

- Workspace Catalog Applications (see page 586)
- Workspace Platform Applications (see page 114)

6.4.6.1.2 Enable a Cluster-scoped Application

Navigate to the workspace containing the clusters you want to deploy to by selecting the appropriate workspace name from the drop-down menu at the top of the DKP dashboard.

1. Select **Applications** from the left navigation bar and find the application you want to deploy to the cluster.
2. Select the the three-dot menu in the desired application's tile and select **Enable** to reach the application enablement page.
   NOTE: The application enablement page can also be reached by selecting **View Details** from the three-dot menu, and then selecting **Enable** from the application's details page.
3. Select the cluster(s) that you're deploying the application to. The available clusters can be sorted by **Name**, **Type**, **Provider** and any **Labels** you've added.
4. Deploy the application to the clusters by selecting **Enable** in the top right corner of the application enablement page.

You are automatically redirected to either the **Applications** or **View Details** page. To view the application enabled in your chosen cluster, navigate to the **Clusters** page on the left navigation bar. The application appears in the **Applications** pane of the appropriate cluster.

> **NOTE:** Once you enable an application at the workspace level, DKP automatically enables that app on any other cluster you create or attach.

6.4.6.1.3 Configure a Cluster-scoped Application

For scenarios where applications require different configurations on a per-cluster basis, navigate to the **Applications** page and select **Edit** from the three-dot menu of the appropriate application to return to the application enablement page.

1. Select the cluster(s) that you're deploying the application to. The available clusters can be sorted by **Name**, **Type**, **Provider** and any **Labels** you've added.
2. Select the **Configuration** tab.
3. The **Configuration** tab contains two separate types of code editors, where you can enter your specified overrides and configurations:
   Workspace Application Configuration: A workspace-level code editor that applies all configurations and overrides to the entirety of the workspace and its clusters for this application.
   Cluster Application Configuration Override: A cluster-scoped code editor that applies configurations and overrides to the cluster specified. These customizations will merge with the workspace application configuration. If there is no cluster-scoped configuration, the workspace configuration applies.
4. If you already have a configuration to apply in a text or .yaml file, you can upload the file by selecting **Upload File**. If you want to download the displayed set of configurations, select **Download File**.
5. Finish configuring the cluster-scoped applications by selecting **Save** in the top right corner of the application enablement page.

You are automatically redirected to either the **Applications** or **View Details** page. To view the custom configurations of the application in the cluster, select the **Configurations** tab on the details page of the application.

> **NOTE:** Editing is disabled in the code boxes displayed in the application's details page. To edit the configuration, select the **Edit** button in the top right of the page and repeat the steps in this section.

6.4.6.1.4 Remove a Cluster-scoped Application

Navigate to the cluster you've deployed your applications to by selecting **Clusters** from the left navigation bar.

1. Click on the Applications tab.
2. Select the three-dot menu in the desired application's tile and select **Uninstall.**
3. A prompt appears to confirm your decision to uninstall the application. Follow the instructions in the prompt and select **Uninstall.**
4. Refresh the page to confirm that the application has been removed from the cluster.

This process only removes the application from the specific cluster you've navigated to. To remove this application from other clusters, navigate to the **Clusters** page and repeat the process.

6.4.6.2 Cluster-scoped Configuration for Existing AppDeployments

Enterprise Gov Advanced

This topic describes how to enable cluster-scoped configuration of applications for **existing** `AppDeployments`. For more information on how to **create** `AppDeployments`, refer to the Application Deployment (see page 1450) section.

6.4.6.2.1 Enable or Disable an App per Cluster and Define a Custom Configuration

Edit the `AppDeployment` resource to modify the configuration of an application per cluster.

6.4.6.2.2 For Better Understanding

When you enable an application for a workspace, you deploy this application to all clusters within that workspace. You can also choose to enable or customize an application on certain clusters within a workspace. This functionality allows you to use DKP in a multi-cluster scenario without restricting the management of multiple clusters from a single workspace.

Your DKP cluster comes bundled with a set of default application configurations. If you want to override the default configuration of your applications, you can define workspace `configOverrides` on top of the default workspace configuration. And if you want to further customize your workplace by enabling applications on a per-cluster basis or by defining per-cluster customizations, you can create and apply `clusterConfigOverrides`.

The cluster-scoped enablement and customization of applications is an **Enterprise-only** feature, which allows the configuration of all workspace Platform (see page 564), DKP catalog (see page 586) and Custom Applications (see page 598) via the **CLI** in your managed and attached clusters regardless of your environment setup (air-gapped or non-air-gapped). This capability is not provided for project applications.

6.4.6.2.3 Prerequisites (per-cluster application configuration)

Enterprise Gov Advanced

- Any application you wish to enable or customize at a cluster level, first needs to be enabled at the workspace-level via an `AppDeployment` (platform application deployment (see page 564), workspace catalog application deployment (see page 594)).
- For custom configurations, you have created a ConfigMap (see page 0) with all the required `spec` fields for each customization you would like to add to an application in a cluster. You can apply a `ConfigMap` to several clusters, or create a `ConfigMap` for each cluster, but the `ConfigMap` object must exist in the Management cluster.
- Determine the name of the workspace where you wish to perform the deployments. You can use the `dkp get workspaces` command to see the list of workspace names and their corresponding namespaces.
- Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace where the cluster is attached:

  ```
  export WORKSPACE_NAMESPACE=<workspace_namespace>
  ```

- Set the `WORKSPACE_NAME` environment variable to the name of the workspace where the cluster is attached:

  ```
  export WORKSPACE_NAME=<workspace_name>
  ```

6.4.6.2.4 Enable an Application per Cluster

Enterprise Gov Advanced

6.4.6.2.4.1 Prerequisites

Ensure you have reviewed the prerequisites (see page 578) section.

6.4.6.2.4.2 Enable an Application per Cluster *for the First Time*

When you enable an application on a workspace, it is deployed to all clusters in the workspace by default. If you would like to only deploy it to a subset of clusters when enabling it on a workspace for the first time, you can follow the steps below:

1. Create an `AppDeployment` for your application, selecting a subset of clusters within the workspace to enable it on. You can use the `dkp get clusters --workspace ${WORKSPACE_NAME}` command to see the list of clusters in the workspace.

   ```
   dkp create appdeployment kube-prometheus-stack --app kube-prometheus-
   stack-44.2.1 --workspace ${WORKSPACE_NAME} --clusters attached-
   cluster1,attached-cluster2
   ```

2. **Optional:** Check the current status (see page 586) of the `AppDeployment` to see the names of the clusters where the application is currently enabled.

6.4.6.2.4.3 Enable or Disable an Application per Cluster *after it has been enabled at the workspace level*

You can enable or disable applications at any time. Once you have enabled the application at the workspace level (see page 0), the `spec.clusterSelector` field (see page 1398) will be populated.

> For clusters that are newly attached into the workspace, all applications enabled for the workspace are automatically enabled on and deployed to the new clusters.

If you want to see on which clusters your application is currently deployed, refer to the Print and Review the Current State of your AppDeployment (see page 548) documentation.

1. Edit the `AppDeployment` YAML by adding or removing the names of the clusters where you want to enable your application in the `clusterSelector` section:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: kube-prometheus-stack
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: kube-prometheus-stack-44.2.1
    kind: ClusterApp
  clusterSelector:
    matchExpressions:
    - key: kommander.d2iq.io/cluster-name
      operator: In
      values:
      - attached-cluster1
      - attached-cluster3-new
EOF
```

6.4.6.2.4.4 Verify the Current Configuration of your Application

Refer to the Verify applications help (see page 0) to connect to the managed or attached cluster and check the status of the deployments.

If you want to know how the `AppDeployment` resource is currently configured, refer to the Print and review the state of your AppDeployment (see page 548) section.

6.4.6.2.5 Customizations per Cluster

Enable or disable a customization per cluster.

- Customize an Application per Cluster (see page 580)
- Disable a Custom Configuration of an Application for a Cluster (see page 583)

6.4.6.2.5.1 Customize an Application per Cluster

Enterprise Gov Advanced

Prerequisites

Ensure you have reviewed the prerequisites (see page 578) section.

Enable a Custom Configuration of an Application for a Cluster

You can customize the application for each cluster occurrence of said application. If you want to customize the application for a cluster that is not yet attached, refer to the instructions below (see page 580), so the

application is deployed with the custom configuration on attachment.
To enable per-cluster customizations:

1. Reference the name of the `ConfigMap` to be applied per cluster in the `spec.clusterConfigOverrides` fields. In this example, you have three different customizations specified in three different `ConfigMaps` for three different clusters in one workspace:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: kube-prometheus-stack
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: kube-prometheus-stack-44.2.1
    kind: ClusterApp
  clusterSelector:
    matchExpressions:
    - key: kommander.d2iq.io/cluster-name
      operator: In
      values:
      - attached-cluster1
      - attached-cluster2
      - attached-cluster3-new
  clusterConfigOverrides:
    - configMapName: kps-cluster1-overrides
      clusterSelector:
        matchExpressions:
        - key: kommander.d2iq.io/cluster-name
          operator: In
          values:
          - attached-cluster1
    - configMapName: kps-cluster2-overrides
      clusterSelector:
        matchExpressions:
        - key: kommander.d2iq.io/cluster-name
          operator: In
          values:
          - attached-cluster2
    - configMapName: kps-cluster3-overrides
      clusterSelector:
        matchExpressions:
        - key: kommander.d2iq.io/cluster-name
          operator: In
          values:
          - attached-cluster3-new
EOF
```

2. If you have not done so yet, create the ConfigMaps (see page 0) referenced in each `clusterConfigOverrides` entry.

- The changes are applied only if the YAML file has a valid syntax.
- Set up only one cluster override `ConfigMap` per cluster. If there are several `ConfigMaps` configured for a cluster, only one will be applied.
- Cluster override `ConfigMaps` must be created on the Management cluster.

Enable a Custom Configuration of an Application for a Cluster at Attachment

You can customize the application configuration for a cluster prior to its attachment, so that the application is deployed with this custom configuration on attachment. This is preferable, if you do not want to redeploy the application with an updated configuration after it has been initially installed, which may cause downtime.

To enable per-cluster customizations, follow these steps **before attaching the cluster**:

1. Set the `CLUSTER_NAME` environment variable to the cluster name that you will give your to-be-attached cluster:

```
export CLUSTER_NAME=<your_attached_cluster_name>
```

2. Reference the name of the `ConfigMap` you want to apply to this cluster in the `spec.clusterConfigOverrides` fields. **You do not need to update the `spec.clusterSelector` field**. In this example, you have the `kps-cluster1-overrides` customization specified for `attached-cluster-1` and a different customization (in `kps-your-attached-cluster-overrides` `ConfigMap`) for your to-be-attached cluster:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: kube-prometheus-stack
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: kube-prometheus-stack-44.2.1
    kind: ClusterApp
  clusterSelector:
    matchExpressions:
    - key: kommander.d2iq.io/cluster-name
```

```
      operator: In
      values:
      - attached-cluster1
  clusterConfigOverrides:
    - configMapName: kps-cluster1-overrides
      clusterSelector:
        matchExpressions:
        - key: kommander.d2iq.io/cluster-name
          operator: In
          values:
          - attached-cluster1
    - configMapName: kps-your-attached-cluster-overrides
      clusterSelector:
        matchExpressions:
        - key: kommander.d2iq.io/cluster-name
          operator: In
          values:
          - ${CLUSTER_NAME}
EOF
```

3. If you have not done so yet, create the ConfigMap (see page 546) referenced for your to-be-attached cluster.

- The changes are applied only if the YAML file has a valid syntax.
- Cluster override `ConfigMaps` must be created on the Management cluster.

Verify the Current Configuration of your Application

Refer to the Verify applications help (see page 0) to connect to the managed or attached cluster and check the status of the deployments.

If you want to know how the `AppDeployment` resource is currently configured, refer to the Print and review the state of your AppDeployment (see page 548) section.

6.4.6.2.5.2 Disable a Custom Configuration of an Application for a Cluster

Enabled customizations are defined in a `ConfigMap`, which, in turn, is referenced in the `spec.clusterConfigOverrides` object of your `AppDeployment`.

1. Review your current configuration to establish what you want to remove:

```
kubectl get appdeployment -n ${WORKSPACE_NAMESPACE} kube-prometheus-stack -o
yaml
```

The output looks similar to this:

```
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: kube-prometheus-stack
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: kube-prometheus-stack-44.2.1
    kind: ClusterApp
  configOverrides:
    name: kube-prometheus-stack-overrides-attached
  clusterSelector:
    matchExpressions:
    - key: kommander.d2iq.io/cluster-name
      operator: In
      values:
      - attached-cluster1
      - attached-cluster2
  clusterConfigOverrides:
    - configMapName: kps-cluster1-overrides
      clusterSelector:
        matchExpressions:
        - key: kommander.d2iq.io/cluster-name
          operator: In
          values:
          - attached-cluster1
    - configMapName: kps-cluster2-overrides
      clusterSelector:
        matchExpressions:
        - key: kommander.d2iq.io/cluster-name
          operator: In
          values:
          - attached-cluster2
```

Here you can see that `kube-prometheus-stack` has been enabled for the `attached-cluster1` and `attached-cluster2`. There is also a custom configuration for each of the clusters: `kps-cluster1-overrides` and `kps-cluster2-overrides`.

2. Edit the file by deleting the `configMapName` entry of the cluster for which you would like to delete the customization. This is located under the `clusterConfigOverrides`:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: kube-prometheus-stack
```

```
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    kind: ClusterApp
    name: kube-prometheus-stack-44.2.1
  configOverrides:
    name: kube-prometheus-stack-ws-overrides
  clusterSelector:
    matchExpressions:
      - key: kommander.d2iq.io/cluster-name
        operator: In
        values:
        - attached-cluster1
  clusterConfigOverrides:
    - configMapName: kps-cluster1-overrides
      clusterSelector:
        matchExpressions:
        - key: kommander.d2iq.io/cluster-name
          operator: In
          values:
          - attached-cluster1
EOF
```

Compare steps one and two for a reference of how an entry should be deleted.

3. Before deleting a `ConfigMap` that contains your customization, **ensure you will NOT require it at a later time**. It is not possible to restore a deleted `ConfigMap`.
   If you choose to delete it, run:

```
kubectl delete configmap <name_configmap> -n ${WORKSPACE_NAMESPACE}
```

It is **NOT** possible to delete a `ConfigMap` that is being actively used and referenced in the `configOverride` of any `AppDeployment`.

Verify the Current Configuration of your Application

Refer to the Verify applications help (see page 0) to connect to the managed or attached cluster and check the status of the deployments.

If you want to know how the `AppDeployment` resource is currently configured, refer to the Print and review the state of your AppDeployment (see page 548) section.

6.4.6.2.6 Verify the Current Application Configuration

Enterprise Gov Advanced

6.4.6.2.6.1 Verify the Current Configuration of your Application

Refer to the Verify applications help (see page 0) to connect to the managed or attached cluster and check the status of the deployments.

If you want to know how the `AppDeployment` resource is currently configured, refer to the Print and review the state of your AppDeployment (see page 548) section.

6.4.7 Workspace Catalog Applications

Enterprise Gov Advanced

Catalog applications are any third-party or open source applications that appear in the Catalog. These applications are deployed to be used for customer workloads.

D2iQ provides DKP Catalog Applications (see page 586) for use in your environment.

- Workspace DKP Catalog Applications (see page 586)
- Deployment of Catalog Applications in Workspaces (see page 594)
- Workspace Catalog Application Upgrades (see page 597)
- Custom Applications (see page 598)

6.4.7.1 Workspace DKP Catalog Applications

Enterprise Gov Advanced

Catalog applications are applications provided by D2iQ for use in your environment.

6.4.7.1.1 Prerequisites

- Ensure your clusters run on a supported Kubernetes version for this release of DKP (see page 1543), and that said Kubernetes version is also compatible with your catalog application version (see page 587).
- For customers with an **Enterprise license** (see page 75) and a multi-cluster environment, we recommend keeping all clusters on the same Kubernetes version. This ensures your DKP catalog application can run on all clusters in a given workspace.

6.4.7.1.1.1 Install the DKP Catalog via the CLI

Follow these steps to install the DKP catalog from the CLI.

1. Refer to Install DKP in an Air-gapped Environment with Catalog Applications (see page 1267) instructions, if you are running in air-gapped environment.
2. Set the `WORKSPACE_NAMESPACE` environment variable to the name of your workspace's namespace:

   ```
   export WORKSPACE_NAMESPACE=<workspace namespace>
   ```

3. Create the `GitRepository`:

   ```
   kubectl apply -f - <<EOF
   apiVersion: source.toolkit.fluxcd.io/v1beta1
   kind: GitRepository
   metadata:
     name: dkp-catalog-applications
     namespace: ${WORKSPACE_NAMESPACE}
     labels:
       kommander.d2iq.io/gitapps-gitrepository-type: catalog
       kommander.d2iq.io/gitrepository-type: catalog
   spec:
     interval: 1m0s
     ref:
       tag: v2.5.2
     timeout: 20s
     url: https://github.com/mesosphere/dkp-catalog-applications
   EOF
   ```

4. Verify that you can see the DKP workspace catalog `Apps` available in the UI (in the Applications section in said workspace), and in the CLI, using `kubectl`:

   ```
   kubectl get apps -n ${WORKSPACE_NAMESPACE}
   ```

6.4.7.1.1.2 Workspace DKP Catalog Applications

Ensure that your DKP Catalog application (see page 0) is compatible with:

- The Kubernetes version in all the Managed and Attached clusters of the workspace where you want to install the catalog application.
- The range of Kubernetes versions supported in this release of DKP (see page 1543).

If your current Catalog application version is not compatible, upgrade the application to a compatible version.

Name	App ID	Compatible Kubernetes versions
kafka-operator-0.20.0	kafka-operator	1.21
kafka-operator-0.20.2	kafka-operator	1.21 - 1.24
kafka-operator-0.23.0-dev.0	kafka-operator	1.21 - 1.25
spark-operator-1.1.6	spark-operator	1.21
spark operator-1.1.17	spark-operator	1.21 - 1.25
zookeeper-operator-0.2.13	zookeeper-operator	1.21 - 1.24
zookeeper-operator-0.2.14	zookeeper-operator	1.21 - 1.25

6.4.7.1.2 Kafka in a Workspace

Enterprise Gov Advanced

Information about the Kafka Operator

Apache Kafka[404] is an open-source distributed event streaming platform used for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications. The Kafka Operator[405] is a Kubernetes operator to automate provisioning, management, autoscaling, and operations of Apache Kafka clusters deployed to Kubernetes. It works by watching custom resources, such as `KafkaClusters`, `KafkaUsers`, and `KafkaTopics`, to provision underlying Kubernetes resources (i.e `StatefulSets`) required for a production-ready Kafka Cluster.

6.4.7.1.2.1 Install

Follow these steps to install the Kafka operator in a workspace. This procedure results in a Kafka operator running in a workspace namespace, ready to manage and create Kafka clusters in any project namespaces. See the Kafka in a Project (see page 619) custom resource documentation for more information on creating Kafka clusters.

404 https://kafka.apache.org/
405 https://banzaicloud.com/docs/supertubes/kafka-operator/

Only install the Kafka operator once per workspace.

1. Follow the generic installation instructions for workspace catalog applications on the Application Deployment (see page 594) page.
2. Within the `AppDeployment`, update the `appRef` to specify the correct `kafka-operator` App. You can find the `appRef.name` by listing the available `Apps` in the workspace namespace:

```
kubectl get apps -n ${WORKSPACE_NAMESPACE}
```

Refer to the Kafka operator Helm Chart documentation[406] for details on custom configuration for the operator.

6.4.7.1.2.2 Uninstall via the CLI

Uninstalling the Kafka operator does not affect existing `KafkaCluster` deployments. After uninstalling the operator, you must manually remove any remaining Custom Resource Definitions (CRDs) from the operator.

1. Delete all of the deployed Kafka custom resources (see Deleting Kafka Custom Resources (see page 0)).
2. Uninstall a Kafka operator `AppDeployment`:

```
kubectl -n <workspace namespace> delete AppDeployment <name of AppDeployment>
```

3. Remove Kafka CRDs:

   NOTE: The CRDs are not finalized for deletion until you delete the associated custom resources.

```
kubectl delete crds kafkaclusters.kafka.banzaicloud.io
kafkausers.kafka.banzaicloud.io kafkatopics.kafka.banzaicloud.io
```

6.4.7.1.2.3 Resources

- Kafka Operator Documentation[407]
- Kafka Operator GitHub Repository[408]
- Sample Kafka Operator Custom Resources[409]

406 https://github.com/banzaicloud/koperator/tree/master/charts/kafka-operator#configuration
407 https://banzaicloud.com/docs/supertubes/kafka-operator/
408 https://github.com/banzaicloud/koperator
409 https://github.com/banzaicloud/koperator/tree/master/config/samples

- Apache Kafka Documentation[410]

6.4.7.1.3 Spark Operator in a Workspace

Enterprise Gov Advanced

How to spin up your Spark Operator

The Kubernetes Operator for Apache Spark aims to make specifying and running Spark applications as easy and idiomatic as running other workloads on Kubernetes. It uses Kubernetes custom resources for specifying, running, and surfacing status of Spark applications. For a complete reference of the custom resource definitions, please refer to the API Definition. For details on its design, please refer to the design documentation. It requires Spark 2.3 and above that supports Kubernetes as a native scheduler backend.

> The default installation is basic, please provide your override configmap to enable desired Spark Operator features.

6.4.7.1.3.1 Install Spark Operator

You can find generic installation instructions for workspace catalog applications on the Application Deployment (see page 594) topic.

> Only install the Spark operator once per workspace.

For details on custom configuration for the operator, refer to the Spark Operator Helm Chart documentation[411].

After you finish the installation, see Spark Operator in a Project (see page 623) custom resource documentation for more information about how to submit your Spark jobs.

6.4.7.1.3.2 Sample Override Configuration File

> Ensure you configure the AppDeployment with the appropriate override configmap.

410 https://kafka.apache.org/documentation/
411 https://github.com/mesosphere/spark-on-k8s-operator/blob/d2iq-master/charts/spark-operator-chart/README.md

- Using UI

```
podLabels:
  owner: john
  team: operations
```

- Using CLI
  See Application Deployment (see page 594) for details.

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: spark-operator-overrides
data:
  values.yaml: |
    configInline:
      podLabels:
        owner: john
        team: operations
EOF
```

6.4.7.1.3.3 Uninstall via the CLI

Uninstalling the Spark Operator does not affect existing **SparkApplication** and **ScheduledSparkApplication** custom resources. You need to manually remove any leftover custom resources and CRDs from the operator. Please refer to deleting Spark Operator custom resources (see page 0).

Follow these steps:

1. Uninstall the Spark Operator `AppDeployment`:

```
kubectl -n <your workspace namespace> delete AppDeployment <name of
AppDeployment>
```

2. Remove the Spark Operator Service Account:

```
# <name of service account> is spark-operator-service-account if you didn't
override the RBAC resources.
```

```
kubectl -n <your workspace namespace> delete serviceaccounts <name of service
account>
```

3. Remove the Spark Operator CRDs:

   NOTE: The CRDs are not finalized for deletion until you delete the associated custom resources.

   ```
   kubectl delete crds scheduledsparkapplications.sparkoperator.k8s.io
   sparkapplications.sparkoperator.k8s.io
   ```

6.4.7.1.3.4 Resources

Here are some resources to learn more about Spark Operator:

- Spark Operator Documentation[412]
- Spark Operator Quick Start Guide[413]
- Spark Operator User Guide[414]
- Spark Documentation[415]

6.4.7.1.4 Zookeeper Operator in a Workspace

Enterprise Gov Advanced

ZooKeeper[416] is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. The ZooKeeper operator[417] is a Kubernetes operator that handles the provisioning and management of ZooKeeper clusters. It works by watching custom resources, such as `ZookeeperClusters`, to provision the underlying Kubernetes resources (`StatefulSets`) required for a production-ready ZooKeeper Cluster.

6.4.7.1.4.1 Install Zookeeper

Follow these steps to install the ZooKeeper operator in a workspace. This procedure results in a ZooKeeper operator running in a workspace namespace, ready to manage and create ZooKeeper clusters in any project namespaces. See the ZooKeeper in a Project (see page 621) custom resource documentation for more information on creating ZooKeeper clusters.

412 https://github.com/mesosphere/spark-on-k8s-operator/blob/d2iq-master/README.md
413 https://github.com/mesosphere/spark-on-k8s-operator/blob/d2iq-master/docs/quick-start-guide.md
414 https://github.com/mesosphere/spark-on-k8s-operator/blob/d2iq-master/docs/user-guide.md
415 https://spark.apache.org/docs/latest/
416 https://zookeeper.apache.org/index.html
417 https://github.com/pravega/zookeeper-operator

Only install the ZooKeeper operator once per workspace.

1. Follow the generic installation instructions for workspace catalog applications on the Application Deployment (see page 594) page.
2. Within the `AppDeployment`, update the `appRef` to specify the correct `zookeeper-operator` App. You can find the `appRef.name` by listing the available `Apps` in the workspace namespace:

```
kubectl get apps -n ${WORKSPACE_NAMESPACE}
```

For details on custom configuration for the operator, please refer to the ZooKeeper operator Helm Chart documentation[418].

6.4.7.1.4.2 Uninstall via the CLI

Uninstalling the ZooKeeper operator will not directly affect any running `ZookeeperClusters`. By default, the operator waits for any `ZookeeperClusters` to be deleted before it will fully uninstall (you can set `hooks.delete: true` in the application configuration to disable this behavior). After uninstalling the operator, you need to manually clean up any leftover Custom Resource Definitions (CRDs).

1. Delete all `ZookeeperClusters`, see Deleting ZooKeeper Clusters (see page 623).
2. Uninstall a ZooKeeper operator `AppDeployment`:

```
kubectl -n <workspace namespace> delete AppDeployment <name of AppDeployment>
```

3. Remove ZooKeeper CRDs:

   WARNING: Once you remove the CRDs, all deployed ZookeeperClusters will be deleted!

```
kubectl delete crds zookeeperclusters.zookeeper.pravega.io
```

6.4.7.1.4.3 Resources

- ZooKeeper Operator Documentation[419]
- ZooKeeper Cluster Helm Chart[420]

418 https://github.com/pravega/zookeeper-operator/tree/master/charts/zookeeper-operator#configuration
419 https://github.com/pravega/zookeeper-operator
420 https://github.com/pravega/zookeeper-operator/tree/master/charts/zookeeper

- ZooKeeper Documentation[421]

6.4.7.2 Deployment of Catalog Applications in Workspaces

Enterprise Gov Advanced

Deploy applications to attached clusters using the CLI

This topic describes how to use the CLI to deploy a workspace catalog application to attached clusters within a workspace. To deploy an application to selected clusters within a workspace, refer to the Cluster-scoped Configuration (see page 577) section of the documentation.

6.4.7.2.1 Prerequisites

Before you begin, you must have:

- A running cluster with Kommander installed (see page 1227). The cluster should be on a supported Kubernetes version for this release of DKP (see page 1537), and also compatible with the catalog application version (see page 587) you wish to install.
- An Attach an Existing Kubernetes Cluster (see page 673) section of the documentation completed.

Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace the attached cluster exists in:

```
export WORKSPACE_NAMESPACE=<workspace_namespace>
```

After creating a GitRepository, use either the DKP UI or the CLI to enable your catalog applications.

From within a workspace, you can enable applications to deploy. Verify that an application has successfully deployed via the CLI (see page 596).

6.4.7.2.2 Enable (and Customize) the Application using the DKP UI

Follow these steps to enable your catalog applications from the DKP UI:

1. **Enterprise only:** from the top menu bar, select your target workspace.
2. Select **Applications** from the sidebar menu to browse the available applications from your configured repositories.
3. Select the three dot button from the bottom-right corner of the desired application card, and then select **Enable**.

421 https://zookeeper.apache.org/documentation

4. If available, select a version from the drop-down menu. This drop-down menu will only be visible if there is more than one version.
5. (Optional) If you want to override the default configuration values, copy your customized values into the text editor under **Configure Service** or upload your YAML file that contains the values:

```
someField: someValue
```

6. Confirm the details are correct, and then select the **Enable** button.

For all applications, you must provide a display name and an ID which is automatically generated based on what you enter for the display name, unless or until you edit the ID directly. The ID must be compliant with Kubernetes DNS subdomain name validation rules[422].

Alternately, you can use the CLI to enable your catalog applications (see page 0).

6.4.7.2.3 Enable the Application using the CLI

See Workspace Catalog Applications (see page 586) for the list of available applications that you can deploy on the attached cluster.

1. Enable a supported application to deploy to your Attached Cluster (see page 673) with an `AppDeployment` resource.
2. Within the `AppDeployment`, define the `appRef` to specify which `App` to enable:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: spark-operator
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: spark-operator-1.1.17
    kind: App
EOF
```

- The `appRef.name` must match the app `name` from the list of available catalog applications.
- Create the resource in the workspace you just created, which instructs Kommander to deploy the `AppDeployment` to the `KommanderCluster`s in the same workspace.

422 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#dns-subdomain-names

6.4.7.2.4 Enable an Application with a Custom Configuration using the CLI

Follow these steps:

Provide the name of a `ConfigMap` in the `AppDeployment`, which provides custom configuration on top of the default configuration:

1.
```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: spark-operator
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: spark-operator-1.1.17
    kind: App
  configOverrides:
    name: spark-operator-overrides
EOF
```

2. Create the `ConfigMap` with the name provided in the step above, with the custom configuration:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: spark-operator-overrides
data:
  values.yaml: |
    configInline:
      uiService:
        enable: false
EOF
```

Kommander waits for the `ConfigMap` to be present before deploying the `AppDeployment` to the managed or attached clusters.

6.4.7.2.5 Verify Applications

The applications are now enabled. Connect to the attached cluster and check the `HelmReleases` to verify the deployment:

```
kubectl get helmreleases -n ${WORKSPACE_NAMESPACE}
```

The output looks similar to this:

```
NAMESPACE              NAME              READY   STATUS
AGE
workspace-test-vjsfq   spark-operator    True    Release reconciliation succeeded
7m3s
```

6.4.7.3 Workspace Catalog Application Upgrades

Enterprise Gov Advanced

Upgrade catalog applications using the CLI and UI

6.4.7.3.1 Prerequisites

- Ensure your clusters run on a supported Kubernetes version for this release of DKP (see page 1543), and that said Kubernetes version is also compatible with your catalog application version (see page 0).
- For customers with an **Enterprise license** (see page 75) and a multi-cluster environment, we recommend keeping all clusters on the same Kubernetes version. This ensures your DKP catalog application can run on all clusters in a given workspace.
- Konvoy installs a new Kubernetes version with each upgrade, therefore it is important that your Catalog application (see page 587) is compatible with the Kubernetes version that comes with this release of DKP.

> If your current Catalog application version is not compatible with this release's Kubernetes version, upgrade the application to a compatible version BEFORE upgrading the Konvoy component on *Managed clusters* (see page 97) or BEFORE upgrading the Kubernetes version on *Attached clusters* (see page 97).

- **For air-gapped environment with catalog applications only:** Ensure you have updated your catalog repository before upgrading. The catalog repository should contain the Docker registry containing the DKP catalog application (see page 597) images and a charts bundle file containing DKP catalog applications (see page 0) charts.

6.4.7.3.1.1 Upgrade Catalog Applications

Before upgrading also, keep in mind the distinction between Platform applications and Catalog applications. Platform applications are deployed and upgraded as a set for each cluster or workspace. Catalog applications are deployed separately, so that you can deploy and upgrade them individually for each project.

Upgrade with UI

Follow these steps to upgrade an application from the DKP UI:

1. From the top menu bar, select your target workspace.
2. Select **Applications** from the sidebar menu.
3. Select the three dot button from the bottom-right corner of the desired application tile, and then select **Edit**.
4. Select the **Version** drop-down, and select a new version. This drop-down will only be available if there is a newer version to upgrade to.
5. Select **Save**.

Upgrade with CLI

Please note that the below commands are using the workspace **name** and not namespace. You can retrieve the workspace name by running the command: `dkp get workspaces`
To view a list of the deployed `apps` to your workspace, you can run the command: `dkp get appdeployments --workspace=<workspace-name>`

```
dkp upgrade catalogapp <appdeployment-name> --workspace=<my-workspace-name> --to-version=<version.number>
```

For example, the following command upgrades the Kafka Operator application, named `kafka-operator-abc`, in a workspace to version `0.23.0-dev.0`:

```
dkp upgrade catalogapp kafka-operator-abc --workspace=my-workspace --to-version=0.23.0-dev.0
```

Platform applications cannot be upgraded on a one-off basis, and must be upgraded in a single process for each workspace. If you attempt to upgrade a platform application with these commands, you receive an error and the application is not upgraded.

6.4.7.4 Custom Applications

Enterprise Gov Advanced

Custom applications are third-party applications you have added to the DKP Catalog.

Custom applications are any third-party applications that are not provided in the DKP Application Catalog. Custom applications can leverage applications from the DKP Catalog or be fully-customized. There is no expectation of support by D2iQ for a Custom application. Custom applications can be deployed on Konvoy clusters or on any D2iQ supported 3rd party Kubernetes distribution.

6.4.7.4.1 Create a Git Repository

Enterprise Gov Advanced

Create a Git Repository in the Workspace namespace

Use the CLI to create the GitRepository resource and add a new repository to your Workspace.

1. Refer to Air-gapped Environment (see page 1258) setup instructions section, if you are running in air-gapped environment.
2. Set the `WORKSPACE_NAMESPACE` environment variable to the name of your workspace's namespace:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

3. Adapt the URL of your Git repository:

   ```
   kubectl apply -f - <<EOF
   apiVersion: source.toolkit.fluxcd.io/v1beta1
   kind: GitRepository
   metadata:
     name: example-repo
     namespace: ${WORKSPACE_NAMESPACE}
     labels:
       kommander.d2iq.io/gitapps-gitrepository-type: dkp
       kommander.d2iq.io/gitrepository-type: catalog
   spec:
     interval: 1m0s
     ref:
       branch: <your-target-branch-name> # e.g., main
     timeout: 20s
     url: https://github.com/<example-org>/<example-repo>
   EOF
   ```

4. Ensure the status of the `GitRepository` signals a ready state:

```
kubectl get gitrepository example-repo -n ${WORKSPACE_NAMESPACE}
```

The repository commit also displays the ready state:

```
NAME           URL                                                        READY
STATUS                                                                    AGE
example-repo https://github.com/example-org/example-repo                 True
Fetched revision: master/6c54bd1722604bd03d25dcac7a31c44ff4e03c6a   11m
```

For more information on the GitRepository resource fields and how to make Flux aware of credentials required to access a private Git repository, see the Flux documentation[423].

6.4.7.4.1.1 Troubleshoot

To troubleshoot issues with adding the GitRepository, review the following logs:

```
kubectl -n kommander-flux logs -l app=source-controller
[...]
kubectl -n kommander-flux logs -l app=kustomize-controller
[...]
kubectl -n kommander-flux logs -l app=helm-controller
[...]
```

6.4.7.4.1.2 Related Information

- Flux[424]
- Flux docs[425]

6.4.7.4.2 Git Repository Structure

Enterprise Gov Advanced

Git repositories must be structured in a specific manner for defined applications to be processed by Kommander.

You must structure your git repository based on the following guidelines, for your applications to be processed properly by Kommander so that they can be deployed.

423 https://fluxcd.io/flux/components/source/gitrepositories/#secret-reference
424 https://fluxcd.io/
425 https://fluxcd.io/docs

6.4.7.4.2.1 Git Repository Directory Structure

Use the following basic directory structure for your git repository:

```
├── helm-repositories
│   ├── <helm repository 1>
│   │   ├── kustomization.yaml
│   │   └── <helm repository name>.yaml
│   └── <helm repository 2>
│     ├── kustomization.yaml
│     └── <helm repository name>.yaml
└── services
    ├── <app name>
    │   ├── <app version1> # semantic version of the app helm chart. e.g., 1.2.3
    │   │   ├── defaults
    │   │   │   ├── cm.yaml
    │   │   │   └── kustomization.yaml
    │   │   ├── <app name>.yaml
    │   │   └── kustomization.yaml
    │   ├── <app version2> # another semantic version of the app helm chart. e.g.,
2.3.4
    │   │   ├── defaults
    │   │   │   ├── cm.yaml
    │   │   │   └── kustomization.yaml
    │   │   ├── <app name>.yaml
    │   │   └── kustomization.yaml
    │   └── metadata.yaml
    └── <another app name>
    ...
```

- Define applications in the `services/` directory.
- You can define multiple versions of an application, under different directories nested under the `services/<app name>/` directory.
- Define application manifests, such as a HelmRelease[426], under each versioned directory `services/<app name>/<version>/` in `<app name>.yaml` which is listed in the `kustomization.yaml` Kubernetes Kustomization file. For more information, see the Kubernetes Kustomization docs[427].
- Define the default values ConfigMap for `HelmReleases` in the `services/<app name>/<version>/defaults` directory, accompanied by a `kustomization.yaml` Kubernetes Kustomization file pointing to the `ConfigMap` file.
- Define the `metadata.yaml` of each application under the `services/<app name>/` directory. For more information, see the Workspace Application Metadata docs (see page 602).

See the DKP Catalog repository[428] for an example of how to structure custom catalog Git repositories.

426 https://fluxcd.io/docs/components/helm/helmreleases/
427 https://kubectl.docs.kubernetes.io/references/kustomize/kustomization/
428 https://github.com/mesosphere/dkp-catalog-applications

6.4.7.4.2.2 Helm Repositories

You must include the `HelmRepository` that is referenced in each `HelmRelease`'s `Chart` spec.

Each `services/<app name>/<version>/kustomization.yaml` must include the path of the YAML file that defines the `HelmRepository`. For example:

```
# services/<app name>/<version>/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - <app name>.yaml
  - ../../../helm-repositories/<helm repository 1>
```

For more information, see the flux documentation about HelmRepositories[429].

6.4.7.4.2.3 Substitution Variables

Some substitution variables[430] are provided.

- `${releaseName}`: For each App deployment, this variable is set to the `AppDeployment` name. Use this variable to prefix the names of any resources that are defined in the application directory in the Git repository so that multiple instances of the same application can be deployed. If you create resources without using the `releaseName` prefix (or suffix) in the name field, there can be conflicts if the same named resource is created in that same namespace.
- `${releaseNamespace}`: The namespace of the Workspace.
- `${workspaceNamespace}`: The namespace of the Workspace that the Workspace belongs to.

6.4.7.4.2.4 Related Information

- Flux[431]
- Flux docs[432]
- Getting started with Flux guide[433]

6.4.7.4.3 Workspace Application Metadata

Enterprise Gov Advanced

429 https://fluxcd.io/docs/components/source/helmrepositories/
430 https://fluxcd.io/docs/components/kustomize/kustomization/#variable-substitution
431 https://fluxcd.io/
432 https://fluxcd.io/docs
433 https://fluxcd.io/docs/get-started/

To display more information about custom applications in the UI, define a metadata.yaml file for each application in the git repository.

You can define how custom applications display in the DKP UI by defining a `metadata.yaml` file for each application in the git repository. You must define this file at `services/<application>/metadata.yaml` for it to process correctly.

You can define the following fields:

Field	Default	Description
displayName	falls back to App ID	Display name of the application for the UI.
description	“”	Short description, should be a sentence or two, displayed in the UI on the application card.
category	general	1 or more categories for this application. Categories are used to group applications in the UI.
overview		Markdown overview used on the application detail page in the UI.
icon		Base64 encoded icon SVG file used for application logos in the UI.
scope	project	List of scopes, can be set only to project or workspace currently.

None of these fields are required for the application to display in the UI.

Here is an example `metadata.yaml` file:

```
displayName: Prometheus Monitoring Stack
description: Stack of applications that collect metrics and provides visualization
and alerting capabilities. Includes Prometheus, Prometheus Alertmanager and Grafana.
category:
  - monitoring
overview: >
  # Overview
  A stack of applications that collects metrics and provides visualization and
alerting capabilities. Includes Prometheus, Prometheus Alertmanager and Grafana.
```

```
  ## Dashboards
  By deploying the Prometheus Monitoring Stack, the following platform applications
and their respective dashboards are deployed. After deployment to clusters in a
workspace, the dashboards are available to access from a respective cluster's detail
page.

  ### Prometheus

  A software application for event monitoring and alerting. It records real-time
metrics in a time series database built using a HTTP pull model, with flexible and
real-time alerting.

  - [Prometheus Documentation - Overview](https://prometheus.io/docs/introduction/
overview/)

  ### Prometheus Alertmanager
  A Prometheus component that enables you to configure and manage alerts sent by the
Prometheus server and to route them to notification, paging, and automation systems.

  - [Prometheus Alertmanager Documentation - Overview](https://prometheus.io/docs/
alerting/latest/alertmanager/)

  ### Grafana
  A monitoring dashboard from Grafana that can be used to visualize metrics collected
by Prometheus.

  - [Grafana Documentation](https://grafana.com/docs/)
icon:
PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHZpZXdCb3g9IjAgMCAzMDAgMzAwIiBzd
HlsZT0iZW5hYmxlLWJhY2tncm91bmQ6bmV3IDAgMCAzMDAgMzAwIiB4bWw6c3BhY2U9InByZXNlcnZlIj48cG
F0aCBkPSJNMTUwIDUwQzk0LjggNTAgNTAgOTQuOCA1MCAxNTBzNDQuOCAxMDAgMTAwIDEwMCAxMDAtNDQuOCA
xMDAtMTAwUzIwNS4yIDUwIDE1MCA1MHptMCAxODcuMmMtMTUuNyAwLTI4LjUtMTAuNS0yOC41LTIzLjRoNTYu
OWMuMSAxMi45LTEyLjcgMjMuNC0yOC40IDIzLjR6bTQ3LTMxLjJoLTk0di0xN2g5NHYxN3ptLS4zLTI1LjloL
TkzLjRjLS4zLS40LS42LS43LS45LTEuMS05LjYtMTEuNy0xMS45LTE3LjgtMTQuMS0yNC4wLS4yIDExLjcgMi
40IDIwIDQuMyAwIDAgNC4zIDEgMTAuNSAyLjEtNi03LTkuNi0xNi05LjYtMjUuMSAwLTIwIDE1LjQtMzcuNiA
5LjgtNTEuNyA1LjQuNCAxMS4yIDExLjQgMTEuNiAyOC41IDUuNy03LjkgOC4xLTIyLjQgOC4xLTMxLjMgMC05
LjIgNi4xLTE5LjkgMTIuMS0yMC4yLTUuNCA4LjkgMS40OIDE2LjUgNy40IDM1LjUgMi4zIDcuMSAyIDE5LjEgM
y43IDI2LjcuNi0xNS44IDMuMy0zOC44IDEzLjMtNDYuNy00LjQgMTAgLjcgMjIuNSA0LjEgMjguNSA1LjYgOS
43IDkgMTcuMSA5IDMxIDAgOS4zLTMuNCAxOC4xLTkuMyAyNSA2LjYtMS4yIDExLjItMi40 IDExLjItMi40bDI
xLjQtNC4yYy4xIDAtMyAxMi44LTE0LjkgMjUuMXoiIHN0eWxlPSJmaWxsOiNmODQzMTEiLz48L3N2Zz4=
```

6.4.7.4.4 Enable a Custom Application from the Workspace Catalog

Enterprise Gov Advanced

Enable a Custom Application from the Workspace Catalog

After creating a GitRepository, you can either use the DKP UI or the CLI to enable your custom applications. To deploy an application to selected clusters within a workspace, refer to the cluster-scoped configuration (see page 577) section.

From within a workspace, you can enable applications to deploy. Verify that an application has successfully deployed via the CLI (see page 0).

6.4.7.4.4.1 Enable the Application Using the UI

Follow these steps:

1. From the top menu bar, select your target workspace.
2. Select **Applications** from the sidebar menu to browse the available applications from your configured repositories.
3. Select the three dot button from the bottom-right corner of the desired application tile, and then select **Enable**.
4. If available, select a version from the drop-down menu. This drop-down menu will only be visible if there is more than one version.
5. (Optional) If you want to override the default configuration values, copy your customized values into the text editor under **Configure Service** or upload your YAML file that contains the values:

   ```
   someField: someValue
   ```

6. Confirm the details are correct, and then select the **Enable** button.

For all applications, you must provide a display name and an ID which is automatically generated based on what you enter for the display name, unless or until you edit the ID directly. The ID must be compliant with Kubernetes DNS subdomain name validation rules[434].

Alternately, you can use the CLI to enable your custom applications (see page 606).

6.4.7.4.4.2 Prerequisites

- Determine the name of the workspace where you wish to perform the deployments. You can use the `dkp get workspaces` command to see the list of workspace names and their corresponding namespaces.
- Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace where the cluster is attached:

434 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#dns-subdomain-names

```
export WORKSPACE_NAMESPACE=<workspace_namespace>
```

6.4.7.4.4.3 Enable the Application using the CLI

Follow these steps:

1. Get the list of available applications to enable using the following command:

   ```
   kubectl get apps -n ${WORKSPACE_NAMESPACE}
   ```

2. Deploy one of the supported applications from the list with an `AppDeployment` resource.
3. Within the `AppDeployment`, define the `appRef` to specify which `App` will be enabled:

   ```
   cat <<EOF | kubectl apply -f -
   apiVersion: apps.kommander.d2iq.io/v1alpha3
   kind: AppDeployment
   metadata:
     name: my-custom-app
     namespace: ${WORKSPACE_NAMESPACE}
   spec:
     appRef:
       name: custom-app-0.0.1
       kind: App
   EOF
   ```

- The `appRef.name` must match the app `name` from the list of available catalog applications.
- Create the resource in the workspace you just created, which instructs Kommander to deploy the `AppDeployment` to the `KommanderCluster`s in the same workspace.

6.4.7.4.4.4 Enable an Application with a Custom Configuration using the CLI

Follow these steps:

1. Provide the name of a `ConfigMap` in the `AppDeployment`, which provides custom configuration on top of the default configuration:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: my-custom-app
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: custom-app-0.0.1
    kind: App
  configOverrides:
    name: my-custom-app-overrides
EOF
```

2. Create the `ConfigMap` with the name provided in the step above, with the custom configuration:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: my-custom-app-overrides
data:
  values.yaml: |
    someField: someValue
EOF
```

Kommander waits for the `ConfigMap` to be present before deploying the `AppDeployment` to the managed or attached clusters in the Workspace.

6.4.7.4.4.5 Verify Applications

After completing the previous steps, your applications are enabled. Connect to the attached cluster and check the `HelmReleases` to verify the deployments:

```
kubectl get helmreleases -n ${WORKSPACE_NAMESPACE}
```

The output looks similar to this:

```
NAMESPACE               NAME            READY   STATUS
AGE
workspace-test-vjsfq    my-custom-app   True    Release reconciliation succeeded
7m3s
```

6.4.8 Workspace Role Bindings

Enterprise Gov Advanced

Workspace Role Bindings grant access to specified Workspace Roles for a specified group of people.

6.4.8.1 Configure Workspace Role Bindings

Before you can create a Workspace Role Binding, ensure you have created a Group. A Kommander Group can contain one or several Identity Provider users, groups or both.

You can assign a role to this Kommander Group:

1. From the top menu bar, select your target workspace.
2. Select **Access Control** in the **Administration** section of the sidebar menu.
3. Select the **Cluster Role Bindings** tab, and then select the **Add Roles** button next to the group you want.
4. Select the Role, or Roles, you want from the drop-down menu and select **Save**.

6.5 Projects

Enterprise Gov Advanced

Multi-cluster Configuration Management

Projects support the management of configMaps, continuous deployments, secrets, services, quotas, and role-based access control and multi-tenant logging by leveraging federated resources. When a Project is created, DKP creates a federated namespace that is propagated to the Kubernetes clusters associated with this Project.

Federation in this context means that a common configuration is pushed out from a central location (DKP) to all Kubernetes clusters, or a pre-defined subset group, under DKP management. This pre-defined subset group of Kubernetes clusters is called a Project.

Projects enable teams to deploy their configurations and services to clusters in a consistent way. Projects enable central IT or a business unit to share their Kubernetes clusters among several teams. Using Projects, DKP leverages Kubernetes Cluster Federation (KubeFed) to coordinate the configuration of multiple Kubernetes clusters.

Kommander allows a user to use labels to select, manually or dynamically, the Kubernetes clusters associated with a Project.

6.5.1 Project Namespace

Project Namespaces isolate configurations across clusters. Individual standard Kubernetes namespaces are automatically created on all clusters belonging to the project. When creating a new project, you can customize the Kubernetes namespace name that is created. It is the grouping of all of these individual standard Kubernetes namespaces that make up the concept of a Project Namespace. A Project Namespace is a Kommander specific concept.

6.5.2 Create a Project

When you create a Project, you must specify a Project Name, a Namespace Name (optional) and a way to allow Kommander to determine which Kubernetes clusters will be part of this project.

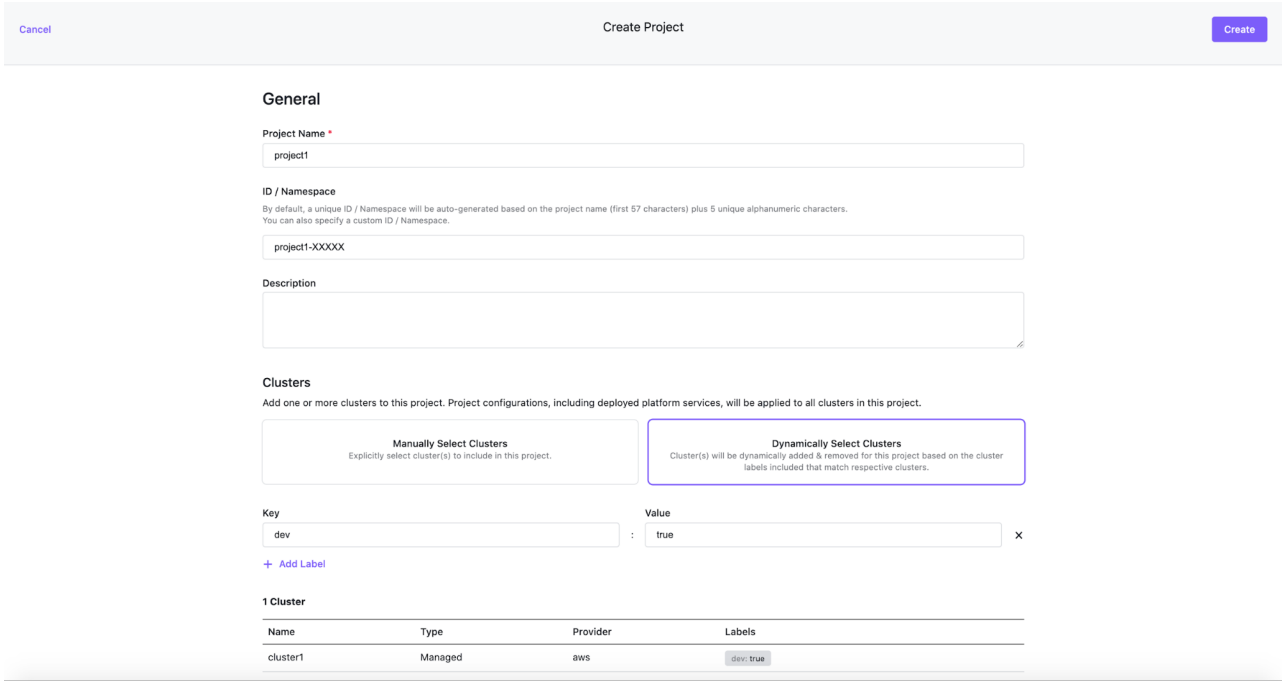
As mentioned previously, a Project Namespace corresponds to a Kubernetes Federated Namespace. By default, the name of the namespace is auto-generated based on the project name (first 57 characters) plus 5 unique alphanumeric characters. You can specify a namespace name, but you must ensure it does not conflict with any existing namespace on the target Kubernetes clusters, that will be a part of the Project.

To determine which Kubernetes clusters will be part of this project, you can either select manually existing clusters or define labels that Kommander will use to dynamically add clusters. The latter is recommended because it will allow you to deploy additional Kubernetes clusters later and to have them automatically associated with Projects based on their labels.

To create a Project, you can either use the DKP UI or create a Project object on the Kubernetes cluster where Kommander is running (using kubectl or the Kubernetes API). The latter allows you to configure Kommander resources in a declarative way. It is available for all kinds of Kommander resources.

6.5.3 Create a Project - UI Method

Here is an example of what it looks like to create a project using the DKP UI:



6.5.4 Create a Project - CLI Method

The following sample is a YAML Kubernetes object for creating a Kommander Project. This example does not work verbatim because it depends on a workspace name that has been previously created and does not exist by default in your cluster. Use this as an example format and fill in the workspace name and namespace name appropriately along with the proper labels.

```
apiVersion: workspaces.kommander.mesosphere.io/v1alpha1
kind: Project
metadata:
  name: My-Project-Name
  namespace: my-project-k8s-namespace-name
spec:
  workspaceRef:
    name: myworkspacename
  namespaceName: myworkspacename-di3tx
  placement:
    clusterSelector:
      matchLabels:
        cluster: prod
```

The following procedures are supported for projects:

- Project Applications (see page 611)
- Project Deployments (see page 636)
- Project Role Bindings (see page 651)
- Project Roles (see page 655)

- Project ConfigMaps (see page 658)
- Project Secrets (see page 659)
- Project Quotas & Limit Ranges (see page 661)
- Project Network Policies (see page 663)

6.5.5 Project Applications

Enterprise Gov Advanced

This section documents the applications and application types that you can utilize with DKP.

Application types are:

- Catalog Applications (see page 617) are either pre-packaged applications from the D2iQ Application Catalog, or custom applications that you maintain for your teams or organization.
  - DKP Applications (see page 618) are applications that are provided by D2iQ and added to the Catalog.
  - Custom Applications (see page 627) are applications you create and add to the Catalog.
- Platform Applications (see page 611) are applications integrated into Kommander.

When deploying and upgrading applications, platform applications come as a bundle; they are tested as a single unit and you must deploy or upgrade them in a single process, for each workspace. This means all clusters in a workspace have the same set and versions of platform applications deployed. Whereas catalog applications are individual, so you can deploy and upgrade them individually, for each project.

6.5.5.1 Project Platform Applications

Enterprise Gov Advanced

How project Platform applications work

The following table describes the list of applications that can be deployed to attached clusters within a project.

Review the Project Platform Application Requirements (see page 616) to ensure that the attached clusters in the project have sufficient resources.

From within a project, you can enable applications to deploy. Verify that an application has successfully deployed via the CLI (see page 564).

6.5.5.1.1 Enable Applications in a Project Using the UI

1. From the top menu bar, select your target workspace.
2. Select **Projects** from the sidebar menu.
3. Select your project from the list.
4. Select the **Applications** tab to browse the available applications.
5. Select the three dot button from the bottom-right corner of the desired application tile, and then select **Enable**.
6. To override the default configuration values, copy your values content into the text editor under **Configure Service** or just upload your yaml file that contains the values:

   ```
   someField: someValue
   ```

7. Select the **Enable** button.

To use the CLI to enable or disable applications, see Application Deployment (see page 564)

There may be dependencies between the applications, which are listed in Project Platform Application Dependencies (see page 615). Review them carefully prior to customizing to ensure that the applications are deployed successfully.

6.5.5.1.2 Platform Applications

NAME	APP ID	Deployed by default
project-grafana-logging-6.38.1	project-grafana-logging	False
project-grafana-loki-0.69.4	project-grafana-loki	False
project-logging-1.0.3	project-logging	False

6.5.5.1.3 Upgrade Platform Applications from the CLI

Platform Applications within a Project are automatically upgraded when the Workspace that a Project belongs to is upgraded. See Upgrade Kommander (see page 1369) for more information on how to upgrade these applications.

6.5.5.1.4 Deploy Project Platform Applications

Enterprise Gov Advanced

Deploy applications to attached clusters in a project using the CLI

This topic describes how to use the CLI to deploy an application to attached clusters within a project. To use the DKP UI to deploy applications, see Deploy Applications in a Project (see page 611).

See Project Platform Applications (see page 611) for a list of all applications and those that are enabled by default.

6.5.5.1.4.1 Prerequisites

Before you begin, you must have:

- A running cluster with Kommander installed (see page 117).
- An existing Kubernetes cluster attached to Kommander (see page 673).

Set the `WORKSPACE_NAME` environment variable to the name of the workspace where the cluster is attached:

```
export WORKSPACE_NAME=<workspace_name>
```

Set the `WORKSPACE_NAMESPACE` environment variable to the namespace of the above workspace:

```
export WORKSPACE_NAMESPACE=$(kubectl get namespace --
selector='workspaces.kommander.mesosphere.io/workspace-name=${WORKSPACE_NAME}' -o
jsonpath='{.items[0].metadata.name}')
```

Set the `PROJECT_NAME` environment variable to the name of the project in which the cluster is included:

```
export PROJECT_NAME=<project_name>
```

Set the `PROJECT_NAMESPACE` environment variable to the name of the above project's namespace:

```
export PROJECT_NAMESPACE=$(kubectl get project ${PROJECT_NAMESPACE} -n $
{WORKSPACE_NAMESPACE} -o jsonpath='{.status.namespaceRef.name}')
```

6.5.5.1.4.2 Deploy the Application

The list of available applications that can be deployed on the attached clusters in a project can be found in the project platform applications topic (see page 611).

1. Deploy one of the supported applications to your existing attached cluster (see page 673) with an `AppDeployment` resource. Provide the `appRef` and application version to specify which `App` is deployed:

```
dkp create appdeployment project-grafana-logging --app project-grafana-
logging-6.38.1 --workspace ${WORKSPACE_NAME} --project ${PROJECT_NAME}
```

2. Create the resource in the project you just created, which instructs Kommander to deploy the `AppDeployment` to the `KommanderClusters` in the same project.

- The `appRef.name` must match the app `name` from the list of available applications.
- Observe that the `dkp create` command must be run with both the `--workspace` and `--project` flags for project platform applications.

6.5.5.1.4.3 Deploy an Application with a Custom Configuration

Follow these steps:

1. Create the `AppDeployment` and provide the name of a `ConfigMap`, which provides custom configuration on top of the default configuration:

```
dkp create appdeployment project-grafana-logging --app project-grafana-
logging-6.38.1 --config-overrides project-grafana-logging-overrides --workspace
${WORKSPACE_NAME} --project ${PROJECT_NAMESPACE}
```

2. Create the `ConfigMap` with the name provided in the step above, with the custom configuration:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${PROJECT_NAMESPACE}
  name: project-grafana-logging-overrides
data:
  values.yaml: |
    datasources:
      datasources.yaml:
        apiVersion: 1
        datasources:
        - name: Loki
```

```
          type: loki
          url: "http://project-grafana-loki-loki-distributed-gateway"
          access: proxy
          isDefault: false
EOF
```

Kommander waits for the `ConfigMap` to be present before deploying the `AppDeployment` to the attached clusters.

6.5.5.1.4.4 Verify Applications

The applications are now enabled.

Export the `project_namespace` with this command:

```
export PROJECT_NAMESPACE=<project_namespace>
```

Connect to the attached cluster and check the `HelmReleases` to verify the deployment:

```
kubectl get helmreleases -n ${PROJECT_NAMESPACE}
NAMESPACE               NAME                        READY   STATUS
AGE
project-test-vjsfq      project-grafana-logging     True    Release reconciliation
succeeded   7m3s
```

Some of the supported applications have dependencies on other applications. See Project Application Dependencies (see page 615) for that table.

6.5.5.1.5 Project Platform Application Dependencies

Enterprise Gov Advanced

Dependencies between project platform applications

There are many dependencies between the applications that are deployed to a project's attached clusters. It is important to note these dependencies when customizing the platform applications to ensure that your services are properly deployed to the clusters. For more information on how to customize platform applications, see Platform Application Deployment (see page 613).

6.5.5.1.5.1 Application Dependencies

When deploying or troubleshooting applications, it helps to understand how applications interact and may require other applications as dependencies.

If an application's dependency does not successfully deploy, the application requiring that dependency does not successfully deploy.

The following sections detail information about the platform applications.

6.5.5.1.5.2 Logging

Collects logs over time from Kubernetes pods deployed in the project namespace. Also provides the ability to visualize and query the aggregated logs.

- project-logging[435]: Defines resources for the Logging Operator which uses them to direct the project's logs to its respective Grafana Loki application.
- project-grafana-loki[436]: A horizontally-scalable, highly-available, multi-tenant log aggregation system inspired by Prometheus.
- project-grafana-logging[437]: Logging dashboard used to view logs aggregated to Grafana Loki.

The project logging applications depend on the workspace logging applications (see page 798) being deployed.

Application	Required Dependencies
project-logging	logging-operator (workspace)
project-grafana-loki	project-logging, grafana-loki (workspace), logging-operator (workspace)
project-grafana-logging	project-grafana-loki

6.5.5.1.6 Project Platform Application Configuration Requirements

ENTERPRISE

435 https://grafana.com/oss/grafana/
436 https://grafana.com/oss/loki/
437 https://grafana.com/oss/grafana/

6.5.5.1.6.1 Project Platform Application Descriptions and Resource Requirements

Platform applications require more resources than solely deploying or attaching clusters into a project. Your cluster must have sufficient resources when deploying or attaching to ensure that the applications are installed successfully.

The following table describes all the platform applications that are available to the clusters in a project, minimum resource and persistent storage requirements, and whether they are enabled by default.

Name	Minimum Resources Suggested	Minimum Persistent Storage Required	Deployed by Default
project-grafana-logging	cpu: 200m memory: 100Mi		No
project-grafana-loki		# of PVs: 3 PV sizes: 10Gi x 3 (total: 30Gi)	No
project-logging			No

6.5.5.2 Project Catalog Applications

Enterprise Gov Advanced

Catalog applications are any third-party or open source applications that appear in the Catalog. These can be DKP applications provided by D2iQ for use in your environment, or Custom Applications (see page 627) that can be used but are not supported by D2iQ.

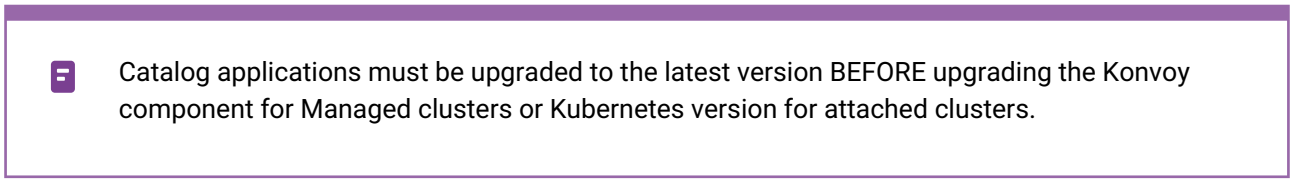
- Upgrading Project Catalog Applications (see page 617)
- Project-level DKP Applications (see page 618)
- Use Custom Resources with Workspace Catalog Applications (see page 627)
- Custom Project Applications (see page 627)

6.5.5.2.1 Upgrading Project Catalog Applications

Enterprise Gov Advanced

Before upgrading your catalog applications, verify the current and supported versions of the application. Also, keep in mind the distinction between Platform applications and Catalog applications. Platform

applications are deployed and upgraded as a set for each cluster or workspace. Catalog applications are deployed separately, so that you can deploy and upgrade them individually for each project.



Catalog applications must be upgraded to the latest version BEFORE upgrading the Konvoy component for Managed clusters or Kubernetes version for attached clusters.

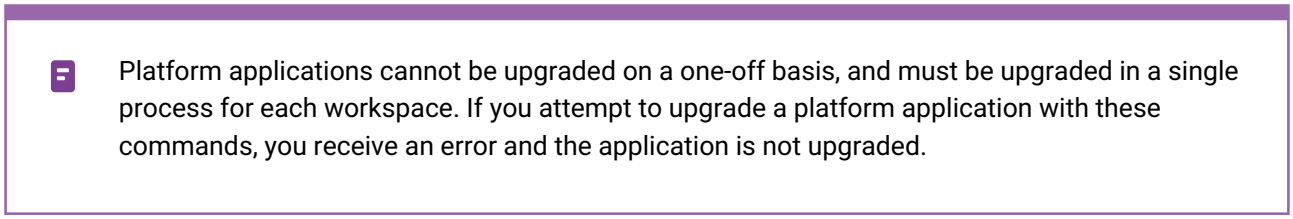
6.5.5.2.1.1 Upgrade with the UI

Follow these steps to upgrade an application from the DKP UI:

1. From the top menu bar, select your target workspace.
2. From the side menu bar, select **Projects**.
3. Select your target project.
4. Select **Applications** from the project menu bar.
5. Select the three dot button from the bottom-right corner of the desired application tile, and then select **Edit**.
6. Select the **Version** drop-down, and select a new version. This drop-down will only be available if there is a newer version to upgrade to.
7. Select **Save**.

6.5.5.2.1.2 Upgrade with the CLI

```
dkp upgrade catalogapp <appdeployment-name> --workspace=my-workspace --project=my-
project --to-version=<version.number>
```



Platform applications cannot be upgraded on a one-off basis, and must be upgraded in a single process for each workspace. If you attempt to upgrade a platform application with these commands, you receive an error and the application is not upgraded.

6.5.5.2.2 Project-level DKP Applications

Enterprise Gov Advanced

DKP applications are catalog applications provided by D2iQ for use in your environment.

Some DKP workspace catalog applications will provision `CustomResourceDefinitions`, which allow you to deploy Custom Resources to a Project. See your DKP workspace catalog application's documentation for instructions.

6.5.5.2.2.1 Kafka in a Project

Enterprise Gov Advanced

Deploying Kafka in a project

Get Started

To get started with creating and managing a Kafka Cluster in a project, you first need to deploy the Kafka operator (see page 588) and the ZooKeeper operator (see page 592) in the workspace where the project exists.

After deploying the Kafka operator, create Kafka Clusters by applying a `KafkaCluster` custom resource **on each attached cluster** in a project's namespace. Refer to the Kafka operator repository[438] for examples of the custom resources and their configurations.

Example Deployment

> If you need to manage these custom resources across all clusters in a project, it is recommended you use Project Deployments (see page 636) which enables you to leverage GitOps to deploy the resources. Otherwise, you will need to create the custom resources manually in each cluster.

This example deployment walks you through first deploying a ZooKeeper cluster and then a Kafka cluster in a project namespace. The result of this procedure is a running ZooKeeper cluster and Kafka cluster ready for use in your project's namespace.

1. Set the `PROJECT_NAMESPACE` environment variable to the name of your project's namespace:

   ```
   export PROJECT_NAMESPACE=<project namespace>
   ```

2. Create a ZooKeeper Cluster custom resource in your project's namespace:

438 https://github.com/banzaicloud/koperator/tree/master/config/samples

```
kubectl apply -f - <<EOF
apiVersion: zookeeper.pravega.io/v1beta1
kind: ZookeeperCluster
metadata:
  name: zookeeper
  namespace: ${PROJECT_NAMESPACE}
spec:
  replicas: 1
EOF
```

3. Check the status of your ZooKeeper cluster using `kubectl`:

```
kubectl -n ${PROJECT_NAMESPACE} get zookeeperclusters
```

4. Download the sample Kafka Cluster[439] file.
5. Update the following attribute `zkAddresses`, replacing `zookeeper-client.zookeeper:2181` with `zookeeper-client.<project namespace>:2181`. You can use `sed` to update the file:
   - MacOS sed

```
# If you are using sed that comes installed on macOS
sed -i '' "s/zookeeper-client.zookeeper:2181/zookeeper-client.$
{PROJECT_NAMESPACE}:2181/g" simplekafkacluster.yaml
```

   - GNU sed

```
# If you are using GNU sed
sed -i "s/zookeeper-client.zookeeper:2181/zookeeper-client.$
{PROJECT_NAMESPACE}:2181/g" simplekafkacluster.yaml
```

6. Verify the file contains the following line:

```
...
zkAddresses:
    - “zookeeper-client.<project namespace>:2181”
...
```

7. Apply the `KafkaCluster` to your project's namespace:

```
kubectl apply -n ${PROJECT_NAMESPACE} -f simplekafkacluster.yaml
```

8. Check the status of your Kafka cluster using `kubectl`:

439 https://raw.githubusercontent.com/banzaicloud/koperator/v0.23.0/config/samples/simplekafkacluster.yaml

```
kubectl -n ${PROJECT_NAMESPACE} get kafkaclusters
```

With both the ZooKeeper cluster and Kafka cluster running in your project's namespace, refer to the Kafka Operator documentation[440] for information on how to test and verify they are working as expected in. When performing those steps, ensure you substitute: `zookeeper-client.<project namespace>:2181` anywhere that the zookeeper client address is mentioned.

Delete Kafka Custom Resources

Follow these steps to delete the Kafka custom resources.

1. View all Kafka resources in the cluster:

   ```
   kubectl get kafkaclusters -A
   kubectl get kafkausers -A
   Kubectl get kafkatopics -A
   ```

2. Delete a `KafkaCluster` example:

   ```
   kubectl -n ${PROJECT_NAMESPACE} delete kafkacluster <name of KafkaCluster>
   ```

Resources

- Kafka Operator Documentation[441]
- Kafka Operator GitHub Repository[442]
- Sample Kafka Operator Custom Resources[443]
- Apache Kafka Documentation[444]

6.5.5.2.2.2 Zookeeper in a Project

Deploying ZooKeeper in a project

Get started

To get started with creating ZooKeeper clusters in your project namespace, you first need to deploy the ZooKeeper operator (see page 592) in the workspace where the project exists.

440 https://banzaicloud.com/docs/supertubes/kafka-operator/test/
441 https://banzaicloud.com/docs/supertubes/kafka-operator/
442 https://github.com/banzaicloud/koperator
443 https://github.com/banzaicloud/koperator/tree/master/config/samples
444 https://kafka.apache.org/documentation/

After you deploy the ZooKeeper operator, you can create ZooKeeper Clusters by applying a `ZookeeperCluster` custom resource **on each attached cluster** in a project's namespace.

A Helm chart[445] exists in the ZooKeeper operator repository that can assist with deploying ZooKeeper clusters.

Example Deployment

If you need to manage these custom resources across all clusters in a project, it is recommended you use Project Deployments (see page 636) which enables you to leverage GitOps to deploy the resources. Otherwise, you will need to create the resources manually in each cluster.

Follow these steps to deploy a ZooKeeper cluster in a project namespace. This procedure results in a running ZooKeeper cluster, ready for use in your project's namespace.

1. Set the `PROJECT_NAMESPACE` environment variable to the name of your project's namespace:

   ```
   export PROJECT_NAMESPACE=<project namespace>
   ```

2. Create a ZooKeeper Cluster custom resource in your project namespace

   ```
   kubectl apply -f - <<EOF
   apiVersion: zookeeper.pravega.io/v1beta1
   kind: ZookeeperCluster
   metadata:
     name: zookeeper
     namespace: ${PROJECT_NAMESPACE}
   spec:
     replicas: 1
   EOF
   ```

3. Check the status of your ZooKeeper cluster using `kubectl`:

   ```
   kubectl get zookeeperclusters -n ${PROJECT_NAMESPACE}
   ```

445 https://github.com/pravega/zookeeper-operator/tree/master/charts/zookeeper

```
NAME        REPLICAS   READY REPLICAS    VERSION   DESIRED VERSION   INTERNAL
ENDPOINT    EXTERNAL ENDPOINT   AGE
zookeeper   1          1                0.2.14    0.2.14            10.100.200
.18:2181   N/A                94s
```

Delete ZooKeeper clusters

Follow these steps to delete the Zookeeper clusters.

1. View `ZookeeperClusters` in all namespaces:

```
kubectl get zookeeperclusters -A
```

2. Delete a specific `ZookeeperCluster`:

```
kubectl -n ${PROJECT_NAMESPACE} delete zookeepercluster <name of
zookeepercluster>
```

Resources

- ZooKeeper Operator Documentation[446]
- ZooKeeper Cluster Helm Chart[447]
- ZooKeeper Documentation[448]

6.5.5.2.2.3 Spark in a Project

Deploying Spark in a project

To run your Spark workloads with Spark Operator, apply the Spark Operator specific custom resources. The Spark Operator works with the following kinds of custom resources:

- `SparkApplication`
- `ScheduledSparkApplication`

See Spark Operator API documentation[449] for more details.

446 https://github.com/pravega/zookeeper-operator
447 https://github.com/pravega/zookeeper-operator/tree/master/charts/zookeeper
448 https://zookeeper.apache.org/documentation
449 https://github.com/mesosphere/spark-on-k8s-operator/blob/d2iq-master/docs/api-docs.md

If you need to manage these custom resources and RBAC resources across all clusters in a project, it is recommended you use Project Deployments (see page 636) which enables you to leverage GitOps to deploy the resources. Otherwise, you will need to create the resources manually in each cluster.

Prerequisites

Follow these steps:

1. Deploy your Spark Operator. See the Spark Operator (see page 590) documentation for more information.
2. Ensure the necessary RBAC resources referenced in your custom resources exist, otherwise the custom resources can fail. See the Spark Operator documentation[450] for details.
    - This is an example of commands for you to create the RBAC resources needed in your project namespace:

```
export PROJECT_NAMESPACE=<project namespace>

kubectl apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: spark-service-account
  namespace: ${PROJECT_NAMESPACE}
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: ${PROJECT_NAMESPACE}
  name: spark-role
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["*"]
- apiGroups: [""]
  resources: ["services"]
  verbs: ["*"]
- apiGroups: [""]
  resources: ["configmaps"]
  verbs: ["*"]
- apiGroups: [""]
```

450 https://github.com/mesosphere/spark-on-k8s-operator/blob/d2iq-master/docs/quick-start-guide.md#about-the-spark-job-namespace

```
  resources: ["persistentvolumeclaims"]
  verbs: ["*"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: spark-role-binding
  namespace: ${PROJECT_NAMESPACE}
subjects:
- kind: ServiceAccount
  name: spark-service-account
  namespace: ${PROJECT_NAMESPACE}
roleRef:
  kind: Role
  name: spark-role
  apiGroup: rbac.authorization.k8s.io
EOF
```

Deploy a Simple SparkApplication

Follow these steps:

1. Create your Project (see page 608) if you don't already have one.
2. Set the `PROJECT_NAMESPACE` environment variable to the name of your project's namespace:

   ```
   export PROJECT_NAMESPACE=<project namespace>
   ```

3. Set the SPARK_SERVICE_ACCOUNT environment variable to one of the following:
   a. `${PROJECT_NAMESPACE}`, if you skipped the step in Prerequisites (see page 624) to create RBAC resources.

      ```
      # This service account is automatically created when you create a
      project and has access to everything in the project namespace.
      export SPARK_SERVICE_ACCOUNT=${PROJECT_NAMESPACE}
      ```

   b. Or set to `spark-service-account`

      ```
      export SPARK_SERVICE_ACCOUNT=spark-service-account
      ```

4. Apply the SparkApplication custom resource in your project namespace

   ```
   kubectl apply -f - <<EOF
   apiVersion: "sparkoperator.k8s.io/v1beta2"
   ```

```
kind: SparkApplication
metadata:
  name: pyspark-pi
  namespace: ${PROJECT_NAMESPACE}
spec:
  type: Python
  pythonVersion: "3"
  mode: cluster
  image: "gcr.io/spark-operator/spark-py:v3.1.1"
  imagePullPolicy: Always
  mainApplicationFile: local:///opt/spark/examples/src/main/python/pi.py
  sparkVersion: "3.1.1"
  restartPolicy:
    type: OnFailure
    onFailureRetries: 3
    onFailureRetryInterval: 10
    onSubmissionFailureRetries: 5
    onSubmissionFailureRetryInterval: 20
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.1.1
    serviceAccount: ${SPARK_SERVICE_ACCOUNT}
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.1.1
EOF
```

Clean up

Follow these steps:

1. View `SparkApplications` in all namespaces:

   ```
   kubectl get sparkapp -A
   ```

2. Deleting a specific `SparkApplication`:

   ```
   kubectl -n ${PROJECT_NAMESPACE} delete sparkapp <name of sparkapplication>
   ```

6.5.5.2.3 Use Custom Resources with Workspace Catalog Applications

Enterprise Gov Advanced

Some workspace catalog applications provision some `CustomResourceDefinition`, which allow you to deploy Custom Resources. Refer to your workspace catalog application's documentation for instructions.

6.5.5.2.4 Custom Project Applications

Enterprise Gov Advanced

Custom applications are third-party applications you have added to the Kommander Catalog.

Custom applications are any third-party applications that are not provided in the DKP Application Catalog. Custom applications can leverage applications from the DKP Catalog or be fully-customized. There is no expectation of support by D2iQ for a Custom application. Custom applications can be deployed on Konvoy clusters or on any D2iQ supported 3rd party Kubernetes distribution.

- Projects - Create a Git Repository (see page 627)
- Projects - Git repository structure (see page 629)
- Projects - Application Metadata (see page 631)
- Enable a Custom Application from the Project Catalog (see page 633)

6.5.5.2.4.1 Projects - Create a Git Repository

Enterprise Gov Advanced

Create a Git Repository in the Project namespace

Use the CLI to create the GitRepository resource and add a new repository to your Project.

1. Refer to air-gapped setup instructions (see page 1264), if you are running in air-gapped environment.
2. Set the `PROJECT_NAMESPACE` environment variable to the name of your project's namespace:

   ```
   export PROJECT_NAMESPACE=<project_namespace>
   ```

3. Adapt the URL of your Git repository.

```
kubectl apply -f - <<EOF
apiVersion: source.toolkit.fluxcd.io/v1beta1
kind: GitRepository
metadata:
  name: example-repo
  namespace: ${PROJECT_NAMESPACE}
spec:
  interval: 1m0s
  ref:
    branch: <your-target-branch-name> # e.g., main
  timeout: 20s
  url: https://github.com/<example-org>/<example-repo>
EOF
```

4. Ensure the status of the `GitRepository` signals a ready state:

```
kubectl get gitrepository example-repo -n ${PROJECT_NAMESPACE}
```

The repository commit also displays the ready state:

```
NAME         URL                                                     READY
STATUS                                                          AGE
example-repo https://github.com/example-org/example-repo             True
Fetched revision: master/6c54bd1722604bd03d25dcac7a31c44ff4e03c6a   11m
```

For more information on the GitRepository resource fields and how to make Flux aware of credentials required to access a private Git repository, see the Flux documentation[451].

Troubleshoot

To troubleshoot issues with adding the GitRepository, review the following logs:

```
kubectl -n kommander-flux logs -l app=source-controller
[...]
kubectl -n kommander-flux logs -l app=kustomize-controller
[...]
kubectl -n kommander-flux logs -l app=helm-controller
[...]
```

Related Information

- Flux[452]

451 https://fluxcd.io/docs/components/source/gitrepositories/
452 https://fluxcd.io/

- Flux docs[453]

6.5.5.2.4.2 Projects - Git repository structure

Enterprise Gov Advanced

Git repositories must be structured in a specific manner for defined applications to be processed by Kommander.

You must structure your git repository based on the following guidelines, for your applications to be processed properly by Kommander so that they can be deployed.

Git Repository Directory Structure

Use the following basic directory structure for your git repository:

```
├── helm-repositories
│   ├── <helm repository 1>
│   │   ├── kustomization.yaml
│   │   └── <helm repository name>.yaml
│   └── <helm repository 2>
│     ├── kustomization.yaml
│     └── <helm repository name>.yaml
└── services
    ├── <app name>
    │   ├── <app version1> # semantic version of the app helm chart. e.g., 1.2.3
    │   │   ├── defaults
    │   │   │   ├── cm.yaml
    │   │   │   └── kustomization.yaml
    │   │   ├── <app name>.yaml
    │   │   └── kustomization.yaml
    │   ├── <app version2> # another semantic version of the app helm chart. e.g.,
2.3.4
    │   │   ├── defaults
    │   │   │   ├── cm.yaml
    │   │   │   └── kustomization.yaml
    │   │   ├── <app name>.yaml
    │   │   └── kustomization.yaml
    │   └── metadata.yaml
    └── <another app name>
    ...
```

- Define applications in the `services/` directory.
- You can define multiple versions of an application, under different directories nested under the `services/<app name>/` directory.

453 https://fluxcd.io/docs

- Define application manifests, such as a HelmRelease[454], under each versioned directory `services/<app name>/<version>/` in `<app name>.yaml` which is listed in the `kustomization.yaml` Kubernetes Kustomization file. For more information, see the Kubernetes Kustomization docs[455].
- Define the default values ConfigMap for `HelmReleases` in the `services/<app name>/<version>/defaults` directory, accompanied by a `kustomization.yaml` Kubernetes Kustomization file pointing to the `ConfigMap` file.
- Define the `metadata.yaml` of each application under the `services/<app name>/` directory. For more information, see the Application Metadata docs (see page 602).

See the DKP Catalog repository[456] for an example of how to structure custom catalog Git repositories.

Helm Repositories

You must include the `HelmRepository` that is referenced in each `HelmRelease`'s `Chart` spec.

Each `services/<app name>/<version>/kustomization.yaml` must include the path of the YAML file that defines the `HelmRepository`. For example:

```
# services/<app name>/<version>/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - <app name>.yaml
  - ../../../helm-repositories/<helm repository 1>
```

For more information, see the flux documentation about HelmRepositories[457].

Substitution Variables

Some substitution variables[458] are provided.

- `${releaseName}`: For each App deployment, this variable is set to the `AppDeployment` name. Use this variable to prefix the names of any resources that are defined in the application directory in the Git repository so that multiple instances of the same application can be deployed. If you create resources without using the `releaseName` prefix (or suffix) in the name field, there can be conflicts if the same named resource is created in that same namespace.
- `${releaseNamespace}`: The namespace of the Project.
- `${workspaceNamespace}`: The namespace of the Workspace that the Project belongs to.

454 https://fluxcd.io/docs/components/helm/helmreleases/
455 https://kubectl.docs.kubernetes.io/references/kustomize/kustomization/
456 https://github.com/mesosphere/dkp-catalog-applications
457 https://fluxcd.io/docs/components/source/helmrepositories/
458 https://fluxcd.io/docs/components/kustomize/kustomization/#variable-substitution

Related Information

- Flux[459]
- Flux docs[460]
- Getting started with Flux guide[461]

6.5.5.2.4.3 Projects - Application Metadata

Enterprise Gov Advanced

To display more information about custom applications in the UI, define a metadata.yaml file for each application in the git repository.

You can define how custom applications display in the DKP UI by defining a `metadata.yaml` file for each application in your git repository. You must define this file at `services/<application>/metadata.yaml` for it to process correctly.

You can define the following fields:

Field	Default	Description
displayName	falls back to App ID	Display name of the application for the UI.
description	“”	Short description, should be a sentence or two, displayed in the UI on the application card.
category	general	1 or more categories for this application. Categories are used to group applications in the UI.
overview		Markdown overview used on the application detail page in the UI.
icon		Base64 encoded icon SVG file used for application logos in the UI.

459 https://fluxcd.io/
460 https://fluxcd.io/docs
461 https://fluxcd.io/docs/get-started/

Field	Default	Description
scope	workspace	List of scopes, can be workspace and/or project currently.

None of these fields are required for the application to display in the UI.

Here is an example `metadata.yaml` file:

```
displayName: Prometheus Monitoring Stack
description: Stack of applications that collect metrics and provides visualization
and alerting capabilities. Includes Prometheus, Prometheus Alertmanager and Grafana.
category:
  - monitoring
overview: >
  # Overview
  A stack of applications that collects metrics and provides visualization and
alerting capabilities. Includes Prometheus, Prometheus Alertmanager and Grafana.

  ## Dashboards
  By deploying the Prometheus Monitoring Stack, the following platform applications
and their respective dashboards are deployed. After deployment to clusters in a
workspace, the dashboards are available to access from a respective cluster's detail
page.

  ### Prometheus

  A software application for event monitoring and alerting. It records real-time
metrics in a time series database built using a HTTP pull model, with flexible and
real-time alerting.

  - [Prometheus Documentation - Overview](https://prometheus.io/docs/introduction/
overview/)

  ### Prometheus Alertmanager
  A Prometheus component that enables you to configure and manage alerts sent by the
Prometheus server and to route them to notification, paging, and automation systems.

  - [Prometheus Alertmanager Documentation - Overview](https://prometheus.io/docs/
alerting/latest/alertmanager/)

  ### Grafana
  A monitoring dashboard from Grafana that can be used to visualize metrics collected
by Prometheus.

  - [Grafana Documentation](https://grafana.com/docs/)
icon:
PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHZpZXdCb3g9IjAgMCAzMDAgMzAwIiBzd
HlsZT0iZW5hYmxlLWJhY2tncm91bmQ6bmV3IDAgMCAzMDAgMzAwIiB4bWw6c3BhY2U9InByZXNlcnZlIj48cG
F0aCBkPSJNMTUwIDUwQzk0LjggNTAgNTAgOTQuOCA1MCAxNTBzNDQuOCAxMDAgMTAwIDEwMCAxMDAtNDQuOCA
```

```
xMDAtMTAwUzIwNS4yIDUwIDE1MCA1MHptMCAxODcuMmMtMTUuNyAwLTI4LjUtMTAuNS0yOC41LTIzLjRoNTYu
OWMuMSAxMi45LTEyLjcgMjMuNC0yOC40IDIzLjR6bTQ3LTMxLjJoLTk0di0xN2g5NHYxN3ptLS4zLTI1LjloL
TkzLjRjLS4zLS40LS42LS43LS45LTEuMS05LjYtMTEuNy0xMS45LTE3LjgtMTQuMS0yNCAwLS4yIDExLjcgMi
40IDIwIDQuMyAwIDAgNC4zIDEgMTAuNSAyLjEtNi03LTkuNi0xNi05LjYtMjUuMSAwLTIwIDE1LjQtMzcuNiA
5LjgtNTEuNyA1LjQuNCAxMS4yIDExLjQgMTEuNiAyOC41IDUuNy03LjkgOC4xLTIyLjQgOC4xLTMxLjMgMC05
LjIgNi4xLTE5LjkgMTIuMS0yMC4yLTUuNCA4LjkgMS40IDE2LjUgNy40IDM1LjUgMi4zIDcuMSAyIDE5LjEgM
y43IDI2LjcuNi0xNS44IDMuMy0zOC44IDEzLjMtNDYuNy00LjQgMTAgLjcgMjIuNSA0LjEgMjguNSA1LjYgOS
43IDkgMTcuMSA5IDMxIDAgOS4zLTMuNCAxOC4xLTkuMyAyNSA2LjYtMS4yIDExLjItMi40IDExLjItMi40bDI
xLjQtNC4yYy4xIDAtMyAxMi44LTE0LjkgMjUuMXoiIHN0eWxlPSJmaWxsOiNmODQzMTEiLz48L3N2Zz4=
```


6.5.5.2.4.4 Enable a Custom Application from the Project Catalog

Enterprise Gov Advanced

Enable a Custom Application from the Project Catalog

After creating a GitRepository, you can either use the DKP UI or the CLI to enable your custom applications.

> From within a project, you can enable applications to deploy. Verify that an application has successfully deployed via the CLI.

Enable the Application using the UI

1. From the top menu bar, select your target workspace.
2. Select **Projects** from the sidebar menu.
3. Select your project from the list.
4. Select the **Applications** tab to browse the available applications from your configured repositories.
5. Select the three dot button from the bottom-right corner of the desired application tile, and then select **Enable**.
6. If available, select a version from the drop-down menu. This drop-down menu will only be visible if there is more than one version.
7. (Optional) If you want to override the default configuration values, copy your values content into the text editor under **Configure Service** or just upload your YAML file that contains the values:

   ```
   someField: someValue
   ```

8. Confirm the details are correct, and then select the **Enable** button.

For all applications, you must provide a display name and an ID which is automatically generated based on what you enter for the display name, unless or until you edit the ID directly. The ID must be compliant with Kubernetes DNS subdomain name validation rules[462].

Alternately, you can use the CLI (see page 0) to enable your custom applications in the section below.

Enable the Application using the CLI

Follow these steps:

1. Set the `PROJECT_NAMESPACE` environment variable to the name of the project's namespace:

   ```
   export PROJECT_NAMESPACE=<project_namespace>
   ```

2. Get the list of available applications to enable using the following command:

   ```
   kubectl get apps -n ${PROJECT_NAMESPACE}
   ```

3. Enable one of the supported applications from the list with an `AppDeployment` resource.

4. Within the `AppDeployment` resource. Provide the `appRef` and application version to specify which `App` is deployed:

   ```
   cat <<EOF | kubectl apply -f -
   apiVersion: apps.kommander.d2iq.io/v1alpha3
   kind: AppDeployment
   metadata:
     name: my-custom-app
     namespace: ${PROJECT_NAMESPACE}
   spec:
     appRef:
       name: custom-app-0.0.1
       kind: App
   EOF
   ```

The `appRef.name` must match the app `name` from the list of available applications.

Enable an Application with a Custom Configuration using the CLI

Follow these steps:

462 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#dns-subdomain-names

1. Provide the name of a `ConfigMap` in the `AppDeployment`, which provides custom configuration on top of the default configuration:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: my-custom-app
  namespace: ${PROJECT_NAMESPACE}
spec:
  appRef:
    name: custom-app-0.0.1
    kind: App
  configOverrides:
    name: my-custom-app-overrides
EOF
```

2. Create the `ConfigMap` with the name provided in the step above, with the custom configuration:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${PROJECT_NAMESPACE}
  name: my-custom-app-overrides
data:
  values.yaml: |
    someField: someValue
EOF
```

Kommander waits for the `ConfigMap` to be present before enabling the `AppDeployment` to the attached clusters in the Project.

Verify Applications

After completing the previous steps, your applications are enabled. Connect to the attached cluster and check the `HelmReleases` to verify the deployments:

```
kubectl get helmreleases -n ${PROJECT_NAMESPACE}
```

The output looks similar to this:

```
NAMESPACE              NAME           READY   STATUS
AGE
```

```
project-test-vjsfq      my-custom-app   True    Release reconciliation succeeded
7m3s
```

Upgrade Custom Applications

You must maintain your custom applications manually. When upgrading DKP, ensure you validate for compatibility issues any custom applications you run against the current version of Kubernetes. We recommend upgrading to the latest compatible application versions as soon as possible.

6.5.5.3 Project AppDeployments

Enterprise Gov Advanced

An `AppDeployment` is a Custom Resource created by DKP with the purpose of deploying applications (platform, DKP catalog and custom applications). For more information about these Custom Resources and how to customize them, refer to the AppDeployments (see page 545) documentation.

6.5.6 Project Deployments

Enterprise Gov Advanced

Use Project Deployments to manage GitOps based Continuous Deployments

You can configure Kommander Projects with GitOps-based Continuous Deployments for federation of your Applications to associated clusters of the project. This is backed by Flux, which enables software and applications to be continuously deployed (CD) using GitOps processes. GitOps enables the application to be deployed as per a manifest that is stored in a Git repository. This ensures that the application deployment can be automated, audited, and declaratively deployed to the infrastructure.

- What is GitOps? (see page 636)
- Continuous Delivery with GitOps (see page 637)
- Continuous Deployment (see page 647)
- Project Deployments Troubleshooting (see page 650)
- View Helm Releases (see page 651)

6.5.6.1 What is GitOps?

GitOps is a modern software deployment strategy. The configuration that describes how your application is deployed to a cluster are stored in a Git repository. The configuration is continuously synchronized from the Git repository to the cluster, ensuring that the specified state of the cluster always matches what is defined in the “GitOps” Git repository.

The benefits of using a GitOps deployment strategy are:

- Familiar, collaborative change and review process. Engineers are intimately familiar with Git-based workflows: branches, pull requests, code reviews, etc. GitOps leverages this experience to control the deployment of software and updates to catch issues early.
- Clear change log and audit trail. The Git commit log serves as an audit trail to answer the question: “who changed what, and when?” Having such information available, you can contact the right people when fixing or prioritizing a production incident to determine the why and correctly resolve the issue as quickly as possible. Additionally, Kommander’s CD component (Flux CD) maintains a separate audit trail in the form of Kubernetes Events, as changes to a Git repository don’t include exactly when those changes were deployed.
- Avoid configuration drift. The scope of manual changes made by operators expands over time. It soon becomes difficult to know which cluster configuration is critical and which is left over from temporary workarounds or live debugging. Over time, changing a project configuration or replicating a deployment to a new environment becomes a daunting task. GitOps supports simple, reproducible deployment to multiple different clusters by having a single source of truth for cluster and application configuration.

That said, there are some cases when live debugging is necessary in order to resolve an incident in the minimum amount of time. In such cases, pull-request-based workflow adds precious time to resolution for critical production outages. Kommander’s CD strategy supports this scenario by letting you disable the auto sync feature. After auto sync is disabled, Flux will stop synchronizing the cluster state from the GitOps git repository. This lets you use `kubectl`, `helm`, or whichever tool you need to resolve the issue.

6.5.6.2 Continuous Delivery with GitOps

DKP enables software and applications to be continuously delivered (CD) using GitOps processes. GitOps enables you to deploy an application according to a manifest that is stored in a Git repository. This ensures that the application deployment can be automated, audited, and declaratively deployed to the infrastructure.

This section contains step-by-step tutorials for performing some common deployment-related tasks using DKP. All tutorials begin with a Prerequisites section that contains links to any steps that need to be taken first. This means you can visit any tutorial to get started.

- Store secrets in GitOps repository using SealedSecrets (see page 637)
- Deploy a Sample App from DKP GitOps (see page 641)

6.5.6.2.1 Store secrets in GitOps repository using SealedSecrets

6.5.6.2.1.1 Securely managing secrets in a GitOps workflow using SealedSecrets

For security reasons, Kubernetes secrets are usually the only resource that cannot be managed with a GitOps workflow. Instead of managing secrets outside of GitOps and having to use a third-party tool like Vault, SealedSecrets provides a way to keep all the advantages of using a GitOps workflow while avoiding exposing secrets. SealedSecrets is composed of two main components:

- A CLI (Kubeseal) to encrypt secrets.
- A cluster-side controller to decrypt the sealed secrets into regular Kubernetes secrets. Only this controller can decrypt sealed secrets, not even the original author.

This tutorial describes how to install these two components, configure the controller, and add or remove sealed secrets.

6.5.6.2.1.2 Set up

These instructions are used as an example. For instructions on the latest release, see the release page[463]. For full documentation on SealedSecrets, see the GitHub repo[464].

6.5.6.2.1.3 Install Kubeseal CLI to Encrypt Your Secrets

- On MacOS

```
brew install kubeseal
```

- On Linux

```
wget https://github.com/bitnami-labs/sealed-secrets/releases/download/v0.18.1/
kubeseal-0.18.1-linux-amd64.tar.gz -O kubeseal
sudo install -m 755 kubeseal /usr/local/bin/kubeseal
```

6.5.6.2.1.4 Install the SealedSecrets Controller on Your Cluster

This controller will be able to decrypt SealedSecrets and create Kubernetes secrets.

1. Create the controller:

```
kubectl apply -f https://github.com/bitnami-labs/sealed-secrets/releases/
download/v0.18.1/controller.yaml
```

2. Fetch the certificate that you will use to encrypt your secrets into sealed secrets:

```
kubeseal --fetch-cert > mycert.pem
```

3. Commit `mycert.pem` to your git repo.

6.5.6.2.1.5 Add a Secret

Secrets can be securely added to Git using sealed secrets:

1. Export the project namespace that you are using for your GitOps repository

463 https://github.com/bitnami-labs/sealed-secrets/releases
464 https://github.com/bitnami-labs/sealed-secrets

```
export PROJECT_NAMESPACE=<your-project-with-gitops>
```

2. Create a Kubernetes secret and pipe it into kubeseal[465] using the certificate `mycert.pem` that you fetched from the controller in the setup:

```
echo '---' >> secrets.yaml
kubectl create secret -n ${PROJECT_NAMESPACE} generic mysecret --dry-run=client
-o yaml --from-literal=my-secret=value | \
    kubeseal --format yaml --cert mycert.pem >> secrets.yaml
```

3. Go to the end of `secrets.yaml` where you just added your new sealed secret. Remove any "creationTimestamp" fields from the YAML.
4. Apply the `secrets.yaml` file to your namespace. If you do not have permission, commit your changes to the repo and let FluxCD apply the changes for you.

```
kubectl apply -f secrets.yaml
```

5. The sealed secret controller will then decrypt the sealed secret and generate a Kubernetes secret from it. Your secret got successfully created by running:

```
kubectl get secret mysecret -n ${PROJECT_NAMESPACE} -o yaml
```

6. If your sealed secret got created successfully but did not generate the matching secret, look at the logs of the controller:

```
kubectl logs -l=name=sealed-secrets-controller -n kube-system
```

7. Commit `secrets.yaml` to your repo if you have not already done so in step 3.

6.5.6.2.1.6 Remove a Secret

1. Following the same example from above in "Adding a secret", now remove the manifest for `mysecret` in `secrets.yaml` and commit those changes to the repo.
2. Delete the SealedSecret in the cluster:

```
kubectl delete SealedSecret -n ${PROJECT_NAMESPACE} mysecret
```

3. Delete the secret itself:

465 https://github.com/bitnami-labs/sealed-secrets#usage

```
kubectl delete secret -n ${PROJECT_NAMESPACE} mysecret
```

6.5.6.2.1.7 Rotate the Controller's Sealing Key

For added security, it is a good practice to rotate the key the controller uses to decrypt sealed secrets. By default, the controller generates a new key every 30 days. When this happens, you need to update the certificate you use to create sealed secrets by fetching the latest one:

```
kubeseal --fetch-cert > mycert.pem
```

Do not forget to commit it back to the repo!

In a disaster case, let's say your cluster gets destroyed, you would lose all your sealing keys, so you would not be able to recreate all the secrets from the sealed secrets in your GitOps repo. For this reason, you might want to back up the sealing keys. To do this every time a new sealing key is generated, run:

```
kubectl get secret -n kube-system -l sealedsecrets.bitnami.com/sealed-secrets-key -o
yaml > sealing-key
```

Then store `sealing-key` with the others in a safe location such as OneLogin Notes or Vault. To restore from a backup after a disaster, recreate all of the sealing keys with `kubectl apply -f sealing-key1 sealing-key2 ...` before starting the controller. If the controller was already started, restart it: `kubectl delete pod -n kube-system -l name=sealed-secrets-controller`

To disable sealing key rotation For example, configure the controller's command in the pod template with `--key-renew-period=0`. See the following YAML file.

```
Pod Template:
  Labels:           name=sealed-secrets-controller
  Service Account:  sealed-secrets-controller
  Containers:
   sealed-secrets-controller:
    Image:      docker.io/bitnami/sealed-secrets-controller:v0.18.1
    Port:       8080/TCP
    Host Port:  0/TCP
    Command:    controller
      --key-renew-period=0
```

If required, edit the controller's manifest with:

```
kubectl edit deployment.apps/sealed-secrets-controller -n kube-system
```

6.5.6.2.2 Deploy a Sample App from DKP GitOps

Enterprise Gov Advanced

Use this procedure to deploy a sample `podinfo` application from DKP Enterprise GitOps.

6.5.6.2.2.1 Prerequisites

- Enterprise DKP installed (see page 117)
- Kommander installed (see page 1499)
- Github account and personal access token[466]
- Add cluster to Kommander (see page 673)
- Setup Workspace and Projects (see page 573)

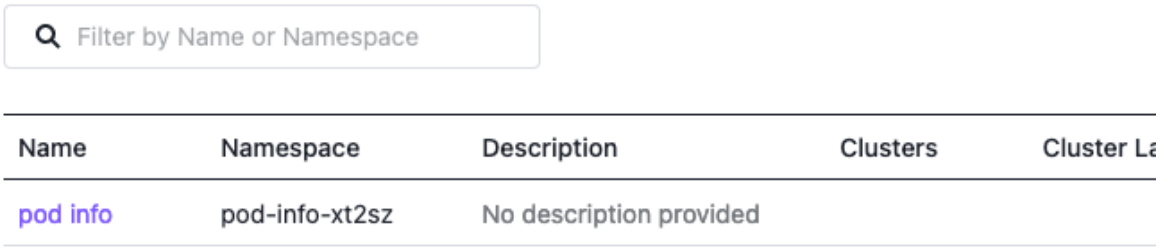
6.5.6.2.2.2 Deployment Steps

> This procedure was run on an AWS cluster with DKP 2.4.0 installed.

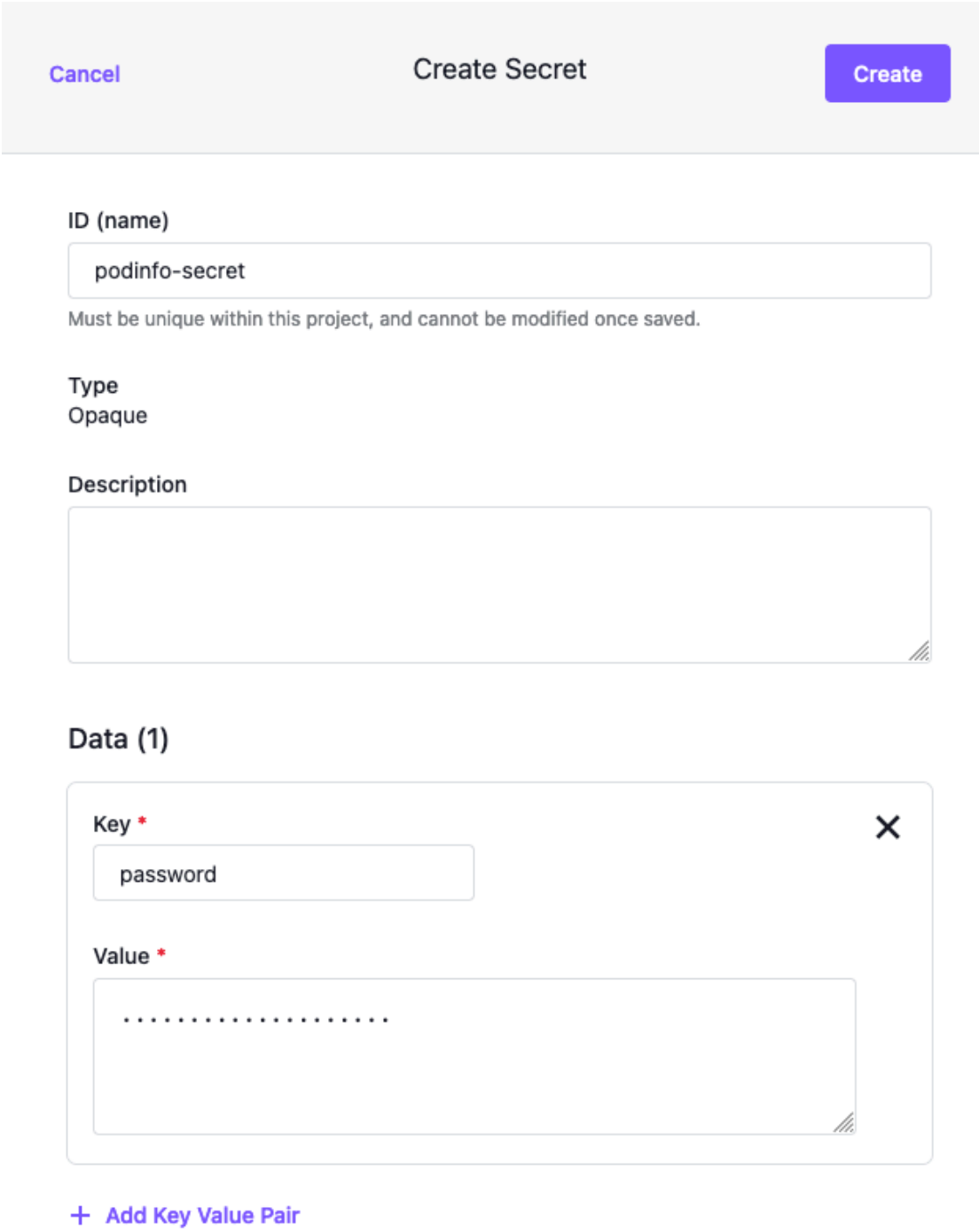
Follow these steps:

1. Ensure you are on the **Default Workspace** (or other workspace you have access to) so that you can create a project.
2. Create a project (see page 608).
   In the working example we name the project **pod-info**. When you create a namespace, Kommander appends five alphanumeric characters. You can opt to select a target cluster for this project from one of the available attached clusters, and then this **(pod-info-xxxxx)** is the namespace used for deployments under the project, for example:

466 https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token



3. [Optional] Create a secret in order to pull from the repository, for private repositories.
    a. Select the Secrets tab and set up your secret according to the Continuous Deployment documentation (see page 647).
    b. Add a **key** and **value** pair for the GitHub personal access token and then select **Create**.

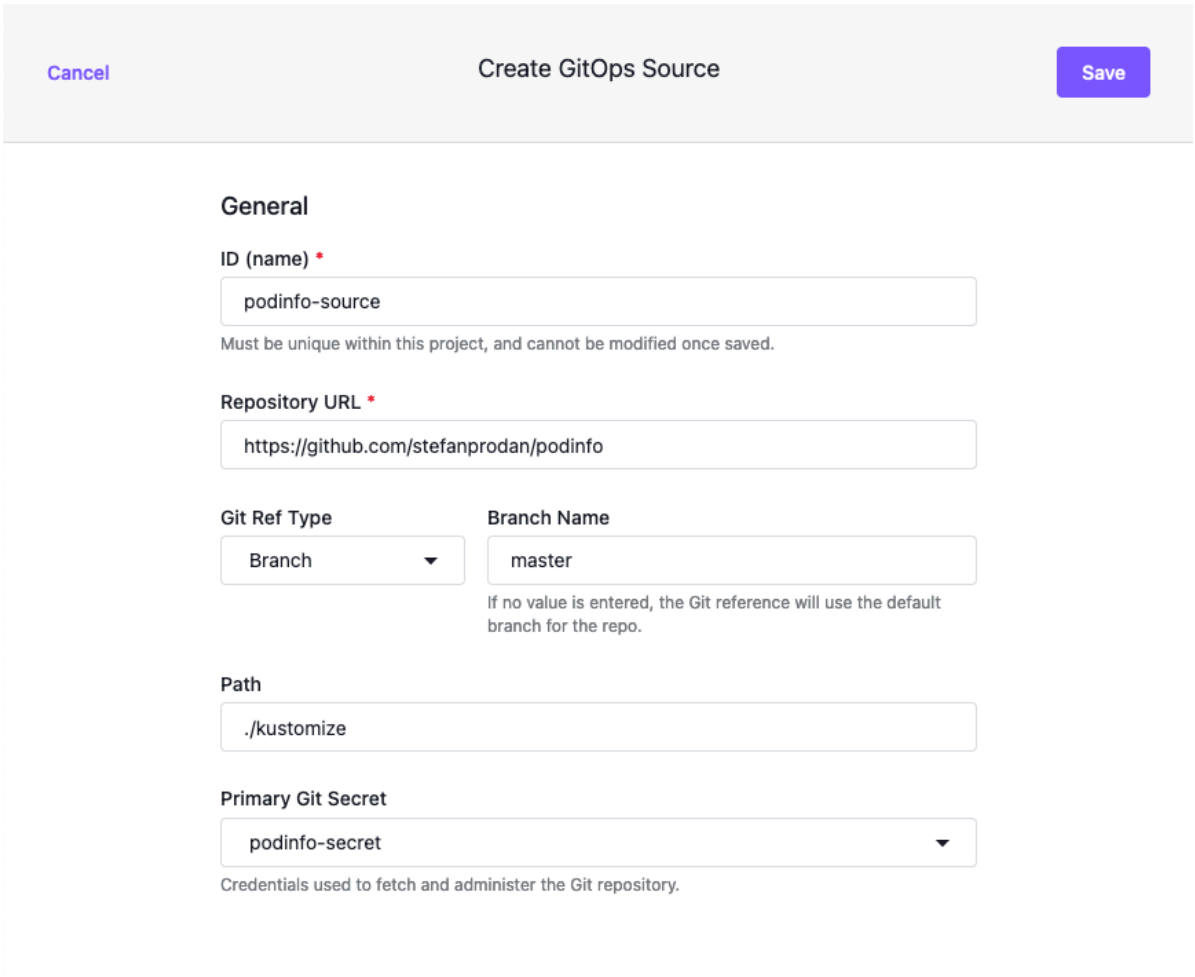


4. Verify that the secret `podinfo-secret` is created on the project namespace in the **managed or attached** cluster:

```
kubectl get secrets -n pod-info-xt2sz --kubeconfig=${CLUSTER_NAME}.conf
```

```
NAME                          TYPE                                  DATA   AGE
default-token-k685t           kubernetes.io/service-account-token   3      94m
pod-info-xt2sz-token-p9k5z    kubernetes.io/service-account-token   3      94m
podinfo-secret                Opaque                                1      1s
tls-root-ca                   Opaque                                1      93m
```

5. Select your project and then select the **CD** tab.
6. Add a GitOps Source, complete the required fields, and then **Save**.
   There are several configurable options such as selecting the **Git Ref Type** but in this example we use the master branch. The **Path** value should contain where the manifests are located. Additionally, the **Primary Git Secret** is the secret (**podinfo-secret**) that you created in the previous step, if you need to access private repositories. This can be disregarded for public repositories.



7. a. Verify the status of `gitrepository` creation with this command (on the attached or managed cluster), and if **READY** is marked as **True**:

```
kubectl get gitrepository -A --kubeconfig=${CLUSTER_NAME}.conf
```

```
NAMESPACE        NAME           URL
AGE    READY   STATUS
kommander-flux   management     https://gitea-http.kommander.svc/kommander/
kommander.git   134m   True    stored artifact for revision 'main/
4fbee486076778c85e14f3196e49b8766e50e6ce'
pod-info-xt2sz   podinfo-source https://github.com/stefanprodan/podinfo
116m   True    stored artifact for revision 'master/
b3b00fe35424a45d373bf4c7214178bc36fd7872'
```

8. Verify the `Kustomization` with this command below (on the attached or managed cluster), and if **READY** is marked as **True**:

```
kubectl get kustomizations -n pod-info-xt2sz --kubeconfig=${CLUSTER_NAME}.conf
```

```
NAME                  AGE    READY    STATUS
originalpodinfo       10m    True     Applied revision: master/
b3b00fe35424a45d373bf4c7214178bc36fd7872
podinfo-source        113m   True     Applied revision: master/
b3b00fe35424a45d373bf4c7214178bc36fd7872
project               116m   True     Applied revision: main/
4fbee486076778c85e14f3196e49b8766e50e6ce
project-tls-root-ca   117m   True     Applied revision: main/
4fbee486076778c85e14f3196e49b8766e50e6ce
```

Note the **port** so that you can use to verify if the app is deployed correctly (on the attached or managed cluster):

```
kubectl get deployments,services -n pod-info-xt2sz --kubeconfig=$
{CLUSTER_NAME}.conf
```

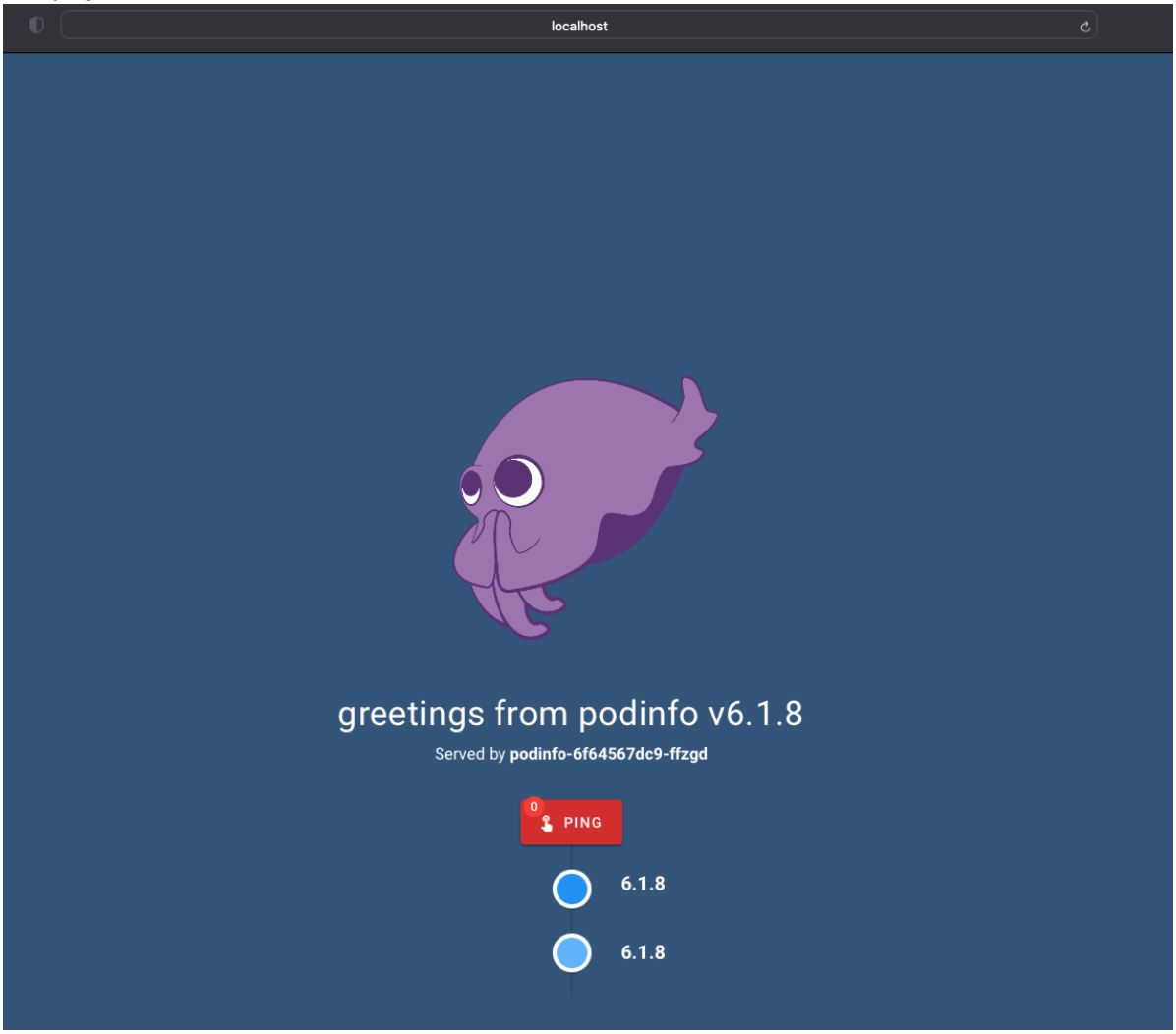
```
NAME                       READY    UP-TO-DATE    AVAILABLE    AGE
deployment.apps/podinfo    2/2      2             2            118m

NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
service/podinfo   ClusterIP   10.99.239.120   <none>        9898/TCP,9999/TCP
118m
```

9. Port forward the **podinfo** service (port **9898**) to verify (on the attached or managed cluster):

```
kubectl port-forward svc/podinfo -n pod-info-xt2sz 9898:9898 --kubeconfig=$
{CLUSTER_NAME}.conf
```

```
Forwarding from 127.0.0.1:9898 -> 9898
Forwarding from [::1]:9898 -> 9898
Handling connection for 9898
Handling connection for 9898
Handling connection for 9898
```

10. Open a browser and type in **localhost:9898**. A successful deployment of the podinfo app gives you this page:



6.5.6.3 Continuous Deployment

Enterprise Gov Advanced

After installing Kommander and configuring your project and its clusters (see page 669), navigate to the **Continuous Deployment (CD)** tab under your Project. Here you create a GitOps source which is a source code management (SCM) repository hosting the application definition. D2iQ recommends that you create a secret first then create a GitOps source accessed by the secret.

6.5.6.3.1 Prerequisites

You must have Kommander installed (see page 1227) to run this procedure.

6.5.6.3.2 Set up a Secret for Accessing GitOps

Create a secret that Kommander uses to deploy the contents of your GitOps repository:

> This dialog box creates a `types.kubefed.io/v1beta1, Kind=FederatedSecret` and this is not yet supported by DKP CLI. Use the GUI, as described above, to create a federated secret or create a `FederatedSecret` manifest and apply it to the project namespace. Learn more about FederatedSecrets (see page 659).

Kommander secrets (for CD) can be configured to support any of the following three authentication methods:

- HTTPS Authentication (described above)
- HTTPS self-signed certificates
- SSH Authentication

The following table describes the fields required for each authentication method:

HTTP Auth	HTTPS Auth (Self-signed)	SSH Auth
username	username	identity
password	password	identity.pub
	caFile	known_hosts

If you are using GitOps by using a GitHub repo as your source, you can create your secret with a personal access token[467]. Then, in the DKP UI, in your project, create a Secret, with a key:value pair of `password` : `<your-token-created-on-github>` . If you are using a GitHub personal access token, you do not need to have a key:value pair of `username` : `<your-github-username>` .

If you are using a secret with your GitHub username and your password, you will need one secret created in the DKP UI, with key:value pairs of `username` : `<your-github-username>` and `password` : `<your-github-password>` . **Note:** if you have multi-factor authentication turned on in your GitHub account, this will not work.

Using a token without a username is valid for GitHub, but other providers (such as GitLab) require both username and tokens.

If you are using a public GitHub repository, you do not need to use a secret.

6.5.6.3.3 Create GitOps Source

After the secret is created, you can view it in the `Secrets` tab. Configure the GitOps source accessed by the secret.

If using an SSH secret, the SCM repo URL needs to be an SSH address. It does not support SCP syntax. The URL format is `ssh://user@host:port/org/repository` .

It takes a few moments for the GitOps Source to be reconciled and the manifests from the SCM repository at the given path to be federated to attached clusters. After the sync is complete, manifests from GitOps source are created in attached clusters.

After a GitOps Source is created, there are various commands that can be executed from the CLI to check the various stages of syncing the manifests.

On the management cluster, check your `GitopsRepository` to ensure that the `CD` manifests have been created successfully.

```
kubectl describe gitopsrepositories.dispatch.d2iq.io -n<PROJECT_NAMESPACE> gitopsdemo
```

467 https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token

```
Name:         gitopsdemo
Namespace:    <PROJECT_NAMESPACE>
...
Events:
  Type    Reason               Age    From                         Message
  ----    ------               ----   ----                         -------
  Normal  ManifestSyncSuccess  1m7s  GitopsRepositoryController  manifests synced to
bootstrap repo
...
```

On the attached cluster, check for your `Kustomization` and `GitRepository` resources. The `status` field reflects the syncing of manifests:

```
kubectl get kustomizations.kustomize.toolkit.fluxcd.io -n<PROJECT_NAMESPACE>
<GITOPS_SOURCE_NAME> -oyaml
```

```
...
status:
  conditions:
  - reason: ReconciliationSucceeded
    status: "True"
    type: Ready
    ...
...
```

Similarly, with `GitRepository` resource:

```
kubectl get gitrepository.source.toolkit.fluxcd.io -n<PROJECT_NAMESPACE>
<GITOPS_SOURCE_NAME> -oyaml
```

```
...
status:
  conditions:
  - reason: GitOperationSucceed
    status: "True"
    type: Ready
    ...
...
```

If there are errors creating the manifests, those events are populated in the status field of the `GitopsRepository` resource on the management cluster, the `GitRepository` and `Kustomization` resources on the attached cluster(s), or both.

6.5.6.3.4 Suspend GitOps Source

There may be times when you need to suspend the auto-sync between the GitOps repository and the associated clusters. This *live debugging* may be necessary to resolve an incident in the minimum amount of time without the overhead of pull request based workflows.

To Suspend the GitOps Source from the DKP UI:

1. From the top menu bar, select your target workspace.
2. Select **Projects** from the sidebar menu.
3. Select your project from the list.
4. Select the **Continuous Deployment (CD)** tab.
5. Select the three dot button to the right of the desired GitOps Source.
6. Select **Suspend** to manually suspend the GitOps reconciliation.

This lets you use `kubectl`, `helm`, or another tool to resolve the issue. After the issue is resolved select **Resume** to sync the updated contents of the GitOps source to the associated clusters.

Similar to **Suspend/Resume**, you can use the **Delete** action to remove the GitOps source. Removing the GitOps source results in removal of all the manifests applied from the GitOps source.

You can have more than one GitOps Source in your Project to deploy manifests from various sources.

Kommander deployments are backed by FluxCD. Please refer to Flux Source Controller[468] and Kustomize controller[469] docs for advanced configuration and more examples.

6.5.6.4 Project Deployments Troubleshooting

Enterprise Gov Advanced

- Events related to federation are stored in respective `FederatedGitRepository`, `FederatedKustomization`, or both resources.
- View the events and logs for `deployments/Kommander-repository-controller` in Kommander namespace, if there are any unexpected errors.
- Enabling the Kommander repository controller for your project namespace causes a number of related Flux controller components[470] to deploy into the namespace. These are necessary for the proper operation of the repository controller and should not be removed.
- Ensure your GitOps repository does not contain any manifests that are cluster-scoped - for example, `Namespace`, `ClusterRole`, `ClusterRoleBinding`, etc. All of the manifests must be namespace-scoped.

468 https://fluxcd.io/docs/components/source/
469 https://fluxcd.io/docs/components/kustomize/
470 https://toolkit.fluxcd.io/components/

- Ensure your GitOps repository does not contain any `HelmRelease` and `Kustomization` resources that are targeting a different namespace than the project namespace.

6.5.6.5 View Helm Releases

Enterprise Gov Advanced

View Helm Releases for Continuous Deployments

In addition to viewing the current GitOps Sources, you can also view the current Helm Releases that have been deployed.

1. From the top menu bar, select your target workspace.
2. Select **Projects** from the sidebar menu.
3. Select your project from the list.
4. Select the **Continuous Deployment (CD)** tab.
5. Select the **Helm Releases** button.

All of the current Helm Release charts are displayed with their Chart Version and the names of the clusters. In this example *daily* is the name of the current cluster being displayed.

6.5.7 Project Role Bindings

Enterprise Gov Advanced

Project Role Bindings grant access to a specified Project Role for a specified group of people

6.5.7.1 Configure Project Role Bindings - UI Method

Before you can create a Project Role Binding, ensure you have created a Group. A Kommander Group can contain one or several Identity Provider users or groups.

You can assign a role to this Kommander Group:

1. From the **Projects** page, select your project.
2. Select the **Role Bindings** tab, then select **Add Roles** next to the group you want.
3. Select the **Role**, or Roles, you want from the drop-down menu, and then select **Save**.

6.5.7.2 Configure Project Role Bindings - CLI Method

A Project role binding can also be created using kubectl:

```
cat << EOF | kubectl create -f -
apiVersion: workspaces.kommander.mesosphere.io/v1alpha1
kind: VirtualGroupProjectRoleBinding
metadata:
  generateName: projectpolicy-
  namespace: ${projectns}
spec:
  projectRoleRef:
    name: ${projectrole}
  virtualGroupRef:
    name: ${virtualgroup}
EOF
```

6.5.7.3 Configure Project Role Bindings to Bind to WorkspaceRoles - CLI Method

You can also create a Project role binding to bind to a WorkspaceRole in certain instances. To list the WorkspaceRoles that you can bind to a Project, run the following command:

```
kubectl get workspaceroles -n ${workspacens} -o=jsonpath="{.items[?
(@.metadata.annotations.workspace\.kommander\.d2iq\.io\/project-default-workspace-
role-for==\"${projectns}\")].metadata.name}"
```

You can bind to any of the above WorkspaceRoles by setting `spec.workspaceRoleRef` in the project role binding:

```
cat << EOF | kubectl create -f -
apiVersion: workspaces.kommander.mesosphere.io/v1alpha1
kind: VirtualGroupProjectRoleBinding
metadata:
  generateName: projectpolicy-
  namespace: ${projectns}
spec:
  workspaceRoleRef:
    name: ${workspacerole}
  virtualGroupRef:
    name: ${virtualgroup}
EOF
```

Note that you must specify either workspaceRoleRef or projectRoleRef to be validated by the admission webhook. Specifying both values is not valid and will cause an error.

Ensure the `projectns`, `workspacens`, `projectrole` (or `workspacerole`) and the `virtualgroup` variables are set before executing the command.

When a Project Role Binding is created, Kommander creates a Kubernetes `FederatedRoleBinding` on the Kubernetes cluster where Kommander is running. You can view this by first finding the name of the project role binding that you created: `kubectl -n ${projectns} get federatedrolebindings.types.kubefed.io`.

Then, view the details like in this example:

```
kubectl -n ${projectns} get federatedrolebindings.types.kubefed.io projectpolicy-
gtct4-rdkwq -o yaml
```

Output:

```
apiVersion: types.kubefed.io/v1beta1
kind: FederatedRoleBinding
metadata:
  creationTimestamp: "2020-06-04T16:19:27Z"
  finalizers:
  - kubefed.io/sync-controller
  generation: 1
  name: projectpolicy-gtct4-rdkwq
  namespace: project1-5ljs9-lhvjl
  ownerReferences:
  - apiVersion: workspaces.kommander.mesosphere.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: VirtualGroupProjectRoleBinding
    name: projectpolicy-gtct4
    uid: 19614de2-4593-433e-82fa-96dc9470e07a
  resourceVersion: "196270"
  selfLink: /apis/types.kubefed.io/v1beta1/namespaces/project1-5ljs9-lhvjl/
federatedrolebindings/projectpolicy-gtct4-rdkwq
  uid: beaffc29-edec-4258-9813-3a17ba27a2a6
spec:
  placement:
    clusterSelector: {}
  template:
    roleRef:
      apiGroup: rbac.authorization.k8s.io
      kind: Role
      name: admin-dbfpj-l6s9g
    subjects:
    - apiGroup: rbac.authorization.k8s.io
      kind: User
      name: user1@d2iq.lab
status:
  clusters:
```

```
  - name: konvoy-5nr5h
  conditions:
  - lastTransitionTime: "2020-06-04T16:19:27Z"
    lastUpdateTime: "2020-06-04T16:19:27Z"
    status: "True"
    type: Propagation
  observedGeneration: 1
```

Then, if you run the following command on a Kubernetes cluster associated with the Project, you'll see a Kubernetes RoleBinding Object, in the corresponding namespace:

```
kubectl -n ${projectns} get rolebinding projectpolicy-gtct4-rdkwq -o yaml
```

Output:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  creationTimestamp: "2020-06-04T16:19:27Z"
  labels:
    kubefed.io/managed: "true"
  name: projectpolicy-gtct4-rdkwq
  namespace: project1-5ljs9-lhvjl
  resourceVersion: "125392"
  selfLink: /apis/rbac.authorization.k8s.io/v1/namespaces/project1-5ljs9-lhvjl/
rolebindings/projectpolicy-gtct4-rdkwq
  uid: 2938398d-437b-4f3a-9cb9-c92e50139196
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: admin-dbfpj-l6s9g
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: user1@d2iq.lab
```

6.5.7.4 Role Binding with VirtualGroup

In DKP, `VirtualGroup` is a list of subjects that can be assigned to several different kinds of roles, including:

- `ClusterRole` for cluster-scoped objects
- `WorkspaceRole` for workspace-scoped objects
- `ProjectRole` for project-scoped objects

In order to define which `VirtualGroup` (s) is assigned to one of these roles, administrators can create corresponding role bindings such as `VirtualGroupClusterRoleBinding`, `VirtualGroupWorkspaceRoleBinding`, and `VirtualGroupProjectRoleBinding`.

For more information about configuring `VirtualGroup`, please refer to the DKP API Documentation (see page 1417).

Note that for `WorkspaceRole` and `ProjectRole`, the referenced `VirtualGroup` and corresponding role and role binding objects need to be in the same namespace. If they are not in the same namespace, the role will not bind to the `VirtualGroup` since it is assumed that the rules set in the role apply to objects that live in that namespace. Whereas for `ClusterRole` which is cluster-scoped, the `VirtualGroupClusterRoleBinding` is also cluster-scoped, even though it references a namespace-scoped `VirtualGroup`.

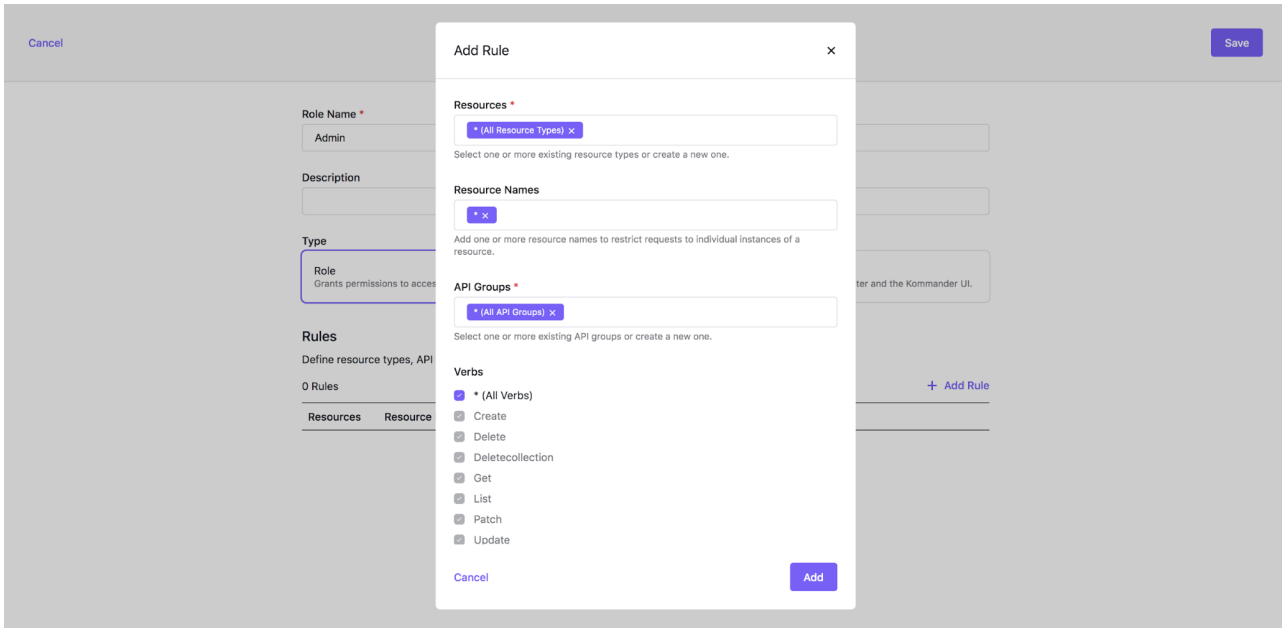
6.5.8 Project Roles

Enterprise Gov Advanced

Project Roles are used to define permissions at the namespace level.

6.5.8.1 Configure Project Role - UI Method

In the example below, a Project Role is created with a single Rule. This Project Role corresponds to an admin role.



You can create a Project Role with several Rules.

6.5.8.2 Configure Project Role - CLI Method

The same Project Role can also be created using kubectl:

```
cat << EOF | kubectl create -f -
apiVersion: workspaces.kommander.mesosphere.io/v1alpha1
kind: ProjectRole
metadata:
  annotations:
    kommander.mesosphere.io/display-name: Admin
  generateName: admin-
  namespace: ${projectns}
spec:
  rules:
  - apiGroups:
    - '*'
    resources:
    - '*'
    verbs:
    - '*'
EOF
```

Ensure the `projectns` variable is set before executing the command.

You can set it using the following command (for a Kommander Project called `project1`, and after setting the `workspacens` as explained in the previous section):

```
projectns=$(kubectl -n ${workspacens} get projects.workspaces.kommander.mesosphere.io
-o jsonpath='{.items[?
(@.metadata.generateName=="project1-")].status.namespaceRef.name}')
```

When a Project Role is created, Kommander creates a Kubernetes `FederatedRole` on the Kubernetes cluster where Kommander is running:

```
kubectl -n ${projectns} get federatedroles.types.kubefed.io admin-dbfpj-l6s9g -o yaml
apiVersion: types.kubefed.io/v1beta1
kind: FederatedRole
metadata:
  creationTimestamp: "2020-06-04T11:54:26Z"
  finalizers:
  - kubefed.io/sync-controller
  generation: 1
  name: admin-dbfpj-l6s9g
  namespace: project1-5ljs9-lhvjl
  ownerReferences:
  - apiVersion: workspaces.kommander.mesosphere.io/v1alpha1
```

```
    blockOwnerDeletion: true
    controller: true
    kind: ProjectRole
    name: admin-dbfpj
    uid: e5f3b2ca-16bf-474d-8305-7be04c034793
  resourceVersion: "75680"
  selfLink: /apis/types.kubefed.io/v1beta1/namespaces/project1-5ljs9-lhvjl/
federatedroles/admin-dbfpj-l6s9g
  uid: 1e5a3d98-b223-4605-bba1-16276a3eb47c
spec:
  placement:
    clusterSelector: {}
  template:
    rules:
    - apiGroups:
      - '*'
      resourceNames:
      - '*'
      resources:
      - '*'
      verbs:
      - '*'
status:
  clusters:
  - name: konvoy-5nr5h
  conditions:
  - lastTransitionTime: "2020-06-04T11:54:26Z"
    lastUpdateTime: "2020-06-04T11:54:26Z"
    status: "True"
    type: Propagation
  observedGeneration: 1
```

Then, if you run the following command on a Kubernetes cluster associated with the Project, you see a Kubernetes Role object in the corresponding namespace:

```
kubectl -n ${projectns} get role admin-dbfpj-l6s9g -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  creationTimestamp: "2020-06-04T11:54:26Z"
  labels:
    kubefed.io/managed: "true"
  name: admin-dbfpj-l6s9g
  namespace: project1-5ljs9-lhvjl
  resourceVersion: "29218"
  selfLink: /apis/rbac.authorization.k8s.io/v1/namespaces/project1-5ljs9-lhvjl/roles/
admin-dbfpj-l6s9g
  uid: f05b998c-4649-4e73-bbfe-c12bc4c86a3c
rules:
- apiGroups:
  - '*'
```

```
  resourceNames:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
```

6.5.9 Project ConfigMaps

Enterprise Gov Advanced

Use ConfigMaps to automate ConfigMaps aretion on your clusters

Project ConfigMaps can be created to make sure Kubernetes ConfigMaps are automatically created on all Kubernetes clusters associated with the Project, in the corresponding namespace.

As reference, a ConfigMap is a key-value pair to store some type of non-confidential data like “name=bob” or “state=CA”. For a full reference to the concept, consult the Kubernetes documentation on the topic of ConfigMaps[471].

6.5.9.1 Configuring Project ConfigMaps - UI Method

The below Project ConfigMap form can be navigated to by:

1. From the top menu bar, select your target workspace.
2. Select **Projects** from the sidebar menu.
3. Select your project from the list.
4. Select the **ConfigMaps** tab to browse the deployed ConfigMaps.
5. Select **+ Create ConfigMap** button.
6. Enter an ID, Description and Data for the ConfigMap, and select the **Create** button.

6.5.9.2 Configuring Project ConfigMaps - CLI Method

A Project ConfigMap is simply a Kubernetes FederatedConfigMap and can be created using kubectl with YAML:

```
cat << EOF | kubectl create -f -
apiVersion: types.kubefed.io/v1beta1
kind: FederatedConfigMap
metadata:
  generateName: cm1-
```

471 https://kubernetes.io/docs/concepts/configuration/configmap/

```
  namespace: ${projectns}
spec:
  placement:
    clusterSelector: {}
  template:
    data:
      key: value
EOF
```

Ensure the `projectns` variable is set before executing the command. This variable is the project namespace (the Kubernetes Namespace associated with the project) that was defined/created when the project itself was initially created.

```
projectns=$(kubectl -n ${workspacens} get projects.workspaces.kommander.mesosphere.io
-o jsonpath='{.items[?
(@.metadata.generateName=="project1-")].status.namespaceRef.name}')
```

Then, if you run the following command on a Kubernetes cluster associated with the Project, you'll see a Kubernetes ConfigMap Object, in the corresponding namespace:

```
kubectl -n ${projectns} get configmap cm1-8469c -o yaml
apiVersion: v1
data:
  key: value
kind: ConfigMap
metadata:
  creationTimestamp: "2020-06-04T16:37:10Z"
  labels:
    kubefed.io/managed: "true"
  name: cm1-8469c
  namespace: project1-5ljs9-lhvjl
  resourceVersion: "131844"
  selfLink: /api/v1/namespaces/project1-5ljs9-lhvjl/configmaps/cm1-8469c
  uid: d32acb98-3d57-421f-a677-016da5dab980
```

6.5.10 Project Secrets

Enterprise Gov Advanced

Project Secrets can be created to make sure a Kubernetes Secrets are automatically created on all the Kubernetes clusters associated with the Project, in the corresponding namespace.

6.5.10.1 Configure Project Secrets - UI Method

1. Select the workspace your project was created in from the workspace selection dropdown in the header.
2. In the sidebar menu, select **Projects**.
3. Select the project you want to configure from the table.
4. Select the **Secrets** tab, and then select the **Create Secret** button.
5. Complete the form and select the **Create** button.

6.5.10.2 Configure Project Secrets - CLI Method

A Project Secret is simply a Kubernetes FederatedConfigSecret and can also be created using kubectl:

```
cat << EOF | kubectl create -f -
apiVersion: types.kubefed.io/v1beta1
kind: FederatedSecret
metadata:
  generateName: secret1-
  namespace: ${projectns}
spec:
  placement:
    clusterSelector: {}
  template:
    data:
      key: dmFsdWU=
EOF
```

Ensure the `projectns` variable is set before executing the command.

```
projectns=$(kubectl -n ${workspacens} get projects.workspaces.kommander.mesosphere.io
-o jsonpath='{.items[?
(@.metadata.generateName=="project1-")].status.namespaceRef.name}')
```

The value of the key is base64 encoded.

If you run the following command on a Kubernetes cluster associated with the Project, you see a Kubernetes Secret Object, in the corresponding namespace:

```
kubectl -n ${projectns} get secret secret1-r9vk2 -o yaml
apiVersion: v1
data:
  key: dmFsdWU=
kind: Secret
metadata:
  creationTimestamp: "2020-06-04T16:51:59Z"
  labels:
    kubefed.io/managed: "true"
  name: secret1-r9vk2
  namespace: project1-5ljs9-lhvjl
  resourceVersion: "137215"
  selfLink: /api/v1/namespaces/project1-5ljs9-lhvjl/secrets/secret1-r9vk2
  uid: e5c6fc1d-93e7-47fe-ae1e-f418f8e35d72
type: Opaque
```

6.5.11 Project Quotas & Limit Ranges

Enterprise Gov Advanced

Project Quotas and Limit Ranges can be set up to limit the number of resources the Project team uses.

6.5.11.1 Creating Project Quotas & Limit Ranges- UI Method

Project Quotas and Limit Ranges can be set up to limit the number of resources the Project team uses. Quotas and Limit Ranges are applied to all project clusters.

1. Select the workspace your project was created in from the workspace selection dropdown in the header.
2. In the sidebar menu, select **Projects**.
3. Select the project you want to configure from the table.
4. Select the **Quotas & Limit Ranges** tab, and then select the **Edit** button.

   Kommander provides a set of default resources for which you can set Quotas. You can also define Quotas for custom resources. We recommend that you set Quotas for CPU and Memory. By using Limit Ranges, you can restrict the resource consumption of individual Pods, Containers, and Persistent Volume Claims in the project namespace. You can also constrain memory and CPU resources consumed by Pods and Containers, and storage resources consumed by Persistent Volume Claims.
5. To add a custom quota, scroll to the bottom of the form and select **Add Quota**.
6. When you are finished, select the **Save** button.

6.5.11.2 Create Project Quotas & Limit Ranges - CLI Method

All the Project Quotas are defined using a Kubernetes FederatedResourceQuota called `kommander` which you can also create/update using kubectl:

```
cat << EOF | kubectl apply -f -
apiVersion: types.kubefed.io/v1beta1
kind: FederatedResourceQuota
metadata:
  name: kommander
  namespace: ${projectns}
spec:
  placement:
    clusterSelector: {}
  template:
    spec:
      hard:
        limits.cpu: "10"
        limits.memory: 1024.000Mi
EOF
```

Ensure the `projectns` variable is set before executing the command.

```
projectns=$(kubectl -n ${workspacens} get projects.workspaces.kommander.mesosphere.io
-o jsonpath='{.items[?
(@.metadata.generateName=="project1-")].status.namespaceRef.name}')
```

Then, if you run the following command on a Kubernetes cluster associated with the Project, you'll see a Kubernetes Secret Object in the corresponding namespace:

```
kubectl -n ${projectns} get resourcequota kommander -o yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  creationTimestamp: "2020-06-05T08:04:37Z"
  labels:
    kubefed.io/managed: "true"
  name: kommander
  namespace: project1-5ljs9-lhvjl
  resourceVersion: "470822"
  selfLink: /api/v1/namespaces/project1-5ljs9-lhvjl/resourcequotas/kommander
  uid: 925b61b4-134b-4c45-915c-96a05b63d3c3
spec:
  hard:
    limits.cpu: "10"
    limits.memory: 1Gi
```

```
status:
  hard:
    limits.cpu: "10"
    limits.memory: 1Gi
  used:
    limits.cpu: "0"
    limits.memory: "0"
```

Similarly, Project Limit Ranges are defined using a FederatedLimitRange object with name `kommander` in the project namespace:

```
cat << EOF | kubectl apply -f -
apiVersion: types.kubefed.io/v1beta1
kind: FederatedLimitRange
metadata:
  name: kommander
  namespace: ${projectns}
spec:
  placement:
    clusterSelector: {}
  template:
    spec:
      limits:
      - type: "Pod"
        max:
          cpu: 500m
          memory: 50Gi
        min:
          cpu: 100m
          memory: 10Gi
      - type: "Container"
        max:
          cpu: 2
          memory: 100Mi
        min:
          cpu: 1
          memory: 10Mi
      - type: "PersistentVolumeClaim"
        max:
          storage: 3Gi
        min:
          storage: 1Gi
EOF
```

6.5.12 Project Network Policies

Enterprise Gov Advanced

Projects are created with a secure-by-default network policy and users needing more flexibility can edit or add more policies to tailor to their unique security needs.

Cluster networking is a critical and central part of Kubernetes that can also be quite challenging. All network communication within and between clusters depends on the presence of a Container Network Interface (CNI) plugin.

6.5.12.1 About Network Plugins

Network plugins ensure that Kubernetes networking requirements are met and surface features needed by network administrators, such as enforcing network policies. Common Network Plugins include Flannel, Calico, and Weave among many others. As an example, D2iQ® Konvoy® uses the Calico CNI plugin by default, but can support others.

Since pods are short-lived, the cluster needs a way to configure the network dynamically as pods are created and destroyed. Plugins provision and manage IP addresses to interfaces and let administrators manage IPs and their assignments to containers, in addition to connections to more than one host, when needed.

6.5.12.2 About Network Policies

By default, and to enable fluid communications within and between clusters, all traffic is authorized between nodes and pods. Most production environments require some kind of traffic flow control at the IP address or port level. An application-centric approach to this uses Network Policies. Pods are isolated by having a Network Policy that selects them, and the configuration of the policy limits the kind of traffic they can receive or send. Network policies do not conflict because they are additive. Pods selected by more than one policy are subject to the union of the policies' ingress and egress rules.

A Network Policy's rules define ingress and egress for network communications between pods and across namespaces. Successful traffic control using network policies is bi-directional. You have to configure both the egress policy on the source pod and the ingress policy on the destination pod to enable the traffic. If either end denies the traffic, it will not flow between the pods.

Since the Kubernetes default is to allow all traffic, it is a common practice to create a default "deny all traffic" rule, and then specifically open up some combination of the pods, ports, and applications as needed.

6.5.12.3 Creating Network Policies

You create network policies in three main parts:

- General information
- Ingress rules
- Egress rules

6.5.12.3.1 General information section

The fields in this part of the form allow you to create a name and description for this policy. Creating a detailed **Description** helps to keep policy functions understandable for additional use and maintenance.

This section also contains the **Pod Selector** fields for selecting pods using either Labels or Expressions. **Labels** added to pod declarations are a common means of identifying individual pods, or creating groups of

pods, in a Kubernetes cluster. **Expressions** are similar to Labels, but allow you to define parameters that identify a range of pods.

The **Policy Types** selections help to define the type of Network Policy you are creating:

- **Default** - automatically includes ingress, and egress is set only if the network policy defines egress rules.
- **Ingress** - this policy applies to ingress traffic for the selected pods, to namespaces using the options you define below, or both.
- **Egress** - this policy applies to egress traffic for the selected pods, to namespaces using the options you define below, or both.

If the Default policy type is too rigid or does not offer what you need, you can select the Ingress or Egress type, or both, and explicitly define the policy with the options that follow. For example, if you do not want this policy to apply to ingress traffic, you would select only Egress, and then define the policy.

To deny all ingress traffic, select the **Ingress** option here and then leave the ingress rules empty.

To deny all egress traffic, select the **Egress** option here and then leave the egress rules empty.

6.5.12.3.2 Ingress rules section

Ingress rules use a combination of **Port** / **Protocol** and **Source** to define the incoming traffic allowed to some or all of the pods in this namespace.

The options under **Sources: From** enable you to define a source either by using the pod selector or by defining an IP block. When using the pod selector method, you can define the namespace, the pods within that namespace, or both.

Namespaces - Selecting a namespace in an ingress rule source permits the pods selected by the pod selector, in your selected namespaces, to receive incoming traffic that meets the other defined criteria. If you have not selected any pods, the rule permits traffic from all pods in the selected namespaces.

Pods - This option selects specific Pods which should be allowed as ingress sources or egress destinations. If you have not selected any namespaces in the namespace selector, this option selects all matching pods in the project namespace. Otherwise, this option selects all matching pods in the selected namespaces.

There also are options to select all namespaces, all pods, or both.

When defining ingress rules using the IP Block method, you define a CIDR and exception conditions. CIDR stands for Classless Inter-Domain Routing and is an IP standard for creating unique network and device identifiers. When grouped together so that they share an initial sequence of bits in their binary representation, the range of addresses creates a CIDR block. The block identity is in an IPv4-like notation including a dotted-decimal address, followed by a slash, then a colon and a number from 0 through 32, for example, 127.0.26.33:31.

6.5.12.3.3 Egress rules section

Egress rules use a similar combination of options to define the outgoing traffic from pods, ranges of pods, or namespaces in a Kommander Project. Port, Protocol, and Destination options for egress rules define the outgoing traffic. You can define your egress rules under **Destination: To**. Ensure the egress policy on the source pods, and the ingress policy on the destination pods, permit traffic in order for the pods to be able to communicate over the network.

6.5.12.4 Network Policy Examples

IMPORTANT: Before you begin each example, ensure you're on the Network Policy page for your project

To navigate to your project's Network Policy page:

1. From the top menu bar, select your target workspace.
2. Select **Projects** from the sidebar menu.
3. Select your project from the list.
4. Select the **Network Policy** tab.

6.5.12.4.1 Ingress: Permit access to API service pods from all namespaces

Suppose you need to create a network policy to permit incoming traffic to API service pods in a specific Kommander Project's namespace from any other pod in any namespace that has the label, `service.corp/users-api-role: client`. For this example, API service pods are those pods created with the Label, `service.corp/users-api-role: api`.

You can limit the policy to just incoming traffic from select namespaces by adding an ingress rule with these characteristics:

- Use Port 8080 to receive incoming TCP traffic
- Refuse traffic from pods unless they are client pods that have a specific Label, such as `service.corp/users-api-role: client`. This example follows a common microservice architecture pattern, microservice-tier-role: access_mode

6.5.12.4.1.1 Configure the general information to access API service pods

1. Select the **+ Create Network Policy** button.
2. Type "microsvc-users-api-allow" in the **ID Name** field.
3. Type "Allow Users microservice clients to reach the APIs provided in this namespace" in the **Description** field.
4. Select **Add** under **Pod Selector** and then select **Match Label**.
5. Set the **Key** to "service.corp/users-api-role" and the **Value** to "API".

6.5.12.4.1.2 Create an Ingress rule to access API service pods

1. Leave **Policy Types** set to Default.
2. Scroll down to **Ingress Rules** and select **+ Add an Ingress Rule**.
3. Select **+ Add Port**, and set the **Port** to 8080 and the **Protocol** to TCP.

6.5.12.4.1.3 Add Sources to access API service pods

1. Select **+ Add Source** and mark the **Select All Namespaces** check box.
2. Select **+ Add Pod Selector**.
3. Select **Match Label**.
4. Set the **Key** value to “service.corp/users-api-role” and set the **Value** to “client”.
5. Scroll up and select **Save**.

6.5.12.4.2 Ingress: Limit pods that access a database to this namespace

Suppose that while deploying an application in a project, you want to protect its database pods by permitting ingress only from API service pods in the current namespace, and prevent ingress from pods in any other namespace.

You can limit the database pods to just the incoming traffic from the current namespaces by adding an ingress rule with these characteristics:

- Use Port 3306 to receive incoming TCP traffic for pods that have the label, `tier: database`
- Refuse traffic from pods unless they have the label, `tier: api`

6.5.12.4.2.1 Configure the general information to access a database

1. Select the **+ Create Network Policy** button.
2. Type “database-access-api-only” in the **ID name** field.
3. Type “Allow MySQL access only from API pods in this namespace” in the **Description** field.
4. Select **Pod Selector** then select **Match Label**.
5. Set the **Key** to “tier” and the **Value** to “database”.

6.5.12.4.2.2 Create an Ingress rule to access a database

1. Select Default for the **Policy Types**.
2. Scroll down to the **Ingress** section.
3. Select **+ Add Ingress Rule**, then select **+ Add Port**.
4. Set the **Port** to “3306” and leave the **Protocol** set to “TCP”.

6.5.12.4.2.3 Add Sources to access a database

1. Select **+ Add Source**.
2. Select **+ Add Pod Selector**.

3. Select **Match Label** and set the **Key** to “tier” and the **Value** to “API”.
4. Scroll up and select **Save**.

6.5.12.4.3 Ingress: Disable but don’t delete ingress rules

Suppose that you want to disable ingress rules temporarily for testing or triaging purposes.

First, you need to create a network policy with one or more ingress rules. You could follow one of the preceding procedures, for example. Then, you need to edit the policy to match the following example:

6.5.12.4.3.1 Edit your Network Policy

1. In the table row belonging to your network policy, click the context menu at the right of the row and select **Edit**.

6.5.12.4.3.2 Disable Ingress rules

1. Update the **Policy Types** so that only **Egress** is selected. If you don’t want to deny *all* egress traffic, ensure that you add an egress rule that suits your preferred level of access. You can add an empty rule to allow all egress traffic.
2. Scroll up and select **Save**.

6.5.12.4.4 Egress: Deny all egress traffic from restricted pods

Suppose that you need to deny all egress traffic from a group of restricted pods. This is a simple egress rule and you can create it following this example and steps:

6.5.12.4.4.1 Configure the General Information to deny Egress

1. Select **+ Add Network Policy**.
2. Type “deny-restricted-egress” in the **ID name** field.
3. Type “Deny egress traffic from restricted pods” in the **Description** field.
4. Select **Pod selector** then select **Match Label.**
5. Set the **Key** to “access” and the **Value** to “restricted”.

6.5.12.4.4.2 Deny Egress traffic

1. Update the **Policy Types** so that only **Egress** is selected. Do not add any egress rules.
2. Scroll up and select **Save**.

6.6 Manage Clusters

View clusters created with Kommander or any connected Kubernetes cluster

Kommander allows you to monitor and manage very large numbers of clusters. Use the features described in this area to connect existing clusters, or to create new clusters whose lifecycle is managed by Konvoy. You can view clusters from the Clusters tab in the navigation pane on the left. You can see the details for a cluster by selecting the **View Details** link at the bottom of the cluster card or the cluster name in either the card or the table view.

6.6.1 Cluster Types

There are several types of clusters that display in the Clusters tab. The cluster type appears in the cluster card just under the cluster name.

The type values include:

- **Attached**: An Attached cluster is one that was not created with Kommander. You cannot manage an Attached cluster's lifecycle, but you can monitor it.
- **Managed**: A Managed cluster is a Konvoy cluster that was created with Kommander. You can use Kommander to manage a Managed cluster's lifecycle.
- **Management**: This is the Konvoy cluster that hosts Kommander.

For more information and definitions of these cluster types, refer to Cluster Types and Concepts (see page 97).

6.6.2 Cluster Statuses

A cluster card's status line displays both the current status and the version of Kubernetes running in the cluster.

The status list includes these values:

Status	Description
Pending	This is the initial state when a cluster is created or connected.
Pending Setup	The cluster has networking restrictions that require additional setup, and is not yet connected or attached.
Loading Data	The cluster has been added to Kommander and we are fetching details about the cluster. This is the status before `Active`.
Active	The cluster is connected to API server.
Provisioning*	The cluster is being created on your cloud provider. This process may take some time.
Provisioned*	The cluster's infrastructure has been created and configured.
Joining	The cluster is being joined to the management cluster for federation.
Joined	The join process is done, and waiting for the first data from the cluster to arrive.
Deleting*	The cluster and its resources are being removed from your cloud provider. This process may take some time.
Error	There has been an error connecting to the cluster or retrieving data from the cluster.
Join Failed	This status can appear when kubefed does not have permission to create entities in the target cluster.
Unjoining	Kubefed is cleaning up after itself, removing all installed resources on the target cluster.

Status	Description
Unjoined	The cluster has been disconnected from the management cluster.
Unjoin Failed	The Unjoin from kubefed failed or there is some other error with deleting or disconnecting.
Unattached*	The cluster was created manually and the infrastructure has been created and configured. However, the cluster is not attached. Review the Manually attach a CLI-created cluster (see page 730) page to resolve this status.

*These statuses only appear on Managed clusters.

6.6.3 Cluster Resources

The Resources graphs on a cluster card show you a cluster's resource requests, limits, and usage. This allows a quick, visual scan of cluster health. Hover over each resource to get specific details for that specific cluster resource.

Resource	Description
CPU Requests	The requested portion of the total allocatable CPU resource for the cluster, measured in number of cores, such 0.5 cores.
CPU Limits	The portion of the total allocatable CPU resource to which the cluster is limited, measured in number of cores, such as 0.5 cores.
CPU Usage	The amount of the allocatable CPU resource being consumed. Cannot be higher than the configured CPU limit. Measured in number of cores, such as 0.5 cores)
Memory Requests	The requested portion of the total allocatable memory resource for the cluster, measured in bytes, such as 64 GiB.

Resource	Description
Memory Limits	The portion of the allocatable memory resource to which the cluster is limited, measured in bytes, such as 64 GiB.
Memory Usage	The amount of the allocatable memory resource being consumed. Cannot be higher than the configured memory limit. Measured in bytes, such as 64 GiB.
Disk Requests	The requested portion of the allocatable ephemeral storage resource for the cluster, measured in bytes, such as 64 GiB.
Disk Limits	The portion of the allocatable ephemeral storage resource to which the cluster is limited, measured in bytes, such as 64 GiB.

For more detailed information, see the Kubernetes documentation[472] about resources.

6.6.4 DKP Platform Applications

Platform Applications are applications that come out of the box with DKP providing functionality like observability, cost management, monitoring, logging, making DKP clusters day 2 production ready right from install.

Platform applications are D2iQ's choice of selected applications from the open source community that are consumed by the platform. You can visit a cluster's detail page to see which platform applications are enabled under the "Platform Applications" section.

Review the Workspace Platform Configuration Requirements (see page 114) to ensure that the attached clusters have sufficient resources. For more information on platform applications and how to customize them, refer to the pages in the (2.5) Platform Application (see page 672) section.

6.6.5 Kubernetes Cluster Federation (KubeFed)

Kubernetes Cluster Federation (KubeFed)[473] allows you to coordinate the configuration of multiple Kubernetes clusters from a single set of APIs in a hosting cluster. KubeFed aims to provide mechanisms for expressing which clusters should have their configuration managed and what that configuration should be. The mechanisms that KubeFed provides are intentionally low-level, and intended to be foundational for more complex multicluster use cases such as deploying multi-geo applications and disaster recovery.

In DKP, KubeFed is used to manage multiple clusters from the management cluster and also to federate various resources. A `KubefedCluster` object is automatically created for each attached cluster and

472 https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
473 https://github.com/kubernetes-sigs/kubefed

joined to the management cluster. Once joined, namespaces can be federated to the clusters - this is how you get workspace and project namespaces created on the attached clusters. From here other resources can be federated into those namespaces, such as ConfigMaps, RBAC, and so on.

See these pages for more information:

- https://github.com/kubernetes-sigs/kubefed/blob/master/docs/concepts.md
- https://github.com/kubernetes-sigs/kubefed/blob/master/docs/userguide.md

6.6.6 Attach a Kubernetes Cluster

Enterprise Gov Advanced

Attach an existing Kubernetes cluster using kubeconfig

6.6.6.1 Attach Kubernetes Cluster

You can attach an existing cluster (whether it is a cluster created with the DKP CLI or by means of another platform) to DKP. At the time of attachment, certain namespaces are created on the cluster, and workspace platform applications are deployed automatically into the newly-created namespaces.

Review the Workspace Platform Application Configuration Requirements (see page 114) to ensure the attached cluster has sufficient resources. For more information on platform applications and customizing them, see Workspace Applications (see page 574).

If the cluster you want to attach was created using Amazon EKS, Azure AKS or Google GKE, create a service account as described in Attach a Cluster with no Networking Restrictions (UI) (see page 679).

Platform applications extend the functionality of Kubernetes and provide ready-to-use logging and monitoring stacks by deploying platform applications when attaching a cluster to Kommander. For more information, refer to Workspace Applications (see page 574).

- Requirements for Attaching an Existing Cluster (see page 673)
- Create a kubeconfig File for Attachment (see page 676)
- Attach a Cluster with no Networking Restrictions (UI) (see page 679)
- Attach a Cluster with Networking Restrictions (see page 693)
- Attach a DKP-created Kubernetes Cluster (see page 730)
- Access a Managed or Attached Cluster (see page 731)
- Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 732)

6.6.6.2 Requirements for Attaching an Existing Cluster

Enterprise Gov Advanced

6.6.6.2.1 Basic Requirements

To attach an existing cluster in the UI, the Application Management cluster must be able to reach the services and the `api-server` of the target cluster.

The cluster you want to attach can be a **DKP-CLI-created** cluster (which will become a Managed cluster (see page 97) upon attachment), or **another Kubernetes cluster** like AKS, EKS, or GKE (which will become an Attached cluster (see page 97) upon attachment).

> DKP does not support attachment of K3s clusters.

For attaching existing clusters without networking restrictions, the requirements depend on which DKP version you are using. Each version of DKP supports a specific range of Kubernetes versions[474]. You must ensure that the target cluster is running a compatible version.

> As of this version of DKP, there is no official method of uninstalling Kommander.

6.6.6.2.2 Create a Default StorageClass

To deploy many of the services on the attached cluster, there must be a default `StorageClass` configured. Run the following command on the cluster you want to attach:

```
kubectl get sc
```

The output should look similar to this. Note the `(default)` after the name:

```
NAME              PROVISIONER       RECLAIMPOLICY   VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION   AGE
ebs-sc (default)   ebs.csi.aws.com   Delete          WaitForFirstConsumer   false
41s
```

If the `StorageClass` is not set as default, add the following annotation to the `StorageClass` manifest:

```
annotations:
  storageclass.kubernetes.io/is-default-class: "true"
```

474 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29923823/2.3.0+Release+Notes#Supported-versions

6.6.6.2.3 Creating Projects and Workspaces

Before you attach clusters, you need to create one or more Workspaces, and we recommend that you also create Projects within your Workspaces. Workspaces (see page 573) give you a logical way to represent your teams and specific configurations. Projects (see page 608) let you define one or more clusters as a group to which Kommander pushes a common configuration. Grouping your existing clusters in Kommander projects and workspaces makes managing their platform services and resources easier and supports monitoring and logging.

> Do not attach a cluster in the "Management Cluster Workspace" workspace. This workspace is reserved for your Application Management cluster only.

6.6.6.2.4 Platform Application Requirements

In addition to the basic cluster requirements, the platform services you want DKP to manage on those clusters will have an impact on the total cluster requirements. The specific combinations of platform applications will make a difference in the requirements for the cluster nodes and their resources (CPU, memory, and storage).

See this table of platform applications (see page 564) that DKP provides by default.

6.6.6.2.5 Attach Existing AKS, EKS and GKE clusters

Attaching an existing AWS cluster requires that the cluster be fully configured and running (see page 958). You must create a separate service account when attaching existing AKS, EKS or Google GKE Kubernetes clusters. This is necessary because the kubeconfig files generated from those clusters are not usable out-of-the-box by Kommander. The kubeconfig files call CLI commands, such as `azure, aws` or `gcloud`, and use locally-obtained authentication tokens. Having a separate service account also allows you to keep access to the cluster specific and isolated to Kommander.

The suggested default cluster configuration includes a control plane pool containing three (3) m5.xlarge nodes and a worker pool containing four (4) m5.2xlarge nodes.

Consider the additional resource requirements for running the platform services you want DKP to manage, and ensure that your existing clusters comply.

To attach an existing EKS cluster, refer to the specific information in Attach Amazon EKS Cluster (see page 1022).

To attach an existing GKE cluster, refer to the specific information in Attach GKE Cluster (see page 688).

6.6.6.2.6 Attach Clusters with an Existing cert-manager Installation

If you are attaching clusters that already have cert-manager installed, the cert-manager `HelmRelease` provided by DKP will fail to deploy, due to the existing cert-manager installation. As long as the pre-existing

cert-manager functions as expected, you can ignore this failure. It will have no impact on the operation of the cluster.

6.6.6.3 Create a kubeconfig File for Attachment

Create a separate service account when attaching existing clusters (for example Amazon EKS, or Azure AKS clusters)

> If you already have a `kubeconfig` file to attach your cluster, go directly to Attach a Cluster with no Networking Restrictions (UI) (see page 679) or Attach a Cluster with Networking Restrictions (see page 693).

The `kubeconfig` files generated from existing clusters are not usable out-of-the-box, because they call provisioner-specific CLI commands (like `aws` commands), and use locally-obtained authentication tokens that are not compatible with DKP. Having a separate service account also allows you to have a dedicated identity for all DKP operations.

To get started, ensure you have kubectl[475] set up and configured with ClusterAdmin[476] for the cluster you want to connect to Kommander.

1. Create the necessary service account:

```
kubectl -n kube-system create serviceaccount kommander-cluster-admin
```

2. Create a token secret for the `serviceaccount`:

```
kubectl -n kube-system create  -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: kommander-cluster-admin-sa-token
  annotations:
    kubernetes.io/service-account.name: kommander-cluster-admin
type: kubernetes.io/service-account-token
EOF
```

3. Verify that the `serviceaccount` token is ready by running this command:

```
kubectl -n kube-system get secret kommander-cluster-admin-sa-token -oyaml
```

475 https://kubernetes.io/docs/tasks/tools/#kubectl
476 https://kubernetes.io/docs/concepts/cluster-administration/

Verify that the `data.token` field is populated. The output should be similar to this:

```
apiVersion: v1
data:
  ca.crt: LS0tLS1CRUdJTiBDR...
  namespace: ZGVmYXVsdA==
  token: ZXlKaGJHY2lPaUpTVX...
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: kommander-cluster-admin
    kubernetes.io/service-account.uid: b62bc32e-b502-4654-921d-94a742e273a8
  creationTimestamp: "2022-08-19T13:36:42Z"
  name: kommander-cluster-admin-sa-token
  namespace: default
  resourceVersion: "8554"
  uid: 72c2a4f0-636d-4a70-9f1c-55a75f15e520
type: kubernetes.io/service-account-token
```

4. Configure the new service account for `cluster-admin` permissions:

```
cat << EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kommander-cluster-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: kommander-cluster-admin
  namespace: kube-system
EOF
```

5. Set up the following environment variables with the access data that is needed for producing a new kubeconfig file:

```
export USER_TOKEN_VALUE=$(kubectl -n kube-system get secret/kommander-cluster-
admin-sa-token -o=go-template='{{.data.token}}' | base64 --decode)
export CURRENT_CONTEXT=$(kubectl config current-context)
export CURRENT_CLUSTER=$(kubectl config view --raw -o=go-
template='{{range .contexts}}{{if eq .name "'''${CURRENT_CONTEXT}'''"}}
{{ index .context "cluster" }}{{end}}{{end}}')
export CLUSTER_CA=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}"{{with
index .cluster "certificate-authority-data" }}{{.}}{{end}}"{{ end }}{{ end }}')
```

```
export CLUSTER_SERVER=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}
{{ .cluster.server }}{{end}}{{ end }}')
```

6. Confirm these variables have been set correctly:

```
export -p USER_TOKEN_VALUE CURRENT_CONTEXT CURRENT_CLUSTER CLUSTER_CA
CLUSTER_SERVER
```

7. Generate a `kubeconfig` file that uses the environment variable values from the previous step:

```
cat << EOF > kommander-cluster-admin-config
apiVersion: v1
kind: Config
current-context: ${CURRENT_CONTEXT}
contexts:
- name: ${CURRENT_CONTEXT}
  context:
    cluster: ${CURRENT_CONTEXT}
    user: kommander-cluster-admin
    namespace: kube-system
clusters:
- name: ${CURRENT_CONTEXT}
  cluster:
    certificate-authority-data: ${CLUSTER_CA}
    server: ${CLUSTER_SERVER}
users:
- name: kommander-cluster-admin
  user:
    token: ${USER_TOKEN_VALUE}
EOF
```

8. This process produces a file in your current working directory called `kommander-cluster-admin-config`. The contents of this file are used in Kommander to attach the cluster.
   Before importing this configuration, verify the `kubeconfig` file can access the cluster:

```
kubectl --kubeconfig $(pwd)/kommander-cluster-admin-config get all --all-
namespaces
```

6.6.6.3.1 Next Step:

Use this `kubeconfig` to:

- Attach a cluster with no additional networking restrictions (see page 679)
- Attach a cluster that has networking restrictions (see page 693)

If a cluster has limited resources to deploy all the federated platform services, it will fail to stay attached in the DKP UI. If this happens, check if there are any pods that are not getting the resources required.

6.6.6.4 Attach a Cluster with no Networking Restrictions (UI)

Enterprise Gov Advanced

For provider-specific instructions, refer to:

- Attach Amazon EKS Cluster (see page 680)
- Attach AKS Cluster (see page 684)
- Attach GKE Cluster (see page 688)

Using the **Add Cluster** option, you can attach an existing Kubernetes or DKP cluster directly to DKP. This enables you to access the multi-cluster management and monitoring benefits that DKP provides, while keeping your existing cluster on its current provider and infrastructure.

How to attach an existing cluster that has no additional networking restrictions

Use this option when you want to attach a cluster that does not require additional access information.

1. From the top menu bar, select your target workspace.
2. On the Dashboard page, select the **Add Cluster** option in the **Actions** dropdown menu at the top right.
3. Select **Attach Cluster**.
4. Select the **No additional networking restrictions** card. Alternatively, if you MUST use network restrictions, stop following the steps below, and see the instructions on the page Attach a Cluster with Network Restrictions (see page 693).
5. In the **Cluster Configuration** section, paste your `kubeconfig` file into the field, or select the **Upload kubeconfig File** button to specify the file.
6. The **Cluster Name** field will automatically populate with the name of the cluster is in the `kubeconfig`. You can edit this field with the name you want for your cluster.
7. The **Context** select list is populated from the `kubeconfig`. Select the desired context with admin privileges from the **Context** select list.
8. Add labels to classify your cluster as needed.
9. Select **Create** to attach your cluster.

6.6.6.4.1 Attach Amazon EKS Cluster

Enterprise Gov Advanced

6.6.6.4.1.1 Attach an existing EKS cluster

You can attach existing Kubernetes clusters to the Management Cluster. After attaching the cluster, you can use the UI to examine and manage (see page 669) this cluster. The following procedure shows how to attach an existing Amazon Elastic Kubernetes Service (EKS) cluster.

6.6.6.4.1.2 Before you Begin

This procedure requires the following items and configurations:

- A fully configured and running Amazon EKS[477] cluster with administrative privileges.
- The current version DKP Enterprise is installed (see page 1269)on your cluster.
- Ensure you have installed `kubectl` in your Management cluster.

This procedure assumes you have an existing and spun up Amazon EKS cluster(s) with administrative privileges. Refer to the Amazon EKS[478] for setup and configuration information.

6.6.6.4.1.3 Attach Amazon EKS Clusters

Access your EKS clusters

1. Ensure you are connected to your EKS clusters. Enter the following commands for each of your clusters:

   ```
   kubectl config get-contexts
   kubectl config use-context <context for first eks cluster>
   ```

2. Confirm `kubectl` can access the EKS cluster:

477 https://aws.amazon.com/eks/
478 https://aws.amazon.com/eks/

```
kubectl get nodes
```

Create a `kubeconfig` File for your EKS cluster

To get started, ensure you have kubectl[479] set up and configured with ClusterAdmin[480] for the cluster you want to connect to Kommander.

1. Create the necessary service account:

   ```
   kubectl -n kube-system create serviceaccount kommander-cluster-admin
   ```

2. Create a token secret for the `serviceaccount`:

   ```
   kubectl -n kube-system create  -f - <<EOF
   apiVersion: v1
   kind: Secret
   metadata:
     name: kommander-cluster-admin-sa-token
     annotations:
       kubernetes.io/service-account.name: kommander-cluster-admin
   type: kubernetes.io/service-account-token
   EOF
   ```

   For more information on Service Account Tokens, refer to this article[481] in our blog.

3. Verify that the `serviceaccount` token is ready by running this command:

   ```
   kubectl -n kube-system get secret kommander-cluster-admin-sa-token -oyaml
   ```

   Verify that the `data.token` field is populated. The output should be similar to this:

   ```
   apiVersion: v1
   data:
     ca.crt: LS0tLS1CRUdJTiBDR...
     namespace: ZGVmYXVsdA==
     token: ZXlKaGJHY2lPaUpTVX...
   kind: Secret
   metadata:
     annotations:
       kubernetes.io/service-account.name: kommander-cluster-admin
   ```

479 https://kubernetes.io/docs/tasks/tools/#kubectl
480 https://kubernetes.io/docs/concepts/cluster-administration/
481 https://eng.d2iq.com/blog/service-account-tokens-in-kubernetes-v1.24/#whats-changed-in-kubernetes-v124

```
    kubernetes.io/service-account.uid: b62bc32e-b502-4654-921d-94a742e273a8
  creationTimestamp: "2022-08-19T13:36:42Z"
  name: kommander-cluster-admin-sa-token
  namespace: default
  resourceVersion: "8554"
  uid: 72c2a4f0-636d-4a70-9f1c-55a75f15e520
type: kubernetes.io/service-account-token
```

4. Configure the new service account for `cluster-admin` permissions:

```
cat << EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kommander-cluster-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: kommander-cluster-admin
  namespace: kube-system
EOF
```

5. Set up the following environment variables with the access data that is needed for producing a new kubeconfig file:

```
export USER_TOKEN_VALUE=$(kubectl -n kube-system get secret/kommander-cluster-
admin-sa-token -o=go-template='{{.data.token}}' | base64 --decode)
export CURRENT_CONTEXT=$(kubectl config current-context)
export CURRENT_CLUSTER=$(kubectl config view --raw -o=go-
template='{{range .contexts}}{{if eq .name "'''${CURRENT_CONTEXT}'''"}}
{{ index .context "cluster" }}{{end}}{{end}}')
export CLUSTER_CA=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}"{{with
index .cluster "certificate-authority-data" }}{{.}}{{end}}"{{ end }}{{ end }}')
export CLUSTER_SERVER=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}
{{ .cluster.server }}{{end}}{{ end }}')
```

6. Confirm these variables have been set correctly:

```
export -p | grep -E 'USER_TOKEN_VALUE|CURRENT_CONTEXT|CURRENT_CLUSTER|
CLUSTER_CA|CLUSTER_SERVER'
```

7. Generate a kubeconfig file that uses the environment variable values from the previous step:

```
cat << EOF > kommander-cluster-admin-config
apiVersion: v1
kind: Config
current-context: ${CURRENT_CONTEXT}
contexts:
- name: ${CURRENT_CONTEXT}
  context:
    cluster: ${CURRENT_CONTEXT}
    user: kommander-cluster-admin
    namespace: kube-system
clusters:
- name: ${CURRENT_CONTEXT}
  cluster:
    certificate-authority-data: ${CLUSTER_CA}
    server: ${CLUSTER_SERVER}
users:
- name: kommander-cluster-admin
  user:
    token: ${USER_TOKEN_VALUE}
EOF
```

8. This process produces a file in your current working directory called `kommander-cluster-admin-config`. The contents of this file are used in Kommander to attach the cluster.
   Before importing this configuration, verify the `kubeconfig` file can access the cluster:

```
kubectl --kubeconfig $(pwd)/kommander-cluster-admin-config get all --all-namespaces
```

Finalize Attachment of your Cluster from the UI

Now that you have kubeconfig, go to the DKP UI and follow these steps below:

1. From the top menu bar, select your target workspace.
2. On the Dashboard page, select the **Add Cluster** option in the **Actions** dropdown menu at the top right.
3. Select **Attach Cluster**.
4. Select the **No additional networking restrictions** card. Alternatively, if you must use network restrictions, stop following the steps below, and see the instructions on the page Attach a cluster WITH network restrictions (see page 693).
5. Upload the kubeconfig file you created in the previous section (or copy its contents) into the **Cluster Configuration** section.
6. The **Cluster Name** field automatically populates with the name of the cluster in the kubeconfig. You can edit this field with the name you want for your cluster.
7. Add labels to classify your cluster as needed.

8. Select **Create** to attach your cluster.

> If a cluster has limited resources to deploy all the federated platform services, it will fail to stay attached in the DKP UI. If this happens, ensure your system has sufficient resources for all pods.

6.6.6.4.1.4 Related Information

For information on related topics or procedures, refer to the following:

- Configuring and Running Amazon EKS Clusters[482]
- Installing and Configuring Konvoy v2.0 or above (see page 1005)
- Installing and Configuring Kommander v2.0 or above (see page 1269)
- Manage Clusters (see page 669)

6.6.6.4.2 Attach AKS Cluster

ENTERPRISE

6.6.6.4.2.1 Attach an existing AKS cluster

You can attach existing Kubernetes clusters to the Management Cluster. After attaching the cluster, you can use the UI to examine and manage (see page 669) this cluster. The following procedure shows how to attach an existing Azure Kubernetes Service (AKS) cluster.

6.6.6.4.2.2 Before you Begin

This procedure requires the following items and configurations:

- A fully configured and running Azure AKS[483]cluster with administrative privileges.
- The current version DKP Enterprise is installed (see page 1269)on your cluster.
- Ensure you have installed `kubectl` in your Management cluster.

> This procedure assumes you have an existing and spun up Azure AKS cluster(s) with administrative privileges. Refer to the Azure AKS[484]for setup and configuration information.

482 https://aws.amazon.com/eks/
483 https://azure.microsoft.com/en-us/products/kubernetes-service/
484 https://azure.microsoft.com/en-us/products/kubernetes-service/

6.6.6.4.2.3 Attach AKS Clusters

Ensure that the `KUBECONFIG` environment variable is set to the Management cluster before attaching by running:

```
export KUBECONFIG=<Management_cluster_kubeconfig>.conf
```

Ensure you have access to your AKS clusters

1. Ensure you are connected to your AKS clusters. Enter the following commands for each of your clusters:

   ```
   kubectl config get-contexts
   kubectl config use-context <context for first AKS cluster>
   ```

2. Confirm `kubectl` can access the AKS cluster:

   ```
   kubectl get nodes
   ```

Create a kubeconfig file for your AKS cluster

To get started, ensure you have kubectl[485] set up and configured with ClusterAdmin[486] for the cluster you want to connect to Kommander.

1. Create the necessary service account:

   ```
   kubectl -n kube-system create serviceaccount kommander-cluster-admin
   ```

2. Create a token secret for the `serviceaccount`:

   ```
   kubectl -n kube-system create  -f - <<EOF
   apiVersion: v1
   kind: Secret
   metadata:
     name: kommander-cluster-admin-sa-token
     annotations:
       kubernetes.io/service-account.name: kommander-cluster-admin
   type: kubernetes.io/service-account-token
   ```

485 https://kubernetes.io/docs/tasks/tools/#kubectl
486 https://kubernetes.io/docs/concepts/cluster-administration/

```
EOF
```

3. Verify that the `serviceaccount` token is ready by running this command:

   ```
   kubectl -n kube-system get secret kommander-cluster-admin-sa-token -oyaml
   ```

   Verify that the `data.token` field is populated. The output should be similar to this:

   ```
   apiVersion: v1
   data:
     ca.crt: LS0tLS1CRUdJTiBDR...
     namespace: ZGVmYXVsdA==
     token: ZXlKaGJHY2lPaUpTVX...
   kind: Secret
   metadata:
     annotations:
       kubernetes.io/service-account.name: kommander-cluster-admin
       kubernetes.io/service-account.uid: b62bc32e-b502-4654-921d-94a742e273a8
     creationTimestamp: "2022-08-19T13:36:42Z"
     name: kommander-cluster-admin-sa-token
     namespace: default
     resourceVersion: "8554"
     uid: 72c2a4f0-636d-4a70-9f1c-55a75f15e520
   type: kubernetes.io/service-account-token
   ```

4. Configure the new service account for `cluster-admin` permissions:

   ```
   cat << EOF | kubectl apply -f -
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRoleBinding
   metadata:
     name: kommander-cluster-admin
   roleRef:
     apiGroup: rbac.authorization.k8s.io
     kind: ClusterRole
     name: cluster-admin
   subjects:
   - kind: ServiceAccount
     name: kommander-cluster-admin
     namespace: kube-system
   EOF
   ```

5. Set up the following environment variables with the access data that is needed for producing a new `kubeconfig` file:

```
export USER_TOKEN_VALUE=$(kubectl -n kube-system get secret/kommander-cluster-
admin-sa-token -o=go-template='{{.data.token}}' | base64 --decode)
export CURRENT_CONTEXT=$(kubectl config current-context)
export CURRENT_CLUSTER=$(kubectl config view --raw -o=go-
template='{{range .contexts}}{{if eq .name "'''${CURRENT_CONTEXT}'''"}}
{{ index .context "cluster" }}{{end}}{{end}}')
export CLUSTER_CA=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}"{{with
index .cluster "certificate-authority-data" }}{{.}}{{end}}"{{ end }}{{ end }}')
export CLUSTER_SERVER=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}
{{ .cluster.server }}{{end}}{{ end }}')
```

6. Confirm these variables have been set correctly:

```
export -p | grep -E 'USER_TOKEN_VALUE|CURRENT_CONTEXT|CURRENT_CLUSTER|
CLUSTER_CA|CLUSTER_SERVER'
```

7. Generate a `kubeconfig` file that uses the environment variable values from the previous step:

```
cat << EOF > kommander-cluster-admin-config
apiVersion: v1
kind: Config
current-context: ${CURRENT_CONTEXT}
contexts:
- name: ${CURRENT_CONTEXT}
  context:
    cluster: ${CURRENT_CONTEXT}
    user: kommander-cluster-admin
    namespace: kube-system
clusters:
- name: ${CURRENT_CONTEXT}
  cluster:
    certificate-authority-data: ${CLUSTER_CA}
    server: ${CLUSTER_SERVER}
users:
- name: kommander-cluster-admin
  user:
    token: ${USER_TOKEN_VALUE}
EOF
```

8. This process produces a file in your current working directory called `kommander-cluster-admin-config`. The contents of this file are used in Kommander to attach the cluster.
   Before importing this configuration, verify the `kubeconfig` file can access the cluster:

```
kubectl --kubeconfig $(pwd)/kommander-cluster-admin-config get all --all-
namespaces
```

Finalize attaching your cluster from the UI

Now that you have `kubeconfig`, go to the DKP UI and follow these steps below:

1. From the top menu bar, select your target workspace.
2. On the Dashboard page, select the **Add Cluster** option in the **Actions** dropdown menu at the top right.
3. Select **Attach Cluster**.
4. Select the **No additional networking restrictions** card. Alternatively, if you must use network restrictions, stop following the steps below, and see the instructions on the page Attach a cluster WITH network restrictions (see page 693).
5. Upload the kubeconfig file you created in the previous section (or copy its contents) into the **Cluster Configuration** section.
6. The **Cluster Name** field automatically populates with the name of the cluster in the kubeconfig. You can edit this field with the name you want for your cluster.
7. Add labels to classify your cluster as needed.
8. Select **Create** to attach your cluster.

If a cluster has limited resources to deploy all the federated platform services, it will fail to stay attached in the DKP UI. If this happens, ensure your system has sufficient resources for all pods.

6.6.6.4.2.4 Related Information

For information on related topics or procedures, refer to the following:

- Installing and Configuring Konvoy v2.0 or above (see page 1068)
- Installing and Configuring Kommander v2.0 or above (see page 1269)
- Manage Clusters (see page 669)

6.6.6.4.3 Attach GKE Cluster

Enterprise Gov Advanced

Attach an existing GKE cluster in DKP

You can attach existing Kubernetes clusters to the Management Cluster. After attaching the cluster, you can use the UI to examine and manage (see page 669) this cluster. The following procedure shows how to attach an existing **standard** GKE cluster.

6.6.6.4.3.1 Before you Begin

This procedure requires the following items and configurations:

- A fully configured and running GKE cluster[487] with a supported Kubernetes version (see page 1543), and administrative privileges.
- The current version of DKP Enterprise installed (see page 1269)on your cluster.
- Ensure you have installed `kubectl` in your Management cluster.

> This procedure assumes you have an existing and spun up GKE cluster with administrator privileges.

6.6.6.4.3.2 Attach GKE Clusters

Ensure you have access to your GKE clusters

1. Ensure you are connected to your GKE clusters. Enter the following commands for each of your clusters:

   ```
   kubectl config get-contexts
   kubectl config use-context <context for first gcloud cluster>
   ```

2. Confirm `kubectl` can access the GKE cluster.

   ```
   kubectl get nodes
   ```

Configure a kubeconfig file

To get started, ensure you have kubectl[488] set up and configured with ClusterAdmin[489] for the cluster you want to connect to Kommander.

487 https://cloud.google.com/kubernetes-engine/docs/concepts/types-of-clusters
488 https://kubernetes.io/docs/tasks/tools/#kubectl
489 https://kubernetes.io/docs/concepts/cluster-administration/

1. Create the necessary service account:

```
kubectl -n kube-system create serviceaccount kommander-cluster-admin
```

2. Create a token secret for the `serviceaccount`:

```
kubectl -n kube-system create  -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: kommander-cluster-admin-sa-token
  annotations:
    kubernetes.io/service-account.name: kommander-cluster-admin
type: kubernetes.io/service-account-token
EOF
```

For more information on Service Account Tokens, refer to this article[490] in our blog.

3. Verify that the `serviceaccount` token is ready by running this command:

```
kubectl -n kube-system get secret kommander-cluster-admin-sa-token -oyaml
```

Verify that the `data.token` field is populated. The output should be similar to this:

```
apiVersion: v1
data:
  ca.crt: LS0tLS1CRUdJTiBDR...
  namespace: ZGVmYXVsdA==
  token: ZXlKaGJHY2lPaUpTVX...
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: kommander-cluster-admin
    kubernetes.io/service-account.uid: b62bc32e-b502-4654-921d-94a742e273a8
  creationTimestamp: "2022-08-19T13:36:42Z"
  name: kommander-cluster-admin-sa-token
  namespace: default
  resourceVersion: "8554"
  uid: 72c2a4f0-636d-4a70-9f1c-55a75f15e520
type: kubernetes.io/service-account-token
```

4. Configure the new service account for `cluster-admin` permissions:

490 https://eng.d2iq.com/blog/service-account-tokens-in-kubernetes-v1.24/#whats-changed-in-kubernetes-v124

```
cat << EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kommander-cluster-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: kommander-cluster-admin
  namespace: kube-system
EOF
```

5. Set up the following environment variables with the access data that is needed for producing a new kubeconfig file:

```
export USER_TOKEN_VALUE=$(kubectl -n kube-system get secret/kommander-cluster-
admin-sa-token -o=go-template='{{.data.token}}' | base64 --decode)
export CURRENT_CONTEXT=$(kubectl config current-context)
export CURRENT_CLUSTER=$(kubectl config view --raw -o=go-
template='{{range .contexts}}{{if eq .name "'''${CURRENT_CONTEXT}'''"}}
{{ index .context "cluster" }}{{end}}{{end}}')
export CLUSTER_CA=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}"{{with
index .cluster "certificate-authority-data" }}{{.}}{{end}}"{{ end }}{{ end }}')
export CLUSTER_SERVER=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}
{{ .cluster.server }}{{end}}{{ end }}')
```

6. Confirm these variables have been set correctly:

```
export -p | grep -E 'USER_TOKEN_VALUE|CURRENT_CONTEXT|CURRENT_CLUSTER|
CLUSTER_CA|CLUSTER_SERVER'
```

7. Generate a kubeconfig file that uses the environment variable values from the previous step:

```
cat << EOF > kommander-cluster-admin-config
apiVersion: v1
kind: Config
current-context: ${CURRENT_CONTEXT}
contexts:
- name: ${CURRENT_CONTEXT}
  context:
    cluster: ${CURRENT_CONTEXT}
    user: kommander-cluster-admin
```

```
    namespace: kube-system
clusters:
- name: ${CURRENT_CONTEXT}
  cluster:
    certificate-authority-data: ${CLUSTER_CA}
    server: ${CLUSTER_SERVER}
users:
- name: kommander-cluster-admin
  user:
    token: ${USER_TOKEN_VALUE}
EOF
```

8. This process produces a file in your current working directory called `kommander-cluster-admin-config`. The contents of this file are used in Kommander to attach the cluster.
   Before importing this configuration, verify the `kubeconfig` file can access the cluster:

   ```
   kubectl --kubeconfig $(pwd)/kommander-cluster-admin-config get all --all-namespaces
   ```

Attach the Cluster

Now that you have a kubeconfig file, go to the DKP UI and follow these steps:

1. Select your target workspace from the top menu bar.
2. Select **Add Cluster** in the **Actions** dropdown menu from the Dashboard page, located at the top-right.
3. Select **Attach Cluster**.
4. Select the **No additional networking restrictions** card.
   Alternatively, if you must use network restrictions, stop following the steps below, and see the instructions on the page Attach a cluster WITH network restrictions (see page 693).
5. Upload the kubeconfig file you created in the previous section (or copy its contents) into the **Cluster Configuration** section.
6. The **Cluster Name** field automatically populates with the name of the cluster in the kubeconfig. You can edit this field with the name you want for your cluster.
7. Add labels to classify your cluster as needed.
8. Select **Create** to attach your cluster.

If a cluster has limited resources to deploy all the federated platform services, it will fail to stay attached in the DKP UI. If this happens, ensure your system has sufficient resources for all pods.

6.6.6.4.3.3 Related Information

For information on related topics or procedures, refer to the following:

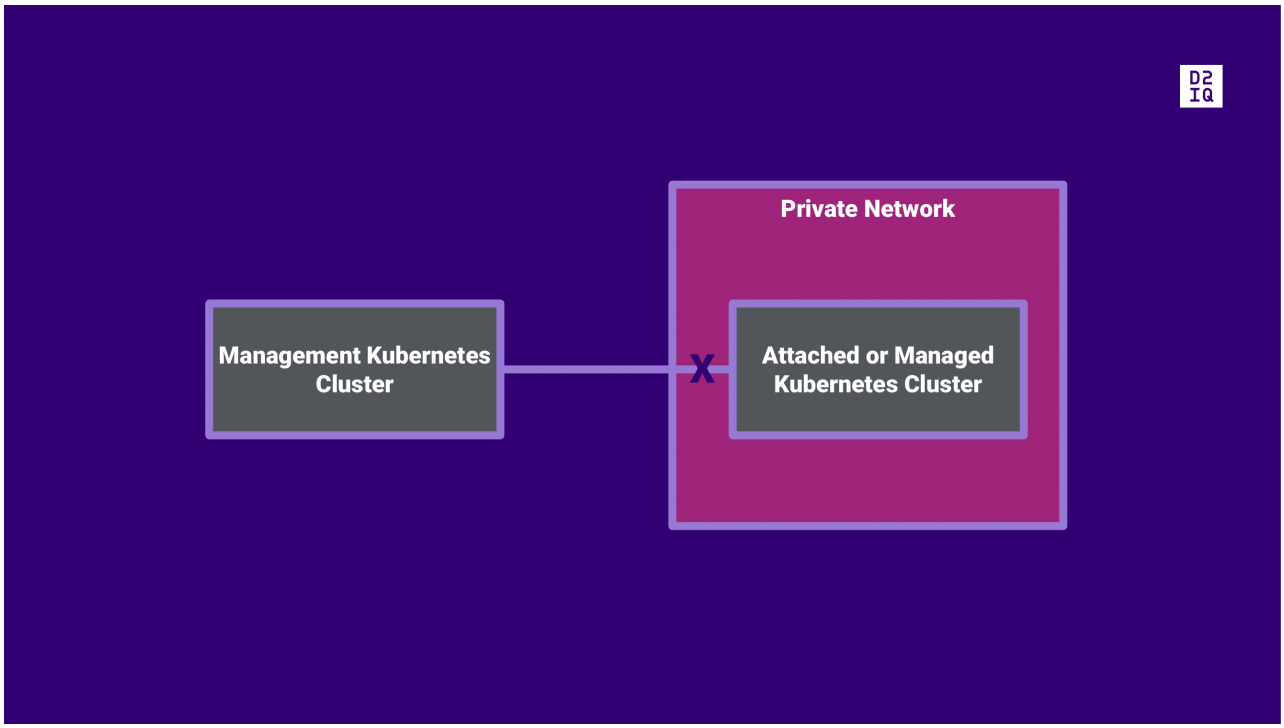
- Installing and Configuring Kommander (see page 1269)
- Manage Clusters (see page 669)

6.6.6.5 Attach a Cluster with Networking Restrictions

Enterprise Gov Advanced

6.6.6.5.1 When do I Need to use a Secure Tunnel?

When attaching a cluster to DKP, the Management cluster initiates an outbound connection to the cluster you wish to attach. This is not possible if the cluster you want to attach (Managed or Attached (see page 97)) has networking restrictions and is not exposed, for example, because it is in a private network, or its API is not accessible from the same network as the Management cluster.



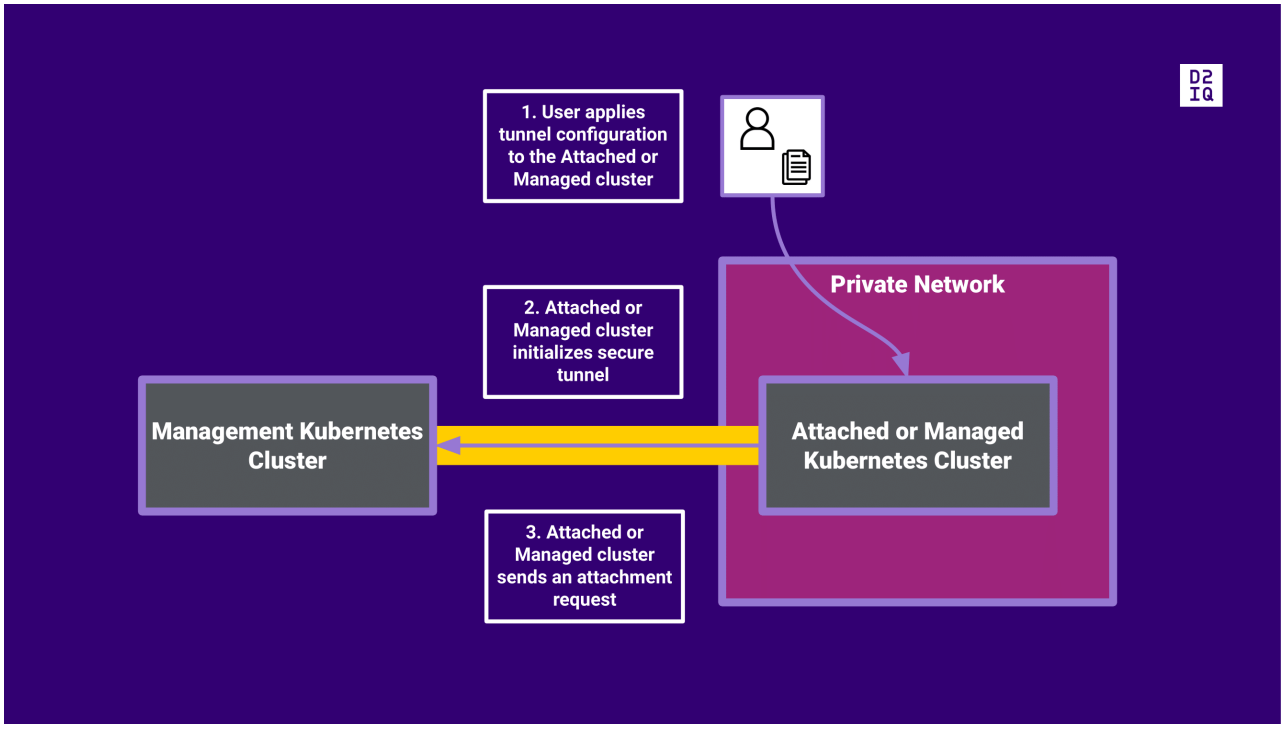
1 Diagram: Attachment not possible when cluster is in a private network.

Use a tunnel when you want to attach a cluster that is in a DMZ, behind a NAT gateway, behind a proxy server, firewall, or requires additional access information.

6.6.6.5.2 How Does the Tunneled Attachment work?

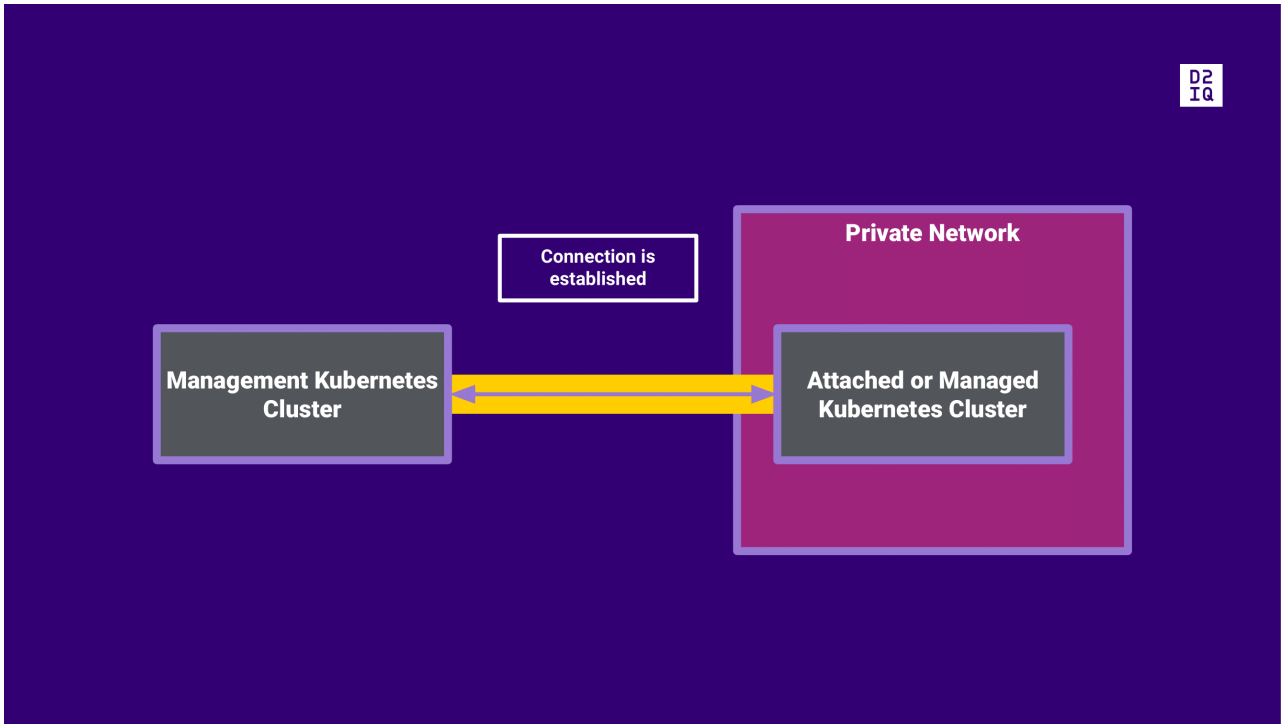
DKP can create a secure tunnel to enable the attachment of clusters that are not directly reachable.

To create a secure tunnel, you must provide a configuration for the tunnel in the cluster you want to attach. After you apply that configuration, the cluster you want to attach will establish a secure tunnel with the Management cluster, and make an attachment request.



2 Diagram: Attachment through a secure tunnel

After the attachment request is accepted and the connection between clusters is established, both clusters will allow bilateral communication.



3 Diagram: Connection between clusters is established

6.6.6.5.3 What are the Steps to Attach a Cluster via a Tunnel?

Ensure you meet the prerequisites for a tunneled attachment, and then attach the cluster with the UI or CLI:

- Prerequisites for a Tunneled Attachment (see page 695)
- Attach a Cluster using a Tunnel via the UI (see page 697)
- Attach a Cluster Using a Tunnel via the CLI (see page 707)
- API documentation (v1alpha1) (see page 722)

6.6.6.5.4 Prerequisites for a Tunneled Attachment

Enterprise Gov Advanced

6.6.6.5.4.1 Before you Begin

- Gain more understanding of this approach by reviewing Attach a Cluster with Networking Restrictions (see page 693).
- Ensure you have reviewed the general Requirements for Attaching an Existing Cluster (see page 673).

6.6.6.5.4.2 Prerequisites

To enable a tunneled attachment, you have the following **additional** prerequisites:

- Ensure that `kubetunnel` is deployed on the Management Cluster (default DKP configuration).
  Use the following command to check if `kubetunnel` is deployed:

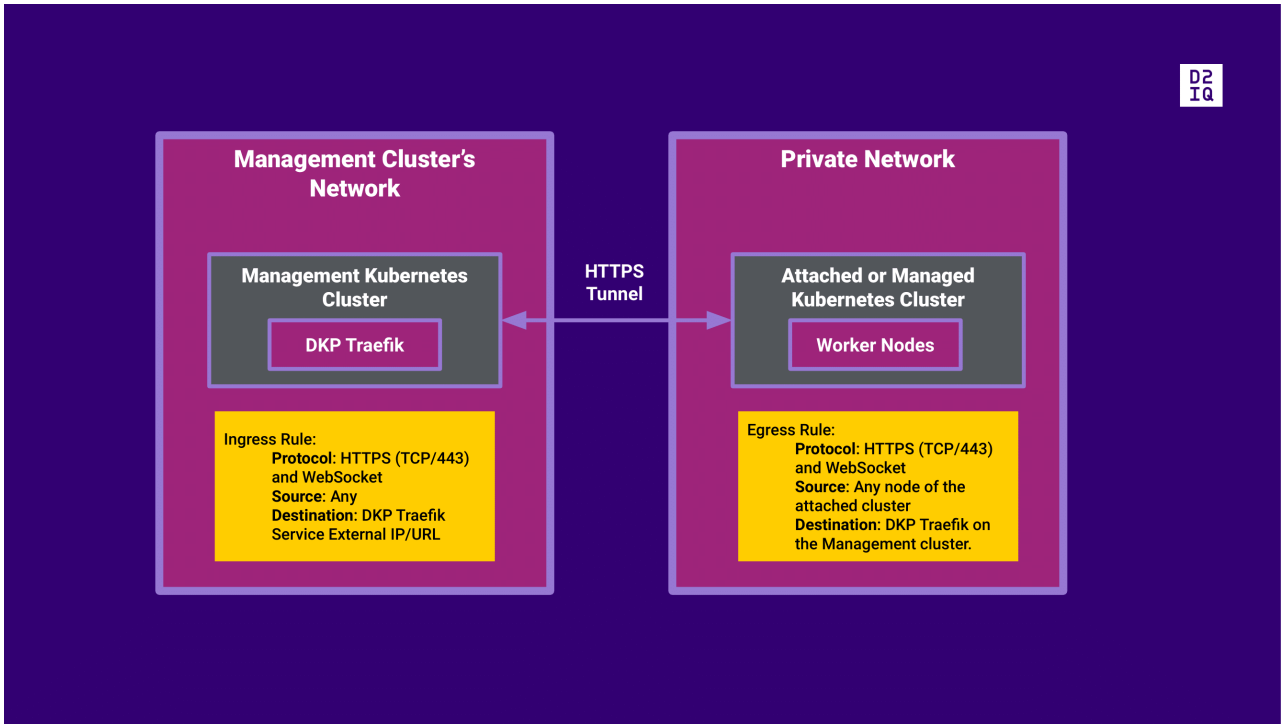
  ```
  kubectl get appdeployments.apps.kommander.d2iq.io -n kommander kubetunnel
  ```

  The output should look similar to this:

  ```
  NAME         APP                 AGE
  kubetunnel   kubetunnel-<version>   5h14m
  ```

- **Firewall rules**:

	The ingress rule on the Management cluster network must allow:	**The egress rule on the Attached or Managed cluster private network must allow:**
Protocol	HTTPS (TCP/443) and WebSocket	HTTPS (TCP/443) and WebSocket
Source	Any	Any node of the Attached or Managed cluster
Destination	DKP Traefik Service External IP/URL	DKP Traefik Service on the Management cluster



Choose your Next Step:

Attach a Cluster using a Tunnel via the UI (see page 697)

Attach a Cluster Using a Tunnel via the CLI (see page 707)

6.6.6.5.5 Attach a Cluster using a Tunnel via the UI

Use the UI to attach an existing cluster that has additional networking restrictions

Ensure you have reviewed and followed the steps in Prerequisites for a Tunneled Attachment (see page 695) before proceeding.

6.6.6.5.5.1 Attach a Cluster Using a Tunnel

Here is a high-level overview of the steps required to attach a cluster via the UI:

- UI: Attach a Cluster (see page 698)
- UI: Finish Attaching the Existing Cluster (see page 699)
- Use the Remote Cluster (see page 700)

6.6.6.5.5.2 UI: Attach a Cluster

1. From the top menu bar, select your target workspace.
2. On the Dashboard page, select the **Add Cluster** option in the **Actions** dropdown menu at the top right.
3. Select **Attach Cluster**.
4. Select the **Cluster has networking restrictions** card to display the configuration page.
5. Enter the **Cluster Name** of the cluster you're attaching.
6. Create additional new Labels as needed.
7. Select the hostname that is the Ingress for the cluster from the **Load Balancer Hostname** dropdown menu. The hostname must match the Kommander Host cluster to which you are attaching your existing cluster with network restrictions.
8. Specify the **URL Path Prefix** for your Load Balancer Hostname. This URL path will serve as the prefix for the specific tunnel services you want to expose on the Kommander management cluster. If no value is specified, the value defaults to `/dkp/tunnel`.

   NOTE: Kommander uses Traefik 2 ingress, which requires explicit definition of strip prefix middleware as a Kubernetes API object, opposed to a simple annotation. Kommander provides default middleware that supports creating tunnels only on the `/dkp/tunnel URL` prefix. This is indicated by using the extra annotation, `traefik.ingress.kubernetes.io/router.middlewares: kommander-stripprefixes-kubetunnel@kubernetescrd` as shown in the code sample that follows. If you want to expose a tunnel on a different URL prefix, you must manage your own middleware configuration.
9. (Optional) Enter a value for the **Hostname** field.
10. If you have not attached this cluster before, you must create a new secret in the **Root CA Certificate** drop down menu. To do this in your Konvoy management cluster, view your base64 encoded Kubernetes secret values to copy and paste into the **Root CA Certificate** field:

    ```
    echo $(kubectl get secret -n cert-manager kommander-ca -o=go-
    template='{{index .data "tls.crt"}}')
    ```

    Otherwise, select from the list of available Secrets.
11. Add any **Extra Annotations** as needed.
12. Select the **Save & Generate kubeconfig** button to generate the kubeconfig file for the network tunnel.

Next Step:

Finish Attaching the Existing Cluster (see page 699)

Related Topic:

For information on the TunnelGateway review the API documentation (v1alpha1) (see page 722).

6.6.6.5.5.3 UI: Finish Attaching the Existing Cluster

Enterprise Gov Advanced

How to apply the kubeconfig file to create the network tunnel to attach an existing cluster

Though the required file is now generated, you still need to apply it to create the network tunnel and complete the attachment process.

1. Select the **Download Manifest** link to download the file you generated previously (see page 693).
2. Copy the `kubectl apply` command from the user interface and paste into your terminal session, DO NOT run it yet.
3. Ensure you substitute the actual name of the file for the variable. Also ensure you use the `--kubeconfig=<managed_cluster_kubeconfig.conf>` flag to run the command on the Attached or Managed cluster. Run the command.
   :note: Running this command starts the attachment process, which may take several minutes to complete. The Cluster details page will be displayed automatically when the cluster attachment process completes.
4. (Optional) Select the **Verify Connection to Cluster** button to send a request to Kommander to refresh the connection information. You can use this option to check to see if the connection is complete, though the Cluster Details page displays automatically when the connection is complete.

> After the initial connection is made, and your cluster becomes viewable as attached in the DKP UI, the attachment, federated add-ons, and platform services will still need to be completed. This may take several additional minutes. If a cluster has limited resources to deploy all the federated platform services, the installation of the federated resources will fail and the cluster may become unreachable in the DKP UI. If this happens, check whether there are any pods that are not getting the resources required.

Next Step:

Now you can learn how to Use the Remote Cluster (see page 700) (this approach uses the CLI).

6.6.6.5.5.4 Use the Remote Cluster

Enterprise Gov Advanced

To access services running on the remote cluster from the Management cluster, connect to the tunnel proxy.

You can use these three methods:

1. If the client program supports use of a kubeconfig file, use the Attached or Managed cluster's kubeconfig.
2. If the client program supports SOCKS5 proxies, use the proxy directly.
3. Otherwise, deploy a proxy server on the Management cluster.

Attached or Managed Cluster Service

These sections require a service to run on the Attached or Managed cluster.

As an example, start the following service:

```
service_namespace=test
service_name=webserver
service_port=8888
service_endpoint=${service_name}.${service_namespace}.svc.cluster.local:$
{service_port}

cat > nginx.yaml <<EOF
apiVersion: v1
kind: Namespace
metadata:
  name: ${service_namespace}
---
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: ${service_namespace}
  name: nginx-deployment
  labels:
    app: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx-app
  template:
    metadata:
      labels:
```

```
        app: nginx-app
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  namespace: ${service_namespace}
  name: ${service_name}
spec:
  selector:
    app: nginx-app
  type: ClusterIP
  ports:
  - targetPort: 80
    port: ${service_port}
EOF

kubectl apply -f nginx.yaml

kubectl rollout status deploy -n ${service_namespace} nginx-deployment
```

On the Attached or Managed cluster, a client Job can access this service using:

```
cat > curl.yaml <<EOF
apiVersion: batch/v1
kind: Job
metadata:
  name: curl
spec:
  template:
    spec:
      containers:
      - name: curl
        image: curlimages/curl:7.76.0
        command: ["curl", "--silent", "--show-error", "http://${service_endpoint}"]
      restartPolicy: Never
  backoffLimit: 4
EOF

kubectl apply -f curl.yaml

kubectl wait --for=condition=complete job curl

podname=$(kubectl get pods --selector=job-name=curl --field-
selector=status.phase=Succeeded -o jsonpath='{.items[0].metadata.name}')
```

```
kubectl logs ${podname}
```

The final command returns the default Nginx web page:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Use of kubeconfig file

This is primarily useful for running `kubectl` commands on the Management cluster to monitor the Attached or Managed cluster.

On the Management cluster, a `kubeconfig` file for the Attached or Managed cluster configured to use the tunnel proxy is available as a Secret. The Secret's name can be identified using:

```
kubeconfig_secret=$(kubectl get tunnelconnector -n ${namespace} ${connector} -o
jsonpath='{.status.kubeconfigRef.name}')
```

After setting `service_namespace` and `service_name` to the service resource, run this command on the Management cluster:

```
cat > get-service.yaml <<EOF
apiVersion: batch/v1
kind: Job
```

```
metadata:
  name: get-service
spec:
  template:
    spec:
      containers:
      - name: kubectl
        image: bitnami/kubectl:1.19
        command: ["kubectl", "get", "service", "-n", "${service_namespace}", "$
{service_name}"]
        env:
        - name: KUBECONFIG
          value: /tmp/kubeconfig/kubeconfig
        volumeMounts:
        - name: kubeconfig
          mountPath: /tmp/kubeconfig
      volumes:
      - name: kubeconfig
        secret:
          secretName: "${kubeconfig_secret}"
      restartPolicy: Never
  backoffLimit: 4
EOF

kubectl apply -n ${namespace} -f get-service.yaml

kubectl wait --for=condition=complete --timeout=5m job -n ${namespace} get-service

podname=$(kubectl get pods -n ${namespace} --selector=job-name=get-service --field-
selector=status.phase=Succeeded -o jsonpath='{.items[0].metadata.name}')

kubectl logs -n ${namespace} ${podname}
```

Direct use of SOCKS5 proxy

To use the SOCKS5 proxy directly, obtain the SOCKS5 proxy endpoint using:

```
proxy_service=$(kubectl get tunnelconnector -n ${namespace} ${connector} -o
jsonpath='{.status.tunnelServer.serviceRef.name}')

socks_proxy=$(kubectl get service -n ${namespace} "${proxy_service}" -o jsonpath='{.s
pec.clusterIP}{":"}{.spec.ports[?(@.name=="proxy")].port}')
```

Provide the value of `${socks_proxy}` as the SOCKS5 proxy to your client.

For example, since `curl` supports SOCKS5 proxies, the Attached or Managed service started above can be accessed from the Management cluster by adding the SOCKS5 proxy to the `curl` command. After setting `service_endpoint` to the service endpoint, on the Management cluster run:

```
cat > curl.yaml <<EOF
apiVersion: batch/v1
kind: Job
metadata:
  name: curl
spec:
  template:
    spec:
      containers:
      - name: curl
        image: curlimages/curl:7.76.0
        command: ["curl", "--silent", "--show-error", "--socks5-hostname", "$
{socks_proxy}", "http://${service_endpoint}"]
      restartPolicy: Never
  backoffLimit: 4
EOF

kubectl apply -f curl.yaml

kubectl wait --for=condition=complete --timeout=5m job curl

podname=$(kubectl get pods --selector=job-name=curl --field-
selector=status.phase=Succeeded -o jsonpath='{.items[0].metadata.name}')

kubectl logs ${podname}
```

The final command returns the same output as for the job on the Attached or Managed cluster, demonstrating that the job on the Management cluster accessed the service running on the Attached or Managed cluster.

Use of deployed proxy on Management cluster

To deploy a proxy on the Management cluster, obtain the SOCKS5 proxy endpoint using:

```
proxy_service=$(kubectl get tunnelconnector -n ${namespace} ${connector} -o
jsonpath='{.status.tunnelServer.serviceRef.name}')

socks_proxy=$(kubectl get service -n ${namespace} "${proxy_service}" -o jsonpath='{.s
pec.clusterIP}{":"}{.spec.ports[?(@.name=="proxy")].port}')
```

Provide the value of `${socks_proxy}` as the SOCKS5 proxy to a proxy deployed on the Management cluster. After setting `service_endpoint` to the service endpoint, on the Management cluster run:

```
cat > nginx-proxy.yaml <<EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: nginx-proxy-crt
```

```
spec:
  secretName: nginx-proxy-crt-secret
  dnsNames:
  - nginx-proxy-service.${namespace}.svc.cluster.local
  issuerRef:
    group: cert-manager.io
    kind: ClusterIssuer
    name: kubernetes-ca
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-proxy
  labels:
    app: nginx-proxy-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-proxy-app
  template:
    metadata:
      labels:
        app: nginx-proxy-app
    spec:
      containers:
      - name: nginx-proxy
        image: mesosphere/ghostunnel:v1.5.3-server-backend-proxy
        args:
        - "server"
        - "--listen=:443"
        - "--target=${service_endpoint}"
        - "--cert=/etc/certs/tls.crt"
        - "--key=/etc/certs/tls.key"
        - "--cacert=/etc/certs/ca.crt"
        - "--unsafe-target"
        - "--disable-authentication"
        env:
        - name: ALL_PROXY
          value: socks5://${socks_proxy}
        ports:
        - containerPort: 443
        volumeMounts:
        - name: certs
          mountPath: /etc/certs
      volumes:
      - name: certs
        secret:
          secretName: nginx-proxy-crt-secret
---
apiVersion: v1
kind: Service
```

```
metadata:
  name: nginx-proxy-service
spec:
  selector:
    app: nginx-proxy-app
  type: ClusterIP
  ports:
  - targetPort: 443
    port: 8765
EOF

kubectl apply -n ${namespace} -f nginx-proxy.yaml

kubectl rollout status deploy -n ${namespace} nginx-proxy

proxy_port=$(kubectl get service -n ${namespace} nginx-proxy-service -o
jsonpath='{.spec.ports[0].port}')
```

Any client running on the Management cluster can now access the service running on the Attached or Managed cluster using the proxy service endpoint. Note in the following that the `curl` job runs in the same namespace as the proxy, to provide access to the CA certificate secret.

```
cat > curl.yaml <<EOF
apiVersion: batch/v1
kind: Job
metadata:
  name: curl
spec:
  template:
    spec:
      containers:
      - name: curl
        image: curlimages/curl:7.76.0
        command:
        - curl
        - --silent
        - --show-error
        - --cacert
        - /etc/certs/ca.crt
        - https://nginx-proxy-service.${namespace}.svc.cluster.local:${proxy_port}
        volumeMounts:
        - name: certs
          mountPath: /etc/certs
      volumes:
      - name: certs
        secret:
          secretName: nginx-proxy-crt-secret
      restartPolicy: Never
  backoffLimit: 4
EOF
```

```
kubectl apply -n ${namespace} -f curl.yaml

kubectl wait --for=condition=complete --timeout=5m job -n ${namespace} curl

podname=$(kubectl get pods -n ${namespace} --selector=job-name=curl --field-
selector=status.phase=Succeeded -o jsonpath='{.items[0].metadata.name}')

kubectl logs -n ${namespace} ${podname}
```

The final command returns the same output as the job on the Attached or Managed cluster, demonstrating that the job on the Management cluster accessed the service running on the Attached or Managed cluster.

Related Topic:

For information on the TunnelGateway review the API documentation (v1alpha1) (see page 722).

6.6.6.5.6 Attach a Cluster Using a Tunnel via the CLI

Use the CLI to attach a Kubernetes Cluster with a Tunnel

Enterprise Gov Advanced

> Ensure you have reviewed and followed the steps in Prerequisites for a Tunneled Attachment (see page 695) before proceeding.

6.6.6.5.6.1 Attach a Cluster Using a Tunnel

Here is a high-level overview of the steps required to attach a cluster via the CLI:

- CLI: Prepare the Management Cluster (see page 707)
- CLI: Create and Configure the Tunnel (see page 710)
- CLI: Use the Remote Cluster (see page 715)

6.6.6.5.6.2 CLI: Prepare the Management Cluster

Enterprise Gov Advanced

Identify the Management Cluster Endpoint

Execute the following command on the Management cluster to obtain the hostname and CA certificate:

```
hostname=$(kubectl get service -n kommander kommander-traefik -o go-template='{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}')
b64ca_cert=$(kubectl get secret -n cert-manager kommander-ca -o=go-
template='{{index .data "tls.crt"}}')
```

Specify a Workspace Namespace

Obtain the desired workspace namespace on the Management cluster for the tunnel gateway:

```
namespace=$(kubectl get workspace default-workspace -o jsonpath="{.status.namespaceRe
f.name}")
```

Alternatively, you can create a new workspace instead of using an existing workspace:
Run the following command, and replace the `<workspace_name>` with the new workspace name:

```
workspace=<workspace_name>
```

Finish creating the workspace:

```
namespace=${workspace}

cat > workspace.yaml <<EOF
apiVersion: workspaces.kommander.mesosphere.io/v1alpha1
kind: Workspace
metadata:
  annotations:
    kommander.mesosphere.io/display-name: ${workspace}
  name: ${workspace}
spec:
  namespaceName: ${namespace}
EOF

kubectl apply -f workspace.yaml
```

You can verify the workspace exists using:

```
kubectl get workspace ${workspace}
```

Create a Tunnel Gateway

Create a tunnel gateway (see page 0) on the Management cluster to listen for tunnel agents on remote clusters:

> Kommander uses Traefik 2 ingress, which requires explicit definition of strip prefix middleware as a Kubernetes API object, opposed to a simple annotation. Kommander provides default middleware that supports creating tunnels only on the `/dkp/tunnel URL` prefix. This is indicated by using the extra annotation, `traefik.ingress.kubernetes.io/router.middlewares: kommander-stripprefixes-kubetunnel@kubernetescrd` as shown in the code sample that follows. If you want to expose a tunnel on a different URL prefix, you must manage your own middleware configuration.

Establish variables for the certificate secret and gateway. Replace the `<gateway_name>` placeholder with the name of the gateway:

```
cacert_secret=kubetunnel-ca
gateway=<gateway_name>
```

Create the `Secret` and `TunnelGateway` objects:

```
cat > gateway.yaml <<EOF
apiVersion: v1
kind: Secret
metadata:
  namespace: ${namespace}
  name: ${cacert_secret}
data:
  ca.crt:
    ${b64ca_cert}
---
apiVersion: kubetunnel.d2iq.io/v1alpha1
kind: TunnelGateway
metadata:
  namespace: ${namespace}
  name: ${gateway}
spec:
  ingress:
    caSecretRef:
      namespace: ${namespace}
      name: ${cacert_secret}
    loadBalancer:
      hostname: ${hostname}
```

```
    urlPathPrefix: /dkp/tunnel
    extraAnnotations:
      kubernetes.io/ingress.class: kommander-traefik
      traefik.ingress.kubernetes.io/router.tls: "true"
      traefik.ingress.kubernetes.io/router.middlewares: kommander-stripprefixes-
kubetunnel@kubernetescrd
EOF

kubectl apply -f gateway.yaml
```

You can verify the gateway exists using the command:

```
kubectl get tunnelgateway -n ${namespace} ${gateway}
```

Next Step:

CLI: Create and Configure the Tunnel (see page 710)

6.6.6.5.6.3 CLI: Create and Configure the Tunnel

Enterprise Gov Advanced

Connect a Remote Cluster

Create a tunnel connector

Create a tunnel connector (see page 726) on the **Management cluster** for the remote cluster.

1. Establish a variable for the connector. Provide the name of the connector, by replacing the `<connector_name>` placeholder:

   ```
   connector=<connector_name>
   ```

2. Create the `TunnelConnector` object:

   ```
   cat > connector.yaml <<EOF
   apiVersion: kubetunnel.d2iq.io/v1alpha1
   kind: TunnelConnector
   metadata:
     namespace: ${namespace}
     name: ${connector}
   ```

```
spec:
  gatewayRef:
    name: ${gateway}
EOF

kubectl apply -f connector.yaml
```

After you create the `TunnelConnector` object, DKP creates a `manifest.yaml`. This `manifest.yaml` contains the configuration information for the components required by the tunnel for a specific cluster.

3. Verify the connector exists:

```
kubectl get tunnelconnector -n ${namespace} ${connector}
```

4. Wait for the tunnel connector to reach `Listening` state and then export the agent manifest:

```
while [ "$(kubectl get tunnelconnector -n ${namespace} ${connector} -o
jsonpath="{.status.state}")" != "Listening" ]
do
  sleep 5
done

manifest=$(kubectl get tunnelconnector -n ${namespace} ${connector} -o
jsonpath="{.status.tunnelAgent.manifestsRef.name}")
while [ -z ${manifest} ]
do
  sleep 5
  manifest=$(kubectl get tunnelconnector -n ${namespace} ${connector} -o
jsonpath="{.status.tunnelAgent.manifestsRef.name}")
done

kubectl get secret -n ${namespace} ${manifest} -o
jsonpath='{.data.manifests\.yaml}' | base64 -d > manifest.yaml
```

The `manifest.yaml` is applied successfully after the command completes.

5. Fetch the `manifest.yaml` to use it in the following section:

```
kubectl get secret -n ${namespace} ${manifest} -o
jsonpath='{.data.manifests\.yaml}' | base64 -d > manifest.yaml
```

When attaching several clusters, ensure that you fetch the `manifest.yaml` of the cluster you are attempting to attach. Using the wrong combination of `manifest.yaml` and cluster will cause the attachment to fail.

Set up the managed cluster

In the following commands, the `--kubeconfig` flag ensures that you set the context to the Attached or Managed cluster. For alternatives and recommendations around setting your context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

1. Apply the `manifest.yaml` file to the **Attached or Managed cluster** and deploy the tunnel agent:

   ```
   kubectl apply --kubeconfig=<managed_cluster_kubeconfig.conf> -f manifest.yaml
   ```

2. Check the status of the created pods:

   ```
   kubectl get pods --kubeconfig=<managed_cluster_kubeconfig.conf> -n kubetunnel
   ```

   After a short time, expect to see a `post-kubeconfig` pod that reaches `Completed` state and a `tunnel-agent` pod that stays in `Running` state.

   ```
   NAME                            READY   STATUS      RESTARTS   AGE
   post-kubeconfig-j2ghk           0/1     Completed   0          14m
   tunnel-agent-f8d9f4cb4-thx8h    0/1     Running     0          14m
   ```

Add the Attached or Managed cluster into Kommander

When you create a cluster using the DKP CLI, it does not attach automatically.

1. On the Management cluster, wait for the tunnel to be connected by the tunnel agent:

   ```
   while [ "$(kubectl get tunnelconnector -n ${namespace} ${connector} -o
   jsonpath="{.status.state}")" != "Connected" ]
   do
     sleep 5
   ```

```
done
```

2. Establish variables for the managed cluster. Replace the `<private_cluster>` placeholder with the name of the managed cluster:

```
managed=<private-cluster>
display_name=${managed}
```

3. Update the `KommanderCluster` object:

```
cat > kommander.yaml <<EOF
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  namespace: ${namespace}
  name: ${managed}
  annotations:
    kommander.mesosphere.io/display-name: ${display_name}
spec:
  clusterTunnelConnectorRef:
    name: ${connector}
EOF

kubectl apply -f kommander.yaml
```

4. Wait for the Attached or Managed cluster to join:

```
while [ "$(kubectl get kommandercluster -n ${namespace} ${managed} -o
jsonpath='{.status.phase}')" != "Joined" ]
do
  sleep 5
done

kubefed=$(kubectl get kommandercluster -n ${namespace} ${managed} -o
jsonpath="{.status.kubefedclusterRef.name}")
while [ -z "${kubefed}" ]
do
  sleep 5
  kubefed=$(kubectl get kommandercluster -n ${namespace} ${managed} -o
jsonpath="{.status.kubefedclusterRef.name}")
done

kubectl wait --for=condition=ready --timeout=60s kubefedcluster -n kube-
federation-system ${kubefed}

kubectl get kubefedcluster -n kube-federation-system ${kubefed}
```

After the command completes, your cluster becomes visible in the DKP UI and you can start using it. Its metrics will be accessible through different dashboards such as Grafana, Karma, etc.

Create a network policy for the tunnel server

This step is optional, but improves **security** by restricting which remote hosts can connect to the tunnel.

1. Apply a network policy that restricts tunnel access to specific namespaces and IP blocks.
   The following example permits connections from:
   - Pods running in the `kommander` and `kube-federation-system` namespace.
   - Remote clusters with IP addresses in the ranges 192.0.2.0 to 192.0.2.255 and 203.0.113.0 to 203.0.113.255.
   - Pods running in namespaces with a label `kubetunnel.d2iq.io/networkpolicy` that match the tunnel name and namespace.

```
cat > net.yaml <<EOF
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  namespace: ${namespace}
  name: ${connector}-deny
  labels:
    kubetunnel.d2iq.io/tunnel-connector: ${connector}
    kubetunnel.d2iq.io/networkpolicy-type: "tunnel-server"
spec:
  podSelector:
    matchLabels:
      kubetunnel.d2iq.io/tunnel-connector: ${connector}
  policyTypes:
  - Ingress
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  namespace: ${namespace}
  name: ${connector}-allow
  labels:
    kubetunnel.d2iq.io/tunnel-connector: ${connector}
    kubetunnel.d2iq.io/networkpolicy-type: "tunnel-server"
spec:
  podSelector:
    matchLabels:
      kubetunnel.d2iq.io/tunnel-connector: ${connector}
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
```

```
        matchLabels:
          kubernetes.io/metadata.name: "kube-federation-system"
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: "kommander"
    - namespaceSelector:
        matchLabels:
          kubetunnel.d2iq.io/networkpolicy: ${connector}-${namespace}
    - ipBlock:
        cidr: 192.0.2.0/24
    - ipBlock:
        cidr: 203.0.113.0/24
EOF

kubectl apply -f net.yaml
```

2. To enable applications running in another namespace to access the attached cluster, add the label `kubetunnel.d2iq.io/networkpolicy=${connector}-${namespace}` to the target namespace:

   ```
   kubectl label ns ${namespace} kubetunnel.d2iq.io/networkpolicy=${connector}-$
   {namespace}
   ```

   All pods in the target namespace can now reach the attached cluster services.

Next Step:

Use the Remote Cluster (see page 700)

6.6.6.5.6.4 CLI: Use the Remote Cluster

Enterprise Gov Advanced

To access services running on the remote cluster from the Management cluster, connect to the tunnel proxy. You can use these three methods:

1. If the client program supports use of a kubeconfig file, use the Attached or Managed cluster's kubeconfig.
2. If the client program supports SOCKS5 proxies, use the proxy directly.
3. Otherwise, deploy a proxy server on the Management cluster.

Attached or Managed Cluster Service

These sections require a service to run on the Attached or Managed cluster.

As an example, start the following service:

```
service_namespace=test
service_name=webserver
service_port=8888
service_endpoint=${service_name}.${service_namespace}.svc.cluster.local:$
{service_port}

cat > nginx.yaml <<EOF
apiVersion: v1
kind: Namespace
metadata:
  name: ${service_namespace}
---
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: ${service_namespace}
  name: nginx-deployment
  labels:
    app: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx-app
  template:
    metadata:
      labels:
        app: nginx-app
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  namespace: ${service_namespace}
  name: ${service_name}
spec:
  selector:
    app: nginx-app
  type: ClusterIP
  ports:
  - targetPort: 80
    port: ${service_port}
EOF
```

```
kubectl apply -f nginx.yaml

kubectl rollout status deploy -n ${service_namespace} nginx-deployment
```

On the Attached or Managed cluster, a client Job can access this service using:

```
cat > curl.yaml <<EOF
apiVersion: batch/v1
kind: Job
metadata:
  name: curl
spec:
  template:
    spec:
      containers:
      - name: curl
        image: curlimages/curl:7.76.0
        command: ["curl", "--silent", "--show-error", "http://${service_endpoint}"]
      restartPolicy: Never
  backoffLimit: 4
EOF

kubectl apply -f curl.yaml

kubectl wait --for=condition=complete job curl

podname=$(kubectl get pods --selector=job-name=curl --field-
selector=status.phase=Succeeded -o jsonpath='{.items[0].metadata.name}')

kubectl logs ${podname}
```

The final command returns the default Nginx web page:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
```

```
<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Use of kubeconfig file

This is primarily useful for running `kubectl` commands on the Management cluster to monitor the Attached or Managed cluster.

On the Management cluster, a `kubeconfig` file for the Attached or Managed cluster configured to use the tunnel proxy is available as a Secret. The Secret's name can be identified using:

```
kubeconfig_secret=$(kubectl get tunnelconnector -n ${namespace} ${connector} -o
jsonpath='{.status.kubeconfigRef.name}')
```

After setting `service_namespace` and `service_name` to the service resource, run this command on the Management cluster:

```
cat > get-service.yaml <<EOF
apiVersion: batch/v1
kind: Job
metadata:
  name: get-service
spec:
  template:
    spec:
      containers:
      - name: kubectl
        image: bitnami/kubectl:1.19
        command: ["kubectl", "get", "service", "-n", "${service_namespace}", "$
{service_name}"]
        env:
        - name: KUBECONFIG
          value: /tmp/kubeconfig/kubeconfig
        volumeMounts:
        - name: kubeconfig
          mountPath: /tmp/kubeconfig
      volumes:
      - name: kubeconfig
        secret:
          secretName: "${kubeconfig_secret}"
      restartPolicy: Never
  backoffLimit: 4
```

```
EOF

kubectl apply -n ${namespace} -f get-service.yaml

kubectl wait --for=condition=complete --timeout=5m job -n ${namespace} get-service

podname=$(kubectl get pods -n ${namespace} --selector=job-name=get-service --field-
selector=status.phase=Succeeded -o jsonpath='{.items[0].metadata.name}')

kubectl logs -n ${namespace} ${podname}
```

Direct use of SOCKS5 proxy

To use the SOCKS5 proxy directly, obtain the SOCKS5 proxy endpoint using:

```
proxy_service=$(kubectl get tunnelconnector -n ${namespace} ${connector} -o
jsonpath='{.status.tunnelServer.serviceRef.name}')

socks_proxy=$(kubectl get service -n ${namespace} "${proxy_service}" -o jsonpath='{.s
pec.clusterIP}{":"}{.spec.ports[?(@.name=="proxy")].port}')
```

Provide the value of `${socks_proxy}` as the SOCKS5 proxy to your client.

For example, since `curl` supports SOCKS5 proxies, the Attached or Managed service started above can be accessed from the Management cluster by adding the SOCKS5 proxy to the `curl` command. After setting `service_endpoint` to the service endpoint, on the Management cluster run:

```
cat > curl.yaml <<EOF
apiVersion: batch/v1
kind: Job
metadata:
  name: curl
spec:
  template:
    spec:
      containers:
      - name: curl
        image: curlimages/curl:7.76.0
        command: ["curl", "--silent", "--show-error", "--socks5-hostname", "$
{socks_proxy}", "http://${service_endpoint}"]
      restartPolicy: Never
  backoffLimit: 4
EOF

kubectl apply -f curl.yaml

kubectl wait --for=condition=complete --timeout=5m job curl
```

```
podname=$(kubectl get pods --selector=job-name=curl --field-
selector=status.phase=Succeeded -o jsonpath='{.items[0].metadata.name}')

kubectl logs ${podname}
```

The final command returns the same output as for the job on the Attached or Managed cluster, demonstrating that the job on the Management cluster accessed the service running on the Attached or Managed cluster.

Use of deployed proxy on Management cluster

To deploy a proxy on the Management cluster, obtain the SOCKS5 proxy endpoint using:

```
proxy_service=$(kubectl get tunnelconnector -n ${namespace} ${connector} -o
jsonpath='{.status.tunnelServer.serviceRef.name}')

socks_proxy=$(kubectl get service -n ${namespace} "${proxy_service}" -o jsonpath='{.s
pec.clusterIP}{":"}{.spec.ports[?(@.name=="proxy")].port}')
```

Provide the value of `${socks_proxy}` as the SOCKS5 proxy to a proxy deployed on the Management cluster. After setting `service_endpoint` to the service endpoint, on the Management cluster run:

```
cat > nginx-proxy.yaml <<EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: nginx-proxy-crt
spec:
  secretName: nginx-proxy-crt-secret
  dnsNames:
  - nginx-proxy-service.${namespace}.svc.cluster.local
  issuerRef:
    group: cert-manager.io
    kind: ClusterIssuer
    name: kubernetes-ca
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-proxy
  labels:
    app: nginx-proxy-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-proxy-app
  template:
```

```
    metadata:
      labels:
        app: nginx-proxy-app
    spec:
      containers:
      - name: nginx-proxy
        image: mesosphere/ghostunnel:v1.5.3-server-backend-proxy
        args:
        - "server"
        - "--listen=:443"
        - "--target=${service_endpoint}"
        - "--cert=/etc/certs/tls.crt"
        - "--key=/etc/certs/tls.key"
        - "--cacert=/etc/certs/ca.crt"
        - "--unsafe-target"
        - "--disable-authentication"
        env:
        - name: ALL_PROXY
          value: socks5://${socks_proxy}
        ports:
        - containerPort: 443
        volumeMounts:
        - name: certs
          mountPath: /etc/certs
      volumes:
      - name: certs
        secret:
          secretName: nginx-proxy-crt-secret
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-proxy-service
spec:
  selector:
    app: nginx-proxy-app
  type: ClusterIP
  ports:
  - targetPort: 443
    port: 8765
EOF

kubectl apply -n ${namespace} -f nginx-proxy.yaml

kubectl rollout status deploy -n ${namespace} nginx-proxy

proxy_port=$(kubectl get service -n ${namespace} nginx-proxy-service -o
jsonpath='{.spec.ports[0].port}')
```

Any client running on the Management cluster can now access the service running on the Attached or Managed cluster using the proxy service endpoint. Note in the following that the `curl` job runs in the same namespace as the proxy, to provide access to the CA certificate secret.

```
cat > curl.yaml <<EOF
apiVersion: batch/v1
kind: Job
metadata:
  name: curl
spec:
  template:
    spec:
      containers:
      - name: curl
        image: curlimages/curl:7.76.0
        command:
        - curl
        - --silent
        - --show-error
        - --cacert
        - /etc/certs/ca.crt
        - https://nginx-proxy-service.${namespace}.svc.cluster.local:${proxy_port}
        volumeMounts:
        - name: certs
          mountPath: /etc/certs
      volumes:
      - name: certs
        secret:
          secretName: nginx-proxy-crt-secret
      restartPolicy: Never
  backoffLimit: 4
EOF

kubectl apply -n ${namespace} -f curl.yaml

kubectl wait --for=condition=complete --timeout=5m job -n ${namespace} curl

podname=$(kubectl get pods -n ${namespace} --selector=job-name=curl --field-
selector=status.phase=Succeeded -o jsonpath='{.items[0].metadata.name}')

kubectl logs -n ${namespace} ${podname}
```

The final command returns the same output as the job on the Attached or Managed cluster, demonstrating that the job on the Management cluster accessed the service running on the Attached or Managed cluster.

Related Topic:

For information on the TunnelGateway review the API documentation (v1alpha1) (see page 722).

6.6.6.5.7 API documentation (v1alpha1)

API Documentation (v1alpha1)

> *This document is automatically generated from the API definition in the code.*

Page Contents

6.6.6.5.7.1 TunnelGateway

Provides an endpoint for remote clusters to connect to the management cluster.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[491]	false
spec		TunnelGatewaySpec (see page 725)	false

6.6.6.5.7.2 TunnelGatewayIngressSpec

Field	Description	Scheme	Required
loadBalancer	Ingress point for the load-balancer. Traffic intended for the service should be sent to these ingress points. If not specified, the controller will derive from the Ingress record status field.	corev1.LoadBalancerIngress	false
host	Restrict access to requests addressed to a specific host or domain using the `IngressRule` format. Defaults to allow all hosts.	string	false

491 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta

Field	Description	Scheme	Required
urlPathPrefix	URL path prefix to prepend to all endpoints. For example, if this field is set to `/ops/portal/kt`, the ingresses created will have URL paths like `/ops/portal/kt/default/cluster1/tunnel-server` and `/ops/portal/kt/default/cluster1/kubeconfig`. Defaults to root path (`/`).	string	false
caSecretRef	A secret reference to the root CA required to verify the ingress endpoints. The secret should have type `Opaque` and contain the key `ca.crt`. If not specified, remote hosts will use their system root CA's to verify the endpoints.	corev1.ObjectReference	false
extraAnnotations	Extra annotations to set on the Ingress object.	map[string]string	false

6.6.6.5.7.3 TunnelGatewayList

Contains a list of `TunnelGateway`.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[492]	false
items		[]TunnelGateway[493]	true

6.6.6.5.7.4 TunnelGatewaySpec

If no ingress is set, the services will only be accessible on `localhost`.

Field	Description	Scheme	Required
ingress	Expose services using an Ingress as specified in the `TunnelGatewayIngressSpec`.	TunnelGatewayIngressSpec (see page 723)	false

6.6.6.5.7.5 KubeconfigWebhookStatus

Status of the kubeconfig webhook.

Field	Description	Scheme	Required
deploymentRef	A reference to the deployment for the kubeconfig webhook.	corev1.LocalObjectReference	false
serviceRef	A reference to the service for the kubeconfig webhook.	corev1.LocalObjectReference	false
ingressRef	A reference to the ingress for the kubeconfig webhook.	corev1.LocalObjectReference	false

492 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
493 https://docs.d2iq.com/dkp/kommander/2.2/clusters/tunnel-cli/api-reference/#TunnelGateway

6.6.6.5.7.6 TunnelAgentStatus

Status of the tunnel agent.

Field	Description	Scheme	Required
manifestsRef	A reference to a secret holding YAML manifests for launching the tunnel agent on the target cluster. The secret is a generic typed secret with filenames as the keys. There might be multiple files in the secret.	corev1.LocalObjectReference	false

6.6.6.5.7.7 TunnelConnector

Describes the local endpoint for the tunnel. A remote cluster will connect to this endpoint to create a tunnel.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[494]	false
spec		TunnelConnectorSpec (see page 727)	false
status		TunnelConnectorStatus (see page 728)	false

6.6.6.5.7.8 TunnelConnectorList

Contains a list of `TunnelConnector`.

494 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta

Field	Description	Scheme	Required
metadata		metav1.ListMeta[495]	false
items		[]TunnelConnector (see page 726)	true

6.6.6.5.7.9 TunnelConnectorSpec

Field	Description	Scheme	Required
gatewayRef	A reference to the `TunnelGateway` object which describes how tunnel services will be exposed outside the current cluster.	corev1.LocalObjectReference	false
proxyPort	The port for the tunnel proxy.	int32	false

495 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta

6.6.6.5.7.10 TunnelConnectorStatus

Field	Description	Scheme	Required
state	State of the tunnel connector: `Starting` - the initial state; `Listening` - the local tunnel server is waiting for the remote agent to connect; `Pending` - the remote agent has connected but the local proxy is not ready; `Connected` - the tunnel is configured and contact to the remote API server succeeded; `Disconnected` - the tunnel is configured but contact to the remote API server failed; `Failed` - an unexpected error occurred, such as not being able to parse the kubeconfig.	TunnelConnectorState	false
tunnelServer	Status of the tunnel server.	TunnelServerStatus (see page 729)	false
kubeconfigWebhook	Status of the kubeconfig webhook.	KubeconfigWebhookStatus (see page 725)	false
tunnelAgent	Status of the tunnel agent.	TunnelAgentStatus (see page 726)	false
serviceAccountRef	A reference to the service account that will be used for registration (of the tunnel agent) and authentication purpose.	corev1.LocalObjectReference	false

Field	Description	Scheme	Required
roleRef	A reference to the role that will be bound to the service account for authorization purpose.	corev1.LocalObjectReference	false
roleBindingRef	A reference to the rolebinding that will be created to bind the service account and the role.	corev1.LocalObjectReference	false
kubeconfigRef	A reference to the secret holding the KUBECONFIG that the clients can use to talk to the API server of the target cluster when it becomes available.	corev1.LocalObjectReference	false
gatewayObservedGeneration	The generation of the linked TunnelGateway object associated with this object. When the linked TunnelGateway object is updated, a controller will update this status field which will in turn trigger a reconciliation of this object.	int64	false

6.6.6.5.7.11 TunnelServerStatus

Status of the tunnel server.

Field	Description	Scheme	Required
deploymentRef	A reference to the deployment for the tunnel server.	corev1.LocalObjectReference	false

Field	Description	Scheme	Required
serviceRef	A reference to the service for the tunnel server.	corev1.LocalObjectReference	false
ingressRef	A reference to the ingress for the tunnel server.	corev1.LocalObjectReference	false

6.6.6.6 Attach a DKP-created Kubernetes Cluster

This section contains the following topics:

- Make a DKP-CLI-created Cluster Managed (see page 730)

When you create a Managed Cluster (see page 97) with the DKP CLI, it does not attach automatically. After creating it, Make a DKP-CLI-created Cluster Managed (see page 730) by attaching it to the Management Cluster (see page 97).

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Turn an Essential Cluster into a Managed Cluster (see page 732).

6.6.6.6.1 Make a DKP-CLI-created Cluster Managed

Enterprise Gov Advanced

Make a DKP CLI cluster managed

When you create a cluster in a Workspace namespace using the DKP CLI, it does not attach automatically.

You will be able to see the new cluster in the UI while it is being provisioned. Once provisioning is completed, the status changes to **Unattached**, and you will be able to attach it as shown in the following section.

6.6.6.6.1.1 Manually Attach a DKP CLI Cluster to the Management Cluster

1. Find out the `name` of the created `Cluster`, so you can reference it later:

   ```
   kubectl -n <workspace_namespace> get clusters
   ```

2. Attach the cluster by creating a `KommanderCluster`:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: <cluster_name>
  namespace: <workspace_namespace>
spec:
  kubeconfigRef:
    name: <cluster_name>-kubeconfig
  clusterRef:
    capiCluster:
      name: <cluster_name>
EOF
```

To avoid manually attaching your cluster, you can set it up to be automatically attached by generating the cluster objects using the DKP CLI `--dry-run -o yaml` flags and creating a cluster as stated in the Advanced Creation of CLI Clusters (see page 760) guide.

6.6.6.7 Access a Managed or Attached Cluster

Enterprise Gov Advanced

How to access a Managed or Attached cluster with credentials

6.6.6.7.1 Access your Clusters Using your UI Administrator Credentials

After the cluster is attached, retrieve a custom `kubeconfig` file from the UI.

1. Select the username in the top right corner, and then select **Generate Token**.
2. Select the cluster name, and follow the instructions to assemble a `kubeconfig` for accessing its Kubernetes API.

If the UI prompts you to log in, use the credentials you normally use to access the UI.

You can also retrieve a custom `kubeconfig` file by visiting the `/token` endpoint on the Kommander cluster domain (example URL: `https://your-server-name.your-region-2.elb.service.com/token/`. Selecting the cluster's name displays the instructions to assemble a `kubeconfig` for accessing its Kubernetes API.

6.6.6.8 Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster

Enterprise Gov Advanced

If you are a DKP Essential customer, you can easily convert your independent clusters into a multi-cluster environment if you upgrade to an Enterprise license.

The pages in this section provides information on how you can turn your Essential Clusters into Managed clusters.

For all the instructions regarding converting an existing Essential cluster into an Enterprise cluster, refer to that entire section of documentation (see page 732).

6.6.7 Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster

Enterprise Gov Advanced

If you are a DKP Essential customer, you can easily convert your independent clusters into a multi-cluster environment if you upgrade to an Enterprise license.

The pages in this section provides information on how you can turn your Essential Clusters into Managed clusters.

6.6.7.1 Prerequisites: General Prerequisites for your Cluster Conversion

Information that you need to know prior to converting your DKP Essential Clusters into DKP Enterprise Managed Clusters.

Enterprise Gov Advanced

Starting with DKP 2.5, you have the option to convert your DKP Essential Clusters into DKP Managed Enterprise Clusters.

6.6.7.1.1 Requirements

To convert your DKP Essential clusters into DKP Managed Enterprise clusters, ensure the following is valid:

- A DKP Management cluster with a valid DKP Enterprise License installed
- At least one installed and running standalone DKP Essential cluster
- All DKP Essential Clusters are upgraded (see page 1365) to the same DKP version as the DKP Managed Enterprise cluster
- The DKP Essential Cluster you want to convert is self-managed (see page 98)
- The DKP Essential Cluster you want to convert only contains its own Cluster API resources and does not contain Cluster API resources from other clusters

See the Licenses (see page 72) page for information on how you can purchase a DKP Enterprise license.

Attaching DKP Enterprise clusters is not supported.

6.6.7.1.2 Downtime Considerations

Your DKP Essential cluster will not be accessible externally for several minutes during the expansion process. Any configuration of the cluster's `Ingress` that requires `traefik-forward-auth` authentication will be affected.

Access from within the cluster via Kubernetes service hostname (for example, http://SERVICE_NAME.NAMESPACE:PORT) is not affected.

6.6.7.1.2.1 Affected DKP Services

- `dkp-ceph-cluster-dashboard`
- `grafana-logging`
- `kube-prometheus-stack-alertmanager`
- `kube-prometheus-stack-grafana`
- `kube-prometheus-stack-prometheus`

- `kubernetes-dashboard`
- `traefik-dashboard`

6.6.7.1.2.2 Other Services

To verify if your services are affected by `traefik-forward-auth`'s downtime, run the following command:

```
kubectl get ingress -n NS <your_customized_ingress_name>
```

Look for the `traefik.ingress.kubernetes.io/router.middlewares` field in the output. If this field contains the value `kommander-forwardauth@kubernetescrd`, your service will be affected by the downtime.

6.6.7.1.2.3 Duration

The `traefik-forward-auth` service is affected starting with the `PreAttachmentCleanup` conversion stage, and will run normally again after `ResumeFluxOperations` is completed. Observe the conversion progress (see page 0) to monitor your cluster's current status.

6.6.7.1.3 Special Considerations for DKP Insights

If you are a participant in the Technical Preview of DKP Insights, you must uninstall Insights prior to converting your DKP Essential clusters into Managed Clusters.

Following conversion, you can re-install Insights.
See the following page for information about installing and uninstalling Insights in your environment: DKP Insights Setup and Configuration[496]

6.6.7.1.4 Next Step

https://d2iq.atlassian.net/wiki/spaces/DENT/pages/213125219/2.5%2BPrerequisites%2BPrepare%2Byour%2BCluster%2Bs%2BExisting%2BConfigurations (see page 734)

6.6.7.1.5 See Also

Troubleshooting (see page 754)

6.6.7.2 Prerequisites: Prepare your Cluster's Existing Configurations

Enterprise Gov Advanced

496 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/62425108/DKP+Insights+Setup+and+Configuration

Before you convert an *Essential* cluster into a *Managed* cluster, review the following information.

6.6.7.2.1 SSO Configuration

After attachment, the SSO configuration of the Management cluster applies to the Managed (formerly Essential) cluster. Any SSO configuration of the former Essential cluster will be deleted.

- If the **Essential cluster has SSO configured**, but the Management cluster has NO configuration, you can copy your Essential cluster's SSO configuration (`dex-controller` resources) to the Management cluster before conversion.
- If your **Management cluster has SSO configured** and the **Essential cluster has another SSO configuration**, you can choose to keep one, or both. To keep the configuration of your Essential cluster, manually copy the `dex-controller` resources to the Management cluster before conversion. DKP maintains the SSO configuration of your Management cluster automatically, unless you manually delete it.

6.6.7.2.2 Domains and Certificates

Any custom domain or certificate configuration you have set up for your Essential cluster remains functional after you turn it into a Managed cluster.

> ⚠ After conversion, any domain or certificate customizations you would like to apply to your Managed cluster must be done via the `KommanderCluster`. This object is now stored in the Management cluster.

6.6.7.2.3 Next Step

Prerequisites: Clone Git Repository from Gitea (see page 735)

6.6.7.3 Prerequisites: Clone Git Repository from Gitea

Enterprise Gov Advanced

After your DKP Essential Cluster is converted into a DKP Enterprise Managed cluster, the old instance of Gitea that is used to host all Git repositories in the DKP Essential Cluster, will not be preserved.

Perform the steps in this page to ensure that you have a local copy of the Management Git Repository in the state it was in prior to undergoing the expansion process.

All DKP Platform Applications will be migrated from the DKP Essential Cluster to the DKP Enterprise Managed Cluster.

1. Prior to turning an DKP Essential cluster to a DKP Enterprise Managed Cluster, clone Gitea using the following command:

```
dkp experimental gitea clone
```

2. Verify that the Git Repository has been successfully cloned to your local environment.

```
cd kommander
git remote -v
# output from git remote -v look like
# origin  https://<YOUR_CLUSTER_INGRESS_HOSTNAME>/dkp/kommander/git/kommander/
kommander (fetch)
# origin  https://<YOUR_CLUSTER_INGRESS_HOSTNAME>/dkp/kommander/git/kommander/
kommander (push)
```

6.6.7.3.1 Next Steps

Back up your Cluster's Applications and Persistent Volumes (see page 736)

6.6.7.4 Back up your Cluster's Applications and Persistent Volumes

Enterprise Gov Advanced

Back up and restore your cluster’s applications before attempting converting your DKP Essential cluster into a DKP Enterprise Managed cluster in the scope of Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 732).

The instructions differ depending on the infrastructure provider of your **DKP Essential cluster**.

For AWS, refer to: Back up an AWS Cluster (see page 736)

For Azure, vSphere, GCP and pre-provisioned environments, refer to: Back up an Azure, vSphere, GCP or Pre-provisioned Cluster (see page 741)

6.6.7.4.1 Back up an AWS Cluster

This section contains the instructions necessary to back up and restore a DKP Essential cluster on the AWS environment in the scope of Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 732).

Here is a summary of the steps:

- Prepare your Cluster for Backup - AWS (see page 737)

- Back up a Cluster - AWS Environment (see page 739)

6.6.7.4.1.1 Prepare your Cluster for Backup - AWS

This section describes how to prepare your cluster on an AWS environment, so it can be backed up before you begin with Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 732).

Prerequisites

- Ensure Velero is installed on your Essential cluster
- Install the Velero CLI (see page 785) (Use at least Velero CLI version 1.10.1)
- Ensure kubectl[497] is installed
- Ensure you have admin rights to the DKP Essential cluster

Prepare your Cluster

Run the following commands in the ***DKP Essential*** **cluster**. For general guidelines on how to set the context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

Prepare Velero

Enable the CSI snapshotting plug-in by providing a custom configuration of Velero.

1. Create an Override with the custom configuration:

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: velero-overrides
  namespace: kommander
data:
  values.yaml: |
    ---
    configuration:
      features: EnableCSI
    initContainers:
      - name: velero-plugin-for-aws
        image: velero/velero-plugin-for-aws:v1.5.2
        imagePullPolicy: IfNotPresent
```

497 https://kubernetes.io/docs/tasks/tools/

```
          volumeMounts:
            - mountPath: /target
              name: plugins
        - name: velero-plugin-for-csi
          image: velero/velero-plugin-for-csi:v0.4.2
          imagePullPolicy: IfNotPresent
          volumeMounts:
            - mountPath: /target
              name: plugins
EOF
```

2. Update the `AppDeployment` to apply the new configuration:

```
cat << EOF | kubectl -n kommander patch appdeployment velero --type='merge' --
patch-file=/dev/stdin
spec:
  configOverrides:
    name: velero-overrides
EOF
```

3. Verify the configuration has been updated before proceeding with the next section:

```
kubectl -n kommander wait --for=condition=Ready kustomization velero
```

The output should look similar to this:

```
kustomization.kustomize.toolkit.fluxcd.io/velero condition met
```

Prepare the AWS IAM Permission

When creating a cluster on AWS, you provided an additional permission as specified in https://docs.d2iq.com/dkp/2.4/iam-artifacts.

For the CSI plugin to function correctly, you must update the existing IAM role to include an additional policy.

Add the AmazonEBSCSIDriverPolicy[498] policy to the control plane role `control-plane.cluster-api-provider-aws.sigs.k8s.io`:

```
aws iam attach-role-policy \
  --role-name control-plane.cluster-api-provider-aws.sigs.k8s.io \
  --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy
```

This will allow the EBS CSI driver, a volume manager, to have enough permissions to create volume snapshots.

498 https://docs.aws.amazon.com/aws-managed-policy/latest/reference/AmazonEBSCSIDriverPolicy.html

The default control plane role name is `control-plane.cluster-api-provider-aws.sigs.k8s.io`. If you customized this name when creating the AWS cluster, replace the default control plane role with the name you assigned to it.

Prepare the CSI Configuration

Configure a `VolumeSnapshotClass` object on the cluster, so Velero can create a volume snapshot:

```
cat << EOF | kubectl apply -f -
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: aws
  labels:
    velero.io/csi-volumesnapshot-class: "true"
driver: ebs.csi.aws.com
deletionPolicy: Delete
parameters:
EOF
```

Next Step:

Back up a Cluster - AWS Environment (see page 739)

6.6.7.4.1.2 Back up a Cluster - AWS Environment

With this workflow, you can back up and restore your cluster's applications. The backup contains all Kubernetes objects and the Persistent Volumes of your DKP applications.

When backing up a cluster that runs on a cloud provider, Velero captures the state of your cluster in a snapshot. Review Velero's list of cloud providers for CSI[499] compatibility.

Run the following commands in the ***DKP Essential* cluster**. For general guidelines on how to set the context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

Back up Instructions

1. Configure Velero to use CSI snapshotting:

499 https://velero.io/docs/v1.10/supported-providers/

```
velero client config set features=EnableCSI
```

2. Create a backup with Velero. Use the following flags to reduce the scope of the backup and only include the applications that are affected during the expansion:

```
velero backup create pre-expansion \
  --include-namespaces="kommander,kommander-default-workspace,kommander-
flux,kubecost" \
  --include-cluster-resources \
  --wait
```

After completion, the output should look similar to this:

```
Backup request "pre-expansion" submitted successfully.
Waiting for backup to complete. You may safely press ctrl-c to stop waiting -
your backup will continue in the background.
.....................................................................................
.....................................................................................
.....................................................................................
.................
Backup completed with status: Completed. You may check for more information
using the commands `velero backup describe pre-expansion` and `velero backup
logs pre-expansion`.
```

Verify the Backup

Review the backup has completed successfully:

```
velero backup describe pre-expansion
```

The following example output will vary depending on your cloud provider. Verify that it shows no errors and the `Phase` is `Completed`:

```
Name:         pre-expansion
Namespace:    kommander
Labels:       velero.io/storage-location=default
Annotations:  velero.io/source-cluster-k8s-gitversion=v1.25.5
              velero.io/source-cluster-k8s-major-version=1
              velero.io/source-cluster-k8s-minor-version=25

Phase:  Completed

Errors:    0
Warnings:  0
```

```
Namespaces:
  Included:  kommander, kommander-default-workspace, kommander-flux, kubecost
  Excluded:  <none>

Resources:
  Included:        *
  Excluded:        <none>
  Cluster-scoped:  included

Label selector:  <none>

Storage Location:  default

Velero-Native Snapshot PVs:  auto

TTL:  720h0m0s

CSISnapshotTimeout:  10m0s

Hooks:  <none>

Backup Format Version:  1.1.0

Started:    2023-03-15 10:40:25 -0400 EDT
Completed:  2023-03-15 10:44:39 -0400 EDT

Expiration:  2023-04-14 10:40:24 -0400 EDT

Total items to be backed up:  5188
Items backed up:              5188

Velero-Native Snapshots: <none included>
```

Next Step

Expand your platform. Use the UI or CLI:

UI: Convert an Essential Cluster into a Managed Cluster (see page 746)

CLI: Convert an Essential Cluster into a Managed Cluster (see page 748)

Related Topics

Restore your Backup and Retry the Cluster Expansion (see page 757)

6.6.7.4.2 Back up an Azure, vSphere, GCP or Pre-provisioned Cluster

This section contains the instructions necessary to back up and restore a DKP Essential cluster on Azure, vSphere, GCP or Pre-provisioned environments in the scope of Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 732).

Here is a summary of the steps:

- Prepare your Cluster for Backup - Azure, vSphere, GCP, Pre-provisioned Environments (see page 742)
- Back up a Cluster - Azure, vSphere, GCP, and Pre-provisioned Environments (see page 744)

6.6.7.4.2.1 Prepare your Cluster for Backup - Azure, vSphere, GCP, Pre-provisioned Environments

This section describes how to prepare your cluster on AWS, Azure, vSphere, Google Cloud or pre-provisioned environment, so it can be backed up before you begin with Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 732).

Prerequisites

- Ensure Velero is installed on your Essential cluster
- Install the Velero CLI (see page 785) (Use at least Velero CLI version 1.10.1)
- Ensure kubectl[500] is installed
- Ensure you have admin rights to the DKP Essential cluster

Prepare your Cluster

Run the following commands in the ***DKP Essential*** **cluster**. For general guidelines on how to set the context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

Prepare Velero

Enable `restic` backup capabilities by providing a custom configuration of Velero.

1. Create an Override with a custom configuration for Velero. This custom configuration deploys the `node-agent` service, which enables `restic`:

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: velero-overrides
  namespace: kommander
data:
  values.yaml: |
    ---
    deployNodeAgent: true
EOF
```

500 https://kubernetes.io/docs/tasks/tools/

2. Reference the created Override in Velero's `AppDeployment` to apply the new configuration:

```
cat << EOF | kubectl -n kommander patch appdeployment velero --type='merge' --
patch-file=/dev/stdin
spec:
  configOverrides:
    name: velero-overrides
EOF
```

3. Wait until the `node-agent` has been deployed:

```
until kubectl get daemonset -A | grep -m 1 "node-agent"; do 0.1 ; done
```

The `node-agent` is ready after a similar output appears:

```
kommander                node-agent
3         3         0       3             0           <none>
```

4. Verify the configuration has been updated before proceeding with the next section:

```
kubectl -n kommander wait --for=condition=Ready kustomization velero
```

The output should look similar to this:

```
kustomization.kustomize.toolkit.fluxcd.io/velero condition met
```

Prepare your Pods for Backup

> ⚠ Run the following commands in the ***DKP Essential*** **cluster**. For general guidelines on how to set the context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

Annotate the `gitea` pods to ensure `restic` backs up the Persistent Volumes (PVs) of the pods that will be affected during the expansion process. These volumes contain the Git repository information of your DKP Essential cluster.

```
kubectl -n kommander annotate pod/gitea-0 backup.velero.io/backup-volumes=data
```

```
kubectl -n kommander annotate pod/gitea-postgresql-0 backup.velero.io/backup-
volumes=data
```

Next Step:

Back up a Cluster - Azure, vSphere, GCP, and Pre-provisioned Environments (see page 744)

6.6.7.4.2.2 Back up a Cluster - Azure, vSphere, GCP, and Pre-provisioned Environments

With this workflow, you can back up and restore your cluster's applications. This backup contains Kubernetes objects and the Persistent Volumes (PVs) of Gitea pods. Given that Gitea's PVs store information on your cluster's state, you will be able to Restore your Cluster (see page 757), if required.

Run the following commands in the ***DKP Essential* cluster**. For general guidelines on how to set the context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

Back up Instructions

1. Create a backup with Velero. Use the following flags to reduce the scope of the backup and only include the applications that are affected during the expansion:

   ```
   velero backup create pre-expansion \
     --include-namespaces="kommander,kommander-default-workspace,kommander-
   flux,kubecost" \
     --include-cluster-resources \
     --snapshot-volumes=false --wait \
     --namespace kommander
   ```

   After completion, the output should look similar to this:

   ```
   Backup request "pre-expansion" submitted successfully.
   Waiting for backup to complete. You may safely press ctrl-c to stop waiting -
   your backup will continue in the background.
   ..........................................................................
   ..........................................................................
   ..........................................................................
   ..............
   Backup completed with status: Completed. You may check for more information
   using the commands `velero backup describe pre-expansion` and `velero backup
   logs pre-expansion`.
   ```

Verify the Backup

1. Ensure the backup has completed successfully:

   ```
   velero backup describe pre-expansion
   ```

   The following example output will vary depending on your cloud provider. Verify that it shows no errors and the `Phase` is `Completed`:

   ```
   Name:         pre-expansion
   Namespace:    kommander
   Labels:       velero.io/storage-location=default
   Annotations:  velero.io/source-cluster-k8s-gitversion=v1.25.5
                 velero.io/source-cluster-k8s-major-version=1
                 velero.io/source-cluster-k8s-minor-version=25

   Phase:  Completed

   Errors:    0
   Warnings:  0

   Namespaces:
     Included:  kommander, kommander-default-workspace, kommander-flux, kubecost
     Excluded:  <none>

   Resources:
     Included:        *
     Excluded:        <none>
     Cluster-scoped:  included

   Label selector:  <none>

   Storage Location:  default

   Velero-Native Snapshot PVs:  auto

   TTL:  720h0m0s

   CSISnapshotTimeout:  10m0s

   Hooks:  <none>

   Backup Format Version:  1.1.0

   Started:    2023-03-15 10:40:25 -0400 EDT
   Completed:  2023-03-15 10:44:39 -0400 EDT

   Expiration:  2023-04-14 10:40:24 -0400 EDT
   ```

```
Total items to be backed up:  5188
Items backed up:              5188

Velero-Native Snapshots: <none included>
```

2. Ensure that the `PodVolumeBackup` objects have been created:

```
kubectl get podvolumebackups -A
```

The output should look similar to this:

```
NAMESPACE   NAME        STATUS      CREATED   NAMESPACE   POD
VOLUME   REPOSITORY ID
UPLOADER TYPE   STORAGE LOCATION   AGE
kommander   ash-59ntg   Completed   39s       kommander   gitea-postgresql-0
data     s3:https://a54904d80411e4d64b572b96cb3ddb62-477717230.us-
west-2.elb.amazonaws.com:8085/dkp-velero/restic/kommander   restic
default            39s
kommander   ash-5vsbf   Completed   42s       kommander   gitea-0
data     s3:https://a54904d80411e4d64b572b96cb3ddb62-477717230.us-
west-2.elb.amazonaws.com:8085/dkp-velero/restic/kommander   restic
default            42s
```

Next Step

Expand your platform. Use the UI or CLI:

UI: Convert an Essential Cluster into a Managed Cluster (see page 746)

CLI: Convert an Essential Cluster into a Managed Cluster (see page 748)

Related Topics

Restore your Backup and Retry the Cluster Expansion (see page 757)

6.6.7.5 UI: Convert an Essential Cluster into a Managed Cluster

Enterprise Gov Advanced

Instructions on how you can convert an Essential Cluster to a Managed cluster with no Networking Restrictions with the DKP UI.

> ⚠ `Ingress` that contains Traefik-Forward-Authentication (see page 841) configuration will be not available during the expansion process, therefore, your DKP Essential cluster will not be accessible externally for several minutes. Access from within the cluster via Kubernetes service hostname (for example, http://SERVICE_NAME.NAMESPACE:PORT) is not affected.
> See Downtime Considerations (see page 733) for more information.

How to attach an existing cluster that has no additional networking restrictions

Use this option when you want to attach a cluster that does not require additional access information.

1. From the top menu bar, select your target workspace.
2. On the Dashboard page, select the **Add Cluster** option in the **Actions** dropdown menu at the top right.
3. Select **Attach Cluster**.
4. Select the **No additional networking restrictions** card. Alternatively, if you MUST use network restrictions, stop following the steps below, and see the instructions on the page Attach a Cluster with Network Restrictions (see page 693).
5. In the **Cluster Configuration** section, paste your `kubeconfig` file into the field, or select the **Upload kubeconfig File** button to specify the file.
6. The **Cluster Name** field will automatically populate with the name of the cluster is in the `kubeconfig`. You can edit this field with the name you want for your cluster.
7. The **Context** select list is populated from the `kubeconfig`. Select the desired context with admin privileges from the **Context** select list.
8. Add labels to classify your cluster as needed.
9. Select **Create** to attach your cluster.

6.6.7.5.1 Verify the Conversion

> ⚠ Run the following commands in the ***Management*** **cluster**. For general guidelines on how to set the context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

1. To verify that the conversion is successful, check the `KommanderCluster` object:

```
kubectl wait --for=condition=AttachmentCompleted kommandercluster <cluster
name> -n ${WORKSPACE_NAMESPACE} --timeout 30m
```

2. The following output appears if the conversion is successful:

```
kommandercluster.kommander.mesosphere.io/<cluster name> condition met
```

After conversion, all Platform Applications will be in the Kommander Namespace in the Managed Cluster.

6.6.7.5.2 Next Steps

Post Conversion Cleanup: Cluster Autoscaler Configuration (see page 749)

6.6.7.5.3 See Also

Troubleshooting (see page 754)

6.6.7.6 CLI: Convert an Essential Cluster into a Managed Cluster

Enterprise Gov Advanced

Instructions on how you can convert an DKP Essential Cluster to a DKP Enterprise Managed Cluster with no Networking Restrictions via the CLI.

`Ingress` that contains Traefik-Forward-Authentication (see page 841) configuration will be not available during the expansion process, therefore, your DKP Essential cluster will not be accessible externally for several minutes. Access from within the cluster via Kubernetes service hostname (for example, http://SERVICE_NAME.NAMESPACE:PORT) is not affected.
See Downtime Considerations (see page 733) for more information.

1. Run the following command in the DKP Enterprise cluster you will have managing your clusters. The cluster name and kubeconfig is from the cluster you are attaching and want to convert to initiate the process of turning an DKP Essential cluster to a DKP Enterprise Managed cluster. The workspace is the workspace name you want the attached cluster to go into.

   ```
   dkp attach cluster --name <essential-cluster-name> --attached-kubeconfig
   <kubeconfig-file-of-essential-cluster> --workspace <workspace-name>
   ```

6.6.7.6.1 Verify the Conversion

> ⚠ Run the following commands in the ***Management*** **cluster**. For general guidelines on how to set the context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

1. To verify that the conversion is successful, check the `KommanderCluster` object:

```
kubectl wait --for=condition=AttachmentCompleted kommandercluster <cluster
name> -n ${WORKSPACE_NAMESPACE} --timeout 30m
```

2. The following output appears if the conversion is successful:

```
kommandercluster.kommander.mesosphere.io/<cluster name> condition met
```

> After conversion, all Platform Applications will be in the `kommander` namespace in the Managed Cluster.

6.6.7.6.2 Next Steps

Post Conversion Cleanup: Cluster Autoscaler Configuration (see page 749)

6.6.7.6.3 See Also

Troubleshooting (see page 754)

6.6.7.7 Post Conversion Cleanup: Cluster Autoscaler Configuration

Follow the steps in this page to ensure that the Cluster Autoscaler is able to function properly after converting your DKP Essential cluster to a DKP Enterprise Managed cluster.

After converting your cluster from Essential to Enterprise, the Cluster Lifecycle Management responsibilities is moved to a single Management cluster.

The Cluster Autoscaler feature also depends on the same Cluster Lifecycle Management components. If you are using the Cluster Autoscaler feature in DKP, you must perform the following steps for this feature to continue to work correctly:

Run the following commands in the ***Management*** **cluster**. For general guidelines on how to set the context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

1. Set the following environment variables with your cluster's details:

```
export CLUSTER_NAME=<>
export WORKSPACE_NAMESPACE=<>
```

2. Apply the Cluster Autoscaler `Deployment` and supporting resources:

```
cat <<EOF | kubectl apply -f -
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: cluster-autoscaler-${CLUSTER_NAME}
  name: cluster-autoscaler-${CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cluster-autoscaler-${CLUSTER_NAME}
  template:
    metadata:
      labels:
        app: cluster-autoscaler-${CLUSTER_NAME}
    spec:
      containers:
      - args:
        - --cloud-provider=clusterapi
        - --node-group-auto-discovery=clusterapi:clusterName=${CLUSTER_NAME}
        - --kubeconfig=/workload-cluster/kubeconfig
        - --clusterapi-cloud-config-authoritative
        - -v5
        command:
        - /cluster-autoscaler
        image: us.gcr.io/k8s-artifacts-prod/autoscaling/cluster-
autoscaler:v1.25.0
        name: cluster-autoscaler
        volumeMounts:
        - mountPath: /workload-cluster
          name: kubeconfig
          readOnly: true
      serviceAccountName: cluster-autoscaler-${CLUSTER_NAME}
```

```
      terminationGracePeriodSeconds: 10
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/master
      - effect: NoSchedule
        key: node-role.kubernetes.io/control-plane
      volumes:
      - name: kubeconfig
        secret:
          items:
          - key: value
            path: kubeconfig
          secretName: ${CLUSTER_NAME}-kubeconfig
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cluster-autoscaler-management-${CLUSTER_NAME}
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-autoscaler-management-${CLUSTER_NAME}
subjects:
- kind: ServiceAccount
  name: cluster-autoscaler-${CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: cluster-autoscaler-${CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-autoscaler-management-${CLUSTER_NAME}
rules:
- apiGroups:
  - cluster.x-k8s.io
  resources:
  - machinedeployments
  - machinedeployments/scale
  - machines
  - machinesets
  verbs:
  - get
  - list
  - update
  - watch
EOF
```

3. Verify the output is similar to the following:

```
deployment.apps/cluster-autoscaler-<cluster-name> created
clusterrolebinding.rbac.authorization.k8s.io/cluster-autoscaler-management-
<cluster-name> created
serviceaccount/cluster-autoscaler-<cluster-name> created
clusterrole.rbac.authorization.k8s.io/cluster-autoscaler-management-<cluster-
name> created
```

4. To check that the status of the deployment has the expected `AVAILABLE` count of `1`, run the following command and verify that the output is similar:

```
$ kubectl get deployment -n $WORKSPACE_NAMESPACE cluster-autoscaler-
$CLUSTER_NAME
NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
cluster-autoscaler-<cluster-name>    1/1     1            1           1m
```

6.6.7.7.1 Next Step

Post Conversion Cleanup: Clusters run on Different Cloud Platforms (see page 752)

6.6.7.7.2 See Also

Troubleshooting (see page 754)

6.6.7.8 Post Conversion Cleanup: Clusters run on Different Cloud Platforms

For a DKP Enterprise Management cluster to manage a cluster hosted in another cloud provider, you must ensure the Management cluster has all the necessary permissions.

6.6.7.8.1 Prerequisites

Prior to running these commands, you must ensure that the DKP Management Enterprise cluster is configured with the necessary platform specific permissions to manage the incoming CAPI objects that backs the infrastructure resources in the target cloud platform.

For example, for the DKP Enterprise Managed cluster to manage CAPI clusters in AWS, refer tohttps://cluster-api-aws.sigs.k8s.io/topics/iam-permissions.html.

DKP supports expanding your platform in the following scenarios:

***DKP Enterprise Management cluster* host provider**	***DKP Enterprise Management cluster* IAM permissions**	***DKP Essential cluster* host provider**
AWS	https://cluster-api-aws.sigs.k8s.io/topics/iam-permissions.html	AWS, GCP, vSphere, Pre-provisioned
GCP	https://cloud.google.com/iam/docs/overview	AWS, GCP, vSphere, Pre-provisioned
vSphere	https://docs.vmware.com/en/vRealize-Operations/Cloud/com.vmware.vcom.config.doc/GUID-F85638E3-937E-4E31-90D0-9D4A5E479292.html	AWS, GCP, vSphere, Pre-provisioned
Azure	https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-ops-guide-iam	Azure
Pre-provisioned	NA	AWS, GCP, vSphere, Pre-provisioned

6.6.7.8.2 Moving the CAPI Resources

1. Following the conversion into a DKP Enterprise managed cluster, run the following command to move the CAPI Objects:

```
dkp move capi-resources --from-kubeconfig <essential_cluster_kubeconfig> --to-kubeconfig <enterprise_cluster_kubeconfig> --to-namespace ${WORKSPACE_NAMESPACE}
```

2. Verify that the output looks similar to the following:

```
✓ Moving cluster resources
```

```
You can now view resources in the moved cluster by using the --kubeconfig flag
with kubectl. For example: kubectl --kubeconfig=<enterprise_cluster_kubeconfig>
get nodes
```

3. After moving the resources, run the following command to remove the CAPI controller manager deployments:

```
dkp delete capi-components --kubeconfig <essential_cluster_kubeconfig>
```

6.6.7.8.3 Next Step

Troubleshooting (see page 754)

6.6.7.9 Troubleshooting

Enterprise Gov Advanced

When an error or failure occurs when converting a DKP Essential cluster to a DKP Enterprise Managed cluster, DKP automatically keeps retrying the cluster's conversion and attachment process. You do not need to trigger it manually.

If the state does not improve after a while, here are some ways in which you can check or troubleshoot the failed conversion:

- Verify the Conversion Status of your Cluster (see page 754)
- Errors Related to CAPI Resources (see page 756)
- Restore your Backup and Retry the Cluster Expansion (see page 757)

6.6.7.9.1 Verify the Conversion Status of your Cluster

Enterprise Gov Advanced

6.6.7.9.1.1 Verify the Conversion

Run the following commands in the ***Management* cluster**. For general guidelines on how to set the context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

1. To verify that the conversion is successful, check the `KommanderCluster` object:

```
kubectl wait --for=condition=AttachmentCompleted kommandercluster <cluster
name> -n ${WORKSPACE_NAMESPACE} --timeout 30m
```

2. The following output appears if the conversion is successful:

```
kommandercluster.kommander.mesosphere.io/<cluster name> condition met
```

If the condition is not met yet, you can observe the conversion process:

6.6.7.9.1.2 Observe the Conversion Process

You can print the state of your cluster's `KommanderCluster` object through the CLI, and observe the cluster conversion process:

```
kubectl get kommandercluster -n ${WORKSPACE_NAMESPACE} <essential_cluster_name> -o
go-template='{{range .status.conditions }}type: {{.type}} {{"\n"}}status: {{.status}}
{{"\n"}}reason: {{.reason}} {{"\n"}}lastTxTime: {{.lastTransitionTime}} {{"\n"}}
message: {{.message}} {{"\n\n"}}{{end}}'
```

The output looks similar to this:

```
type: IngressAddressReady
status: False
reason: IngressServiceNotFound
lastTxTime: 2023-02-24T14:58:18Z
message: Ingress service object was not found in the cluster

type: IngressCertificateReady
status: True
reason: Ready
lastTxTime: 2023-02-24T14:49:09Z
message: Certificate is up to date and has not expired

type: CAPIResourceMoved
status: True
reason: Succeeded
lastTxTime: 2023-02-24T14:50:56Z
message: Moved CAPI resources from attached cluster to management cluster

type: PreAttachmentCleanup
status: True
reason: Succeeded
lastTxTime: 2023-02-24T14:54:47Z
message: pre-attach cleanup succeeded

# [...]
```

6.6.7.9.1.3 Next Step:

Errors Related to CAPI Resources (see page 756)

6.6.7.9.2 Errors Related to CAPI Resources

Enterprise Gov Advanced

6.6.7.9.2.1 Failed Condition Reason: `FailedToIdentifyCAPIResources`

kubeconfigRef points to the wrong secret

Verify the `KommanderCluster` object of both your Essential and Enterprise clusters. The `spec.kubeconfigRef.name` of each object should point to a valid `kubeconfig` secret.

1. Download the referenced `kubeconfig` to your local machine:

```
kubectl get secret -n <WORKSPACE_NAMESPACE> <cluster_name>-kubeconfig -o
jsonpath='{.data.value}' | base64 --decode > <cluster_name>-kubeconfig
```

2. Verify if the `kubeconfig` is valid:

```
kubectl get namespaces -A --kubeconfig <cluster_name>-kubeconfig
```

3. Verify the output of the previous command:

No errors in the output

If your output shows no errors, the error message is not related to a `kubeconfig`.

Error in the output

If the output shows an error, delete the `KommanderCluster` object via CLI:

```
kubectl delete kommandercluster -n <WORKSPACE_NAMESPACE> <WRONG_KOMMANDER_CLUSTER>
```

At this particular stage and in the context of converting your cluster, deleting your `KommanderCluster` will not affect your environment. However, DO NOT delete your `KommanderCluster` in other scenarios, as it detaches the referenced cluster from the Management cluster.

Finally, restart the cluster conversion process with the UI (see page 746) or CLI (see page 748).

Essential cluster has more than one instance of `v1beta1/clusters.cluster.x-k8s.io`

> ⚠ D2iQ does not support converting Essential clusters that contain the Cluster API resources of more than one cluster.

Ensure your Essential cluster only contains its own CAPI resources and does not contain the CAPI resources of other clusters.

6.6.7.9.2.2 Next Topic:

Restore your Backup and Retry the Cluster Expansion (see page 757)

6.6.7.9.3 Restore your Backup and Retry the Cluster Expansion

When an error or failure occurs when converting a DKP Essential cluster to a DKP Enterprise Managed cluster, DKP automatically keeps retrying the cluster's conversion and attachment process. You do not need to trigger it manually.

If you must interrupt the expansion process to restore your cluster and retry the expansion procedure, follow these instructions:

- Restore your Cluster (see page 757)
- Move your CAPI Resources Back to the DKP Essential Cluster (see page 758)
- Verify the Restore Process and Retry the Expansion (see page 759)

6.6.7.9.3.1 Restore your Cluster

Prerequisites

- You have backed up your cluster (see page 736).
- The cluster expansion (see page 732) you attempted was not successful.

Restore the Cluster

> ⚠ Switch between **DKP Enterprise Management** and **DKP Essential** clusters for the following commands. For general guidelines on how to set the context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

1. Delete the `KommanderCluster` object on the **DKP Enterprise Management** cluster:

```
kubectl -n <WORKSPACE_NAMESPACE> delete kommandercluster
<KOMMANDER_CLUSTER_NAME> --wait=false
```

2. Disable the Flux controllers on the **DKP Essential** cluster to interrupt the expansion process:

```
kubectl -n kommander-flux delete deployment -l app.kubernetes.io/
instance=kommander-flux
```

3. Delete the `kube-federation-system` namespace on the **DKP Essential** cluster:

```
kubectl get ns kube-federation-system -o json | jq '.spec.finalizers = []' |
kubectl replace --raw "/api/v1/namespaces/kube-federation-system/finalize" -f -
```

4. Restore your cluster's configuration on the **DKP Essential** cluster:

```
velero restore create pre-expansion --from-backup pre-expansion --existing-
resource-policy update --wait --namespace kommander
```

Next Step:

Move your CAPI Resources Back to the DKP Essential Cluster (see page 758)

6.6.7.9.3.2 Move your CAPI Resources Back to the DKP Essential Cluster

Previous Step:

Restore your Cluster (see page 757)

Move your cluster's CAPI Resources

Because the created backup does not include CAPI resources, you will also have to move the resources back to the DKP Essential cluster.

Ensure you replace `<CAPI_CLUSTER_NAME>` with the name of the DKP Essential cluster you were converting in the current workflow. If you accidentally provide the CAPI cluster name of another Managed cluster, the command will move the CAPI resources of the incorrect cluster to the DKP Essential cluster.

1. Retrieve your Managed cluster's kubeconfig and write it to the `<essential.conf>` file:

```
KUBECONFIG=management.conf ./dkp get kubeconfig -n <WORKSPACE_NAMESPACE> -c
<CAPI_CLUSTER_NAME> > essential.conf
```

2. Move your CAPI resources back to the DKP Essential cluster:

```
dkp move capi-resources --from-kubeconfig management.conf --to-kubeconfig
<essential.conf> -n <WORKSPACE_NAMESPACE> --to-namespace default
```

Next Step:

Verify the Restore Process and Retry the Expansion (see page 759)

6.6.7.9.3.3 Verify the Restore Process and Retry the Expansion

Previous Step:

Move your CAPI Resources Back to the DKP Essential Cluster (see page 758)

Verify the Restore Process

Verify that you successfully restored your cluster:

```
velero restore describe pre-expansion --namespace kommander
```

The output looks similar to this:

```
Name:         pre-expansion
Namespace:    kommander
Labels:       <none>
Annotations:  <none>

Phase:                       Completed
Total items to be restored:  2411
Items restored:              2411

...
```

Retry the Expansion Process

Execute the cluster expansion again, as shown in Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 732).

Related Topic:

Troubleshooting (see page 754)

6.6.8 Advanced Creation of CLI Clusters

Create CLI clusters

> ⚠ This feature is for advanced users and users in unique environments only. We highly recommend using other documented methods to create clusters whenever possible.

6.6.8.1 Generate Cluster Objects

Depending on your infrastructure, DKP CLI can generate a set of cluster objects that can be customized for unusual use cases. Visit the Advanced Configuration documentation for AWS (see page 958) for an example on how to use the `--dry-run` and `--output` flags to create a set of cluster objects.

6.6.8.2 Use the Upload YAML Form

1. In the selected workspace Dashboard, select the **Add Cluster** option in the **Actions** dropdown menu at the top right.
2. On the Add Cluster page, select **Upload YAML to Create a Cluster** and provide advanced cluster details:
   - **Workspace**: The workspace where this cluster belongs (if within the Global workspace).
   - **Cluster YAML**: Paste or upload your customized set of cluster objects into this field. Only valid YAML is accepted.
   - **Add Labels**: By default, your cluster has labels that reflect the infrastructure provider provisioning. For example, your AWS cluster may have a label for the data center region and `provider: aws`. Cluster labels are matched to the selectors created for Projects (see page 608). Changing a cluster label may add or remove the cluster from projects.
3. Click **Create** to begin provisioning the DKP CLI cluster. This step may take a few minutes, taking time for the cluster to be ready and fully deploy its components. The cluster automatically tries to join, and should resolve after it is fully-provisioned.

6.6.9 Management Cluster

A guide for the Management Cluster and the Management Cluster Workspace

When you install Kommander, the host cluster is attached in the **Management Cluster Workspace** as the **Management Cluster**, called **Management Cluster** in the Global workspace dashboard and **Kommander Host** inside the Management Cluster Workspace. This allows the Management Cluster to be included in Projects (see page 608) and enables the management of its Platform Applications (see page 611) from the Management Cluster Workspace.

> Do not attach a cluster in the "Management Cluster Workspace" workspace. This workspace is reserved for your Kommander Management cluster only.

6.6.9.1 Editing

As an attached cluster, you can edit the Management Cluster to add or remove Labels, then you can use these labels to include the Management Cluster in Projects inside of the Management Cluster Workspace.

6.6.9.2 Disconnecting

The Management Cluster cannot be disconnected from the GUI like other attached clusters. Because of this, the Management Cluster Workspace cannot be deleted from the GUI as it will always have the Management Cluster inside itself.

6.6.10 Cluster Applications

Applications are installed by the management cluster. You can visit a cluster's detail page to see the application dashboards enabled from the deployed applications under the **Application Dashboards** section.

Under the **Applications** section of the cluster's detail page, you can view the workspace applications enabled for the cluster, grouped by category.

In this section, you can also view the current status of the enabled applications on the cluster, on each application card. Hovering on the status displays details about the status of the application.

Cluster applications can have one of the following statuses:

Status	Description
Enabled	The application is enabled, but the status on the cluster is not available.
Pending	The application is waiting to be deployed.
Deploying	The application is currently being deployed to the cluster.

Status	Description
Deployed	The application has successfully been deployed to the cluster.
Deploy Failed	The application failed to deploy to the cluster.

Review the Workspace Platform Application Configuration Requirements (see page 114) to ensure that the attached clusters have sufficient resources. For more information on applications and how to customize them, see Workspace Applications (see page 586).

6.6.10.1 Custom Cluster Application Dashboard Cards

You can add custom application dashboard cards to the cluster detail page's Applications section by creating a `ConfigMap` on the cluster. The `ConfigMap` must have a `kommander.d2iq.io/application` label applied through the CLI, and must contain both `name` and `dashboardLink` data keys to be displayed. Upon creation of the `ConfigMap`, the DKP UI displays a card corresponding to the data provided in the `ConfigMap`. Custom application cards have a Kubernetes icon, and can link to a service running in the cluster, or use an absolute URL to link to any accessible URL.

6.6.10.1.1 ConfigMap Example

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: "my-app"
  namespace: "app-namespace"
  labels:
    "kommander.d2iq.io/application": "my-app"
data:
  name: "My Application"
  dashboardLink: "/path/to/app"
```

Key	Description	Required
`metadata.labels."kommander.d2iq.io/application"`	The application name (ID).	X

Key	Description	Required
`data.name`	The display name that describes the application and displays on the custom application card in the user interface.	X
`data.dashboardLink`	The link to the application. This can be an absolute link, `https://www.d2iq.com` or a relative link, `/dkp/kommander/dashboard.` If you use a relative link, the link is built using the cluster's path as the base of the URL to the application.	X
`data.docsLink`	Link to documentation about the application. This is displayed on the application card, but omitted if not present.	
`data.category`	Category with which to group the custom application. If not provided, the application is grouped under the category, "None."	
`data.version`	A version string for the application. If not provided, "N/A" is displayed on the application card in the user interface.	

Use a command similar to this to create a new custom application `ConfigMap`:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: "my-app"
  namespace: "default"
  labels:
    "kommander.d2iq.io/application": "my-app"
data:
```

```
  name: "My Application"
  dashboardLink: "/path/to/app"
EOF
```

6.6.11 Custom Domains and Certificates Configuration

Configure a custom domain and certificate for your Management or any Managed/Attached cluster

DKP supports configuring a custom domain name per cluster, so you can access the DKP UI and other platform services via that domain. Additionally, you can provide a custom certificate for the domain, or one can be issued automatically by Let's Encrypt (or other certificate authorities supporting the ACME protocol).

The configuration path is the same regardless of whether you are configuring a custom domain and certificate on the Management cluster, or a Managed/Attached cluster. However, you can choose to set up a customized domain and certificate for the Management cluster during DKP installation.

- Customize a domain or certificate in the **Management cluster**, during installation (see page 1230).
- Customize a domain or certificate in the **Management or a Managed/Attached Cluster**, after installation (see page 765).

To customize the domain or certificate for a specific cluster after the installation of DKP, adapt or create an API YAML to specify the custom domain and certification details.

Here is an overview of the information provided in this section:

- Why Should you set up a Custom Domain or Certificate? (see page 764)
- Configure Custom Domains or Custom Certificates post Kommander Installation (see page 765)

6.6.11.1 Why Should you set up a Custom Domain or Certificate?

6.6.11.1.1 Reasons for Using a Custom DNS Domain

DKP supports the customization of domains to allow you to use your own domain or hostname for your services. For example, you can set up your DKP UI or any of your clusters to be accessible with your custom domain name instead of the domain provided by default.

- To set up a custom domain (without a custom certificate), refer to Configure a Custom Domain without a Custom Certificate (see page 1240).

6.6.11.1.2 Reasons for Using a Custom Certificate

DKP's default CA identity supports the encryption of data exchange and traffic (between your client and your environment's server). To configure an additional security layer that validates your environment's server authenticity, DKP supports configuring a custom certificate issued by a trusted Certificate Authority either directly in a Secret or managed automatically using the ACME protocol (for example, Let's Encrypt).

Changing the default certificate for any of your clusters can be helpful. For example, you can adapt it to classify your DKP UI or any other type of service as trusted (when accessing a service via a browser).

To set up a custom domain and certificate, refer to the following pages respectively:

- Configure a custom domain and certificate as part of the cluster's installation process (see page 1232). This is only possible for your Management/Essential cluster (see page 97).
- Update your cluster's current domain and certificate configuration as part of your cluster's Day 2 operations (see page 765). You can do this for any cluster type (see page 97) in your environment.

> Using Let's Encrypt or other public ACME certificate authorities **does not work in air-gapped scenarios**, as these services require connection to the Internet for their setup. For air-gapped environments, you can either use self-signed certificates issued by the cluster (the default configuration), or a certificate created manually using a trusted Certificate Authority.

6.6.11.1.3 Next Step:

Configure Custom Domains or Custom Certificates (see page 765)

6.6.11.2 Configure Custom Domains or Custom Certificates post Kommander Installation

If you have not installed DKP yet, you can also configure a custom domain during the installation of DKP (see page 1230).

Once you have installed the Kommander component of DKP, you can configure a custom domain and certificate by modifying the `KommanderCluster` object of your cluster.

6.6.11.2.1 Important Concepts

IssuerRef, ClusterIssuerRef or certificateSecretRef?

If you use a certificate issued and managed automatically by `cert-manager`, you need an *Issuer* or *Cluster Issuer* that you reference in your `KommanderCluster` resource. The referenced object must contain the information of your certificate provider.

If you want to use a manually-created certificate, you need a *secret* that you reference in your `KommanderCluster` resource.

Management, Managed or Attached cluster? Location of the KommanderCluster and Issuer objects

In the Management (see page 97) or Essential (see page 98) cluster, both the `KommanderCluster` and *issuer* objects are stored on the same cluster. The *issuer* can be referenced as an `Issuer`, `ClusterIssuer` or `certificateSecretRef`.

In Managed (see page 97) and Attached (see page 97) clusters, the `KommanderCluster` object is stored on the Management cluster. The `Issuer`, `ClusterIssuer` or `certificateSecretRef` is stored on the Managed or Attached cluster.

6.6.11.2.2 Configuration

You have two options to update the `KommanderCluster` resource and establish a custom domain and certificate.

Expand the instructions depending on whether you need to reference an automatically-generated certificate or a manually-generated certificate:

Use an automatically-generated certificate

Use a certificate that is managed automatically and supported by `cert-manager`:

1. Create an `Issuer` or `ClusterIssuer` with your certificate provider information. Store this object in the cluster where you want to customize the certificate and domain.
   For an example of how to do this, refer to Configure your Custom Domain and Certificate (see page 1232).
2. Update the `KommanderCluster` by referencing the **name of the created** `Issuer` or `ClusterIssuer` in the `spec.ingress.issuerRef` field.
   Enter the custom domain name in the `spec.ingress.hostname` field:

```
cat <<EOF | kubectl -n <workspace_namespace> --kubeconfig
<management_cluster_kubeconfig> patch \
kommandercluster <cluster_name>  --type='merge' --patch-file=/dev/stdin
spec:
  ingress:
    hostname: <cluster_hostname>
    issuerRef:
      name: <issuer_name>
      kind: Issuer # or ClusterIssuer depending on the issuer config
EOF
```

Use a manually-generated certificate

Use a manually-created certificate that is **customized for your hostname**.

1. Obtain or create a certificate that is customized for your hostname. Store this object in the workspace namespace of the target cluster.
2. Create a secret with the certificate in the cluster's namespace. Give it a name by replacing `<certificate_secret_name>`:

```
kubectl create secret generic -n "${WORKSPACE_NAMESPACE}"
 <certificate_secret_name> \
  --from-file=ca.crt=$CERT_CA_PATH \
  --from-file=tls.crt=$CERT_PATH \
  --from-file=tls.key=$CERT_KEY_PATH \
  --type=kubernetes.io/tls
```

3. Update the `KommanderCluster` by referencing this **secret** in the `spec.ingress.certificateSecretRef` field and provide the custom domain name in the `spec.ingress.hostname`:

```
cat <<EOF | kubectl -n <workspace_namespace> --kubeconfig
<management_cluster_kubeconfig> patch \
kommandercluster <cluster_name>  --type='merge' --patch-file=/dev/stdin
spec:
  ingress:
    hostname: <cluster_hostname>
    certificateSecretRef:
      name: <certificate_secret_name>
EOF
```

In order for Kommander to access the secret containing the certificate, it must be located in the workspace namespace of the target cluster.

6.6.11.2.2.1 Next Step:

Verify and Troubleshoot Configuration Status (see page 769)

6.6.11.2.2.2 Related topics

Why Should you set up a Custom Domain or Certificate? (see page 764)

https://d2iq.atlassian.net/wiki/spaces/DENT/pages/137593048/Configuration+Example+with+Let+s+Encrypt (see page 767)

Advanced Configuration: ClusterIssuer (see page 1236)

6.6.11.2.3 Configuration Example with Let's Encrypt

6.6.11.2.3.1 Configure a Custom Certificate with Let's Encrypt

Let's Encrypt is one of the Certificate Authorities (CA) supported by cert-manager. To set up a Let's Encrypt certificate, create an `Issuer` or `ClusterIssuer` in the target cluster and then reference it in the `issuerRef` field of the `KommanderCluster` resource.

1. Create the Let's Encrypt ACME cert-manager issuer:

```
cat <<EOF | kubectl apply -f -
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: custom-acme-issuer
spec:
  acme:
    email: <your_email>
    server: https://acme-v02.api.letsencrypt.org/directory
    privateKeySecretRef:
      name: kommander-acme-issuer-account
    solvers:
      - dns01:
          route53:
            region: us-east-1
            role: arn:aws:iam::YYYYYYYYYYYY:role/dns-manager
EOF
```

2. Configure the Management cluster to use your `custom-domain.example.com` with a certificate issued by Let's Encrypt by referencing the created `ClusterIssuer`:

```
cat <<EOF | kubectl -n kommander --kubeconfig <management_cluster_kubeconfig>
patch \ kommandercluster host-cluster  --type='merge' --patch-file=/dev/stdin
spec:
  ingress:
    hostname: custom-domain.example.com
    issuerRef:
      name: custom-acme-issuer
      kind: ClusterIssuer
EOF
```

6.6.11.2.3.2 Next Step:

Verify and Troubleshoot Configuration Status (see page 769)

6.6.11.2.3.3 Related Topics

Why Should you set up a Custom Domain or Certificate? (see page 764)

Configure Custom Domains or Custom Certificates (see page 765)

6.6.11.2.4 Verify and Troubleshoot Configuration Status

If you want to ensure the customization for a domain and certificate is completed, or if you want to obtain more information on the status of the customization, display the status information for the `KommanderCluster`.

1. Inspect the modified `KommanderCluster` object:

   ```
   kubectl describe kommandercluster -n <workspace_name> <cluster_name>
   ```

2. If the ingress is still being provisioned, the output looks similar to this:

   ```
   [...]
   Conditions:
     Last Transition Time:  2022-06-24T07:48:31Z
     Message:               Ingress service object was not found in the cluster
     Reason:                IngressServiceNotFound
     Status:                False
     Type:                  IngressAddressReady
   [...]
   ```

   If the provisioning has been completed, the output looks similar to this:

   ```
   [...]
   Conditions:
     Last Transition Time:  2022-06-28T13:43:33Z
     Message:               Ingress service address has been provisioned
     Reason:                IngressServiceAddressFound
     Status:                True
     Type:                  IngressAddressReady
     Last Transition Time:  2022-06-28T13:42:24Z
     Message:               Certificate is up to date and has not expired
     Reason:                Ready
     Status:                True
     Type:                  IngressCertificateReady
   [...]
   ```

   The same command also prints the actual customized values for the `KommanderCluster.Status.Ingress`. Here is an example:

   ```
   [...]
      ingress:
       address: 172.20.255.180
       caBundle: LS0tLS1CRUdJTiBD...<output has been shortened>...DQVRFLS0tLS0K
   [...]
   ```

6.6.11.2.4.1 Related Topics

Why Should you set up a Custom Domain or Certificate? (see page 764)

https://d2iq.atlassian.net/wiki/spaces/DENT/pages/137593048/Configuration+Example+with+Let+s+Encrypt (see page 767)

Configure Custom Domains or Custom Certificates (see page 765)

6.6.12 Disconnect or Delete Clusters

6.6.12.1 Disconnect or delete a cluster

6.6.12.2 Disconnect vs. Delete

When you attach a cluster to Kommander that was **not created with Kommander**, you can later detach it. This does not alter the running state of the cluster, but simply removes it from the DKP UI. User workloads, platform services, and other Kubernetes resources are not cleaned up at detach.

After successfully detaching the cluster, manually disconnect the attached cluster's Flux installation from the management Git repository. Otherwise, changes to apps in the managed cluster's workspace will still be reflected on the cluster you just detached. Ensure your `dkp` configuration references the target cluster. You can do this by setting the `KUBECONFIG` environment variable to the appropriate kubeconfig file location[501]. An alternative to initializing the `KUBECONFIG` environment variable is to use the `-kubeconfig=cluster_name.conf` flag. Then, run `kubectl -n kommander-flux patch gitrepo management -p '{"spec":{"suspend":true}}' --type merge` to make the cluster's workloads not managed by Kommander, anymore.

If you **created a managed cluster with Kommander**, you cannot disconnect the cluster, but you can **delete** the cluster. This completely removes the cluster and all of its cloud assets.

We recommend deleting a managed cluster via the DKP UI. Deleting clusters via the CLI will not remove all DKP resources, in which case you will have to follow the troubleshooting instructions (see page 771) to finalize the detachment.

501 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

If you delete the Management (Konvoy) cluster, you can not use Kommander to delete any Managed clusters created by Kommander. If you want to delete all clusters, ensure you delete any Managed clusters before finally deleting the Management cluster.

6.6.12.2.1 Statuses

See Statuses (see page 669) for a list of possible states a cluster can have when it is getting disconnected or deleted.

6.6.12.3 Troubleshooting

I cannot detach an attached cluster that is “Pending”

OR

The cluster I deleted via CLI still appears in the UI with “Error” state

Sometimes detaching or deleting a Kubernetes cluster causes that cluster to get stuck in a “Pending” or “Error” state. This can happen because the wrong `kubeconfig` file is used, or the cluster is just not reachable. In order to detach the cluster, so it does not show in the UI, follow these steps:

1. Determine the `KommanderCluster` resource backing the cluster you tried to attach/detach:

   ```
   kubectl -n WORKSPACE_NAMESPACE get kommandercluster
   ```

   Replace `WORKSPACE_NAMESPACE` with the actual current workspace name. You can find this name by going to `https://YOUR_CLUSTER_DOMAIN_OR_IP_ADDRESS/dkp/kommander/dashboard/workspaces` in your browser.

2. Delete the cluster:

   ```
   kubectl -n WORKSPACE_NAMESPACE delete kommandercluster CLUSTER_NAME
   ```

3. If the resource does not go after a short time, remove its finalizers:

   ```
   kubectl -n WORKSPACE_NAMESPACE patch kommandercluster CLUSTER_NAME --type json
   -p '[{"op":"remove", "path":"/metadata/finalizers"}]'
   ```

   This removes the cluster from the DKP UI.

6.7 Backup and Restore

For production clusters, regular maintenance should include routine backup operations to ensure data integrity and reduce the risk of data loss due to unexpected events. Backup operations should include the cluster state, application state, and the running configuration of both stateless and stateful applications in the cluster.

DKP stores all data as CRDs in the Kubernetes API and you can back up and restore it using the following procedure.

- Configure Velero with Rook Ceph Storage (see page 772)
- Use Velero with AWS (see page 772)
- Use Velero with Azure (see page 778)
- Back up with Velero (see page 785)
- Use Velero with GCP (see page 791)

6.7.1 Configure Velero with Rook Ceph Storage

For default installations, DKP deploys Velero[502] integrated with Rook Ceph[503], operating inside the same cluster.

For production use-cases, D2iQ advises to provide an *external* storage class to use with Rook Ceph[504]. See the Rook Ceph in DKP (see page 894) for more information.

6.7.1.1 Cloud Provider Storage

If you do not want to use Rook Ceph for storing Velero backups and you can configure Velero to use cloud provider storage (see page 778).

6.7.2 Use Velero with AWS

Configure Velero to use S3 buckets as storage for its backup operations.

6.7.2.1 Overview

Follow these procedures to set up Velero with AWS S3:

- Velero with AWS S3 - Prepare your Environment (see page 773)
- Velero with AWS S3 - Configure Velero (see page 774)
- Velero with AWS S3 - Establish a Backup Location (see page 777)

502 https://velero.io/
503 https://rook.io/
504 https://rook.io/

6.7.2.2 Velero with AWS S3 - Prepare your Environment

6.7.2.2.1 Prerequisites

- Ensure you have installed Velero (included in the default DKP installation).
- Ensure you have installed the Velero CLI (see page 785).
- You have created an S3 bucket with AWS[505].

6.7.2.2.1.1 Environment variables:

- Set the `BUCKET` environment variable to the name of the S3 bucket you want to use as backup storage:

  ```
  export BUCKET=<aws-bucket-name>
  ```

- Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace. Replace `<workspace_namespace>` with the name of the target workspace:

  ```
  export WORKSPACE_NAMESPACE=<workspace_namespace>
  ```

  This can be the `kommander` namespace for the Management (see page 97) cluster or any other additional workspace namespace for Attached (see page 97) or Managed (see page 97) clusters. To list all available workspaces and clusters, use the dkp get clusters -A (see page 1497) command.

- Set the `CLUSTER_NAME` environment variable. Replace `<target_cluster>` with the name of the cluster where you want to set up Velero:

  ```
  export CLUSTER_NAME=<target_cluster>
  ```

6.7.2.2.2 Prepare your AWS Credentials

See the official docs[506] for details on how to use IAM roles instead of static credentials.

1. Create a file containing your static AWS credentials:
   In this example, the file's name is `credentials-velero`.

505 https://docs.aws.amazon.com/AmazonS3/latest/userguide/creating-bucket.html
506 https://github.com/vmware-tanzu/velero-plugin-for-aws

```
cat << EOF > aws-credentials
  [default]
  aws_access_key_id=<REDACTED>
  aws_secret_access_key=<REDACTED>
EOF
```

2. Create a secret on the cluster where you are installing and configuring Velero by referencing the file created in the previous step. This can be the Management (see page 97), a Managed (see page 97) or an Attached (see page 97) cluster:
   In this example, the secret's name is `velero-aws-credentials`.

```
kubectl create secret generic -n ${WORKSPACE_NAMESPACE} velero-aws-credentials
--from-file=aws=aw
```

6.7.2.2.3 Next Step:

Velero with AWS S3 - Configure Velero (see page 774)

6.7.2.3 Velero with AWS S3 - Configure Velero

Customize Velero to allow the configuration of a non-default backup location.

1. Create a `ConfigMap` for the Velero configuration:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: velero-overrides
data:
  values.yaml: |
    configuration:
      backupStorageLocation:
        bucket: ${BUCKET}
        config:
          region: <AWS_REGION> # such as us-west-2
          s3ForcePathStyle: "false"
          insecureSkipTLSVerify: "false"
          s3Url: ""
          # profile should be set to the AWS profile name mentioned in the
secret
          profile: default
    credentials:
      # With the proper IAM permissions with access to the S3 bucket,
```

```
      # you can attach the EC2 instances using the IAM Role, OR fill in
"existingSecret" OR "secretContents" below.
      #
      # Name of a pre-existing secret (if any) in the Velero namespace
      # that should be used to get IAM account credentials.
      existingSecret: velero-aws-credentials
      # The key must be named "cloud", and the value corresponds to the entire
content of your IAM credentials file.
      # For more information, consult the documentation for the velero plugin
for AWS at:
      # [AWS] https://github.com/vmware-tanzu/velero-plugin-for-aws/blob/main/
README.md
      secretContents:
        # cloud: |
        #   [default]
        #   aws_access_key_id=<REDACTED>
        #   aws_secret_access_key=<REDACTED>
EOF
```

2. Patch the Velero `AppDeployment` to reference the created `ConfigMap` with the Velero overrides:
   a. To update Velero in **all clusters in a workspace**:

   ```
   cat << EOF | kubectl -n ${WORKSPACE_NAMESPACE} patch appdeployment
   velero --type="merge" --patch-file=/dev/stdin
   spec:
     configOverrides:
       name: velero-overrides
   EOF
   ```

   b. To update Velero for a **specific cluster** in a workspace, see Customize an Application per Cluster (see page 580) .

3. Check the `ConfigMap` on the `HelmRelease` object:

   ```
   kubectl wait --for=jsonpath='{.spec.valuesFrom[1].name}'=velero-overrides
   HelmRelease/velero -n ${WORKSPACE_NAMESPACE}
   ```

4. Verify that the Velero pod is running:

   ```
   kubectl get pods -A --kubeconfig=${CLUSTER_NAME}.conf |grep velero
   ```

You can also configure Velero by editing the `kommander.yaml` and rerunning the installation. To follow this *alternative* configuration path, expand the following section:

Alternative Configuration path for Management/Essential clusters

6.7.2.3.1 Configure Velero on the Management Cluster

1. Output the Kommander configuration to `kommander.yaml`

```
dkp install kommander -o yaml --init > kommander.yaml
```

   a. Add the Velero configuration to the `apps.velero.values` section:
   NOTE: This process has been tested to work with plugins for AWS v1.1.0. Newer versions of these plugins can be used, but have not been tested by D2iQ.

```
...
  velero:
    values: |
      configuration:
      backupStorageLocation:
        bucket: ${BUCKET}
        config:
          region: <AWS_REGION> # such as us-west-2
          s3ForcePathStyle: "false"
          insecureSkipTLSVerify: "false"
          s3Url: ""
          # profile should be set to the AWS profile name mentioned in
the secret
          profile: default
      credentials:
        # With the proper IAM permissions with access to the S3 bucket,
        # you can attach the EC2 instances using the IAM Role, OR fill
in "existingSecret" OR "secretContents" below.
        #
        # Name of a pre-existing secret (if any) in the Velero namespace
        # that should be used to get IAM account credentials.
        existingSecret: velero-aws-credentials
        # The key must be named "cloud", and the value corresponds to
the entire content of your IAM credentials file.
        # For more information, consult the documentation for the velero
plugin for AWS at:
        # [AWS] https://github.com/vmware-tanzu/velero-plugin-for-aws/
blob/main/README.md
        secretContents:
          # cloud: |
          #   [default]
          #   aws_access_key_id=<REDACTED>
          #   aws_secret_access_key=<REDACTED>
...
```

2. Run `dkp install kommander` using the `kommander.yaml` configuration file:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

3. After running `dkp install kommander` see the ConfigMap on the `HelmRelease` object:

```
kubectl wait --for=jsonpath='{.spec.valuesFrom[1].name}'=velero-overrides
HelmRelease/velero -n kommander
```

4. Check the Helm releases that the new Velero configuration has been applied:

```
kubectl get helmrelease -n ${WORKSPACE_NAMESPACE} --kubeconfig=$
{CLUSTER_NAME}.conf
```

5. Verify that the Velero pod is running:

```
kubectl get pods -A --kubeconfig=${CLUSTER_NAME}.conf |grep velero
```

6. Create the backup storage location:

```
velero backup-location create -n ${WORKSPACE_NAMESPACE} <aws-backup-location-
name> \
  --provider aws \
  --bucket ${BUCKET} \
  --config region=<AWS_REGION> \
  --credential=velero-aws-credentials=aws
```

7. Check that the backup storage location is `Available` and referencing the correct S3 bucket:

```
kubectl get backupstoragelocations -n kommander -oyaml
```

6.7.2.3.2 Next Step:

Velero with AWS S3 - Establish a Backup Location (see page 777)

6.7.2.4 Velero with AWS S3 - Establish a Backup Location

6.7.2.4.1 Create a Backup Storage Location

1. Create a location for the backup by pointing to an existing S3 bucket:

```
velero backup-location create -n ${WORKSPACE_NAMESPACE} <aws-backup-location-
name> \
  --provider aws \
  --bucket ${BUCKET} \
  --config region=<AWS_REGION> \
  --credential=velero-aws-credentials=aws
```

2. Check that the backup storage location is `Available` and that it references the correct S3 bucket:

```
kubectl get backupstoragelocations -n ${WORKSPACE_NAMESPACE} -oyaml
```

If the BackupStorageLocation is not Available, view any error events by using: `kubectl describe backupstoragelocations -n ${WORKSPACE_NAMESPACE}`

6.7.2.4.2 Create a Test Backup

1. Create a test backup for AWS:

```
velero backup create aws-velero-testbackup -n ${WORKSPACE_NAMESPACE} --
kubeconfig=${CLUSTER_NAME}.conf --storage-location <aws-backup-location-name>
--snapshot-volumes=false
```

2. View your backup:

```
velero backup describe aws-velero-testbackup
```

6.7.2.4.3 Next Step:

Back up with Velero (see page 785)

6.7.3 Use Velero with Azure

Configure Velero to use Azure Blob Storage as storage for its backup operations.

6.7.3.1 Overview

Follow these procedures to set up Velero with Azure Blob Storage:

6.7.3.2 Velero with Azure Blob Containers - Prepare your Environment

6.7.3.2.1 Prerequisites

- Ensure you have installed Velero (included in the default DKP installation).
- Ensure you have installed the Velero CLI (see page 785).
- Ensure you have installed the Azure CLI[507].
- Ensure you have sufficient access rights to the Azure storage environment and blob container you want to use for backup. For more information about data authorization, see the official Azure blob storage documentation[508].

6.7.3.2.2 Prepare your Environment

1. Create a container in Azure blob storage[509].
2. Set the `BLOB_CONTAINER` environment variable to the name of the blob container you created to use as backup storage:

   ```
   export BLOB_CONTAINER=<Azure-blob-container-name>
   ```

3. Set up a storage account and resource group[510].
4. Set the `AZURE_BACKUP_RESOURCE_GROUP` variable to the name of the resource group you created:

   ```
   AZURE_BACKUP_RESOURCE_GROUP=<azure-resource-group-name>
   ```

5. Set the `AZURE_STORAGE_ACCOUNT_ID` variable to the unique identifier of the storage account you want to use for the backup:
   See https://learn.microsoft.com/en-us/azure/storage/common/storage-account-get-info?toc=%2Fazure%2Fstorage%2Fblobs%2Ftoc.json&bc=%2Fazure%2Fstorage%2Fblobs%2Fbreadcrumb%2Ftoc.json&tabs=azure-cli#get-the-resource-id-for-a-storage-account to obtain the ID. The output

507 https://learn.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest
508 https://learn.microsoft.com/en-us/azure/storage/common/authorize-data-access?toc=%2Fazure%2Fstorage%2Fblobs%2Ftoc.json&bc=%2Fazure%2Fstorage%2Fblobs%2Fbreadcrumb%2Ftoc.json
509 https://learn.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-portal#create-a-container
510 https://learn.microsoft.com/en-us/azure/storage/common/storage-account-create?tabs=azure-cli#create-a-storage-account-1

shows the entire location path of the storage account. You only need the last part, or storage account name, to set the variable.

```
AZURE_STORAGE_ACCOUNT_ID=<storage-account-name>
```

6. Set the `AZURE_BACKUP_SUBSCRIPTION_ID` variable to the unique identifier of the subscription you want to use for the backup:
   See https://learn.microsoft.com/en-us/cli/azure/account?view=azure-cli-latest#az-account-list to obtain the ID.

   ```
   AZURE_BACKUP_SUBSCRIPTION_ID=<azure-subscription-id>
   ```

7. Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace.
   Replace `<workspace_namespace>` with the name of the target workspace:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

   This can be the `kommander` namespace for the Management (see page 97) cluster or any other additional workspace namespace for Attached (see page 97) or Managed (see page 97) clusters. To list all available workspace namespaces, use the `kubectl get kommandercluster -A` command.

8. Set the `CLUSTER_NAME` environment variable. Replace `<target_cluster>` with the name of the cluster where you want to set up Velero:

   ```
   export CLUSTER_NAME=<target_cluster>
   ```

6.7.3.2.3 Prepare your Azure Credentials

> See https://learn.microsoft.com/en-us/azure/storage/blobs/authorize-data-operations-portal for more details on authorization.

1. Create a `credentials-velero` file with the information required to create a secret. Use the same credentials that you employed when creating the cluster (see page 1033).
   These credentials **should not** be Base64 encoded, because Velero will not read them properly.
   Replace the variables in `<...>` with your environment's information. See your Microsoft Azure account to look up the values.

```
cat << EOF > ./credentials-velero
AZURE_SUBSCRIPTION_ID=${AZURE_BACKUP_SUBSCRIPTION_ID}
AZURE_TENANT_ID=<AZURE_TENANT_ID>
AZURE_CLIENT_ID=<AZURE_CLIENT_ID>
AZURE_CLIENT_SECRET=<AZURE_CLIENT_SECRET>
AZURE_BACKUP_RESOURCE_GROUP=${AZURE_BACKUP_RESOURCE_GROUP}
AZURE_CLOUD_NAME=AzurePublicCloud
EOF
```

2. Use the `credentials-velero` file to create the secret:

```
kubectl create secret generic -n ${WORKSPACE_NAMESPACE} velero-azure-
credentials --from-file=azure=credentials-velero --kubeconfig=$
{CLUSTER_NAME}.conf
```

6.7.3.2.4 Next Step:

Velero with Azure Blob Containers - Configure Velero (see page 781)

6.7.3.3 Velero with Azure Blob Containers - Configure Velero

Customize Velero to allow the configuration of a non-default backup location.

1. Create a `ConfigMap` to allow Velero to use Azure blob containers as backup storage location:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: velero-overrides
data:
  values.yaml: |
    initContainers:
      - name: velero-plugin-for-microsoft-azure
        image: velero/velero-plugin-for-microsoft-azure:v1.5.1
        imagePullPolicy: IfNotPresent
        volumeMounts:
          - mountPath: /target
            name: plugins
    credentials:
      extraSecretRef: velero-azure-credentials
EOF
```

2. Patch the Velero `AppDeployment` to reference the created `ConfigMap` with the Velero overrides:
   a. To update Velero in **all clusters in a workspace**:

```
cat << EOF | kubectl -n ${WORKSPACE_NAMESPACE} patch appdeployment
velero --type="merge" --patch-file=/dev/stdin
spec:
  configOverrides:
    name: velero-overrides
EOF
```

   b. To update Velero for a **specific cluster** in a workspace, see Customize an Application per Cluster (see page 580).

3. Check the `ConfigMap` on the `HelmRelease` object:

```
kubectl get hr -n kommander velero -o jsonpath='{.spec.valuesFrom[?
(@.name=="velero-overrides")]}'
```

   The output looks like this if the deployment is successful:

```
{"kind":"ConfigMap","name":"velero-overrides"}
```

4. Verify that the Velero pod is running:

```
kubectl get pods -A --kubeconfig=${CLUSTER_NAME}.conf |grep velero
```

You can also configure Velero by editing the `kommander.yaml` and rerunning the installation. To follow this *alternative* configuration path, expand the following section:

Alternative Configuration Path for Management/Essential Clusters

1. Refresh the `kommander.yaml` to add the customization of Velero:
   ⚠️ Before running this command, ensure the `kommander.yaml` is the configuration file you are currently using for your environment. Otherwise, your previous configuration will be lost. ⚠️

```
dkp install kommander -o yaml --init > kommander.yaml
```

2. Configure DKP to load the plugins and to include the secret in the `apps.velero` section:
   ℹ️ This process has been tested to work with plugin Azure v1.5.1. More recent versions of these plugins can be used, but have not been tested by D2iQ.

```
...
```

```
  velero:
    values: |
      initContainers:
        - name: velero-plugin-for-microsoft-azure
          image: velero/velero-plugin-for-microsoft-azure:v1.5.1
          imagePullPolicy: IfNotPresent
          volumeMounts:
            - mountPath: /target
              name: plugins
      credentials:
        extraSecretRef: velero-azure-credentials
...
```

3. Use the modified `kommander.yaml` configuration to install this Velero configuration:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

4. Check the `ConfigMap` on the `HelmRelease` object:

```
kubectl get hr -n kommander velero -o jsonpath='{.spec.valuesFrom[?
(@.name=="velero-overrides")]}'
```

The output looks like this if the deployment is successful:

```
{"kind":"ConfigMap","name":"velero-overrides"}
```

5. Verify that the Velero pod is running:

```
kubectl get pods -A --kubeconfig=${CLUSTER_NAME}.conf |grep velero
```

6.7.3.3.1 Next Step:

Velero with Azure Blob Containers - Establish a Backup Location (see page 783)

6.7.3.4 Velero with Azure Blob Containers - Establish a Backup Location

6.7.3.4.1 Create a Backup Storage Location

1. Create a location for the backup by pointing to an existing Azure container:
   Ensure you set the required environment variables as specified in Velero with Azure Blob

Containers - Prepare your Environment (see page 779).

Replace `<azure-backup-location-name>` with a name for the backup location.

```
velero backup-location create <azure-backup-location-name> -n $
{WORKSPACE_NAMESPACE} \
--provider azure \
--bucket ${BLOB_CONTAINER} \
--config resourceGroup=${AZURE_BACKUP_RESOURCE_GROUP},storageAccount=$
{AZURE_STORAGE_ACCOUNT_ID},subscriptionId=${AZURE_BACKUP_SUBSCRIPTION_ID} \
--credential=velero-azure-credentials=azure --kubeconfig=${CLUSTER_NAME}.conf
```

2. Check that the backup storage location is `Available` and that it references the correct Azure container:

```
kubectl get backupstoragelocations -n ${WORKSPACE_NAMESPACE} -oyaml
```

6.7.3.4.2 Create a Test Backup

1. Create a test backup that is stored in the location you created in the previous section:

```
velero backup create azure-velero-testbackup -n ${WORKSPACE_NAMESPACE} \
--kubeconfig=${CLUSTER_NAME}.conf \
--storage-location <azure-backup-location-name> \
--snapshot-volumes=false
```

2. View your backup:

```
velero backup describe azure-velero-testbackup
```

Troubleshooting

1. If your backup wasn't created, Velero may have had an issue installing the plugin. If the plugin was not installed, run this command:

```
velero plugin add velero/velero-plugin-for-microsoft-azure:v1.5.1 -n $
{WORKSPACE_NAMESPACE}
```

2. Confirm your `backupstoragelocation` was configured correctly

```
kubectl get backupstoragelocations -n ${WORKSPACE_NAMESPACE}
```

If your backup storage location is “Available”, proceed creating a test backup.

```
NAME              PHASE       LAST VALIDATED   AGE   DEFAULT
<azure-backup-location-name>       Available   38s              60m
```

6.7.3.4.3 Next Step:

Back up with Velero (see page 785)

6.7.4 Back up with Velero

DKP provides Velero[511] by default, to support backup and restore operations for your Kubernetes clusters and persistent volumes.

- Install the Velero CLI (see page 785)
- Regular Backup Operations (see page 786)
- Restore a cluster from backup (see page 788)
- Backup Service Diagnostics (see page 790)

6.7.4.1 Install the Velero CLI

Although installing the Velero command-line interface is optional and independent of deploying a DKP cluster, having access to the command-line interface provides several benefits. For example, you can use the Velero command-line interface to back up or restore a cluster on demand, or to modify certain settings without changing the Velero configuration.

- By default, DKP sets up Velero to use Rook Ceph over TLS using a self-signed certificate.
- As a result, when using certain commands, you may be asked to use the `--insecure-skip-tls-verify` flag.

Again, the default setup is not suitable for production use-cases.

See the instructions to install the Velero command-line interface[512] for more information.

In DKP, the Velero platform application is installed in the `kommander` namespace, instead of `velero`. Thus, after installing the CLI, we recommend that you set the Velero CLI `namespace` config option so that subsequent Velero CLI invocations will use the correct namespace:

```
velero client config set namespace=kommander
```

511 https://velero.io/
512 https://velero.io/docs/v1.5/basic-install/#install-the-cli

6.7.4.2 Regular Backup Operations

Velero provides the following basic administrative functions to back up production clusters:

- Set a Backup Schedule (see page 0)
- Run On-demand Backups (see page 0)
- Restore from a Backup Archive (see page 788)

If you require a custom backup location, see how to create one for AWS (see page 777), Azure (see page 783), or GCP (see page 795).

6.7.4.2.1 Prepare your Environment

Before you modify a schedule or create an on-demand backup, set the following environment variables:

1. Specify the workspace namespace of the cluster for which you want to configure the backup:

```
export WORKSPACE_NAMESPACE=<workspace_namespace>
```

2. Specify the cluster for which you want to create the backup:

```
export CLUSTER_NAME=<target_cluster_name>
```

6.7.4.2.2 Set a Backup Schedule

By default, DKP configures a regular, automatic backup of the cluster's state in Velero. The default settings do the following:

- Create daily backups
- Save the data from all namespaces

DKP default backups do not support the creation of Volume Snapshots.

These default settings take effect after the cluster is created. If you install DKP with the default platform services deployed, the initial backup starts after the cluster is successfully provisioned and ready for use.

6.7.4.2.3 Create Backup Schedules

The Velero CLI provides an easy way to create alternate backup schedules. For example:

```
velero create schedule <backup-schedule-name> -n ${WORKSPACE_NAMESPACE} \
--kubeconfig=${CLUSTER_NAME}.conf \
--snapshot-volumes=false \
--schedule="@every 8h"
```

6.7.4.2.4 Change Default Backup Service Settings

1. Check the backup schedules currently configured for the cluster:

   ```
   velero get schedules
   ```

2. Delete the `velero-default` schedule:

   ```
   velero delete schedule velero-default
   ```

3. Replace the default schedule with your custom settings:

   ```
   velero create schedule velero-default -n ${WORKSPACE_NAMESPACE} \
   --kubeconfig=${CLUSTER_NAME}.conf \
   --snapshot-volumes=false \
   --schedule="@every 24h"
   ```

6.7.4.2.5 Create Backup Schedule for a Specific Namespace

You can also create backup schedules for specific namespaces. Creating a backup for a specific namespace can be useful for clusters running multiple apps operated by multiple teams. For example:

```
velero create schedule <backup-schedule-name> \
--include-namespaces=kube-system,kube-public,kommander \
--snapshot-volumes=false \
--schedule="@every 24h"
```

The Velero command line interface provides many more options worth exploring. You can also find tutorials for disaster recovery[513] and cluster migration[514] on the Velero community site.

513 https://velero.io/docs/v0.11.0/disaster-case
514 https://velero.io/docs/v0.11.0/migration-case

6.7.4.2.6 Back up on Demand

In some cases, you might find it necessary to create a backup outside the regularly-scheduled interval. For example, if you are preparing to upgrade any components or modify your cluster configuration, perform a backup before taking that action.

Create a backup by running the following command:

```
velero backup create <backup-name> -n ${WORKSPACE_NAMESPACE} \
--kubeconfig=${CLUSTER_NAME}.conf \
--snapshot-volumes=false
```

6.7.4.3 Restore a cluster from backup

6.7.4.3.1 Prerequisites

Before attempting to restore the cluster state using the Velero command-line interface, verify the following requirements:

- The backend storage, Rook Ceph Cluster, is still operational.
- The Velero platform service in the cluster is still operational.
- The Velero platform service is set to a `restore-only-mode` to avoid having backups run while restoring.

6.7.4.3.2 Restoring from Backup

To list the available backup archives for your cluster, run the following command:

```
velero backup get
```

To set Velero to a `restore-only-mode`, run the following command:

```
velero server --restore-only=true
```

This mode is being deprecated and will be removed in Velero in Velero v2.0. Use read-only backup storage locations instead.

Finally, check your deployment to verify that the configuration change was applied correctly:

```
helm get values -n kommander velero
```

To restore cluster data on demand from a selected backup snapshot available in the cluster, run a command similar to the following:

```
velero restore create --from-backup <BACKUP-NAME>
```

Important: If you are restoring using Velero from the default setup (and not using an external bucket or blob to store your backups (see page 778)), you may see an error when describing or viewing the logs of your backup restore. This is a known issue when restoring from an object store that is not accessible from outside your cluster. However, you can review the success of the backup restore by confirming the Phase is `Completed` and not in error, and viewing the logs by running these `kubectl` commands:

```
kubectl logs -l name=velero -n kommander --tail -1
```

6.7.4.3.3 Restore your DKP Management Cluster

When restoring a backup to the Management Cluster, you must adjust configuration to avoid restore errors.

6.7.4.3.3.1 Prior to restoring a backup

1. Ensure the following `ResourceQuota` setup is not configured on your cluster (this `ResourceQuota` will be automatically restored)

   ```
   kubectl -n kommander delete resourcequota one-kommandercluster-per-kommander-workspace
   ```

2. Turn off the `Workspace` validation webhooks. Otherwise, you will not restore Workspaces with pre-configured namespaces. If the validation webhook named `kommander-validating` is present, it must be modified with this command:

```
kubectl patch validatingwebhookconfigurations kommander-validating \
  --type json \
  --patch '[
    {
      "op": "remove",
      "path": "/webhooks/0/rules/3/operations/0"
    }
  ]'
```

6.7.4.3.3.2 After restoring a backup

1. Verify that the `ResourceQuota` named `one-kommandercluster-per-kommander-workspace` is restored
2. Add the removed `CREATE` webhook rule operation with:

```
kubectl patch validatingwebhookconfigurations kommander-validating \
  --type json \
  --patch '[
    {
      "op": "add",
      "path": "/webhooks/0/rules/3/operations/0",
      "value": "CREATE"
    }
  ]'
```

6.7.4.4 Backup Service Diagnostics

You can check whether the Velero service is currently running on your cluster through the Kubernetes dashboard (accessible through the DKP UI on the Management Cluster), or by running the following `kubectl` command:

```
kubectl get all -A | grep velero
```

If the Velero platform service application is currently running, you can generate diagnostic information about Velero backup and restore operations. For example, you can run the following commands to retrieve, back up, and restore information that you can use to assess the overall health of Velero in your cluster:

```
velero get schedules
velero get backups
velero get restores
velero get backup-locations
velero get snapshot-locations
```

6.7.5 Use Velero with GCP

Configure Velero to use Google Cloud Storage Buckets as storage for its backup operations.

6.7.5.1 Overview

Follow these procedures to set up Velero with Azure Blob Storage:

- Velero with Google Cloud Storage Buckets - Prepare your Environment (see page 791)
- Velero with Google Cloud Storage Buckets - Configure Velero (see page 792)
- Velero with Google Cloud Storage Buckets - Establish a Backup Location (see page 795)

6.7.5.2 Velero with Google Cloud Storage Buckets - Prepare your Environment

6.7.5.2.1 Prerequisites

- Ensure you have installed Velero (included in the default DKP installation).
- Ensure you have installed the Velero CLI (see page 785).
- Ensure you have installed the gcloud CLI[515].
- **Optional:** You can install the gsutil CLI[516] (or opt to create buckets through the GCS Console)
- Ensure you have created a GCS bucket[517].
- Ensure you have sufficient access rights to the bucket you want to use for backup. For more information about GCP-related access control, refer to the official documentation of the Google Cloud Storage platform[518].

6.7.5.2.2 Set Environment Variables

- Set the `BUCKET` environment variable to the name of the GCS container you want to use as backup storage:

  ```
  export BUCKET=<GCS-bucket-name>
  ```

- Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace. Replace `<workspace_namespace>` with the name of the target workspace:

  ```
  export WORKSPACE_NAMESPACE=<workspace_namespace>
  ```

515 https://cloud.google.com/sdk/docs/install
516 https://cloud.google.com/storage/docs/gsutil_install
517 https://cloud.google.com/storage/docs/creating-buckets
518 https://cloud.google.com/storage/docs/access-control

This can be the `kommander` namespace for the Management (see page 97) cluster or any other additional workspace namespace for Attached (see page 97) or Managed (see page 97) clusters. To list all available workspace namespaces, use the `kubectl get kommandercluster -A` command.

- Set the `CLUSTER_NAME` environment variable. Replace `<target_cluster>` with the name of the cluster where you want to set up Velero:

```
export CLUSTER_NAME=<target_cluster>
```

6.7.5.2.3 Prepare your Google Credentials

You can store your backups in **Google Cloud Platform/GCS buckets**.

> See https://cloud.google.com/storage/docs/creating-buckets#required-roles for more information on setting up access to your bucket.

1. Create a `credentials-velero` file with the information required to create a secret.
   Replace `<service-account-email>` with the email address you used to grant permissions to your bucket. The address usually follows the format `<service-account-user>@<gcp-project>.iam.gserviceaccount.com`.

```
gcloud iam service-accounts keys create credentials-velero \
    --iam-account <service-account-email>
```

2. Use the `credentials-velero` file to create the secret:

```
kubectl create secret generic -n ${WORKSPACE_NAMESPACE} velero-gcp-credentials
--from-file=gcp=credentials-velero --kubeconfig=${CLUSTER_NAME}.conf
```

6.7.5.2.4 Next Step:

Velero with Google Cloud Storage Buckets - Configure Velero (see page 792)

6.7.5.3 Velero with Google Cloud Storage Buckets - Configure Velero

Customize Velero to allow the configuration of a non-default backup location.

1. Create a `ConfigMap` to allow Velero to use GCS buckets as backup storage location:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: velero-overrides
data:
  values.yaml: |
    initContainers:
      - name: velero-plugin-for-gcp
        image: velero/velero-plugin-for-gcp:v1.5.0
        imagePullPolicy: IfNotPresent
        volumeMounts:
          - mountPath: /target
            name: plugins
    credentials:
      extraSecretRef: velero-gcp-credentials
EOF
```

2. Patch the Velero `AppDeployment` to reference the created `ConfigMap` with the Velero overrides:
   a. To update Velero in **all clusters in a workspace**:

   ```
   cat << EOF | kubectl -n ${WORKSPACE_NAMESPACE} patch appdeployment
   velero --type="merge" --patch-file=/dev/stdin
   spec:
     configOverrides:
       name: velero-overrides
   EOF
   ```

   b. To update Velero for a **specific cluster** in a workspace, see Customize an Application per Cluster (see page 580).

3. Check the `ConfigMap` on the `HelmRelease` object:

   ```
   kubectl get hr -n kommander velero -o jsonpath='{.spec.valuesFrom[?
   (@.name=="velero-overrides")]}'
   ```

   The output looks like this if the deployment is successful:

   ```
   {"kind":"ConfigMap","name":"velero-overrides"}
   ```

4. Ensure that the Velero pod is running:

   ```
   kubectl get pods -A --kubeconfig=${CLUSTER_NAME}.conf |grep velero
   ```

You can also configure Velero by editing the `kommander.yaml` and rerunning the installation. To follow this *alternative* configuration path, expand the following section:

Alternative Configuration Path for Management/Essential Clusters

6.7.5.3.1 Configure Velero on the Management Cluster

1. Refresh the `kommander.yaml` to add the customization of Velero:
   ⚠️ Before running this command, ensure the `kommander.yaml` is the configuration file you are currently using for your environment. Otherwise, your previous configuration will be lost. ⚠️

   ```
   dkp install kommander -o yaml --init > kommander.yaml
   ```

2. Configure DKP to load the plugins and to include the secret in the `apps.velero` section:
   ℹ️ This process has been tested to work with plugin GCP v1.5.0. More recent versions of these plugins can be used, but have not been tested by D2iQ.

   ```
   ...
     velero:
       values: |
         initContainers:
           - name: velero-plugin-for-gcp
             image: velero/velero-plugin-for-gcp:v1.5.0
             imagePullPolicy: IfNotPresent
             volumeMounts:
               - mountPath: /target
                 name: plugins
         credentials:
           extraSecretRef: velero-gcp-credentials
   ...
   ```

3. Use the modified `kommander.yaml` configuration to install this Velero configuration:

   ```
   dkp install kommander --installer-config kommander.yaml --kubeconfig=$
   {CLUSTER_NAME}.conf
   ```

4. Check the `ConfigMap` on the `HelmRelease` object:

   ```
   kubectl get hr -n kommander velero -o jsonpath='{.spec.valuesFrom[?
   (@.name=="velero-overrides")]}'
   ```

   The output looks like this if the deployment is successful:

```
{"kind":"ConfigMap","name":"velero-overrides"}
```

5. Verify that the Velero pod is running:

```
kubectl get pods -A --kubeconfig=${CLUSTER_NAME}.conf |grep velero
```

6.7.5.3.2 Next Step:

Velero with Google Cloud Storage Buckets - Establish a Backup Location (see page 795)

6.7.5.4 Velero with Google Cloud Storage Buckets - Establish a Backup Location

6.7.5.4.1 Create a Backup Storage Location

1. Create a location for the backup by pointing to an existing GCS bucket:
   Ensure you set the required environment variables as specified in Velero with Google Cloud Storage Buckets - Prepare your Environment (see page 791).
   Replace `<gcp-backup-location-name>` with a name for the backup location.

```
velero backup-location create <gcp-backup-location-name> -n $
{WORKSPACE_NAMESPACE} \
--provider gcp \
--bucket $BUCKET \
--credential=velero-gcp-credentials=gcp
```

2. Check that the backup storage location is `Available` and that it references the correct GCS bucket:

```
kubectl get backupstoragelocations -n ${WORKSPACE_NAMESPACE} -oyaml
```

6.7.5.4.2 Create a Test Backup

1. Create a test backup that is stored in the location you created in the previous section:

```
velero backup create gcp-velero-testbackup -n ${WORKSPACE_NAMESPACE} \
--kubeconfig=${CLUSTER_NAME}.conf \
--storage-location <gcp-backup-location-name> \
```

```
--snapshot-volumes=false
```

2. View your backup:

```
velero backup describe gcp-velero-testbackup
```

Troubleshooting

1. If your backup wasn't created, Velero may have had an issue installing the plugin. If the plugin was not installed, run this command:

```
velero plugin add velero/velero-plugin-for-gcp:v1.5.0 -n ${WORKSPACE_NAMESPACE}
```

2. Confirm your `backupstoragelocation` was configured correctly

```
kubectl get backupstoragelocations -n ${WORKSPACE_NAMESPACE}
```

If your backup storage location is "Available", proceed creating a test backup.

```
NAME              PHASE       LAST VALIDATED   AGE   DEFAULT
<gcp-backup-location-name>       Available   38s              60m
```

6.7.5.4.3 Next Step:

Back up with Velero (see page 785)

6.8 Logging

D2iQ Kubernetes Platform (DKP) comes with a pre-configured logging stack that allows you to collect and visualize pod and admin log data at the Workspace level. The logging stack is also multi-tenant capable. Multi-tenancy is enabled at the Project level through role-based access control (RBAC).

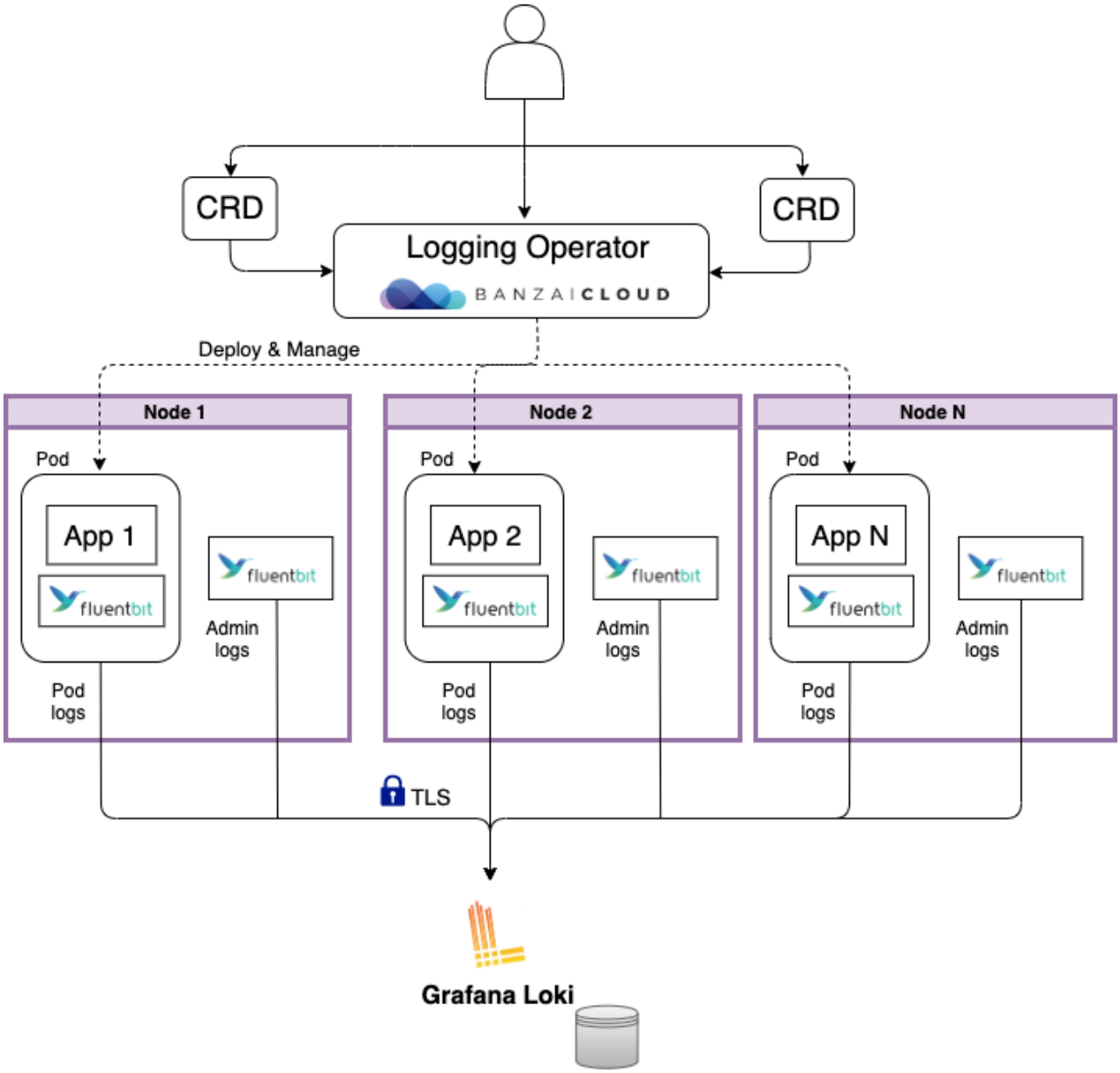
By default, logging is disabled on managed and attached clusters. You need to enable the logging stack applications explicitly on the workspace to make use of these capabilities.

The primary components of the logging stack include these platform services:

- BanzaiCloud Logging-operator
- Grafana and Grafana Loki
- Fluent Bit and Fluentd

In addition to these platform services, logging relies on other software and system facilities, including the container runtime, the journald facility, and systemd configuration are used to collect logs and messages from all the machines in the cluster.

The following diagram illustrates how different components of the logging stack collect log data and provide information about clusters:



The DKP logging stack aggregates logs from applications and nodes running inside your cluster.

DKP uses the BanzaiCloud Logging-operator to manage the Fluent Bit and Fluentd deployments that collect pod logs, using Kubernetes API extensions called custom resources. The custom resources allow users to declare logging configurations using `kubectl` commands. The Fluent Bit instance deployed by Logging-operator gathers pod logs data and sends it to Fluentd, which forwards it to the appropriate Grafana Loki servers based on the configuration defined in custom resources.

Loki then indexes the log data by label and stores it for querying. Loki maintains log order integrity but does not index the log messages themselves, which improves its efficiency and lowers its footprint.

- Admin-level logs (see page 798)
- Enable Workspace-level Logging (see page 798)
- Multi-Tenant Logging Overview (see page 808)
- Fluent Bit (see page 818)
- Scaling the Logging Stack (see page 824)
- Configuring Loki to use AWS S3 Storage in DKP (see page 827)

6.8.1 Admin-level logs

DKP also includes a Fluentbit instance to collect admin-level log information which is sent to the workspace Grafana Loki that's running on the cluster. The admin log information includes:

- Logs for host processes managed by systemd
- Kernel logs
- Kubernetes audit logs

This approach helps to isolate the more sensitive logs from Logging-operator, eliminating the possibility that users might gain inadvertent access to that data.

See Fluent Bit (see page 818) for more information about these logs.

On the Management cluster, the Fluentbit application is disabled by default. The amount of admin logs ingested to Loki requires additional disk space to be configured on the `rook-ceph-cluster`. Enabling admin logs may use around 2GB/day per node. See Rook Ceph in DKP (see page 894) for more details on how to configure the Ceph Cluster.

6.8.2 Enable Workspace-level Logging

6.8.2.1 How to enable Workspace-level Logging for use with DKP.

Logging is disabled by default on managed and attached clusters. You will need to enable logging features explicitly at the Workspace level, if you want to capture and view log data.

You must perform these procedures to enable multi-tenant logging at the Project level as well.

- Logging Prerequisites (see page 799)
- Enable Logging Applications through the UI (see page 799)
- Create AppDeployments to Enable Workspace Logging (see page 800)

- Override ConfigMap to Restrict Logging to Specific Namespaces (see page 801)
- Override ConfigMap to Modify the Storage Retention Period in Workspace Grafana Loki (see page 803)
- Verify Cluster Logging Stack Installation (see page 805)
- View Cluster Log Data (see page 806)

6.8.2.2 Logging Prerequisites

Before you begin, you must:

- Be a cluster administrator with permissions to configure cluster-level platform services.
- Set a default storage class (see page 1254) on each attached cluster for successful Loki deployment.

For more information, see Default Storage Providers in DKP (see page 103) .

6.8.2.3 Enable Logging Applications through the UI

6.8.2.3.1 How to enable the logging stack through the UI for Workspace-level logging

You can enable the Workspace logging stack to all attached clusters within the Workspace through the UI. If you prefer to enable the logging stack with `kubectl`, review how you create AppDeployments to Enable Workspace Logging (see page 800).

To enable workspace-level logging in DKP using the UI, follow these steps:

1. From the top menu bar, select your target workspace.
2. Select **Applications** from the sidebar menu.
3. Ensure `traefik` and `cert-manager` are enabled on your cluster. These are deployed by default unless you modified your configuration.
4. Scroll to the **Logging** applications section.
5. Select the three dot button from the bottom-right corner of the cards for Rook Ceph and Rook Ceph Cluster, then click **Enable**. On the **Enable Workspace Platform Application** page, you can add a customized configuration for settings that best fit your organization. You can leave the configuration settings unchanged to enable with default settings.
6. Select **Enable** at the top right of the page.
7. Repeat the process for the Grafana Loki, Logging Operator, and Grafana Logging applications.
8. You can verify the cluster logging stack installation by waiting until the cards have a Deployed checkmark on the Cluster Application (see page 761) page, or you can verify the Cluster Logging Stack installation via the CLI (see page 805).
9. Then, you can view cluster log data (see page 806).

We do not recommend installing Fluent Bit, which is responsible for collecting admin logs, unless you have configured the Grafana Loki Ceph Cluster Bucket with sufficient storage space. The amount of admin logs ingested to Loki requires additional disk space to be configured on the `rook-ceph-cluster`. Enabling admin logs may use around 2GB/day per node. See Rook Ceph in DKP (see page 894) for more details on how to configure the Ceph Cluster.

6.8.2.4 Create AppDeployments to Enable Workspace Logging

How to create AppDeployments to enable Workspace-level logging

Workspace logging AppDeployments enable and deploy the logging stack to all attached clusters within the workspace. Use the DKP UI to enable the logging applications, or, alternately, use the CLI to create the AppDeployments.

To enable logging in DKP using the CLI, follow these steps on the **management** cluster:

1. Execute the following command to get the name and namespace of your workspace:

   ```
   dkp get workspaces
   ```

   And copy the values under the `NAME` and `NAMESPACE` columns for your workspace.

2. Export the `WORKSPACE_NAME` variable:

   ```
   export WORKSPACE_NAME=<WORKSPACE_NAME>
   ```

3. Export the `WORKSPACE_NAMESPACE` variable:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
   ```

4. Ensure that Cert-Manager and Traefik are enabled in the workspace. If you want to find if the applications are enabled on the management cluster workspace, you can run:

   ```
   dkp get appdeployments --workspace ${WORKSPACE_NAME}
   ```

5. You can confirm that the applications are deployed on the **managed** or **attached** cluster by running this `kubectl` command in that cluster. (Ensure you switch to the correct context or kubeconfig[519] of the attached cluster for the following `kubectl` command.)

519 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

```
kubectl get helmreleases -n ${WORKSPACE_NAMESPACE}
```

6. Copy these commands and execute them on the **management** cluster from a command line to create the Logging-operator, Grafana-loki, and Grafana-logging AppDeployments:

```
dkp create appdeployment logging-operator --app logging-operator-3.17.10 --
workspace ${WORKSPACE_NAME}
dkp create appdeployment rook-ceph --app rook-ceph-1.10.11 --workspace $
{WORKSPACE_NAME}
dkp create appdeployment rook-ceph-cluster --app rook-ceph-cluster-1.10.11 --
workspace ${WORKSPACE_NAME}
dkp create appdeployment grafana-loki --app grafana-loki-0.69.4 --workspace $
{WORKSPACE_NAME}
dkp create appdeployment grafana-logging --app grafana-logging-6.38.1 --
workspace ${WORKSPACE_NAME}
```

Then, you can verify the cluster logging stack installation (see page 805).

To deploy the applications to selected clusters within the workspace, refer to the cluster-scoped configuration (see page 577) section.

We do not recommend installing Fluent Bit, which is responsible for collecting admin logs, unless you have configured the Rook Ceph Cluster with sufficient storage space. Enabling admin logs via Fluent Bit may use around 2GB/day per node. See Rook Ceph in DKP (see page 894) for more details on how to configure the Rook Ceph Cluster.

To install Fluent Bit, create the AppDeployment:

```
dkp create appdeployment fluent-bit --app fluent-bit-0.21.6 --workspace $
{WORKSPACE_NAME}
```

6.8.2.5 Override ConfigMap to Restrict Logging to Specific Namespaces

How to override the logging configMap to restrict logging to specific namespaces.

As a cluster administrator, you may have a need to limit, or restrict, logging activities only to certain namespaces. Kommander allows you to do this by creating an override configMap that modifies the logging configuration created in the Create AppDeployment for Workspace Logging (see page 800) procedure.

6.8.2.5.1 Prerequisites

- Implement each of the steps listed in Enable Workspace-level Logging (see page 798).
- Ensure that log data is available before you execute this procedure.

6.8.2.5.2 Create and use the Override Entries

To create and use the override configMap entries, follow these steps:

1. Execute the following command to get the namespace of your workspace:

   ```
   dkp get workspaces
   ```

   And copy the value under the `NAMESPACE` column for your workspace.

2. Set the `WORKSPACE_NAMESPACE` variable to the namespace copied in the previous step:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
   ```

3. Identify one or more namespaces to which you want to restrict logging.

4. Create a file named `logging-operator-logging-overrides.yaml` and paste the following YAML code into it to create the overrides configMap:

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: logging-operator-logging-overrides
     namespace: ${WORKSPACE_NAMESPACE}
   data:
     values.yaml: |
       ---
       clusterFlows:
       - name: cluster-containers
         spec:
           globalOutputRefs:
           - loki
           match:
           - exclude:
               namespaces:
               - <your-namespace>
               - <your-other-namespace>
   ```

5. Add the relevant namespace values for `metadata.namespace` and the `clusterFlows[0].spec.match[0].exclude.namespaces` values at the end of the file, and save the file.
6. Use the following command to apply the YAML file:

```
kubectl apply -f logging-operator-logging-overrides.yaml
```

7. Edit the `logging-operator` AppDeployment to set the value of `spec.configOverrides.name` to `logging-operator-logging-overrides`.
(Refer to Deploy an application with a custom configuration (see page 0) for more information)

```
dkp edit appdeployment -n ${WORKSPACE_NAMESPACE} logging-operator
```

After your editing is complete, the AppDeployment resembles this example:

```
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: logging-operator
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: logging-operator-3.17.10
    kind: ClusterApp
  configOverrides:
    name: logging-operator-logging-overrides
```

8. Perform actions that generate log data, both in the specified namespaces *and* the namespaces you mean to exclude.
9. Verify that the log data contains only the data you expected to receive.

6.8.2.6 Override ConfigMap to Modify the Storage Retention Period in Workspace Grafana Loki

DKP configures Grafana Loki to retain log metadata and logs for 1 week, using the Compactor[520]. This retention policy can be customized.

The minimum retention period is 24h.

520 https://grafana.com/docs/loki/latest/operations/storage/boltdb-shipper/#compactor

To customize the retention policy using `configOverrides`, run these commands on the **management** cluster:

1. Execute the following command to get the namespace of your workspace:

   ```
   dkp get workspaces
   ```

   Copy the value under the `NAMESPACE` column for your workspace.

2. Set the `WORKSPACE_NAMESPACE` variable to the namespace copied in the previous step:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
   ```

3. Create a `ConfigMap` with custom configuration values for Grafana Loki. Since the retention configuration is nested in a `config` string, you must copy the entire block. The following example sets the retention period to 360 hours (15 days). Refer to Grafana Loki's Retention Configuration documentation[521] for more information about this field.

   ```
   cat <<EOF | kubectl apply -f -
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: grafana-loki-custom-overrides
     namespace: ${WORKSPACE_NAMESPACE}
   data:
     values.yaml: |
       loki:
         structuredConfig:
           limits_config:
             retention_period: 360h
   EOF
   ```

4. Edit the `grafana-loki` AppDeployment to set the value of `spec.configOverrides.name` to `grafana-loki-custom-overrides` (Refer to Deploy a service with a custom configuration (see page 0) for more information).

   ```
   dkp edit appdeployment -n ${WORKSPACE_NAMESPACE} grafana-loki
   ```

   After your editing is complete, the AppDeployment resembles this example:

   ```
   apiVersion: apps.kommander.d2iq.io/v1alpha3
   ```

521 https://grafana.com/docs/loki/latest/operations/storage/retention/#retention-configuration

```
kind: AppDeployment
metadata:
  name: grafana-loki
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: grafana-loki-0.69.10
    kind: ClusterApp
  configOverrides:
    name: grafana-loki-custom-overrides
```

6.8.2.7 Verify Cluster Logging Stack Installation

How to verify the cluster's logging stack installed successfully.

You must wait for the cluster's logging stack `HelmReleases` to deploy before attempting to configure or use the logging features.

Run the following commands on the **management** cluster:

1. Execute the following command to get the namespace of your workspace:

   ```
   dkp get workspaces
   ```

   And copy the value under the `NAMESPACE` column for your workspace.

2. Set the `WORKSPACE_NAMESPACE` variable to the namespace copied in the previous step:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
   ```

   Ensure you switched to the correct context or kubeconfig[522] of the attached cluster for the following kubectl commands.

3. Check the deployment status using this command on the **attached** cluster:

   ```
   kubectl get helmreleases -n ${WORKSPACE_NAMESPACE}
   ```

NOTE: It may take some time for these changes to take effect, based on the duration configured for the Flux GitRepository reconciliation.

522 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

When the logging stack is successfully deployed, you will see output that includes the following `HelmReleases`:

```
NAME                          READY   STATUS                             AGE
grafana-logging               True    Release reconciliation succeeded   15m
logging-operator              True    Release reconciliation succeeded   15m
logging-operator-logging      True    Release reconciliation succeeded   15m
grafana-loki                  True    Release reconciliation succeeded   15m
rook-ceph                     True    Release reconciliation succeeded   15m
rook-ceph-cluster             True    Release reconciliation succeeded   15m
object-bucket-claims          True    Release reconciliation succeeded   15m
```

Then, you can View Cluster Log Data (see page 806).

6.8.2.8 View Cluster Log Data

How to view the cluster's log data after enabling logging.

Though you enable logging at the Workspace level, viewing the log data is done at the cluster level, using the cluster's Grafana logging URL.

Run the following commands on the **management** cluster:

1. Execute the following command to get the namespace of your workspace:

   ```
   dkp get workspaces
   ```

   And copy the value under the `NAMESPACE` column for your workspace.

2. Set the `WORKSPACE_NAMESPACE` variable to the namespace copied in the previous step:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
   ```

   Run the following commands on the **attached** cluster to access the Grafana UI:

   Ensure you switched to the correct context or kubeconfig[523] of the attached cluster for the following kubectl commands:

3. Get the Grafana URL:

523 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

```
kubectl get ingress -n ${WORKSPACE_NAMESPACE} grafana-logging -o go-
template='https://{{with index .status.loadBalancer.ingress 0}}
{{or .hostname .ip}}{{end}}{{with index .spec.rules 0}}{{with index .http.paths
0}}{{.path }}{{end}}{{end}}{{"\n"}}'
```

To view logs in Grafana:

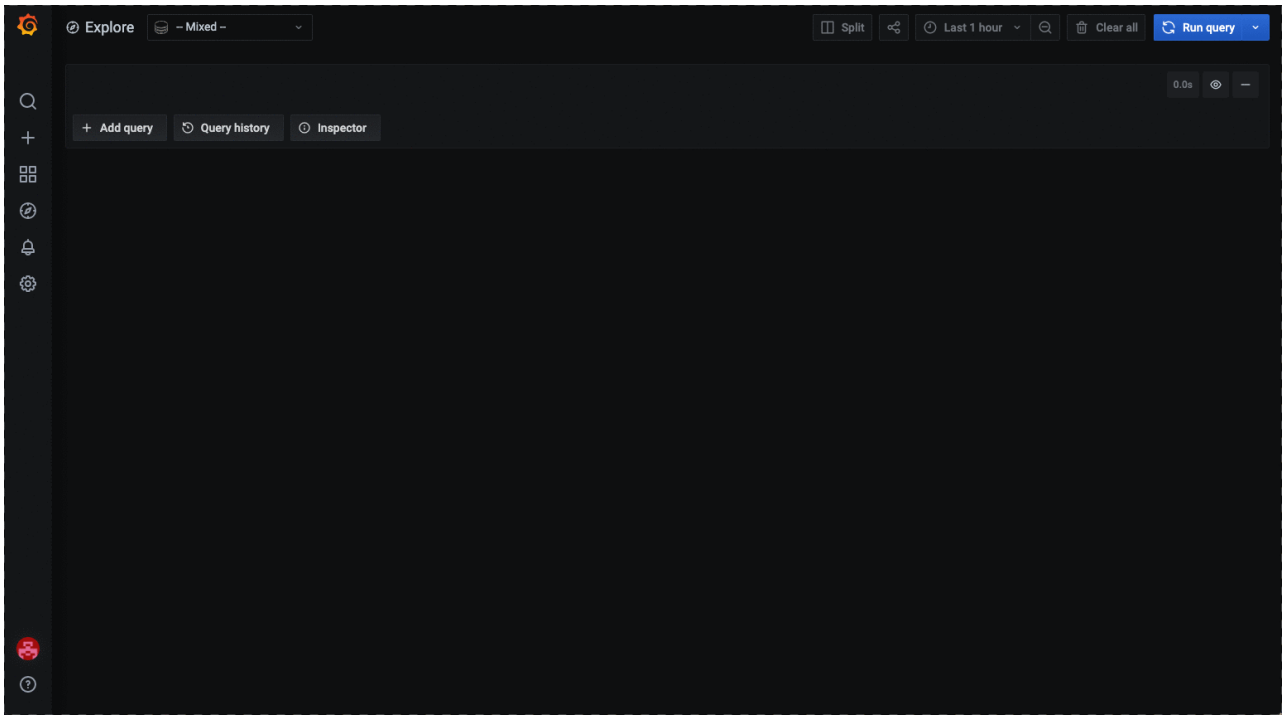
1. Go to the Explore tab:

```
kubectl get ingress -n ${WORKSPACE_NAMESPACE} grafana-logging -o go-
template='https://{{with index .status.loadBalancer.ingress 0}}
{{or .hostname .ip}}{{end}}{{with index .spec.rules 0}}{{with index .http.paths
0}}{{.path }}{{end}}{{end}}/explore{{"\n"}}'
```

2. You may be prompted to log in using the SSO flow. See Authentication and Authorization (see page 829) for more information.
3. At the top of the page, change the datasource to `Loki`.

See the Grafana Loki documentation[524] for more on how to use the interface to view and query logs.

@



524 https://grafana.com/docs/grafana/v7.5/datasources/loki/

⚠ Cert-Manager and Traefik must be deployed in the attached cluster to be able to access the Grafana UI. These are deployed by default on the workspace.

6.8.3 Multi-Tenant Logging Overview

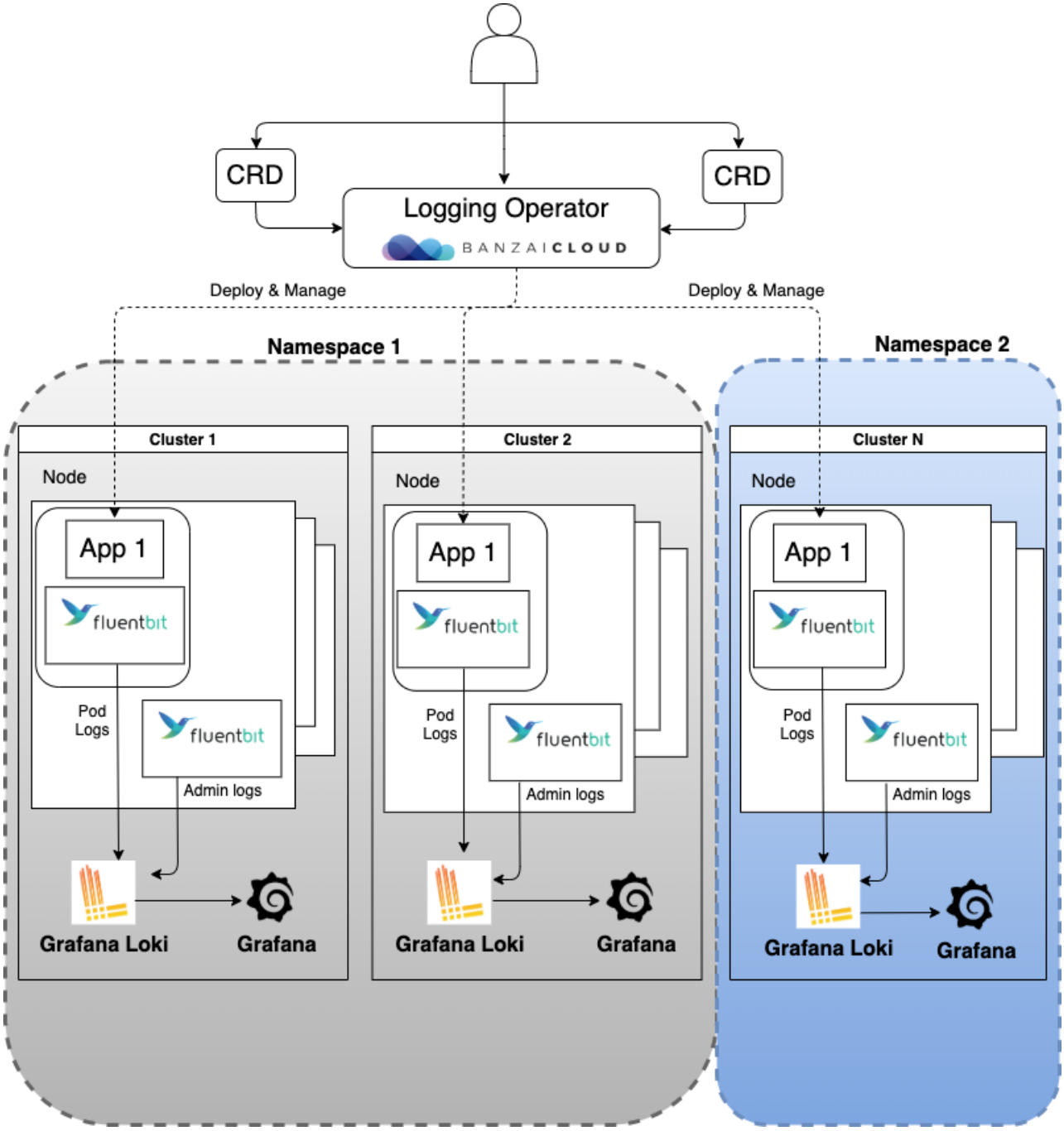
Enterprise Gov Advanced

Multi-tenant logging in Kommander is built on the DKP Logging architecture. Multi-tenant logging uses the same components as the base logging architecture:

- BanzaiCloud logging-operator
- Grafana Loki
- Grafana

⚠ You must perform the Workspace-level Logging (see page 798) procedures as a prerequisite to enable multi-tenant logging at the Project level as well.

Access to log data is done at the namespace level, through the use of Projects within Kommander as shown in the diagram:



Each Project namespace has a logging-operator “Flow” that sends its pod logs to its own Loki server. A custom controller deploys corresponding Loki and Grafana servers in each namespace, and defines a logging-operator Flow in each namespace that forwards its pod logs to its respective Loki server. There is a corresponding Grafana server for visualizations for each namespace.

For the convenience of cluster Administrators, a cluster-scoped Loki/Grafana instance pair is deployed with a corresponding Logging-operator `ClusterFlow` that directs pod logs from all namespaces to the pair. A cluster Administrator can grant access either to none of the logs, or to all logs collected from all pods in a

given namespace. Assigning teams to specific namespaces enables the team members to see only the logs for the namespaces they own.

As with any endpoint, if an Ingress controller is in use in the environment, take care that the ingress rules do not supersede the RBAC permissions and thus prevent access to the logs.

Cluster Administrators will need to monitor and adjust resource usage to prevent operational difficulties or excessive use on a *per namespace* basis.

- Enable Multi-tenant Logging (see page 810)
- Create a Project for Logging (see page 811)
- Create Project-level Logging AppDeployments (see page 811)
- Override ConfigMap to Modify the Storage Retention Period in Project Grafana Loki (see page 812)
- Verify Project Logging Stack Installation (see page 814)
- View Project Log Data (see page 815)

6.8.3.1 Enable Multi-tenant Logging

Enterprise Gov Advanced

6.8.3.1.1 Prerequisites

Before you begin, you must:

- Enable Workspace-level Logging (see page 798) before you can configure multi-tenant logging.
- Be a cluster administrator with permissions to configure cluster level platform services.

6.8.3.1.2 Multi-tenant Logging Enablement Process

The steps required to enable multi-tenant logging include:

1. Create a Project. (see page 811)
2. Create the required Project-level AppDeployments. (see page 811)
3. Verify that the Project logging stack is installed. (see page 814)
4. View a Project's log data. (see page 815)

Get started with multi-tenant logging by Creating a Project (see page 811).

6.8.3.2 Create a Project for Logging

Enterprise Gov Advanced

To enable multi-tenant logging, you must first Create a Project (see page 608) and its namespace. Users assigned to this namespace will be able to access log data for only that namespace and not others.

Then, you can create project-level AppDeployments for use in multi-tenant logging (see page 811).

6.8.3.3 Create Project-level Logging AppDeployments

Enterprise Gov Advanced

How to create Project-level AppDeployments for use in multi-tenant logging

You must create AppDeployments in the Project namespace to enable and deploy the logging stack to all clusters within a Project. You can use the CLI to do this, or use the DKP UI to enable the logging applications.

To create the AppDeployments needed for Project-level logging, follow these steps **on the management cluster**:

1. Determine the name and namespace of the workspace that your project is in. You can use the `dkp get workspaces` command to see the list of workspace names and their corresponding namespaces.

   ```
   dkp get workspaces
   ```

   Copy the values under the `NAME` and `NAMESPACE` columns for your workspace.

2. Export the `WORKSPACE_NAME` variable:

   ```
   export WORKSPACE_NAME=<WORKSPACE_NAME>
   ```

3. Export the `WORKSPACE_NAMESPACE` variable:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
   ```

4. Execute the following command to get the namespace of your project:

   ```
   kubectl get projects -n ${WORKSPACE_NAMESPACE}
   ```

Copy the value under the `NAME` column for your project. This may NOT be identical to the Display Name of the `Project`.

5. Export the `PROJECT_NAME` variable:

```
export PROJECT_NAME=<PROJECT_NAME>
```

6. Copy these commands and execute them from a command line:

```
dkp create appdeployment project-grafana-loki --app project-grafana-loki-0.48.6
 --workspace ${WORKSPACE_NAME} --project ${PROJECT_NAME}
dkp create appdeployment project-grafana-logging --app project-grafana-
logging-6.38.1 --workspace ${WORKSPACE_NAME} --project ${PROJECT_NAME}
```

Then, you can Verify the Project Logging Stack Installation (see page 814)for multi-tenant logging.

6.8.3.4 Override ConfigMap to Modify the Storage Retention Period in Project Grafana Loki

Enterprise Gov Advanced

DKP configures Project Grafana Loki to retain log metadata and logs for 1 week, using the Compactor[525]. This retention policy can be customized.

The minimum retention period is 24h.

To customize the retention policy using `configOverrides`, run these commands on the **management** cluster:

1. Determine the namespace of the workspace that your project is in. You can use the `dkp get workspaces` command to see the list of workspace names and their corresponding namespaces.

```
dkp get workspaces
```

Copy the value under the `NAMESPACE` column for your workspace. This may NOT be identical to the Display Name of the `Workspace`.

525 https://grafana.com/docs/loki/latest/operations/storage/boltdb-shipper/#compactor

2. Set the `WORKSPACE_NAMESPACE` variable to the namespace copied in the previous step:

```
export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
```

3. Get the namespace of your project:

```
kubectl get projects --namespace ${WORKSPACE_NAMESPACE}
```

Copy the value under the `PROJECT NAMESPACE` column for your workspace. This may NOT be identical to the Display Name of the `Project`.

4. Set the `PROJECT_NAMESPACE` variable to the namespace copied in the previous step:

```
export PROJECT_NAMESPACE=<PROJECT_NAMESPACE>
```

5. Create a `ConfigMap` with custom configuration values for Grafana Loki. Since the retention configuration is nested in a `config` string, you must copy the entire block. The following example sets the retention period under to 360 hours (15 days). Refer to Grafana Loki's Retention Configuration documentation[526] for more information about this field.

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: project-grafana-loki-custom-overrides
  namespace: ${PROJECT_NAMESPACE}
data:
  values.yaml: |
    loki:
      structuredConfig:
        limits_config:
          retention_period: 360h
EOF
```

6. Run the following command on the **management** cluster to reference the `configOverrides` in the `project-grafana-loki` AppDeployment:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
```

526 https://grafana.com/docs/loki/latest/operations/storage/retention/#retention-configuration

```
  name: project-grafana-loki
  namespace: ${PROJECT_NAMESPACE}
spec:
  appRef:
    name: project-grafana-loki-0.48.6
    kind: ClusterApp
  configOverrides:
    name: project-grafana-loki-custom-overrides
EOF
```

6.8.3.5 Verify Project Logging Stack Installation

Enterprise Gov Advanced

How to verify the project logging stack installation for multi-tenant logging

You must wait for the Project's logging stack `HelmReleases` to deploy before configuring or using the Project-level logging features, including multi-tenancy:

Run the following commands on the **management** cluster:

1. Determine the namespace of the workspace that your project is in. You can use the `dkp get workspaces` command to see the list of workspace names and their corresponding namespaces.

   ```
   dkp get workspaces
   ```

   Copy the value under the `NAMESPACE` column for your workspace.

2. Export the `WORKSPACE_NAMESPACE` variable:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
   ```

3. Execute the following command to get the namespace of your project

   ```
   kubectl get projects -n ${WORKSPACE_NAMESPACE}
   ```

   Copy the value under `PROJECT NAMESPACE` column for your project. This may NOT be identical to the Display Name of the `Project`.

4. Export the `PROJECT_NAMESPACE` variable:

   ```
   export PROJECT_NAMESPACE=<PROJECT_NAMESPACE>
   ```

Run the following commands on the **managed** or **attached** cluster. For this, ensure you switch to the correct context or kubeconfig[527] of the cluster for the following `kubectl` commands:

5. Check the deployment status using this command on the attached cluster:

```
kubectl get helmreleases -n ${PROJECT_NAMESPACE}
```

NOTE: It may take some time for these changes to take effect, based on the duration configured for the Flux GitRepository reconciliation.
When successfully deployed, you will see output that includes the following `HelmReleases`:

```
NAMESPACE              NAME                                  READY   STATUS
AGE
${PROJECT_NAMESPACE}   project-grafana-logging               True    Release
reconciliation succeeded   15m
${PROJECT_NAMESPACE}   project-grafana-loki                  True    Release
reconciliation succeeded   11m
${PROJECT_NAMESPACE}   project-loki-object-bucket-claims     True    Release
reconciliation succeeded   11m
```

Then, you can View Project Log Data (see page 815) within multi-tenant logging.

6.8.3.6 View Project Log Data

Enterprise Gov Advanced

How to view project log data within multi-tenant logging

You can only view the log data for a Project to which you have been granted access.

6.8.3.6.1 Access Project Grafana's UI

Run the following commands on the **management** cluster:

1. Determine the namespace of the workspace that your project is in. You can use the `dkp get workspaces` command to see the list of workspace names and their corresponding namespaces.

```
dkp get workspaces
```

527 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

Copy the value under the `NAMESPACE` column for your workspace.

2. Export the `WORKSPACE_NAMESPACE` variable:

```
export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
```

3. Execute the following command to get the namespace of your project

```
kubectl get projects -n ${WORKSPACE_NAMESPACE}
```

Copy the value under `PROJECT NAMESPACE` column for your project. This may NOT be identical to the Display Name of the `Project`.

4. Export the `PROJECT_NAMESPACE` variable:

```
export PROJECT_NAMESPACE=<PROJECT_NAMESPACE>
```

Run the following commands **on the attached cluster** to access the Project Grafana's UI:

Ensure you switched to the correct context or kubeconfig[528] of the attached cluster for the following kubectl commands:

5. Get the Grafana URL:

```
kubectl get ingress -n ${PROJECT_NAMESPACE} ${PROJECT_NAMESPACE}-project-
grafana-logging -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}{{with
index .spec.rules 0}}{{with index .http.paths 0}}{{.path }}{{end}}{{end}}
{{"\n"}}'
```

6.8.3.6.2 View Logs in Grafana

1. Go to the `Explore` tab:

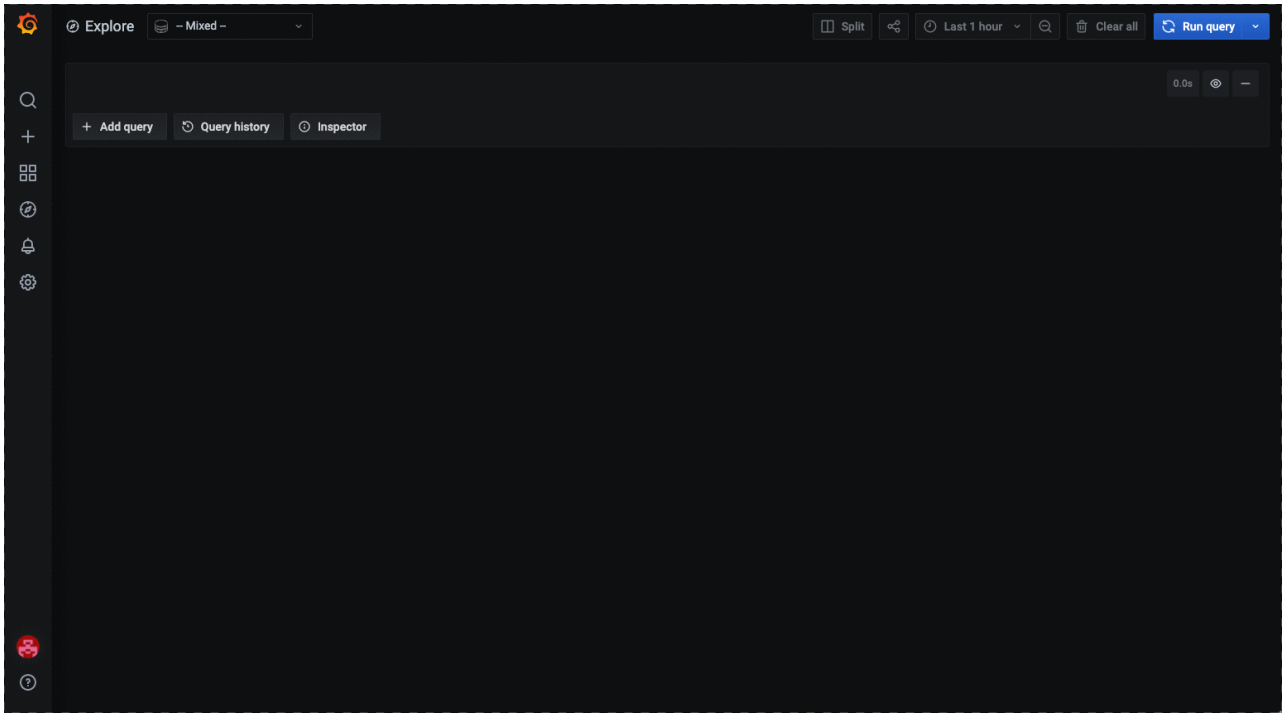
```
kubectl get ingress -n ${PROJECT_NAMESPACE} ${PROJECT_NAMESPACE}-project-
grafana-logging -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}{{with
index .spec.rules 0}}{{with index .http.paths 0}}{{.path }}{{end}}{{end}}/
explore{{"\n"}}'
```

2. You may be prompted to log in using the SSO flow. See Authentication and Authorization (see page 829) for more information.

528 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

3. At the top of the page, change the data source to `Loki`.

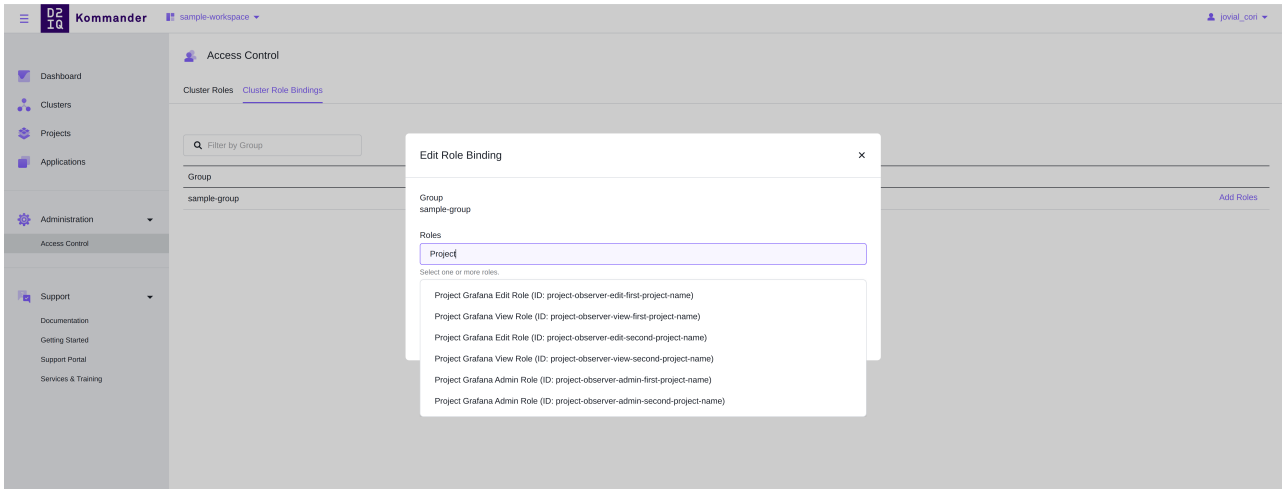
See the Grafana Loki documentation[529] for more on how to use the interface to view and query logs.



> ⚠ Cert-Manager and Traefik must be deployed in the attached cluster to be able to access the Grafana UI. These are deployed by default on the workspace.

You can configure workspace policy (see page 796) to restrict access to the Project logging Grafana UI. Each Grafana instance in a Project has a unique URL at the cluster level. Consider creating a `WorkspaceRoleBinding` that maps to a `ClusterRoleBinding`, on attached cluster(s), for each Project level Grafana instance. For example, If you have a group named `sample-group` and two projects named `first-project` and `second-project` in `sample-workspace` workspace, then the Role Bindings look similar to the following:

529 https://grafana.com/docs/grafana/v7.5/datasources/loki/



Select the correct role bindings for each group for a project at the workspace level.

6.8.4 Fluent Bit

Fluent Bit[530] is the DKP choice of open-source log collection and forwarding tool.

On the Management cluster, the Fluentbit application is disabled by default. The amount of admin logs ingested to Loki requires additional disk space to be configured on the `rook-ceph-cluster`. Enabling admin logs may use around 2GB/day per node. See Rook Ceph in DKP (see page 894) for more details on how to configure the Rook Ceph Cluster.

6.8.4.1 Audit Log Collection

Auditing[531] in Kubernetes provides a way to chronologically document the actions taken on a cluster. On Kommander, by default, audit logs are collected and stored for quick indexing. Viewing and accessing can be done via the Grafana logging UI.

To adjust the default Audit Policy log backend[532] configuration, you must modify the log retention settings by Configuring the Control Plane (see page 1351) before creating the cluster. This needs to be done prior to creating the cluster since it cannot be edited after creation.

- Collecting systemd Logs from a Non-default Path (see page 819)

530 https://fluentbit.io/
531 https://kubernetes.io/docs/tasks/debug-application-cluster/audit/
532 https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/#log-backend

6.8.4.2 Related Information

For information on related topics or procedures, refer to the following:

- Logging (see page 796)

6.8.4.3 Collecting systemd Logs from a Non-default Path

By default, Fluent Bit pods are configured to collect `systemd` logs from the `/var/log/journal/` path on cluster nodes.

If `systemd-journald` running as a part of the OS on the nodes uses a different path for writing logs, you will need to override configuration of the `fluent-bit` AppDeployment to make Fluent Bit collect `systemd` logs.

To configure the Fluent Bit AppDeployment to collect `systemd` logs from a non-default path, follow these steps (all `kubectl` and `dkp` invocations refer to the **management** cluster):

1. Execute the following command to get the namespace of the workspace in which you would like to configure Fluent Bit:

   ```
   dkp get workspaces
   ```

   And copy the value under the `NAMESPACE` column for your workspace.

2. Set the `WORKSPACE_NAMESPACE` variable to the namespace copied in the previous step:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
   ```

3. Identify the `systemd-journald` log data storage path on the nodes of the clusters in the workspace by using the OS documentation and examining the `systemd` configuration.

   Usually it will be either `/var/log/journal` (typically used when `systemd-journald` is configured to store logs permanently; in this case the default Fluent Bit configuration should work) or `/run/log/journal` (typically used when `systemd-journald` is configured to use a volatile storage).

4. Extract the default Helm values used by the Fluent Bit App:

   ```
   kubectl get -n ${WORKSPACE_NAMESPACE} configmaps fluent-bit-0.20.9-d2iq-
   defaults -o=jsonpath='{.data.values\.yaml}' > fluent-bit-values.yaml
   ```

5. Edit the resulting file `fluent-bit-values.yaml` by removing all sections except for `extraVolumes`, `extraVolumeMounts` and `config.inputs`. The result should look similarly to this:

```
extraVolumes:
# we create this to have a persistent tail-db directory an all nodes
# otherwise a restarted fluent-bit would rescrape all tails
- name: tail-db
  hostPath:
    path: /var/log/tail-db
    type: DirectoryOrCreate
# we create this to get rid of error messages that would appear on non control-
plane nodes
- name: kubernetes-audit
  hostPath:
    path: /var/log/kubernetes/audit
    type: DirectoryOrCreate
# needed for kmsg input plugin
- name: uptime
  hostPath:
    path: /proc/uptime
    type: File
- name: kmsg
  hostPath:
    path: /dev/kmsg
    type: CharDevice

extraVolumeMounts:
- name: tail-db
  mountPath: /tail-db
- name: kubernetes-audit
  mountPath: /var/log/kubernetes/audit
- name: uptime
  mountPath: /proc/uptime
- name: kmsg
  mountPath: /dev/kmsg

config:
  inputs: |
    # Collect audit logs, systemd logs, and kernel logs.
    # Pod logs are collected by the fluent-bit deployment managed by logging-
operator.
    [INPUT]
        Name tail
        Alias kubernetes_audit
        Path /var/log/kubernetes/audit/*.log
        Parser kubernetes-audit
        DB /tail-db/audit.db
        Tag audit.*
        Refresh_Interval 10
```

```
        Rotate_Wait 5
        Mem_Buf_Limit 135MB
        Buffer_Chunk_Size 5MB
        Buffer_Max_Size 20MB
        Skip_Long_Lines Off
    [INPUT]
        Name systemd
        Alias kubernetes_host
        DB /tail-db/journal.db
        Tag host.*
        Max_Entries 1000
        Read_From_Tail On
        Strip_Underscores On
    [INPUT]
        Name kmsg
        Alias kubernetes_host_kernel
        Tag kernel
```

6. Add the following item to the list under the `extraVolumes` key:

```
- name: kubernetes-host
  hostPath:
    path: <path to systemd logs on the node>
    type: Directory
```

7. Add the following item to the list under the `extraVolumeMounts` key:

```
- name: kubernetes-host
  mountPath: <path to systemd logs on the node>
```

These items will make Kubernetes mount systemd logs into Fluent Bit pods.

8. Add the following line into the `[INPUT]` entry identified by `Name systemd` and `Alias kubernetes_host`.

```
Path <path to systemd logs on the node>
```

This is needed to make Fluent Bit actually collect the mounted logs

9. Assuming that the path to systemd logs on the node is `/run/log/journal`, the result will look similarly to this:

```
extraVolumes:
# we create this to have a persistent tail-db directory an all nodes
# otherwise a restarted fluent-bit would rescrape all tails
- name: tail-db
```

```
  hostPath:
    path: /var/log/tail-db
    type: DirectoryOrCreate
# we create this to get rid of error messages that would appear on non control-
plane nodes
- name: kubernetes-audit
  hostPath:
    path: /var/log/kubernetes/audit
    type: DirectoryOrCreate
# needed for kmsg input plugin
- name: uptime
  hostPath:
    path: /proc/uptime
    type: File
- name: kmsg
  hostPath:
    path: /dev/kmsg
    type: CharDevice
- name: kubernetes-host
  hostPath:
    path: /run/log/journal
    type: Directory

extraVolumeMounts:
- name: tail-db
  mountPath: /tail-db
- name: kubernetes-audit
  mountPath: /var/log/kubernetes/audit
- name: uptime
  mountPath: /proc/uptime
- name: kmsg
  mountPath: /dev/kmsg
- name: kubernetes-host
  mountPath: /run/log/journal

config:
  inputs: |
    # Collect audit logs, systemd logs, and kernel logs.
    # Pod logs are collected by the fluent-bit deployment managed by logging-
operator.
    [INPUT]
        Name tail
        Alias kubernetes_audit
        Path /var/log/kubernetes/audit/*.log
        Parser kubernetes-audit
        DB /tail-db/audit.db
        Tag audit.*
        Refresh_Interval 10
        Rotate_Wait 5
        Mem_Buf_Limit 135MB
        Buffer_Chunk_Size 5MB
        Buffer_Max_Size 20MB
```

```
        Skip_Long_Lines Off
    [INPUT]
        Name systemd
        Alias kubernetes_host
        Path /run/log/journal
        DB /tail-db/journal.db
        Tag host.*
        Max_Entries 1000
        Read_From_Tail On
        Strip_Underscores On
    [INPUT]
        Name kmsg
        Alias kubernetes_host_kernel
        Tag kernel
```

10. Create a `ConfigMap` manifest with override values from `fluent-bit-values.yaml`:

```
cat <<EOF >fluent-bit-overrides.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: fluent-bit-overrides
data:
  values.yaml: |
$(cat fluent-bit-values.yaml | sed 's/^/    /g')
EOF
```

11. Create a `ConfigMap` from the manifest above:

```
kubectl apply -f fluent-bit-overrides.yaml
```

12. Edit the `fluent-bit` AppDeployment to set the value of `spec.configOverrides.name` to the name of the created `ConfigMap`. (You can use the steps in the procedure, Deploy an Application with a Custom Configuration (see page 819) as a guide.)

```
dkp edit appdeployment -n ${WORKSPACE_NAMESPACE} fluent-bit
```

After your editing is complete, the AppDeployment resembles this example:

```
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: fluent-bit
  namespace: ${WORKSPACE_NAMESPACE}
```

```
spec:
  appRef:
    name: fluent-bit-0.20.9
    kind: ClusterApp
  configOverrides:
    name: fluent-bit-overrides
```

13. Log in into the Grafana logging UI of your workspace and verify that logs with a label `log_source=kubernetes_host` are now present in Loki.

6.8.5 Scaling the Logging Stack

6.8.5.1 Introduction

Depending on the application workloads you run on your clusters, you may find that the default settings for the DKP logging stack do not meet your needs. In particular, if your workloads produce lots of log traffic, you may find you need to adjust the logging stack components to properly capture all the log traffic. Follow the suggestions below to tune the logging stack components as needed.

6.8.5.2 Logging Operator

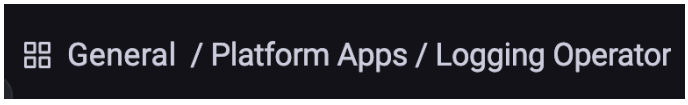
In a high log traffic environment, `fluentd` usually becomes the bottleneck of the logging stack. According to https://banzaicloud.com/docs/one-eye/logging-operator/operation/scaling/:

The typical sign of this is when `fluentd` *cannot handle its* ***buffer***[533] *directory size growth for more than the configured or calculated (timekey + timekey_wait) flush interval.*

For metrics to monitor, please refer to https://docs.fluentd.org/monitoring-fluentd/monitoring-prometheus#metrics-to-monitor.

6.8.5.2.1 Grafana dashboard

In DKP, if the `Prometheus Monitoring` (kube-prometheus-stack) platform application is enabled, you can view the Logging Operator dashboard in the Grafana UI.



You can also improve fluentd throughput by disabling the buffering for `loki` clusterOutput.

6.8.5.2.2 Example Configuration

You can see an example configuration of the logging operator in Logging Stack Application Sizing Recommendations (see page 549).

533 https://banzaicloud.com/docs/one-eye/logging-operator/configuration/plugins/outputs/buffer/

For more information, refer to:

- https://docs.fluentd.org/deployment/performance-tuning-single-process

6.8.5.3 Grafana Loki

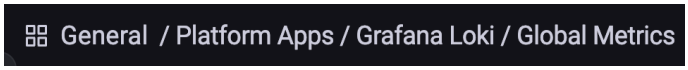
DKP deploys Loki in Microservice mode[534] – this provides you with the highest flexibility in terms of scaling.

In a high log traffic environment, we recommend:

- Ingester should be the first component to be considered for scaling up.
- Distributor should be scaled up only when the existing Distributor is experiencing stress due to high computing resource usage.
    - Usually, the number of Distributor pods should be much lower than the number of Ingester pods

6.8.5.3.1 Grafana dashboard

In DKP, if `Prometheus Monitoring` (kube-prometheus-stack) platform app is enabled, you can view the Loki dashboards in Grafana UI and here is one of the Loki dashboard:



6.8.5.3.2 Example Configuration

You can see an example config of Loki at Logging Stack Application Sizing Recommendations (see page 549).

For more information, refer to:

- https://grafana.com/docs/loki/latest/fundamentals/architecture/components/
- https://grafana.com/docs/loki/latest/operations/scalability/
- https://grafana.com/docs/loki/latest/best-practices/

6.8.5.4 Rook Ceph

Ceph is the default S3 storage provider. In DKP, a Rook Ceph Operator and a Rook Ceph Cluster are deployed together to have a Ceph Cluster.

6.8.5.4.1 Storage

The default configuration of Rook Ceph Cluster in DKP has a 33% overhead in data storage for redundancy. Meaning, if the data disks allocated for your Rook Ceph Cluster is 1000Gb, 750Gb will be used to store your data. Thus, it is important to account for that in planning the capacity of your data disks to prevent issues.

534 https://grafana.com/docs/loki/latest/fundamentals/architecture/deployment-modes/#microservices-mode

6.8.5.4.1.1 ObjectBucketClaim storage limit

ObjectBucketClaim has a storage limit option to prevent your S3 bucket from growing over a limit. In DKP this is enabled by default.

Thus, after you size up your Rook Ceph Cluster for more storage, it is important to also increase the storage limit of your ObjectBucketClaims of your `grafana-loki` and/or `project-grafana-loki`.

To change it for `grafana-loki`, please provide an override configmap in `rook-ceph-cluster` platform app to override `dkp.grafana-loki.maxSize`

To change it for `project-grafana-loki`, please provide an override configmap in `project-grafana-loki` platform app to override `dkp.project-grafana-loki.maxSize`

6.8.5.4.2 Example Configuration

You can see an example config at Rook Ceph Cluster Sizing Recommendations (see page 554).

6.8.5.4.3 Ceph OSD CPU considerations

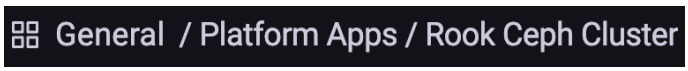
ceph-osd is the object storage daemon for the Ceph distributed file system. It is responsible for storing objects on a local file system and providing access to them over the network.

If you determine that the Ceph OSD component is the bottleneck, then you may wish to consider increasing the CPU allocated to it.

See this page for more info: https://ceph.io/en/news/blog/2022/ceph-osd-cpu-scaling/

6.8.5.4.4 Grafana dashboard

In DKP, if the `Prometheus Monitoring` (kube-prometheus-stack) platform app is enabled, you can view the Ceph dashboards in the Grafana UI. Below is one of the Ceph dashboard:



6.8.5.5 Audit Log

6.8.5.5.1 Overhead

Enabling audit logging requires additional computing and storage resources.
When you enable audit logging by enabling the `kommander-fluent-bit` AppDeployment, inbound traffic to the logging stack increases the log traffic by approximately 3-4 more times.

Thus, when enabling the audit log, consider scaling up all components in the logging stack mentioned above.

6.8.5.5.2 Fine-tuning audit log Fluent Bit

If you are certain that you only need to collect a subset of the logs that the default config[535] makes the `kommander-fluent-bit` pods collect, you can add your own override configmap to `kommander-fluent-bit` with proper Fluent Bit `INPUT` , `FILTER` , `OUTPUT` settings. This helps reduce the audit log traffic.

To see the default configuration of Fluent Bit, see the Release Notes (see page 1537) > **Components and Applications**.

For more information, refer to:

- Admin-level logs (see page 798)
- https://docs.fluentbit.io/manual/administration/configuring-fluent-bit/classic-mode/configuration-file

6.8.6 Configuring Loki to use AWS S3 Storage in DKP

Follow the instructions on this page to configure Loki to use AWS S3 Storage in DKP

6.8.6.1 Configuring Loki

The easiest way to get started with using AWS S3 storage with Grafana Loki is to use a set of static AWS credentials.

1. Execute the following command to get the namespace of your workspace:

   ```
   dkp get workspaces
   ```

2. Set the `WORKSPACE_NAMESPACE` variable to the namespace copied in the previous step:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE
   ```

3. Create a secret containing the static AWS S3 credentials. The secret is then mounted into each of the Grafana Loki pods as environment variables.

   ```
   kubectl create secret generic dkp-aws-s3-creds -n${WORKSPACE_NAMESPACE} \
   --from-literal=AWS_ACCESS_KEY_ID=<key id> \
   --from-literal=AWS_SECRET_ACCESS_KEY=<secret key>
   ```

4. Create a config overrides ConfigMap to update the storage configuration.
   NOTE: This can also be added to the installer configuration if you are configuring Grafana Loki on the Management Cluster.

535 https://github.com/mesosphere/kommander-applications/blob/v2.5.0/services/fluent-bit/0.21.6/defaults/cm.yaml

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: grafana-loki-overrides
  namespace: ${WORKSPACE_NAMESPACE}
data:
  values.yaml: |
    loki:
      annotations:
        secret.reloader.stakater.com/reload: dkp-aws-s3-creds
      structuredConfig:
        storage_config:
          aws:
            s3: s3://<region>/<bucket name>
    ingester:
      extraEnvFrom:
        - secretRef:
            name: dkp-aws-s3-creds
    querier:
      extraEnvFrom:
        - secretRef:
            name: dkp-aws-s3-creds
    queryFrontend:
      extraEnvFrom:
        - secretRef:
            name: dkp-aws-s3-creds
    compactor:
      extraEnvFrom:
        - secretRef:
            name: dkp-aws-s3-creds
    ruler:
      extraEnvFrom:
        - secretRef:
            name: dkp-aws-s3-creds
    distributor:
      extraEnvFrom:
        - secretRef:
            name: dkp-aws-s3-creds
EOF
```

5. Update the `grafana-loki` AppDeployment to apply the configuration override.
   NOTE: If you use the Kommander CLI installation configuration file, you don't need this step

```
cat << EOF | kubectl -n ${WORKSPACE_NAMESPACE} patch appdeployment grafana-loki
--type="merge" --patch-file=/dev/stdin
spec:
  configOverrides:
    name: grafana-loki-overrides
```

```
EOF
```

The reloader annotation only works in DKP 2.5.2+. Any changes to the AWS credential secret will not automatically reload some Loki pods. In this scenario, The ingester and compactor pods need to be manually restarted.
For more information, refer to https://github.com/grafana/helm-charts/issues/1905.

6.9 Security

Security architecture.

- Authentication and authorization architecture (see page 829)
- OpenID Connect (OIDC) Introduction (see page 830)
- SAML Connector (see page 834)
- Policy Controls (see page 837)
- Traefik-Forward-Authentication in DKP (TFA) (see page 841)

6.9.1 Authentication and authorization architecture

6.9.1.1 Details on distributed authentication and authorization between clusters

6.9.1.2 Authentication

DKP UI comes with a pre-configured authentication Dex[536] identity broker and provider.

Kubernetes, Kommander, and Dex do not store any user identities. The Kommander installation comes with default admin static credentials. These credentials should only be used to access the **DKP UI** for configuring an external identity provider. Currently, there is no way to update these credentials, so they should be treated as backup credentials and not used for normal access.

The DKP UI admin credentials are stored as a secret. They never leave the boundary of the UI cluster and are never shared to any other cluster.

536 https://github.com/dexidp/dex

The Dex service issues an OIDC ID token[537] on successful user authentication. Other platform applications use the ID token as an authentication proof. User identity to the Kubernetes API server is provided by the `kube-oidc-proxy` platform service that reads the identity from an ID token. Web requests to DKP UI access are authenticated by the traefik forward auth[538] platform service.

The kube-oidc-proxy[539] service authenticates kubectl CLI requests using the Kubernetes API Server Go library. This library requires that if an **email_verified** claim is present, it must be set to **true**, even if the `insecureSkipEmailVerified: true` flag is configured in the Dex connector. Thus, ensure that the OIDC provider is configured to set the email_verified field to 'true'.

A user identity is shared across a UI cluster and all other attached clusters.

6.9.1.2.1 Attached clusters

A newly attached cluster has federated `kube-oidc-proxy`, `dex-k8s-authenticator`, and `traefik-forward-auth` platform applications. These platform applications are configured to accept UI cluster Dex issued ID tokens.

When the `traefik-forward-auth` is used as a Traefik Ingress authenticator[540], it checks if the user identity was issued by the Kommander cluster Dex service. An anonymous user is redirected to the UI cluster Dex service to authenticate and confirm their identity.

Never enter your own credentials on any of the attached clusters. On the UI cluster use the static admin credentials or an external identity provider (IDP).

6.9.1.3 Authorization

There is no centralized authorization component in Kommander. Each component and service makes its own authorization decisions based on user identity.

6.9.2 OpenID Connect (OIDC) Introduction

6.9.2.1 An introduction to OpenID Connect (OIDC) Authentication in Kubernetes

All Kubernetes clusters have two categories of users: service accounts and normal users. Kubernetes manages authentication for service accounts, but the cluster administrator, or a separate service, manages authentication for normal users.

---

537 https://openid.net/specs/openid-connect-core-1_0.html#IDToken
538 https://github.com/mesosphere/traefik-forward-auth
539 https://github.com/jetstack/kube-oidc-proxy
540 https://docs.traefik.io/v2.4/providers/kubernetes-ingress/

Kommander configures the cluster to use OpenID Connect (OIDC), a popular and extensible user authentication method, and installs Dex, a popular, open-source software product that integrates your existing Identity Providers with Kubernetes.

To begin, set up an Identity Provider with Dex, then use OIDC as the Authentication method.

6.9.2.2 Identity Provider

An Identity Provider (IdP) is a service that lets you manage identity information for users, including groups. A cluster created in Kommander uses Dex as its IdP. Dex, in turn, delegates to one or more external IdPs.

If you use already use one or more of the following IdPs, you can configure Dex to use them:

Name	Supports Refresh Tokens	Supports Groups Claim	Supports `preferred_username` Claim	Status	Notes
LDAP[541]	yes	yes	yes	stable	
GitHub[542]	yes	yes	yes	stable	
SAML 2.0[543]	no	yes	no	stable	
GitLab[544]	yes	yes	yes	beta	
OpenID Connect[545]	yes	yes	yes	beta	Includes Salesforce, Azure, etc.
Google[546]	yes	yes	yes	alpha	
LinkedIn[547]	yes	no	no	beta	
Microsoft[548]	yes	yes	no	beta	

541 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/ldap.md
542 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/github.md
543 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/saml.md
544 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/gitlab.md
545 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/oidc.md
546 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/google.md
547 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/linkedin.md
548 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/microsoft.md

Name	Supports Refresh Tokens	Supports Groups Claim	Supports `preferred_username` Claim	Status	Notes
AuthProxy[549]	no	no	no	alpha	Authentication proxies such as Apache2 mod_auth, etc.
Bitbucket Cloud[550]	yes	yes	no	alpha	
OpenShift[551]	no	yes	no	stable	

NOTE: These are the Identity Providers supported by Dex 2.22.0[552], the version used by DKP.

6.9.2.3 Add Login Connectors

Kommander uses Dex to provide OpenID Connect single sign-on (SSO) to the cluster. Dex can be configured to use multiple connectors, including GitHub, LDAP, and SAML 2.0. The Dex Connector documentation[553] describes how to configure different connectors. You can add the configuration as the values field in the Dex application. An example Dex configuration provided to the Kommander CLI's `install` command would look similar to this:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  dex:
    values: |
      config:
        connectors:
        - type: oidc
          id: google
          name: Google
          config:
            issuer: https://accounts.google.com/o/oauth2/v2/auth
            clientID: YOUR_CLIENT_ID
            clientSecret: YOUR_CLIENT_SECRET
```

549 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/authproxy.md
550 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/bitbucketcloud.md
551 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/openshift.md
552 https://github.com/dexidp/dex/blob/v2.22.0/README.md
553 https://dexidp.io/docs/connectors/

```
            redirectURI: https://DKP_CLUSTER_DOMAIN/dex/callback
            scopes:
            - openid
            - profile
            - email
            insecureSkipEmailVerified: true
            insecureEnableGroups: true
            userIDKey: email
            userNameKey: email
[...]
```

6.9.2.4 Change the Access Token Lifetime

By default, the client access token lifetime is 24 hours. After this time, the token expires and cannot be used to authenticate. See the Dex documentation[554] for more information on access token expiration and rotation settings.

Here is an example configuration for extending the token lifetime to 48 hours:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  dex:
    values: |
      config:
        expiry:
          idTokens: "48h"
[...]
```

6.9.2.5 Authentication

OpenID Connect is an extension of the OAuth2 authentication protocol[555]. As required by OAuth2, the client must be registered with Dex. Do this by passing the name of the application and a callback/redirect URI. These handle the processing of the OpenID token after the user authenticates successfully. After registration, Dex returns a `client_id` and a `secret`. Authentication requests use these between the client and Dex to identify the client.

Users access Kommander in two ways:

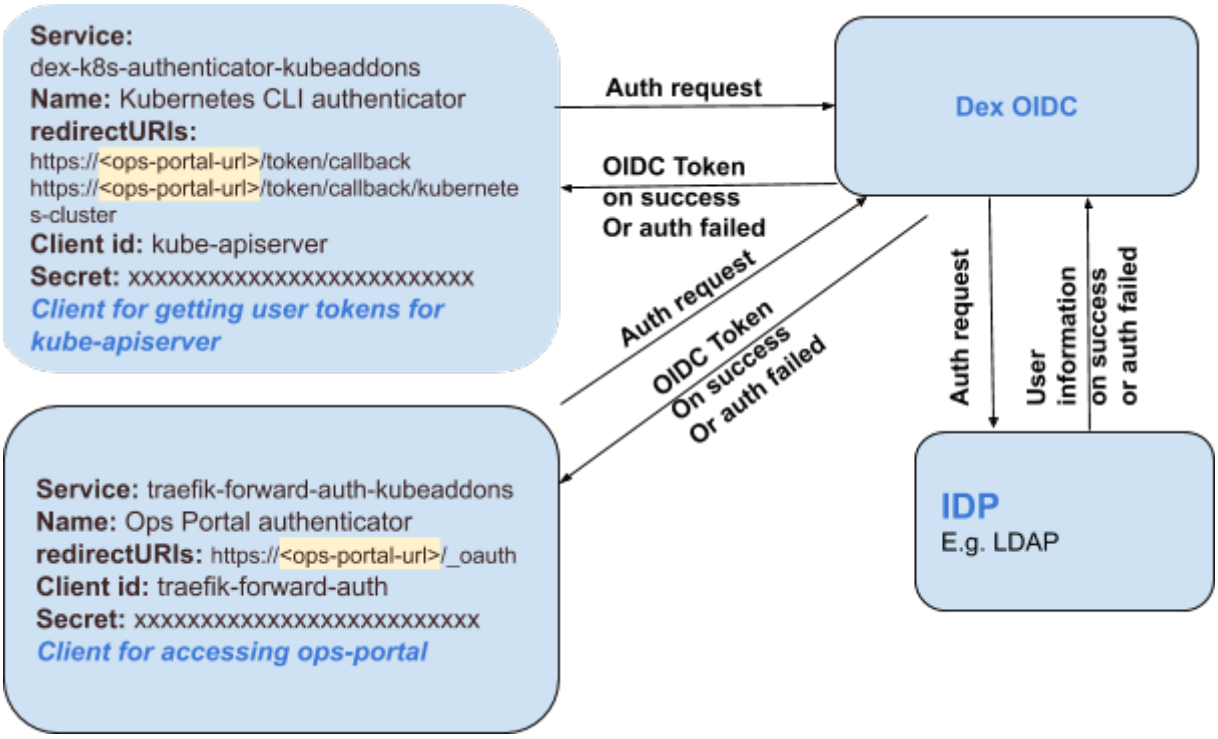
- To interact with Kubernetes API, usually through `kubectl`.
- To interact with the DKP UI, which has GUI dashboards for Prometheus, Grafana, etc.

In Kommander, Dex comes pre-configured with a client for these access use cases. The clients talk to Dex for authentication. Dex talks to the configured Identity Provider, or IdP, (for example LDAP, SAML, etc) to perform the actual task of authenticating the user.

554 https://dexidp.io/docs/id-tokens/#expiration-and-rotation-settings
555 https://oauth.net/2/

If the user authenticates successfully, Dex pulls the user's information from the IdP and forms an OpenID token. The token contains this information and returns it to the respective client's callback URL. The client or end user uses this token for communicating with the DKP UI or Kubernetes API respectively.

This figure illustrates these components and their interaction at a high level:



6.9.3 SAML Connector

6.9.3.1 Connect your Kommander cluster to an IdP using SAML

6.9.3.2 Connect Kommander to an IdP Using SAML

This procedure configures your Kommander cluster to use SAML, to connect to an identity provider (IdP).

1. Install DKP (see page 117).
2. Configure the IdP

   Provide the issuer URL and the Assertion Consumer Service (ACS) or callback URL to your IdP. The issuer URL points to the authentication endpoint at the service provider (Dex), which issues a request towards the IdP via the user agent.

   The issuer URL follows this schema:

   ```
   https://<your-cluster-host>/dex
   ```

The ACS URL points to the service provider (Dex) endpoint that receives SAML assertions issued by the IdP.

The ACS or callback URL should look like this:

```
https://<your-cluster-host>/dex/callback
```

Depending on the IdP, you might be asked to provide the configuration in some form of an XML snippet. See the following example, making sure to replace `<your-cluster-host>` with your URL:

```
<?xml version="1.0" encoding="UTF-8"?>
<EntityDescriptor xmlns="urn:oasis:names:tc:SAML:2.0:metadata" entityID="https:
//<your-cluster-host>/dex">
    <SPSSODescriptor AuthnRequestsSigned="false" WantAssertionsSigned="true"
 protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
        <NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-format:persistent</
NameIDFormat>
        <AssertionConsumerService index="0" isDefault="true" Binding="urn:oasis
:names:tc:SAML:2.0:bindings:HTTP-POST" Location="https://<your-cluster-host>/
dex/callback" />
    </SPSSODescriptor>
</EntityDescriptor>
```

3. Modify the `dex` configuration:

   For this step, get the following from your IdP:

   - single sign-on URL or SAML URL -> `ssoURL`
   - base64 encoded, PEM encoded CA certificate -> `caData`
   - username attribute name in SAML response -> `usernameAttr`
   - email attribute name in SAML response -> `emailAttr`

   From above you need:

   - issuer URL -> `entityIssuer`
   - callback URL -> `redirectURI`

   Ensure you base64 encode the contents of the PEM file. As an example, the prefix of the contents will result into this exact base64 prefix:

   ```
   LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tC[...]
   ```

   You can add the configuration as the values field in the `dex` application. An example `dex` configuration provided to the Kommander CLI's install command (see page 1227) should look similar to:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  dex:
    values: |
      config:
      connectors:
      - type: saml
          id: saml
          name: SAML
          config:
            ssoURL: < url for POST request >
            caData: < base64 PEM encoded CA for the IdP server >
            redirectURI: https://<your-cluster-host>/dex/callback
            entityIssuer: https://<your-cluster-host>/dex
            usernameAttr: < user attribute in saml response >
            emailAttr: < email attribute in saml response >
[...]
```

4. Modify the `traefik-foward-auth-mgmt` configuration and add a whitelist:

   This step is required to give access to a user to the DKP UI. For each user, you must give Access to Kubernetes resources (see page 513) and add an entry in the `whitelist` below.

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
...
  traefik-forward-auth-mgmt:
    values: |
      traefikForwardAuth:
        allowedUser:
          valueFrom:
            secretKeyRef: null
        whitelist:
        - < allowed email addresses >
```

5. Run `kommander install --installer-config kommander.yaml` to deploy modified `dex`.
6. Visit `https://<your-cluster-host>/dkp/kommander/dashboard` to login to the DKP UI.
7. Select `Launch Console` and follow the authentication steps to complete the procedure.

6.9.4 Policy Controls

6.9.4.1 Workload Policy Controls

6.9.4.2 Enforce Policies Using Gatekeeper

6.9.4.2.1 Learn how to enforce policies using Gatekeeper

Gatekeeper[556] is the policy controller for Kubernetes, allowing organizations to enforce configurable policies using the Open Policy Agent[557], a policy engine for Cloud Native environments hosted by CNCF as a graduated-level project.

This tutorial describes how to use Gatekeeper to enforce policies by rejecting non-compliant resources. Specifically, this tutorial describes two constraints as a way to use Gatekeeper as an alternative to Pod Security Policies[558]:

- Prevent the running of privileged pods (see page 0)
- Prevent mounting host path volumes (see page 0)

6.9.4.2.2 Before you Begin

Before starting this tutorial, verify the following:

- You must have access to a Linux, macOS, or Windows computer with a supported operating system version.
- You must have a properly deployed and running cluster. For information about deploying Kubernetes with default settings on different types of infrastructures, see the Choose Infrastructure[559] topic.
- If you install Kommander with a custom configuration, make sure you enabled Gatekeeper.

6.9.4.2.3 Use Gatekeeper

Gatekeeper uses the OPA Constraint Framework[560] to describe and enforce policy. Before you can define a constraint, you must first define a `ConstraintTemplate`, which describes both the `Rego` (a powerful query language) that enforces the constraint and the schema of the constraint. The schema of the constraint allows an admin to fine-tune the behavior of a constraint, much like arguments to a function.

The Gatekeeper repository includes a library of policies[561] to replace Pod Security Policies which you will use in the following tutorials.

---

556 https://github.com/open-policy-agent/gatekeeper
557 https://github.com/open-policy-agent/opa
558 https://kubernetes.io/docs/concepts/policy/pod-security-policy/
559 https://docs.d2iq.com/dkp/konvoy/2.2/choose-infrastructure/
560 https://github.com/open-policy-agent/frameworks/tree/master/constraint
561 https://github.com/open-policy-agent/gatekeeper-library/tree/master/library/pod-security-policy

6.9.4.2.3.1 Prevent privileged pods

Define the ConstraintTemplate

Create the privileged pod policy constraint template `k8spspprivilegedcontainer` by running the following command:

```
kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper-library/master/library/pod-security-policy/privileged-containers/template.yaml
```

Define the Constraint

Constraints are then used to inform Gatekeeper that the admin wants to enforce a ConstraintTemplate, and how.

Create the privileged pod policy constraint `psp-privileged-container` by running the following command:

```
kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper-library/master/library/pod-security-policy/privileged-containers/samples/psp-privileged-container/constraint.yaml
```

Test that the constraint is enforced

Create a privileged pod by running the following command:

```
kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper-library/master/library/pod-security-policy/privileged-containers/samples/psp-privileged-container/example_disallowed.yaml
```

You should see the following output:

```
Error from server ([denied by psp-privileged-container] Privileged container is not allowed: nginx, securityContext: {"privileged": true}): error when creating "https://raw.githubusercontent.com/open-policy-agent/gatekeeper-library/master/library/pod-security-policy/privileged-containers/samples/psp-privileged-container/example_disallowed.yaml": admission webhook "validation.gatekeeper.sh" denied the request: [denied by psp-privileged-container] Privileged container is not allowed: nginx, securityContext: {"privileged": true}
```

6.9.4.2.3.2 Prevent host path volumes

Define the ConstraintTemplate

Create the host path volume policy constraint template `k8spsphostfilesystem` by running the following command:

```
kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper-library/master/library/pod-security-policy/host-filesystem/template.yaml
```

Define the Constraint

Constraints are then used to inform Gatekeeper that the admin wants to enforce a ConstraintTemplate, and how.

Create the host path volume policy constraint `psp-host-filesystem` by running the following command to only allow `/foo` to be mounted as a host path volume:

```
cat <<EOF | kubectl apply -f -
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sPSPHostFilesystem
metadata:
  name: psp-host-filesystem
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
  parameters:
    allowedHostPaths:
    - readOnly: true
      pathPrefix: "/foo"
EOF
```

Test that the constraint is enforced

Create a privileged pod by running the following command:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
```

```
kind: Pod
metadata:
  name: nginx-host-filesystem
  labels:
    app: nginx-host-filesystem
spec:
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
      readOnly: true
  volumes:
  - name: cache-volume
    hostPath:
      path: /tmp # directory location on host
EOF
```

You should see the following output:

```
Error from server ([denied by psp-host-filesystem] HostPath volume {"hostPath": {"pat
h": "/tmp", "type": ""}, "name": "cache-volume"} is not allowed, pod: nginx-host-
filesystem. Allowed path: [{"readOnly": true, "pathPrefix": "/foo"}]): error when
creating "STDIN": admission webhook "validation.gatekeeper.sh" denied the request:
[denied by psp-host-filesystem] HostPath volume {"hostPath": {"path": "/tmp", "type":
""}, "name": "cache-volume"} is not allowed, pod: nginx-host-filesystem. Allowed
path: [{"readOnly": true, "pathPrefix": "/foo"}]
```

Test that the constraint allows the allowed host paths

Create a pod that mounts an allowed host path by running the following command:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: nginx-host-filesystem
  labels:
    app: nginx-host-filesystem
spec:
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
```

```
      readOnly: true
  volumes:
  - name: cache-volume
    hostPath:
      path: /foo # directory location on host
EOF
```

You should see the following output:

```
pod/nginx-host-filesystem created
```

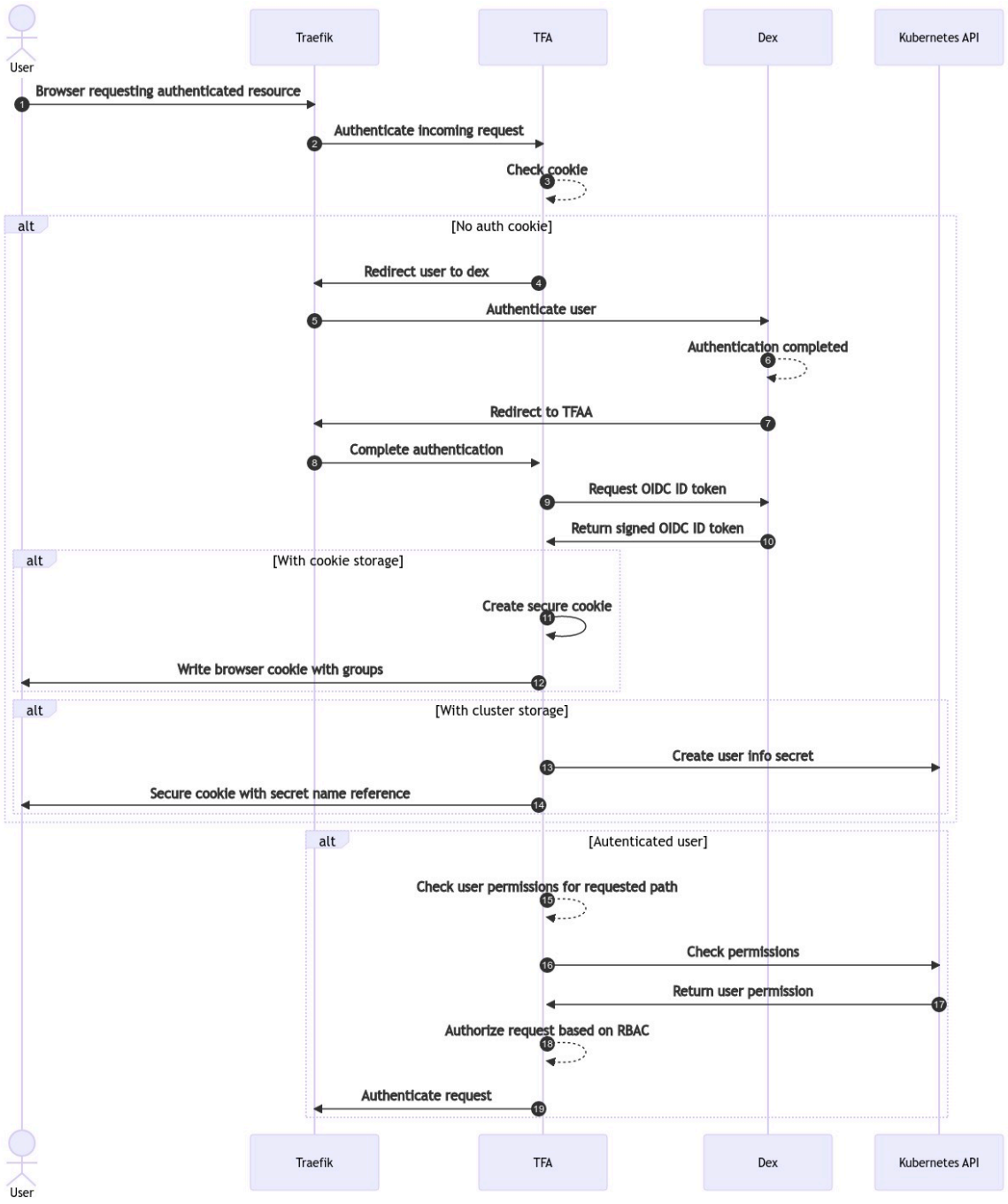
6.9.5 Traefik-Forward-Authentication in DKP (TFA)

6.9.5.1 TFA Overview

Traefik-Forward-Authentication (TFA) is a vital component in DKP. It handles authentication and authorization of web (HTTP) requests and how users can access the DKP UI. Regardless of the authentication method you choose (user and password, SSO, 2FA, etc.), TFA is a component that authorizes HTTP clients to enter your environment.

TFA is one of the standard applications in the Kommander component of DKP. It is deployed by a controller on all attached, managed, and management clusters.

6.9.5.2 TFA Authentication Workflow



6.9.5.3 Default TFA Configuration in DKP

In the default configuration in DKP, TFA stores all authentication information about users via the browser cookies. When TFA authenticates users, it stores the user's metadata in encrypted browser cookies.

Subsequent requests following initial authentication will use these cookies to recognize users without the need to re-authenticate them.

NOTE:

- The cookies are securely encrypted, so they cannot be modified by users.
- The cookies contain the RBAC username.
- The cookies contain a list of groups that users are associated with.

6.9.5.4 Cluster Storage Option

The browser cookie storage is limited to a maximum of 4Kb per cookie. However, if a user is assigned to a large number of groups, this limitation can be exceeded and this will return a response 500-internal server error, meaning that the user will be unable to access any web services.

To work around the cookie storage size limit, TFA can be configured to store the metadata claims in the cluster as a Kubernetes secret instead of in the browser. To do so, the `clusterStorage` option can be configured in the Traefik Forward Auth application when installing Kommander.

In order to enable the `clusterStorage` option, add the following to the cluster.yaml when installing Kommander:

```
traefik-forward-auth-mgmt:
  values: |
    clusterStorage:
      enabled: true
      namespace: kommander
```

If the `clusterStorage` feature is enabled, automatic garbage collection will delete the secrets after 12 hours. Keep in mind that enabling this feature will have performance implications for web requests, because TFA needs to load the secret to retrieve the user groups for each HTTP API request. Because of this, we recommend first trying to reduce the number of groups associated users and only enabling this option if that cannot be accomplished.

For additional information about traefik-forward-authentication, see the TFA page on Github[562].

562 https://github.com/mesosphere/traefik-forward-auth

6.10 Networking

6.10.1 Configure networking for Konvoy cluster

This section describes different networking components that come together to form a Konvoy networking stack. It assumes familiarity with Kubernetes networking.

6.10.2 Networking Service

A Service[563] is an API resource that defines a logical set of pods and a policy by which to access them, and is an abstracted manner to expose applications as network services.

Kubernetes gives pods their own IP addresses and a single DNS name for a set of pods. Services are used as entrypoints to load-balance the traffic across the pods. A selector determines the set of Pods targeted by a Service.

For example, if you have a set of pods that each listen on TCP port `9191` and carry a label `app=MyKonvoyApp`, as configured in the following:

```
apiVersion: v1
kind: Service
metadata:
  name: my-konvoy-service
  namespace: default
spec:
  selector:
    app: MyKonvoyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9191
```

This specification creates a new `Service` object named `"my-konvoy-service"`, that targets TCP port `9191` on any pod with the `app=MyKonvoyApp` label.

563 https://kubernetes.io/docs/concepts/services-networking/service/#service-resource

Kubernetes assigns this Service an IP address. In particular, the `kube-proxy` implements a form of virtual IP for Services of type other than `ExternalName`.

- The name of a Service object must be a valid DNS label name.
- A Service is not a Platform Service.

6.10.2.1 Service Topology

Service Topology[564] is a mechanism in Kubernetes to route traffic based upon the Node topology of the cluster. For example, you can configure a Service to route the traffic to endpoints on specific nodes, or even based on the region or availability zone of the node's location.

To enable this new feature in your Kubernetes cluster, use the feature gates `--feature-gates="ServiceTopology=true,EndpointSlice=true"` flag. After enabling, you can control Service traffic routing by defining the `topologyKeys` field in the Service API object.

In the following example, a Service defines `topologyKeys` to be routed to endpoints only in the same zone:

```
apiVersion: v1
kind: Service
metadata:
  name: my-konvoy-service
  namespace: default
spec:
  selector:
    app: MyKonvoyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9191
  topologyKeys:
  - "topology.kubernetes.io/zone"
```

If the value of the `topologyKeys` field does not match any pattern, the traffic is rejected.

564 https://kubernetes.io/docs/concepts/services-networking/service-topology/#using-service-topology

6.10.2.2 EndpointSlices

EndpointSlices[565] are an API resource that appears as a scalable and more manageable solution to network endpoints within a Kubernetes cluster. They allow for distributing network endpoints across multiple resources with a limit of 100 endpoints per EndpointSlice.

An EndpointSlice contains references to a set of endpoints, and the control plane takes care of creating EndpointSlices for any Service that has a selector specified. These EndpointSlices include references to all the pods that match the Service selector.

Like Services, the name of a EndpointSlice object must be a valid DNS subdomain name.

In this example, here's a sample EndpointSlice resource for the example Kubernetes Service:

```
apiVersion: discovery.k8s.io/v1beta1
kind: EndpointSlice
metadata:
  name: konvoy-endpoint-slice
  namespace: default
  labels:
    kubernetes.io/service-name: my-konvoy-service
addressType: IPv4
ports:
- name: http
  protocol: TCP
  port: 80
endpoints:
- addresses:
  - "192.168.126.168"
  conditions:
    ready: true
  hostname: ip-10-0-135-39.us-west-2.compute.internal
  topology:
    kubernetes.io/hostname: ip-10-0-135-39.us-west-2.compute.internal
    topology.kubernetes.io/zone: us-west2-b
```

6.10.2.3 DNS for Services and Pods

Every new Service object in Kubernetes gets assigned a DNS name. The Kubernetes DNS component schedules a DNS name for the pods and services created on the cluster, and then the Kubelets are configured so containers can resolve these DNS names.

Considering previous examples, assume there is a Service named `my-konvoy-service` in the Kubernetes namespace `default`. A Pod running in namespace `default` can look up this service by performing a DNS query for `my-konvoy-service`. A Pod running in namespace `kommander` can look up this service by performing a DNS query for `my-konvoy-service.default`.

565 https://kubernetes.io/docs/concepts/services-networking/endpoint-slices/#endpointslice-resource

In general, a pod has the following DNS resolution:

```
pod-ip-address.namespace-name.pod-name.cluster-domain.example.
```

Similarly, a service has the following DNS resolution:

```
service-name.namespace-name.svc.cluster-domain.example.
```

You can find additional information about all the possible record types and layout here[566].

6.10.2.4 Ingress and Networking

Ingress[567] is an API resource that manages external access to the services in a cluster through HTTP or HTTPS. It offers name-based virtual hosting, SSL termination and load balancing when exposing HTTP/HTTPS routes from outside to services in the cluster.

The traffic policies are controlled by rules as part of the Ingress definition. Each rule defines the following details:

- An optional host to which apply the rules.
- A list of paths or routes which has an associated backend defined with a Service `name`, a port `name` and `number`.
- A backend is a combo of a Service and port names, or a custom resource backend defined as a CRD. Consequently HTTP/HTTPS requests to the Ingress that matches the host and path of the rule are sent to the listed backend.

An example of an Ingress specification is:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: konvoy-ingress
  namespace: default
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /path
        pathType: Prefix
        backend:
          service:
            name: my-konvoy-service
            port:
```

566 https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/
567 https://kubernetes.io/docs/concepts/services-networking/ingress/#what-is-ingress

```
            number: 80
```

In Kommander, you can expose services to the outside world using Ingress objects.

6.10.2.4.1 Ingress Controllers

In contrast with the controllers in the Kubernetes control plane, Ingress controllers are not started with a cluster so you need to choose the desired Ingress controller.

An Ingress controller has to be deployed in a cluster for the Ingress definitions to work.

Kubernetes as a project currently supports and maintains GCE and nginx controllers.

These are four of the most known Ingress controllers[568]:

- HAProxy Ingress[569] is a highly customizable community-driven ingress controller for HAProxy.
- NGINX offers support and maintenance for the NGINX Ingress Controller[570] for Kubernetes.
- Traefik[571] is a fully featured Ingress controller (Let's Encrypt, secrets, http2, websocket), and has commercial support.
- Ambassador API Gateway[572] EXPERIMENTAL is an Envoy based Ingress controller with community and commercial support.

In Kommander, `Traefik` deploys by default as a well-suited Ingress controller.

6.10.2.5 Network Policies

NetworkPolicy is an API resource that controls the traffic flow at port level 3 or 4, or at the IP address level. It enables defining constraints on how a pod communicates with various network services such as `endpoints` and `services`.

A Pod can be restricted to talk to other network services through a selection of the following identifiers:

- Namespaces that have to access. There can be pods that are not allowed to talk to other namespaces.
- Other allowed IP blocks regardless of the node or IP address assigned to the targeted Pod.
- Other allowed Pods.

An example of a NetworkPolicy specification is:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: network-konvoy-policy
  namespace: default
```

568 https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/#additional-controllers
569 https://haproxy-ingress.github.io/
570 https://www.nginx.com/products/nginx-ingress-controller
571 https://github.com/containous/traefik
572 https://www.getambassador.io/

```
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          app: MyKonvoyApp
    - podSelector:
        matchLabels:
          app: MyKonvoyApp
    ports:
    - protocol: TCP
      port: 6379
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 5978
```

As shown in the example, when defining a pod or namespace based NetworkPolicy, you use a selector to specify what traffic is allowed to and from the Pod(s).

6.10.2.5.1 Adding Entries to Pod /etc/hosts with HostAliases

The Pod API resource definition has a `HostAliases` field that allows adding entries to the Pod's container `/etc/hosts` file. This field overrides the hostname resolution when DNS and other options are not applicable.

For example, to resolve `foo.node.local`, `bar.node.local` to `127.0.0.1` and `foo.node.remote`, `bar.node.remote` to `10.1.2.3`, configure the `HostAliases` values as follows:

```
apiVersion: v1
kind: Pod
metadata:
  name: hostaliases-konvoy-pod
spec:
  restartPolicy: Never
  hostAliases:
```

```
  - ip: "127.0.0.1"
    hostnames:
    - "foo.node.local"
    - "bar.node.local"
  - ip: "10.1.2.3"
    hostnames:
    - "foo.node.remote"
    - "bar.node.remote"
  containers:
  - name: cat-hosts
    image: busybox
    command:
    - cat
    args:
    - "/etc/hosts"
```

6.10.3 Required Domains

6.10.3.1 This section describes the required domains for DKP

You must have access to the following domains through the customer networking rules so that Kommander can download all required images:

- http://docker.io
- http://gcr.io
- http://k8s.gcr.io
- mcr.microsoft.com[573]
- nvcr.io
- http://quay.io
- http://us.gcr.io
- registry.k8s.io[574]

In an air-gapped installation, these domains do not need to be accessible.

573 http://mcr.microsoft.com
574 http://registry.k8s.io/

6.10.4 Configure Ingress for load balancing

6.10.4.1 Learn how to configure Ingress settings for load balancing (layer-7)

Ingress is the name used to describe an API object that manages external access to the services in a cluster. Typically, an Ingress exposes HTTP and HTTPS routes from outside the cluster to services running within the cluster.

The object is called an Ingress because it acts as a gateway for inbound traffic. The Ingress receives inbound requests and routes them according to the rules you defined for the **Ingress resource** as part of your cluster configuration.

Expose an application running on your cluster by configuring an Ingress for load balancing (layer-7).

6.10.4.2 Prerequisites

Before you begin, you must:

- Have access to a Linux, macOS, or Windows computer with a supported operating system version.
- Have a properly deployed and running cluster.

6.10.4.3 Expose a pod using an Ingress (L7)

1. Deploy two web application Pods on your Kubernetes cluster by running the following command:

```
kubectl run --restart=Never --image hashicorp/http-echo --labels app=http-
echo-1 --port 80 http-echo-1 -- -listen=:80 --text="Hello from http-echo-1"
kubectl run --restart=Never --image hashicorp/http-echo --labels app=http-
echo-2 --port 80 http-echo-2 -- -listen=:80 --text="Hello from http-echo-2"
```

2. Expose the Pods with a service type of ClusterIP by running the following commands:

```
kubectl expose pod http-echo-1 --port 80 --target-port 80 --name "http-echo-1"
kubectl expose pod http-echo-2 --port 80 --target-port 80 --name "http-echo-2"
```

3. Create the Ingress to expose the application to the outside world by running the following command:

```
cat <<EOF | kubectl create -f -
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: kommander-traefik
    traefik.ingress.kubernetes.io/router.tls: "true"
```

```
  generation: 7
  name: echo
  namespace: default
spec:
  rules:
  - http:
      paths:
      - backend:
          service:
            name: http-echo-1
            port:
              number: 80
        path: /echo1
        pathType: Prefix
  - http:
      paths:
      - backend:
          service:
            name: http-echo-2
            port:
              number: 80
        path: /echo2
        pathType: Prefix
EOF
```

The configuration settings in this example illustrate:

- setting the `kind` to `Ingress`.
- setting the `service.name` to be exposed as each `backend`.

4. Run the following command to get the URL of the load balancer created on AWS for the Traefik service:

```
kubectl get svc kommander-traefik -n kommander
```

This command displays the internal and external IP addresses for the exposed service. (Note that IP addresses and host names are for illustrative purposes. Always use the information from your own cluster)

```
NAME                TYPE           CLUSTER-IP    EXTERNAL-IP
PORT(S)                                  AGE
kommander-traefik   LoadBalancer   10.0.24.215
abf2e5bda6ca811e982140acb7ee21b7-37522315.us-west-2.elb.amazonaws.com   80:3116
9/TCP,443:32297/TCP,8080:31923/TCP   4h22m
```

5. Validate that you can access the web application Pods by running the following commands: (Note that IP addresses and host names are for illustrative purposes. Always use the information from your own cluster)

```
curl -k https://abf2e5bda6ca811e982140acb7ee21b7-37522315.us-
west-2.elb.amazonaws.com/echo1
curl -k https://abf2e5bda6ca811e982140acb7ee21b7-37522315.us-
west-2.elb.amazonaws.com/echo2
```

6.10.5 Ingress

6.10.5.1 Traefik Ingress Controller

Kubernetes Ingress resources expose HTTP and HTTPS routes from outside the cluster to services within the cluster. In Kommander, the Traefik[575] Ingress controller is installed by default and provides access to the DKP UI.

An Ingress performs the following:

- Gives Services externally-reachable URLs
- Load balances traffic
- Terminates SSL/TLS sessions
- Offers name-based virtual hosting

An Ingress controller fulfills the Ingress with a load balancer.

A cluster can have multiple Ingress controllers. D2iQ recommends adding your own Ingress controllers for your applications. The Traefik Ingress controller that Kommander installs for access to the DKP UI can be replaced later if a different solution is a better fit. Using your own Ingress controller in parallel for your own business requirements ensures that you are not limited by any future changes in Kommander.

6.10.5.2 Traefik v2.4

Traefik is a modern HTTP reverse proxy and load balancer that deploys microservices with ease. Kommander currently installs Traefik v2.4 by default on every cluster. Traefik creates a service of type `LoadBalancer`. In the cloud, the cloud provider creates the appropriate load balancer. In an on-premises deployment, by default, it uses MetalLB.

Traefik listens to the Kubernetes API and automatically generates and updates the routes without any further configuration or intervention so that the Services selected by the Ingress resources are connected to the outside world. Further, Traefik supports a rich set of functionality such as Name-based routing, Path-based routing, Traffic splitting, etc.

Major features highlighted in the Traefik documentation:

- Continuously updates its configuration (No restarts!)
- Supports multiple load balancing algorithms
- Provides HTTPS to your microservices
- Circuit breakers, retry
- A clean web UI
- Websocket, HTTP/2, GRPC ready

575 https://landscape.cncf.io/card-mode?category=service-proxy&grouping=category&selected=traefik

- Provides metrics (Rest, Prometheus, Datadog, StatsD, InfluxDB)
- Keeps access logs (JSON, CLF)
- Exposes a Rest API
- Packaged as a single binary file (made with go) and available as a docker image

6.10.5.3 Related Information

For information on related topics or procedures, refer to the following:

- List of Ingress controllers[576]
- Traefik v2.4 docs[577]
- Configure Ingress for load balancing (see page 851)
- Load Balancing (see page 854)

6.10.6 Load Balancing

In a Kubernetes cluster, depending on the flow of traffic direction, there are two kinds of load balancing:

- Internal load balancing for the traffic within a Kubernetes cluster
- External load balancing for the traffic coming from outside the cluster

6.10.6.1 Load Balancing for Internal Traffic

Load balancing within a Kubernetes cluster is accessed through a `ClusterIP` service type. `ClusterIP` presents a single IP address to the client and load balances the traffic going to this IP to the backend servers. The actual load balancing happens using `iptables` or IPVS configuration. The `kube-proxy` Kubernetes component programs these. The `iptables` mode of operation uses DNAT[578] rules to distribute direct traffic to real servers, whereas IPVS[579] leverages in-kernel transport-layer load-balancing. Read a comparison between these two methods[580]. By default, `kube-proxy` runs in `iptables` mode. The kube-proxy configuration can be altered by updating the `kube-proxy` configmap in the `kube-system` namespace.

6.10.6.2 Load Balancing for External Traffic

External traffic destined for the Kubernetes service requires a service of type `LoadBalancer`, through which external clients connect to your internal service. Under the hood, it uses a load balancer provided by the underlying infrastructure to direct the traffic.

576 https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/
577 https://doc.traefik.io/traefik/v2.4/
578 https://www.linuxtopia.org/Linux_Firewall_iptables/x4013.html
579 https://en.wikipedia.org/wiki/IP_Virtual_Server
580 https://www.projectcalico.org/comparing-kube-proxy-modes-iptables-or-ipvs/

6.10.6.2.1 In the Cloud

In cloud deployments, the load balancer is provided by the cloud provider. For example, in AWS, the service controller communicates with the AWS API to provision an ELB service which targets the cluster nodes.

6.10.6.2.2 On-premises

For an on-premises deployment, DKP ships with MetalLB[581], which provides load-balancing services. The environments that use MetalLB are pre-provisioned and vSphere infrastructures.

For more information on how to configure MetalLB for these environments, refer to the following pages:

- Pre-provisioned Configure MetalLB (see page 1121)
- Configure MetalLB for a vSphere infrastructure (see page 1159)

6.10.6.2.3 Custom Load Balancer for External Traffic

If you want to use a non-DKP load balancer for external traffic, see Install Kommander with an External Load Balancer (see page 1242).

6.10.7 External DNS

This section describes how to use `external-dns` to maintain your DNS records

You require a DNS record when setting up a custom domain for your cluster (see page 764). You can either create one manually, or set up the `external-dns` service to manage your DNS record automatically.

If you choose to use `external-dns` to maintain your DNS records, the `external-dns` will take care of pointing the DNS record to the ingress of the cluster automatically.

The following example shows how to configure `external-dns` to manage DNS records in AWS Route 53 automatically:

1. Open the `kommander.yaml` file:
   a. If you do not have a `kommander.yaml` file, initialize the configuration file (see page 1227), so you can edit it in the following steps. **WARNING**: Initialize this file only ONCE, otherwise you will overwrite previous customizations.
   b. If you have initialized the configuration file already, open the `kommander.yaml` with the editor of your choice.
2. Adjust the `app` section of your `kommander.yaml` file to include these values:

```
apps:
  external-dns:
```

581 https://metallb.universe.tf/concepts/

```
    enabled: true
    values: |
      aws:
        credentials:
          secretKey: <secret-key>
          accessKey: <access-key>
        region: <provider-region>
        preferCNAME: true
      policy: upsert-only
      txtPrefix: local-
      domainFilters:
        - <example.com>
```

3. In the same `app` section, adjust the `traefik` section to include the following:

```
traefik:
    enabled: true
    values: |
      service:
        annotations:
          external-dns.alpha.kubernetes.io/hostname: <mycluster.example.com>
```

Refer to the external-dns documentation[582] for more information, as well as further instructions on how to configure `external-dns` to use other DNS providers like Google Cloud DNS, CloudFlare, or on-site providers.

6.10.7.1 Kommander managed clusters

Enterprise Gov Advanced

To configure external DNS for attached clusters, the External DNS application must be enabled in the Kommander Workspace (see page 586).

6.10.7.1.1 Configure External DNS using the UI

1. Select the workspace your cluster is attached to from the top navigation bar.
2. Navigate to the **Applications** overview.
3. Search for the External DNS application and select **Enable** from the *three-dot menu* on the bottom right of the application card.

582 https://artifacthub.io/packages/helm/bitnami/external-dns

4. On the enable page, you can apply a specific configuration. Select **Download File,** which provides a default example you can adjust to your needs. Once all changes are made, its content can either be copied and pasted into the text field or uploaded using the **Upload File** button.

Here is an example configuration. Refer to external-dns documentation[583] for more information:

```
aws:
    credentials:
        secretKey: <secret-key>
        accessKey: <access-key>
    region: <provider-region>
    preferCNAME: true
policy: upsert-only
txtPrefix: local-
domainFilters:
    - <example.com>
```

6.10.8 Use Istio as a Microservice

Istio helps you manage cloud-based deployments by providing an open-source service mesh to connect, secure, control, and observe microservices.

This topic describes how to expose an application running on the DKP cluster using the LoadBalancer (layer-4) service type.

6.10.8.1 Prerequisites

Before you begin, verify the following:

- You must have access to a Linux, macOS, or Windows computer with a supported operating system version.
- You must have a properly deployed and running cluster.
- Determine the name of the workspace where you wish to perform the deployment. You can use the `dkp get workspaces` command to see the list of workspace names and their corresponding namespaces.
- Set the `WORKSPACE_NAME` environment variable to the name of the workspace where the cluster is attached:

  ```
  export WORKSPACE_NAME=<workspace_name>
  ```

583 https://artifacthub.io/packages/helm/bitnami/external-dns

6.10.8.2 Deploy Istio Using DKP

1. Review the list of available applications (see page 1546) to obtain the current *APP ID* and *version* for Istio and it's dependencies (see page 572), as you will need this information to execute the following commands.
2. Install Istio's dependencies (see page 572) with an `AppDeployment` resource. Replace the `<APPID>` and `<APPID-version>` variables with the information of the application:
   Note: The required value for the `--app` flag consists of the *APP ID* and *version* in the `APPID-version` format.

   ```
   dkp create appdeployment <APPID> --app <APPID-version> --workspace $
   {WORKSPACE_NAME}
   ```

3. Enable the deployment of Istio to your managed or attached cluster (see page 97):

   ```
   dkp create appdeployment istio --app istio-1.16.2 --workspace ${WORKSPACE_NAME}
   ```

- Create the resource in the workspace you just created, which instructs Kommander to deploy the `AppDeployment` to the `KommanderCluster`s in the same workspace.
- Observe that the `dkp create` command must be run with the `WORKSPACE_NAME` instead of the `WORKSPACE_NAMESPACE` flag.

6.10.8.2.1 Download the Istio Command Line Utility

1. Pull a copy of the corresponding Istio command line to your system:

   ```
   curl -L https://istio.io/downloadIstio | ISTIO_VERSION=1.16.2 sh -
   ```

2. Change to the Istio directory and set your PATH environment variable by running the following commands:

   ```
   cd istio*
   export PATH=$PWD/bin:$PATH
   ```

3. Run the following `istioctl` command and view the subsequent output:

```
istioctl version
```

```
client version: <your istio version here>
control plane version: <your istio version here>
data plane version: <your istio version here> (1 proxies)
```

6.10.8.2.2 Deploy a Sample Application on Istio

The Istio `bookinfo` sample application is composed of four separate microservices that demonstrate various Istio features.

1. Deploy the sample `bookinfo` application on the Kubernetes cluster by running the following commands:

   IMPORTANT: Ensure your `dkp` configuration references the cluster where you deployed Istio by setting the `KUBECONFIG` environment variable, or using the `--kubeconfig` flag, in accordance with Kubernetes conventions[584].

   ```
   kubectl apply -f <(istioctl kube-inject -f samples/bookinfo/platform/kube/
   bookinfo.yaml)
   kubectl apply -f samples/bookinfo/networking/bookinfo-gateway.yaml
   ```

2. Get the URL of the load balancer created on AWS for this service by running the following command:

   ```
   kubectl get svc istio-ingressgateway -n istio-system
   ```

   The command displays output similar to the following:

   ```
   NAME                   TYPE           CLUSTER-IP    EXTERNAL-IP
   PORT(S)
   AGE
   istio-ingressgateway   LoadBalancer   10.0.29.241
   a682d13086ccf11e982140acb7ee21b7-2083182676.us-west-2.elb.amazonaws.com   15020:3
   0380/TCP,80:31380/TCP,443:31390/TCP,31400:31400/TCP,15029:30756/TCP,15030:31420
   /TCP,15031:31948/TCP,15032:32061/TCP,15443:31232/TCP   110s
   ```

3. Open a browser and navigate to the external IP address for the load balancer to access the application.

584 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

For example, the external IP address in the sample output is `a682d13086ccf11e982140acb7ee21b7-2083182676.us-west-2.elb.amazonaws.com`, enabling you to access the application using the following URL: `http://a682d13086ccf11e982140acb7ee21b7-2083182676.us-west-2.elb.amazonaws.com/productpage`

4. Follow the steps in the Istio BookInfo Application[585] documentation to understand the different Istio features.

For more information, see Istio's official documentation[586].

6.11 GPUs

This section describes NVIDIA GPU support on Kommander. DKP supported NVIDIA driver[587] version is 470.x. GPUs, such as those made by AMD, are not currently supported. This document assumes familiarity with Kubernetes GPU support. More information about GPUs in AWS environment can be found in the Advanced AWS (see page 988) section.

6.11.1 Kommander GPU Overview

GPU support on Kommander uses the NVIDIA[588]GPU operator. Through the NVIDIA GPU operator application, Kommander configures the container runtime to run GPU containers, and installs all the necessary items to power up the NVIDIA GPU devices.

The following components provide NVIDIA GPU support on Kommander:

- `libnvidia-container` and `nvidia-container-runtime`: GPU Support in Kommander depends on the containerd runtime. `libnvidia-container` and `nvidia-container-runtime` fit between `containerd` and `runc`, simplifying the container runtime integration with the GPU.
- NVIDIA Device Plugin[589]: Kommander makes use of NVIDIA GPUs using this Kubernetes device plugin. It allows GPU enabled containers to run on Kubernetes, tracking the number of available GPUs on each node and their health.
- NVIDIA Data Center GPU Manager[590]: Contains a Prometheus exporter that provides NVIDIA GPU metrics.

Kommander runs these components as daemonsets, making them easier to manage and upgrade across all GPU nodes.

For more information from NVIDIA, see the Getting Started[591] page for NVIDIA.

585 https://istio.io/docs/examples/bookinfo/
586 https://istio.io/latest/docs/
587 https://www.nvidia.com/Download/Find.aspx
588 https://github.com/NVIDIA/gpu-operator
589 https://github.com/NVIDIA/k8s-device-plugin
590 https://developer.nvidia.com/dcgm
591 https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/getting-started.html#chart-customization-options

6.11.2 GPU Prerequisites

6.11.2.1 Configure GPU for Kommander clusters

6.11.2.2 Prerequisites

Before you begin, you must:

- Ensure nodes provide an NVIDIA GPU.
- If you are using a public cloud service such as AWS (see page 988), create an AMI with KIB using the instructions on the KIB for GPU (see page 1302) page.
- If you are deploying in a pre-provisioned environment, ensure that you have created the appropriate secret for your GPU nodepool (see page 210) and have uploaded the appropriate artifacts to each node. See the GPU only steps section (see page 0)on the Pre-provisioned Prerequisites Air-gapped page for additional information.

Specific instructions must be followed for enabling `nvidia-gpu-operator` depending on if you want to deploy the app on a Management cluster or a Attached or a Managed cluster.

- For instructions on enabling the NVIDIA platform application on a Management cluster, follow the instructions in the **NVIDIA Platform Application Management Cluster** (see page 861) section.
- For instructions on enabling the NVIDIA platform application on attached or managed clusters, follow the instructions in the **NVIDIA Platform Application Attached or Managed Cluster** (see page 864) section.

Once `nvidia-gpu-operator` has been enabled depending on the cluster type, proceed to the **Select the correct Toolkit version for your NVIDIA GPU Operator** on each of those pages.

6.11.3 NVIDIA Platform Application Management Cluster

Instructions on enabling the NVIDIA platform application on a Management cluster

6.11.3.1 Enable NVIDIA Platform Application on Kommander for Management Cluster

If you intend to run applications that make use of GPU's on your cluster, you should install the NVIDIA GPU operator. To enable NVIDIA GPU support when installing Kommander on a management cluster, perform the following steps:

1. Create an installation configuration file:

```
dkp install kommander --init > install.yaml
```

2. Append the following to the apps section in the `install.yaml` file to enable Nvidia platform services.

```
apps:
  nvidia-gpu-operator:
   enabled: true
```

3. Install Kommander using the configuration file you created:

```
dkp install kommander --installer-config ./install.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

   In the previous command, the `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you set the context to **install Kommander on the right cluster**. For alternatives and recommendations around setting your context, refer to Provide a context for your commands (see page 99).

4. Proceed to the Select the correct Toolkit version for your NVIDIA GPU Operator (see page 861) section.

TIP: Sometimes, applications require a longer period of time to deploy, which causes the installation to time out. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.

6.11.3.2 Select the Correct Toolkit Version for your NVIDIA GPU Operator

The NVIDIA Container Toolkit allows users to run GPU accelerated containers. The toolkit includes a container runtime library and utilities to automatically configure containers to leverage NVIDIA GPU and must be configured correctly according to your base operating system.

6.11.3.2.1 Kommander (Management Cluster) Customization

1. Select the correct Toolkit version based on your OS:

   The **NVIDIA Container Toolkit** allows users to run GPU accelerated containers. The toolkit includes a container runtime library and utilities to automatically configure containers to leverage NVIDIA GPU and must be configured correctly according to your base operating system.

Centos 7.9/RHEL 7.9:
If you're using Centos 7.9 or RHEL 7.9 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1227) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.10.0-centos7
```

RHEL 8.4/8.6 and SLES 15 SP3
If you're using RHEL 8.4/8.6 or SLES 15 SP3 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1227) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.10.0-ubi8
```

Ubuntu 18.04 and 20.04
If you're using Ubuntu 18.04 or 20.04 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1227) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.11.0-ubuntu20.04
```

2. Install Kommander, using the configuration file you created:

```
dkp install kommander --installer-config ./install.yaml
```

In the previous command, the `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you set the context to install Kommander on the right cluster. For alternatives and recommendations around setting your context, refer to Provide a context for your commands (see page 99).

> **TIP:** Sometimes, applications require a longer period of time to deploy, which causes the installation to time out. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.

6.11.4 NVIDIA Platform Application Attached or Managed Cluster

Instructions on enabling the NVIDIA platform application on attached or managed clusters

6.11.4.1 Enable NVIDIA Platform Application on Attached or Managed Clusters

If you intend to run applications that utilize GPU's on Attached or Managed clusters, you must enable the `nvidia-gpu-operator` platform application in the workspace.

> To use the UI to enable the application, refer to the https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919665/Platform+Applications#Customize-a-workspace%E2%80%99s-applications (see page 0) page.
>
> To use the CLI, refer to the Deploy Platform Applications via CLI (see page 564) page.
>
> If only a subset of attached or managed clusters in the workspace are utilizing GPU's, refer to Enable an Application per Cluster (see page 579) on how to only enable the `nvidia-gpu-operator` on specific clusters.

After you have enabled the `nvidia-gpu-operator` app in the workspace on the necessary clusters, proceed to the next section.

6.11.4.2 Select the Correct Toolkit Version for your NVIDIA GPU Operator

The NVIDIA Container Toolkit allows users to run GPU accelerated containers. The toolkit includes a container runtime library and utilities to automatically configure containers to leverage NVIDIA GPU and must be configured correctly according to your base operating system.

6.11.4.2.1 Workspace (Attached and Managed clusters) Customization

> Refer to AppDeployment resources (see page 545) for how to use the CLI to customize the platform application on a workspace.
>
> If specific attached/managed clusters in the workspace require different configurations, refer to Customize an Application per Cluster (see page 580) for how to do this.

1. Select the correct Toolkit version based on your OS and create a `ConfigMap` with these configuration override values:

   Centos 7.9/RHEL 7.9:
   If you're using Centos 7.9 or RHEL 7.9 as the base operating system for your GPU enabled nodes, set the toolkit.version parameter in your install.yaml to the following:

   ```
   cat <<EOF | kubectl apply -f -
   apiVersion: v1
   kind: ConfigMap
   metadata:
     namespace: ${WORKSPACE_NAMESPACE}
     name: nvidia-gpu-operator-overrides-attached
   data:
     values.yaml: |
       toolkit:
         version: v1.10.0-centos7
   EOF
   ```

   RHEL 8.4/8.6 and SLES 15 SP3
   If you're using RHEL 8.4/8.6 or SLES 15 SP3 as the base operating system for your GPU enabled nodes, set the toolkit.version parameter in your install.yaml to the following:

   ```
   cat <<EOF | kubectl apply -f -
   apiVersion: v1
   kind: ConfigMap
   metadata:
     namespace: ${WORKSPACE_NAMESPACE}
     name: nvidia-gpu-operator-overrides-attached
   data:
     values.yaml: |
       toolkit:
         version: v1.10.0-ubi8
   EOF
   ```

Ubuntu 18.04 and 20.04
If you're using Ubuntu 18.04 or 20.04 as the base operating system for your GPU enabled nodes, set the toolkit.version parameter in your install.yaml to the following:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: nvidia-gpu-operator-overrides-attached
data:
  values.yaml: |
    toolkit:
      version: v1.11.0-ubuntu20.04
EOF
```

2. Note the name of this `ConfigMap` (`nvidia-gpu-operator-overrides-attached`) and use it to set the necessary `nvidia-gpu-operator` `AppDeployment` spec fields depending on the scope of the override. Alternatively, you can also use the UI to pass in the configuration overrides for the app per workspace or per cluster.

6.11.5 Kommander GPU Settings and Troubleshoot

6.11.5.1 Validate and troubleshoot GPU

6.11.5.2 Validate that the Application has Started Correctly

Run the following command to validate that your application has started correctly:

```
kubectl get pods -A | grep nvidia
```

The output should be similar to the following:

```
nvidia-container-toolkit-daemonset-7h2l5   1/1   Running     0              150m
nvidia-container-toolkit-daemonset-mm65g   1/1   Running     0              150m
nvidia-container-toolkit-daemonset-mv7xj   1/1   Running     0              150m
nvidia-cuda-validator-pdlz8              0/1     Completed   0              150m
nvidia-cuda-validator-r7qc4              0/1     Completed   0              150m
nvidia-cuda-validator-xvtqm              0/1     Completed   0              150m
nvidia-dcgm-exporter-9r6rl               1/1     Running     1 (149m ago)   150m
nvidia-dcgm-exporter-hn6hn               1/1     Running     1 (149m ago)   150m
```

```
nvidia-dcgm-exporter-j7g7g                1/1     Running     0              150m
nvidia-dcgm-jpr57                         1/1     Running     0              150m
nvidia-dcgm-jwldh                         1/1     Running     0              150m
nvidia-dcgm-qg2vc                         1/1     Running     0              150m
nvidia-device-plugin-daemonset-2gv8h      1/1     Running     0              150m
nvidia-device-plugin-daemonset-tcmgk      1/1     Running     0              150m
nvidia-device-plugin-daemonset-vqj88      1/1     Running     0              150m
nvidia-device-plugin-validator-9xdqr      0/1     Completed   0              149m
nvidia-device-plugin-validator-jjhdr      0/1     Completed   0              149m
nvidia-device-plugin-validator-llxjk      0/1     Completed   0              149m
nvidia-operator-validator-9kzv4           1/1     Running     0              150m
nvidia-operator-validator-fvsr7           1/1     Running     0              150m
nvidia-operator-validator-qr9cj           1/1     Running     0              150m
```

If you are seeing errors, ensure that you set the container toolkit version appropriately based on your OS, as described in the previous section.

6.11.5.3 NVIDIA GPU Monitoring

Kommander uses the NVIDIA Data Center GPU Manager[592] to export GPU metrics towards Prometheus. By default, Kommander has a Grafana dashboard called `NVIDIA DCGM Exporter Dashboard` to monitor GPU metrics. This GPU dashboard is shown in Kommander's Grafana UI.

6.11.5.4 NVIDIA MIG Settings

MIG stands for Multi-Instance-GPU. It is a mode of operation for future NVIDIA GPUs that allows the user to partition a GPU into a set of MIG devices. Each set appears to the software that is consuming them as a mini-GPU with a fixed partition of memory and a fixed partition of compute resources.

NOTE: MIG is only available for the following NVIDIA devices: H100, A100, and A30.

6.11.5.4.1 To Configure MIG

1. Set the MIG strategy according to your GPU topology.
   - `mig.strategy` should be set to mixed when MIG mode is not enabled on all GPUs on a node.
   - `mig.strategy` should be set to single when MIG mode is enabled on all GPUs on a node and they have the same MIG device types across all of them.

   For the Management Cluster, this can be set at install time by modifying the Kommander configuration file to add configuration for the `nvidia-gpu-operator` application:

592 https://developer.nvidia.com/dcgm

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  nvidia-gpu-operator:
    values: |
      mig:
        strategy: single
...
```

Or by modifying the `clusterPolicy` object for the GPU operator once it has already been installed.

2. Set the MIG profile for the GPU you are using. In our example, we are using the A30 GPU that supports the following MIG profiles:

```
4 GPU instances @ 6GB each
2 GPU instances @ 12GB each
1 GPU instance @ 24GB
```

Set the mig profile by labeling the node ${NODE} with the profile as in the example below:

```
kubectl label nodes ${NODE} nvidia.com/mig.config=all-1g.6gb --overwrite
```

3. Check the node labels to see if the changes were applied to your MIG enabled GPU node

```
kubectl get no -o json | jq .items[0].metadata.labels
```

```
"nvidia.com/mig.config": "all-1g.6gb",
  "nvidia.com/mig.config.state": "success",
  "nvidia.com/mig.strategy": "single"
```

4. Deploy a sample workload:

```
apiVersion: v1
kind: Pod
metadata:
  name: cuda-vector-add
spec:
  restartPolicy: OnFailure
  containers:
      - name: cuda-vectoradd
```

```
      image: "nvidia/samples:vectoradd-cuda11.2.1"
      resources:
        limits:
          nvidia.com/gpu: 1

  nodeSelector:
    "nvidia.com/gpu.product": NVIDIA-A30-MIG-1g.6gb
```

If the workload successfully finishes, then your GPU has been properly MIG partitioned.

6.11.5.5 Troubleshooting NVIDIA GPU Operator on Kommander

In case you run into any errors with NVIDIA GPU Operator, here are a couple commands you can run to troubleshoot:

1. Connect (using SSH or similar) to your GPU enabled nodes and run the `nvidia-smi` command. Your output should be similar to the following example:

```
  [ec2-user@ip-10-0-0-241 ~]$ nvidia-smi
  Thu Nov  3 22:52:59 2022

+-----------------------------------------------------------------------------+
  | NVIDIA-SMI 470.82.01    Driver Version: 470.82.01    CUDA Version: 11.4
|
  |-------------------------------+----------------------
+----------------------+
  | GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC
|
  | Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M.
|
  |                               |                      |               MIG M.
|
  |
==============================+======================+======================|
  |   0  Tesla T4            On   | 00000000:00:1E.0 Off |                    0
 |
  | N/A   54C    P8    11W /  70W |      0MiB / 15109MiB |      0%      Default
|
  |                               |                      |                  N/A
|
  +-------------------------------+----------------------
+----------------------+

+-----------------------------------------------------------------------------+
  | Processes:
|
  |  GPU   GI   CI        PID   Type   Process name                  GPU Memory
|
```

```
  |        ID   ID                                                    Usage
|
  |
=============================================================================|
  |  No running processes found
|

+-----------------------------------------------------------------------------+
```

2. Another common issue is having a misconfigured toolkit version, resulting in NVIDIA pods in a bad state
   For example:

```
nvidia-container-toolkit-daemonset-jrqt2                          1/1
Running                 0             29s
nvidia-dcgm-exporter-b4mww                                        0/1     Error
1 (9s ago)    16s
nvidia-dcgm-pqsz8                                                 0/1
CrashLoopBackOff        1 (13s ago)   27s
nvidia-device-plugin-daemonset-7fkzr                              0/1     Init:0
/1                0             14s
nvidia-operator-validator-zxn4w                                   0/1
Init:CrashLoopBackOff   1 (7s ago)    11s
```

   To modify the toolkit version, run the following commands to modify the `AppDeployment` for the `nvidia gpu operator` application:

   • Provide the name of a `ConfigMap` with the custom configuration in the `AppDeployment`:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: nvidia-gpu-operator
  namespace: kommander
spec:
  appRef:
    kind: ClusterApp
    name: nvidia-gpu-operator-1.11.1
  configOverrides:
    name: nvidia-gpu-operator-overrides
EOF
```

   • Create the `ConfigMap` with the name provided in the previous step, which provides the custom configuration on top of the default configuration in the config map, set the version appropriately:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: kommander
  name: nvidia-gpu-operator-overrides
data:
  values.yaml: |
    toolkit:
      version: v1.10.0-centos7
EOF
```

3. If a node has an NVIDIA GPU installed and the `nvidia-gpu-operator` application is enabled on the cluster, but the node is still not accepting GPU workloads, it's possible that the nodes do not have a label that indicates there is an NVIDIA GPU present.
   By default the GPU operator will attempt to configure nodes with the following labels present, which are usually applied by the node feature discovery component:

```
    "feature.node.kubernetes.io/pci-10de.present":      "true",
    "feature.node.kubernetes.io/pci-0302_10de.present": "true",
    "feature.node.kubernetes.io/pci-0300_10de.present": "true",
```

   If these labels are not present on a node that you know contains an NVIDIA GPU, you can manually label the node using the following command:

```
kubectl label node ${NODE} feature.node.kubernetes.io/
pci-0302_10de.present=true
```

6.11.5.6 Disable NVIDIA GPU Operator Platform Application on Kommander

1. Delete all GPU workloads on the GPU nodes where the NVIDIA GPU Operator platform application is present.
2. Delete the existing NVIDIA GPU Operator AppDeployment using the following command:

```
kubectl delete appdeployment -n kommander nvidia-gpu-operator
```

3. Wait for all NVIDIA related resources in the `Terminating` state to be cleaned up. You can check pod status with the following command:

```
kubectl get pods -A | grep nvidia
```

For information on how to delete nodepools, refer to Pre-provisioned Create and Delete Node Pools (see page 1123)

6.11.6 GPU Toolkit Versions

The **NVIDIA Container Toolkit** allows users to run GPU accelerated containers. The toolkit includes a container runtime library and utilities to automatically configure containers to leverage NVIDIA GPU and must be configured correctly according to your base operating system.

Centos 7.9/RHEL 7.9:
If you're using Centos 7.9 or RHEL 7.9 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1227) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.10.0-centos7
```

RHEL 8.4/8.6 and SLES 15 SP3
If you're using RHEL 8.4/8.6 or SLES 15 SP3 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1227) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.10.0-ubi8
```

Ubuntu 18.04 and 20.04
If you're using Ubuntu 18.04 or 20.04 as the base operating system for your GPU enabled nodes, set the

`toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1227) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.11.0-ubuntu20.04
```

6.11.7 Enable GPU Usage After Installing DKP

If you want to enable your cluster to run GPU nodes after installing DKP, enable the correct Toolkit version for your operating system in the `nvidia-gpu-operator` AppDeployment.

If you have not installed the Kommander component of DKP yet, set the Toolkit version in the Kommander Installer Configuration file (see page 872) and skip this section.

1. Create a `ConfigMap` with the necessary configuration overrides to set the correct Toolkit version (see page 872). For example, if you're using Centos 7.9 or RHEL 7.9 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter:

   ```
   cat <<EOF | kubectl apply -f -
   apiVersion: v1
   kind: ConfigMap
   metadata:
     namespace: kommander
     name: nvidia-gpu-operator-overrides
   data:
     values.yaml: |
       toolkit:
         version: v1.10.0-centos7
   EOF
   ```

   Take the correct GPU Toolkit version from this page: GPU Toolkit Versions (see page 872)

2. Update the `nvidia-gpu-operator` AppDeployment in the `kommander` namespace to reference the `ConfigMap` you created:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: nvidia-gpu-operator
  namespace: kommander
spec:
  appRef:
    name: nvidia-gpu-operator-1.11.1
    kind: ClusterApp
  configOverrides:
    name: nvidia-gpu-operator-overrides
EOF
```

6.12 Monitoring and Alerts

Using DKP you can monitor the state of the cluster and the health and availability of the processes running on the cluster. By default, Kommander provides monitoring services using a pre-configured monitoring stack based on the Prometheus open-source project and its broader ecosystem.

The default DKP monitoring stack:

- Provides in-depth monitoring of Kubernetes components and platform services.
- Includes a default set of Grafana dashboards to visualize the status of the cluster and its platform services.
- Supports predefined critical error and warning alerts. These alerts notify immediately if there is a problem with cluster operations or availability.

By incorporating Prometheus, Kommander visualizes all the exposed metrics from your different nodes, Kubernetes objects, and platform service applications running in your cluster. The default monitoring stack also enables you to add metrics from any of your deployed applications, making those applications part of the overall Prometheus metrics stream.

- Recommendations (see page 874)
- Grafana Dashboards (see page 877)
- Cluster Metrics (see page 879)
- Configure Alerts Using AlertManager (see page 880)
- Centralized Monitoring (see page 885)
- Centralized Cost Monitoring (see page 888)
- Monitoring Applications Using Prometheus (see page 891)
- Set Storage Capacity for Prometheus (see page 893)

6.12.1 Recommendations

Enterprise Gov Advanced

Recommended settings for monitoring and collecting metrics for Kubernetes, platform services, and applications deployed on the cluster

D2iQ conducts routine performance testing of Kommander. The following table provides recommended settings, based on cluster size and increasing workloads, that maintain a healthy Prometheus monitoring deployment.

> The resource settings reflect some settings but do not represent the exact structure to be used in the platform service configuration.

6.12.1.1 Prometheus

Cluster Size	Number of Pods	Number of Services	Resource settings

10	1k	250	`resources:`
			`  limits:`
			`  cpu: 500m`
			`  memory: 2192Mi`
			`  requests:`
			`  cpu: 100m`
			`  memory: 500Mi`
			`storage: 35Gi`

25	1k	250	`resources:`
			`  limits:`
			`  cpu: 2`
			`  memory: 6Gi`
			`  requests:`
			`  cpu: 1`
			`  memory: 3Gi`
			`storage: 60Gi`

50	1.5k	500	<pre>resources:
			  limits:
			    cpu: 7
			    memory:
			28Gi
			  requests:
			    cpu: 2
			    memory: 8Gi
			storage: 100Gi</pre>

100	3k	1k	<pre>resources:
			  limits:
			    cpu: 12
			    memory:
			50Gi
			  requests:
			    cpu: 10
			    memory:
			48Gi
			storage: 100Gi</pre>

200	10k	3k	<pre>resources:
			  limits:
			    cpu: 20
			    memory:
			80Gi
			  requests:
			    cpu: 15
			    memory:
			50Gi
			storage: 100Gi</pre>

300	15k	6k	<pre>resources:
			  limits:
			  cpu: 35
			  memory:
			150Gi
			  requests:
			  cpu: 25
			  memory:
			120Gi
			storage: 100Gi</pre>

6.12.2 Grafana Dashboards

With Grafana, you can query and view collected metrics in easy-to-read graphs. Kommander ships with a set of default dashboards including:

- Kubernetes Components: API Server, Nodes, Pods, Kubelet, Scheduler, StatefulSets and Persistent Volumes
- Kubernetes USE method: Cluster and Nodes
- Calico
- etcd
- Prometheus

Find the complete list of default enabled dashboards on GitHub[593].

To disable all of the default dashboards, follow these steps to define an overrides ConfigMap:

1. Create a file named `kube-prometheus-stack-overrides.yaml` and paste the following YAML code into it to create the overrides ConfigMap:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-prometheus-stack-overrides
  namespace: <your-workspace-namespace>
data:
 values.yaml: |
   ---
   grafana:
     defaultDashboardsEnabled: false
```

2. Use the following command to apply the YAML file:

593 https://github.com/prometheus-community/helm-charts/tree/main/charts/kube-prometheus-stack/templates/grafana/dashboards-1.14

```
kubectl apply -f kube-prometheus-stack-overrides.yaml
```

3. Edit the `kube-prometheus-stack` AppDeployment to replace the `spec.configOverrides.name` value with `kube-prometheus-stack-overrides`. (You can use the steps in the procedure, Deploy an application with a custom configuration (see page 580) as a guide.) When your editing is complete, the AppDeployment will resemble this code sample:

```
apiVersion: apps.kommander.d2iq.io/v1alpha2
kind: AppDeployment
metadata:
  name: kube-prometheus-stack
  namespace: <your-workspace-namespace>
spec:
  appRef:
    name: kube-prometheus-stack-44.2.1
    kind: ClusterApp
  configOverrides:
    name: kube-prometheus-stack-overrides
```

To access the Grafana UI, browse to the landing page and then search for the Grafana dashboard, for example, `https://<CLUSTER_URL>/dkp/grafana`.

6.12.2.1 Add Custom Dashboards

In Kommander you can define your own custom dashboards. You can use a few methods to import dashboards[594] to Grafana.

One method is to use ConfigMaps to import dashboards[595]. Below are steps on how to create a ConfigMap with your dashboard definition.

For simplicity, this section assumes the desired dashboard definition is in `json` format:

```
{
    "annotations": {
        "list": []
    },
    "description": "etcd sample Grafana dashboard with Prometheus",
    "editable": true,
    "gnetId": null,
    "hideControls": false,
    "id": 6,
    "links": [],
    "refresh": false,
    ...
```

594 https://github.com/grafana/helm-charts/tree/main/charts/grafana#import-dashboards
595 https://github.com/grafana/helm-charts/tree/main/charts/grafana#sidecar-for-dashboards

```
}
```

After creating your custom dashboard json, insert it into a ConfigMap and save it as `etcd-custom-dashboard.yaml`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: etcd-custom-dashboard
  labels:
    grafana_dashboard: "1"
data:
  etcd.json: |
    {
        "annotations": {
            "list": []
        },
        "description": "etcd sample Grafana dashboard with Prometheus",
        "editable": true,
        "gnetId": null,
        "hideControls": false,
        "id": 6,
        "links": [],
        "refresh": false,
        ...
    }
```

Apply the ConfigMap, which automatically gets imported to Grafana using the Grafana dashboard sidecar[596]:

```
kubectl apply -f etcd-custom-dashboard.yaml
```

6.12.3 Cluster Metrics

The `kube-prometheus-stack` is deployed by default on the management cluster and attached clusters. This stack deploys the following Prometheus components to expose metrics from nodes, Kubernetes units, and running apps:

- prometheus-operator: orchestrates various components in the monitoring pipeline.
- prometheus: collects metrics, saves them in a time series database, and serves queries.
- alertmanager: handles alerts sent by client applications such as the Prometheus server.
- node-exporter: deployed on each node to collect the machine hardware and OS metrics.
- kube-state-metrics: simple service that listens to the Kubernetes API server and generates metrics about the state of the objects.
- grafana: monitors and visualizes metrics.
- service monitors: collects internal Kubernetes components.

596 https://github.com/grafana/helm-charts/tree/main/charts/grafana#sidecar-for-dashboards

DKP has a listener on the `metrics.k8s.io/v1beta1/nodes` resource, which updates your backend store when that value changes. We then poll that backend store every 5 seconds, so the metrics are updated in realtime every 5-seconds without the need to refresh your view.

A detailed description of the exposed metrics can be found in the kube-state-metrics documentation on GitHub[597]. The `service-monitors` collect internal Kubernetes components but can also be extended to monitor customer apps as explained in the section Monitor Applications, on this page below.

6.12.4 Configure Alerts Using AlertManager

To keep your clusters and applications healthy and drive productivity forward, you need to stay informed of all events occurring in your cluster. DKP helps you to stay informed of these events by using the `alertmanager` of the `kube-prometheus-stack`.

Kommander is configured with pre-defined alerts to monitor four specific events. You receive alerts related to:

- State of your nodes
- System services managing the Kubernetes cluster
- Resource events from specific system services
- Prometheus expressions exceeding some pre-defined thresholds

Some examples of the alerts currently available are:

- CPUThrottlingHigh
- TargetDown
- KubeletNotReady
- KubeAPIDown
- CoreDNSDown
- KubeVersionMismatch

A complete list with all the pre-defined alerts can be found on GitHub[598].

6.12.4.1 Prerequisites

- Determine the name of the workspace where you wish to perform the actions. You can use the `dkp get workspaces` command to see the list of workspace names and their corresponding namespaces.
- Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace where the cluster is attached:

597 https://github.com/kubernetes/kube-state-metrics/tree/master/docs#exposed-metrics
598 https://github.com/prometheus-community/helm-charts/tree/main/charts/kube-prometheus-stack/templates/prometheus/rules-1.14

```
export WORKSPACE_NAMESPACE=<workspace_namespace>
```

6.12.4.1.1 Configure Alert Rules

Use override `ConfigMaps` to configure alert rules.

You can enable or disable the default alert rules by providing the desired configuration in an overrides ConfigMap. For example, if you want to disable the default `node` alert rules, follow these steps to define an overrides ConfigMap:

1. Create a file named `kube-prometheus-stack-overrides.yaml` and paste the following YAML code into it to create the overrides ConfigMap:

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: kube-prometheus-stack-overrides
     namespace: ${WORKSPACE_NAMESPACE}
   data:
    values.yaml: |
      ---
      defaultRules:
        rules:
          node: false
   ```

2. Use the following command to apply the YAML file:

   ```
   kubectl apply -f kube-prometheus-stack-overrides.yaml
   ```

3. Edit the `kube-prometheus-stack` AppDeployment to replace the `spec.configOverrides.name` value with `kube-prometheus-stack-overrides`. (You can use the steps in the procedure, Deploy an application with a custom configuration (see page 880) as a guide.)

   ```
   dkp edit appdeployment -n ${WORKSPACE_NAMESPACE} kube-prometheus-stack
   ```

   After your editing is complete, the AppDeployment resembles this example:

   ```
   apiVersion: apps.kommander.d2iq.io/v1alpha2
   kind: AppDeployment
   metadata:
     name: kube-prometheus-stack
     namespace: ${WORKSPACE_NAMESPACE}
   ```

```
spec:
  appRef:
    name: kube-prometheus-stack-34.9.3
    kind: ClusterApp
  configOverrides:
    name: kube-prometheus-stack-overrides
```

4. To disable all rules, create an overrides ConfigMap with this YAML code:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-prometheus-stack-overrides
  namespace: ${WORKSPACE_NAMESPACE}
data:
 values.yaml: |
   ---
   defaultRules:
     create: false
```

5. Alert rules for the Velero platform service are turned off by default. You can enable them with the following overrides ConfigMap. They should be enabled only if the `velero` platform service is enabled. If platform services are disabled disable the alert rules to avoid alert misfires.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-prometheus-stack-overrides
  namespace: ${WORKSPACE_NAMESPACE}
data:
  values.yaml: |
    ---
    mesosphereResources:
      rules:
        velero: true
```

6. To create a custom alert rule named `my-rule-name`, create the overrides ConfigMap with this YAML code:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-prometheus-stack-overrides
  namespace: ${WORKSPACE_NAMESPACE}
data:
  values.yaml: |
    ---
```

```
    additionalPrometheusRulesMap:
      my-rule-name:
        groups:
        - name: my_group
          rules:
          - record: my_record
            expr: 100 * my_record
```

After you set up your alerts, you can manage each alert using the Prometheus web console to mute or unmute firing alerts, and perform other operations. For more information about configuring `alertmanager`, see the Prometheus website[599].

To access the Prometheus Alertmanager UI, browse to the landing page and then search for the Prometheus Alertmanager dashboard, for example `https://<CLUSTER_URL>/dkp/alertmanager`.

6.12.4.1.2 Notify Prometheus Alerts in Slack

To hook up the Prometheus `alertmanager` notification system, you need to overwrite the existing configuration.

1. The following file, named `alertmanager.yaml`, configures `alertmanager` to use the Incoming Webhooks feature of Slack (`slack_api_url: https://hooks.slack.com/services/<HOOK_ID>`) to fire all the alerts to a specific channel `#MY-SLACK-CHANNEL-NAME`.

```
global:
  resolve_timeout: 5m
  slack_api_url: https://hooks.slack.com/services/<HOOK_ID>

route:
  group_by: ['alertname']
  group_wait: 2m
  group_interval: 5m
  repeat_interval: 1h

  # If an alert isn't caught by a route, send it to slack.
  receiver: slack_general
  routes:
    - match:
        alertname: Watchdog
      receiver: "null"

receivers:
  - name: "null"
  - name: slack_general
    slack_configs:
      - channel: '#MY-SLACK-CHANNEL-NAME'
```

599 https://prometheus.io/docs/alerting/configuration/

```
        icon_url: https://avatars3.githubusercontent.com/u/3380462
        send_resolved: true
        color: '{{ if eq .Status "firing" }}danger{{ else }}good{{ end }}'
        title: '{{ template "slack.default.title" . }}'
        title_link: '{{ template "slack.default.titlelink" . }}'
        pretext: '{{ template "slack.default.pretext" . }}'
        text: '{{ template "slack.default.text" . }}'
        fallback: '{{ template "slack.default.fallback" . }}'
        icon_emoji: '{{ template "slack.default.iconemoji" . }}'

templates:
  - '*.tmpl'
```

2. The following file, named `notification.tmpl`, is a template that defines a pretty format for the fired notifications:

```
{{ define "__titlelink" }}
{{ .ExternalURL }}/#/alerts?receiver={{ .Receiver }}
{{ end }}

{{ define "__title" }}
[{{ .Status | toUpper }}{{ if eq .Status "firing" }}:{{ .Alerts.Firing | len }}
{{ end }}] {{ .GroupLabels.SortedPairs.Values | join " " }}
{{ end }}

{{ define "__text" }}
{{ range .Alerts }}
{{ range .Labels.SortedPairs }}*{{ .Name }}*: `{{ .Value }}`
{{ end }} {{ range .Annotations.SortedPairs }}*{{ .Name }}*: {{ .Value }}
{{ end }} *source*: {{ .GeneratorURL }}
{{ end }}
{{ end }}

{{ define "slack.default.title" }}{{ template "__title" . }}{{ end }}
{{ define "slack.default.username" }}{{ template "__alertmanager" . }}{{ end }}
{{ define "slack.default.fallback" }}{{ template "slack.default.title" . }} |
{{ template "slack.default.titlelink" . }}{{ end }}
{{ define "slack.default.pretext" }}{{ end }}
{{ define "slack.default.titlelink" }}{{ template "__titlelink" . }}{{ end }}
{{ define "slack.default.iconemoji" }}{{ end }}
{{ define "slack.default.iconurl" }}{{ end }}
{{ define "slack.default.text" }}{{ template "__text" . }}{{ end }}
```

3. Finally, apply these changes to `alertmanager` as follows. Set `${WORKSPACE_NAMESPACE}` to the workspace namespace that `kube-prometheus-stack` is deployed in:

```
kubectl create secret generic -n ${WORKSPACE_NAMESPACE} \
  alertmanager-kube-prometheus-stack-alertmanager \
```

```
  --from-file=alertmanager.yaml \
  --from-file=notification.tmpl \
  --dry-run=client --save-config -o yaml | kubectl apply -f -
```

6.12.5 Centralized Monitoring

Enterprise Gov Advanced

Monitor clusters, created with Kommander, on any attached cluster

Kommander provides centralized monitoring, in a multi-cluster environment, using the monitoring stack running on any attached clusters. Centralized monitoring is provided by default in every managed or attached cluster.

Managed or attached clusters are distinguished by a monitoring ID. The monitoring ID corresponds to the kube-system namespace UID of the cluster. To find a cluster's monitoring ID, you can go to the Clusters tab on the DKP UI (in the relevant workspace), or go to the **Clusters** page in the **Global** workspace:

```
https://<CLUSTER_URL>/dkp/kommander/dashboard/clusters
```

Select the `View Details` link on the attached cluster card, and then select the **Configuration** tab, and find the monitoring ID under **Monitoring ID (clusterId)**.

You may also search or filter by monitoring IDs on the Clusters page, linked above.

You can also run this kubectl command, **using the correct cluster's context or kubeconfig**, to look up the cluster's kube-system namespace UID to determine which cluster the metrics and alerts correspond to:

```
kubectl get namespace kube-system -o jsonpath='{.metadata.uid}'
```

6.12.5.1 Centralized Metrics

Managed and attached clusters collects and presents metrics from all attached clusters remotely using Thanos. You can visualize these metrics in Grafana using a set of provided dashboards.

The Thanos Query[600] component is installed on attached and managed clusters. Thanos Query queries the Prometheus instances on the attached clusters, using a Thanos sidecar running alongside each Prometheus container. Grafana is configured with Thanos Query as its datasource, and comes with pre-installed dashboards for a global view of all attached clusters. The `Thanos Query` dashboard is also installed, by default, to monitor the Thanos Query component.

NOTE: Metrics from clusters are read remotely from Kommander; they are not backed up. If a attached cluster goes down, Kommander no longer collects or presents its metrics, including past data.

You can access the centralized Grafana UI at:

600 https://thanos.io/v0.5/components/query/#query

```
https://<CLUSTER_URL>/dkp/kommander/monitoring/grafana
```

NOTE: This is a separate Grafana instance than the one installed on all attached clusters. It is dedicated specifically to components related to centralized monitoring.

Optionally, if you want to access the Thanos Query UI (essentially the Prometheus UI), the UI is accessible at:

```
https://<CLUSTER_URL>/dkp/kommander/monitoring/query
```

You can also check that the attached cluster's Thanos sidecars are successfully added to Thanos Query by going to:

```
https://<CLUSTER_URL>/dkp/kommander/monitoring/query/stores
```

The preferred method to view the metrics for a specific cluster is to go directly to that cluster's Grafana UI.

6.12.5.1.1 Adding Custom Dashboards

You can also define custom dashboards for centralized monitoring on Kommander. There are a few methods to import dashboards[601] to Grafana. For simplicity, assume the desired dashboard definition is in `json` format:

```
{
    "annotations":
    ...
    # Complete json file here
    ...
    "title": "Some Dashboard",
    "uid": "abcd1234",
    "version": 1
}
```

After creating your custom dashboard json, insert it into a ConfigMap and save it as `some-dashboard.yaml`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: some-dashboard
  labels:
    grafana_dashboard_kommander: "1"
data:
  some_dashboard.json: |
```

601 https://github.com/mesosphere/charts/tree/master/stable/grafana#import-dashboards

```
{
  "annotations":
  ...
  # Complete json file here
  ...
  "title": "Some Dashboard",
  "uid": "abcd1234",
  "version": 1
}
```

Apply the ConfigMap, which will automatically get imported to Grafana via the Grafana dashboard sidecar:

```
kubectl apply -f some-dashboard.yaml
```

6.12.5.2 Centralized Alerts

A centralized view of alerts, from attached clusters, is provided using an alert dashboard called Karma[602]. Karma aggregates all alerts from the Alertmanagers running in the attached clusters, allowing you to visualize these alerts on one page. Using the Karma dashboard, you can get an overview of each alert and filter by alert type, cluster, and more.

Silencing alerts using the Karma UI is currently not supported.

You can access the Karma dashboard UI at:

```
https://<CLUSTER_URL>/dkp/kommander/monitoring/karma
```

When there are no attached clusters, the Karma UI displays an error message `Get https://placeholder.invalid/api/v2/status: dial tcp: lookup placeholder.invalid on 10.0.0.10:53: no such host`. This is expected, and the error disappears when clusters are connected.

602 https://github.com/prymitive/karma

6.12.5.2.1 Federating Prometheus Alerting Rules

You can define additional Prometheus alerting rules[603] on attached and managed clusters and federate them to all of the attached clusters by following these instructions. To use these instructions you must install the kubefedctl CLI[604].

1. Enable the PrometheusRule type for federation.

```
kubefedctl enable PrometheusRules --kubefed-namespace kommander
```

2. Modify the existing alertmanager configuration.

```
kubectl edit PrometheusRules/kube-prometheus-stack-alertmanager.rules -n
kommander
```

3. Append a sample rule.

```
- alert: MyFederatedAlert
  annotations:
    message: A custom alert that will always fire.
  expr: vector(1)
  labels:
    severity: warning
```

4. Federate the rules you just modified.

```
kubefedctl federate PrometheusRules kube-prometheus-stack-alertmanager.rules --
kubefed-namespace kommander -n kommander
```

5. Ensure that the clusters selection (`status.clusters`) is appropriately set for your desired federation strategy and check the propagation status[605].

```
kubectl get federatedprometheusrules kube-prometheus-stack-alertmanager.rules
-n kommander -oyaml
```

6.12.6 Centralized Cost Monitoring

Enterprise Gov Advanced

603 https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/
604 https://github.com/kubernetes-sigs/kubefed/blob/master/docs/installation.md#kubefedctl-cli
605 https://github.com/kubernetes-sigs/kubefed/blob/master/docs/userguide.md#propagation-status

Monitoring costs of all attached clusters with Kubecost

Kubecost[606], running on Kommander, provides centralized cost monitoring for all attached clusters. This feature, installed by default in the management cluster, provides a centralized view of Kubernetes resources used on all attached clusters.

By default, up to 15 days of cost metrics are retained, with no backup to an external store.

6.12.6.1 Adding a Kubecost License Key to your Clusters

By default, DKP is deployed with the free version of Kubecost, which provides cost monitoring for individual clusters and metric retention for up to 15 days. Licensed plans from Kubecost with additional features are also available. For more information, refer to the Kubecost Pricing[607] page.

The following instructions provide information on how you can add a Kubecost license key to your clusters if you have purchased one.

The license key must be applied to the Centralized Kubecost application running on the Management cluster.

1. Obtain license key from Kubecost. For more information, refer to the Kubecost Pricing[608] page.
2. Access the `centralized-kubecost` dashboard with the following link:

   ```
   https://<CLUSTER_URL>/dkp/kommander/kubecost/frontend/
   ```

3. From the dashboard, select the **Settings** icon, then select “Add License Key”.
   NOTE: Alternatively, the dashboard can be accessed via the following link: `https://<CLUSTER_URL>/dkp/kommander/kubecost/frontend/settings`
4. Input your license key, then select “Update.”
5. After the license key has been added, the licensed features become available in the Kubecost UI.

There is a free trial of the Kubecost Enterprise plan that allows you to preview enterprise features for 30 days. To access this, select “Start Free Trial” in the **Settings** pane.

606 https://kubecost.com/
607 https://www.kubecost.com/pricing/
608 https://www.kubecost.com/pricing/

6.12.6.1.1 Considerations when Adding a License Key

Until you add a license key, Kubecost caches the context of the cluster you first navigate from. This means that if you navigate to the Kubecost UI via the dashboard button in the Application Dashboards tab of any cluster, you will not be able to access the centralized Kubecost UI until you clear your browser cookies/ cache.

6.12.6.2 Centralized Costs

Using Thanos, the management cluster collects cost metrics remotely from each attached cluster. Costs from the last day and the last 7 days are displayed for each cluster, workspace, and project in the respective DKP UI pages. Further cost analysis and details can be found in the centralized Kubecost UI running on Kommander, at:

```
https://<CLUSTER_URL>/dkp/kommander/kubecost/frontend/detail.html#&agg=cluster
```

For more information on cost allocation metrics and how to navigate this view in the Kubecost UI, see the Kubecost docs on Kubernetes Cost Allocation[609].

To identify the clusters in Kubecost, use the cluster's monitoring ID. The monitoring ID corresponds to the kube-system namespace UID of the cluster. To find the cluster's monitoring ID, select the **Clusters** tab on the DKP UI in the relevant workspace, or go to the **Clusters** page in the **Global** workspace:

```
https://<CLUSTER_URL>/dkp/kommander/dashboard/clusters
```

Select **View Details** on the attached cluster card. Select the **Configuration** tab, and find the monitoring ID under **Monitoring ID (clusterId)**.

You can also search or filter by monitoring ID on the **Clusters** page.

To look up a cluster's kube-system namespace UID directly using the CLI, run the following kubectl command, **using the cluster's context or kubeconfig**.

```
kubectl get namespace kube-system -o jsonpath='{.metadata.uid}'
```

6.12.6.2.1 Kubecost

Kubecost integrates directly with the Kubernetes API and cloud billing APIs to give you real-time visibility into your Kubernetes spend and cost allocation. By monitoring your Kubernetes spend across clusters, you can avoid overspend caused by uncaught bugs or oversights. With a cost monitoring solution in place you can realize the full potential and cost of these resources and avoid over-provisioning resources.

To customize pricing and out of cluster costs for AWS, you must apply these settings using the Kubecost UI running on each attached cluster. You can access the attached cluster's Kubecost Settings page at:

609 https://docs.kubecost.com/using-kubecost/getting-started/cost-allocation

```
https://<MANAGED_CLUSTER_URL>/dkp/kubecost/frontend/settings.html
```


Make sure you access the cluster's Kubecost UI linked above, not the centralized Kubecost UI running on the Kommander management cluster.

6.12.6.2.1.1 AWS

For more accurate AWS Spot pricing, follow these steps[610] to configure a data feed for the AWS Spot instances.

To allocate out of cluster costs for AWS, visit this guide[611].

6.12.6.2.2 Grafana dashboards

A set of Grafana dashboards providing visualization of cost metrics is provided in the centralized Grafana UI:

```
https://<CLUSTER_URL>/dkp/kommander/monitoring/grafana
```

These dashboards give a global view of accumulated costs from all attached clusters. From the navigation in Grafana, you can find these dashboards by selecting those tagged with `cost`, `metrics`, and `utilization`.

6.12.6.3 Related Information

For information on related topics or procedures, refer to the following:

- Kubecost documentation[612]

6.12.7 Monitoring Applications Using Prometheus

Before attempting to monitor your own applications, you should be familiar with the Prometheus conventions for exposing metrics. In general, there are two key recommendations:

- You should expose metrics using an HTTP endpoint named `/metrics`.
- The metrics you expose must be in a format that Prometheus can consume.

By following these conventions, you ensure that your application metrics can be consumed by Prometheus itself or by any Prometheus-compatible tool that can retrieve metrics, using the Prometheus client endpoint.

610 https://docs.kubecost.com/using-kubecost/getting-started/spot-checklist#implementing-spot-nodes-in-your-cluster
611 https://docs.kubecost.com/install-and-configure/advanced-configuration/cloud-integration/aws-cloud-integrations/aws-out-of-cluster
612 https://docs.kubecost.com/

The `kube-prometheus-stack` for Kubernetes provides easy monitoring definitions for Kubernetes services and deployment and management of Prometheus instances. It provides a Kubernetes resource called `ServiceMonitor`.

By default, the `kube-prometheus-stack` provides the following service monitors to collect internal Kubernetes components:

- kube-apiserver
- kube-scheduler
- kube-controller-manager
- etcd
- kube-dns/coredns
- kube-proxy

The operator is in charge of iterating over all of these `ServiceMonitor` objects and collecting the metrics from these defined components.

The following example illustrates how to retrieve application metrics. In this example, there are:

- Three instances of a simple app named `my-app`
- The sample app listens and exposes metrics on port 8080
- The app is assumed to already be running

To prepare for monitoring of the sample app, create a service that selects the pods that have `my-app` as the value defined for their app label setting.

The service object also specifies the port on which the metrics are exposed. The `ServiceMonitor` has a label selector to select services and their underlying endpoint objects. For example:

```
kind: Service
apiVersion: v1
metadata:
  name: my-app
  namespace: my-namespace
  labels:
    app: my-app
spec:
  selector:
    app: my-app
  ports:
  - name: metrics
    port: 8080
```

This service object is discovered by a `ServiceMonitor`, which defines the selector to match the labels with those defined in the service. The app label must have the value `my-app`.

In this example, in order for `kube-prometheus-stack` to discover this `ServiceMonitor`, add a specific label `prometheus.kommander.d2iq.io/select: "true"` in the `yaml`:

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: my-app-service-monitor
  namespace: my-namespace
  labels:
    prometheus.kommander.d2iq.io/select: "true"
spec:
  selector:
    matchLabels:
      app: my-app
  endpoints:
  - port: metrics
```

In this example, you would modify the Prometheus settings to have the operator collect metrics from the service monitor by appending the following configuration to the overrides ConfigMap:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-prometheus-stack-overrides
  namespace: ${WORKSPACE_NAMESPACE}
data:
  values.yaml: |
    ---
    prometheus:
      additionalServiceMonitors:
        - name: my-app-service-monitor
          selector:
            matchLabels:
              app: my-app
          namespaceSelector:
            matchNames:
              - my-namespace
          endpoints:
            - port: metrics
              interval: 30s
```

Official documentation about using a `ServiceMonitor` to monitor an app with the Prometheus-operator on Kubernetes can be found on this GitHub repository[613].

6.12.8 Set Storage Capacity for Prometheus

Follow the steps in this page to set a specific storage capacity for Prometheus.

When defining the requirements of a cluster, you can specify the capacity and resource requirements of Prometheus by modifying the settings in the overrides ConfigMap definition as shown below:

613 https://github.com/coreos/prometheus-operator/blob/master/Documentation/user-guides/getting-started.md#related-resources

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-prometheus-stack-overrides
  namespace: ${WORKSPACE_NAMESPACE}
data:
  values.yaml: |
    ---
    prometheus:
      prometheusSpec:
        resources:
          limits:
            cpu: "4"
            memory: "8Gi"
          requests:
            cpu: "2"
            memory: "6Gi"
      storageSpec:
        volumeClaimTemplate:
          spec:
            resources:
              requests:
                storage: "100Gi"
```

6.13 Storage for Applications

DKP ships with a Rook Ceph cluster, that is used as the primary blob storage for various DKP components in the logging stack and backups.

The pages in this section provide an overview of the Rook Ceph application in DKP, including information about its components, resource requirements, storage configuration information, and dashboard.

- Rook Ceph in DKP (see page 894)
- BYOS (Bring Your Own Storage) to DKP Clusters (see page 901)

6.13.1 Rook Ceph in DKP

DKP ships with a Rook Ceph cluster, that is used as the primary blob storage for various DKP components in the logging stack and backups.

The pages in this section provide an overview of the Rook Ceph application in DKP, including information about its components, resource requirements, storage configuration information, and dashboard.

The Ceph instance installed by DKP is intended only for use by the logging stack and `velero` platform applications.

If you have an instance of Ceph that is managed outside of the DKP lifecycle, see Bring Your Own Storage to DKP Clusters (see page 901).

- Rook Ceph in DKP - Prerequisites (see page 895)
- Rook Ceph Configuration (see page 896)
- Rook Ceph Dashboard (see page 900)

6.13.1.1 Rook Ceph in DKP - Prerequisites

Important information you should know prior to using Rook Ceph in DKP

The Ceph instance installed by DKP is intended only for use by the logging stack and `velero` platform applications.

If you have an instance of Ceph that is managed outside of the DKP lifecycle, see Bring Your Own Storage to DKP Clusters (see page 901).

If you do not plan on using any of the logging stack components such as `grafana-loki` or `velero` for backups, then you do not need Rook Ceph for your installation and you can disable it by adding the following to your installer config file:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  ...
  ...
  grafana-loki:
    enabled: false
  ...
  rook-ceph:
    enabled: false
  rook-ceph-cluster:
    enabled: false
  ...
  velero:
    enabled: false
...
```

You must enable `rook-ceph` and `rook-ceph-cluster` if any of the following is true:

- If `grafana-loki` is enabled.
- If `velero` is enabled. If you applied config overrides for `velero` to use a storage that is external to your cluster, (see page 778)then you do not need Ceph to be installed.

For more information about Ceph, refer to the following:

- Intro to Ceph — Ceph Documentation[614]
- Rook — Rook Ceph Documentation[615]

See Rook Ceph Configuration (see page 896) for information on how you can configure Ceph for you Ceph Envirnoment.

6.13.1.2 Rook Ceph Configuration

This page contains information about configuring Rook Ceph in your DKP Environment.

The Ceph instance installed by DKP is intended only for use by the logging stack and `velero` platform applications.

If you have an instance of Ceph that is managed outside of the DKP lifecycle, see Bring Your Own Storage to DKP Clusters (see page 901).

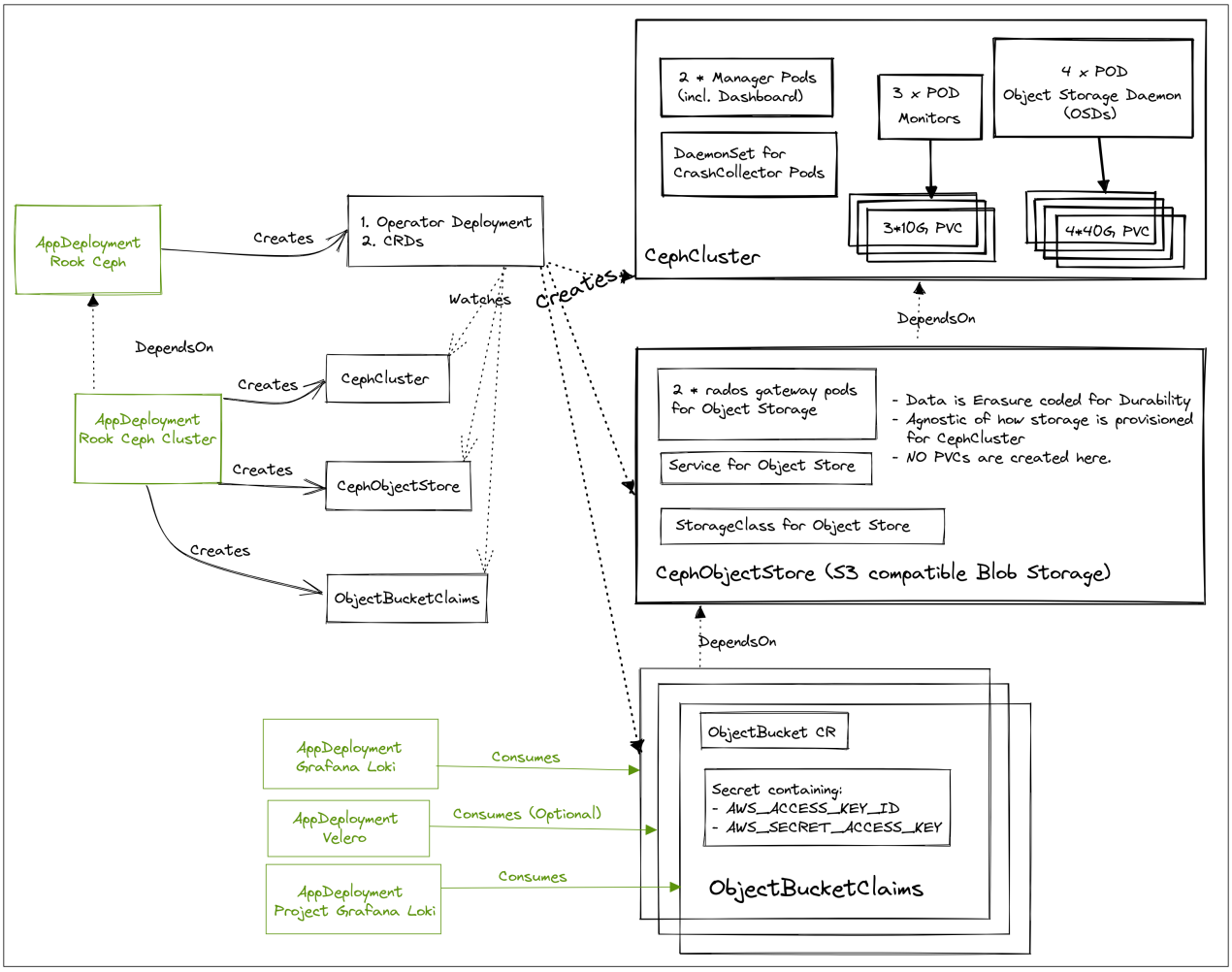
6.13.1.2.1 Components of a Rook Ceph Cluster

Ceph supports creating clusters in different modes as listed in CephCluster CRD - Rook Ceph Documentation[616]. DKP, specifically is shipped with a PVC Cluster, as documented in PVC Storage Cluster - Rook Ceph Documentation[617]. It is recommended to use the PVC mode to keep the deployment and upgrades simple and agnostic to technicalities with node draining.

Ceph cannot be your CSI Provisioner when installing in PVC mode as Ceph relies on an existing CSI provisioner to bind the PVCs created by it. It is possible to use Ceph as your CSI provisioner, but that is outside the scope of this document. If you have an instance of Ceph that acts as the CSI Provisoner, then it is possible to reuse it for your DKP Storage needs. See BYOS (Bring Your Own Storage) to DKP Clusters (see page 901) for information on reusing existing Ceph.

When you create `AppDeployments` for `rook-ceph` and `rook-ceph-cluster` platform applications results in the deployment of various components as listed in the following diagram:

614 https://docs.ceph.com/en/quincy/start/intro/
615 https://rook.io/docs/rook/v1.10/Getting-Started/intro/
616 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/
617 https://rook.io/docs/rook/v1.10/CRDs/Cluster/pvc-cluster/#pvc-storage-only-for-monitors



4 Rook Ceph Cluster Components

Items highlighted in green are user-facing and configurable.

Please refer to Rook Ceph Storage Architecture[618] and Ceph Architecture[619] for an in-depth explanation of the inner workings of the components outlined in the above diagram.

For additional details about the data model, refer to the Rook Ceph Data Model[620] page.

6.13.1.2.2 Resource Requirements

The following is a non-exhaustive list of the resource requirements for long running components of Ceph:

618 https://rook.io/docs/rook/v1.10/Getting-Started/storage-architecture/
619 https://docs.ceph.com/en/quincy/architecture/
620 https://github.com/rook/rook/blob/release-1.10/design/ceph/data-model.md

Type	Resources	Total
CPUs	100m x # of mgr instances (default 2) 250m x # of mon instances (default 3) 250m x # of osd instances (default 4) 100m x # of crashcollector instances (Daemonset i.e., # of nodes) 250m x # of rados gateway replicas (default 2)	~2000m CPU
Memory	512Mi x # of mgr instances (default 2) 512Gi x # of mon instances (default 3) 1Gi x # of osd instances (default 4) 500Mi x # of rados gateway replicas (default 2)	~8Gi Memory
Disk	4 x 40Gi PVCs with `Block` mode for ObjectStorageDaemons[621] 3 x 10Gi PVCs with `Block` or `FileSystem` mode for Mons[622]	190Gi

Your default `StorageClass` should support creation of `PersistentVolume`s that satisfy the `PersistentVolumeClaim`s created by Ceph with `volumeMode: Block`.

6.13.1.2.3 Ceph Storage Configuration

Ceph is highly configurable and can support Replication or Erasure Coding to ensure data durability[623]. DKP is configured to use Erasure Coding for maximum efficiency.

Primer on Replication Strategies

Replication and Erasure Coding are the two primary methods for storing data in a durable fashion in any distributed system.

6.13.1.2.3.1 Replication[624]

- For a replication factor of N, data has N copies (including the original copy)
- Smallest possible replication factor is 2 (usually this means 2 storage nodes).
    - With replication factor of 2, data has 2 copies and this tolerates loss of one copy of data.
- Storage efficiency : `(1/N) * 100` percentage. For example,
    - If `N=2`, then efficiency is `50%`.
    - If `N=3`, then efficiency is `33%` so on.

621 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings
622 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#mon-settings
623 https://en.wikipedia.org/wiki/Durability_(database_systems)
624 https://en.wikipedia.org/wiki/Replication_(computing)

- Fault Tolerance : `N-1` nodes can be lost without loss of data. For example,
    - If `N=2` , then atmost 1 node can be lost without data loss.
    - If `N=3` , then atmost 2 nodes can be lost without data loss and so on.

6.13.1.2.3.2 Erasure Coding[625]

- Slices an object into `k` data fragments and computes `m` parity fragments. The erasure coding scheme gaurentees that data can be recreated using any `k` fragments out of `k+m` fragments.
- The `k + m = n` fragments are spread across (`>=n`) Storage Nodes to offer durability.
- Since `k` out of `n` fragments (could be parity or could be data fragments) are needed for recreation of data, at most `m` fragments can be lost without loss of data.
- The smallest possible count is `k = 2` , `m = 1` i.e., `n = k + m = 3` . This works only if there are at least `n = 3` storage nodes.
- Storage efficiency: `k/(k+m) * 100` percentage. For example,
    - If `k=2, m=1` , then efficiency is `67%`
    - If `k=3, m=1` , then efficiency is `75%` and so on.
- Fault Tolerance: `m` nodes can be lost without loss of data. For example:
    - If `k=3, m=1` then atmost 1 out of 4 nodes can be lost without data loss.
    - If `k=4, m=2` then atmost 2 out of 6 nodes can be lost without data loss and so on.

The default configuration creates a `CephCluster` that creates 4 x `PersistentVolumeClaim` s of `40G` each, resulting in `160G` of raw storage. Erasure coding ensures durability with `k=3` data bits and `m=1` parity bits. This gives a storage efficiency of `75%` (refer to the primer above for calculation), which means `120G` of disk space is available for consumption by services like `grafana-loki` , `project-grafana-loki` , and `velero` .

It is possible to override replication strategy for logging stack (`grafana-loki`) and `velero` backups. Refer to the default configmap for the `CephObjectStore` at services/rook-ceph-cluster/1.10.3/defaults/cm.yaml#L126-L175[626] and override the replication strategy according to your needs by referring to CephObjectStore CRD[627] documentation.

For more information about configuring storage in Rook Ceph, refer to the following pages:

625 https://en.wikipedia.org/wiki/Erasure_code
626 https://github.com/mesosphere/kommander-applications/blob/v2.4.0/services/rook-ceph-cluster/1.10.3/defaults/cm.yaml#L126-L175
627 https://www.rook.io/docs/rook/v1.10/CRDs/Object-Storage/ceph-object-store-crd/

- Ceph OSD Management - Rook Ceph Documentation[628] - for general information on how to configure Object Storage Daemons (OSDs).
- Ceph Configuration - Rook Ceph Documentation[629] - for information on how to set up auto-expansion of OSDs.

See Also:

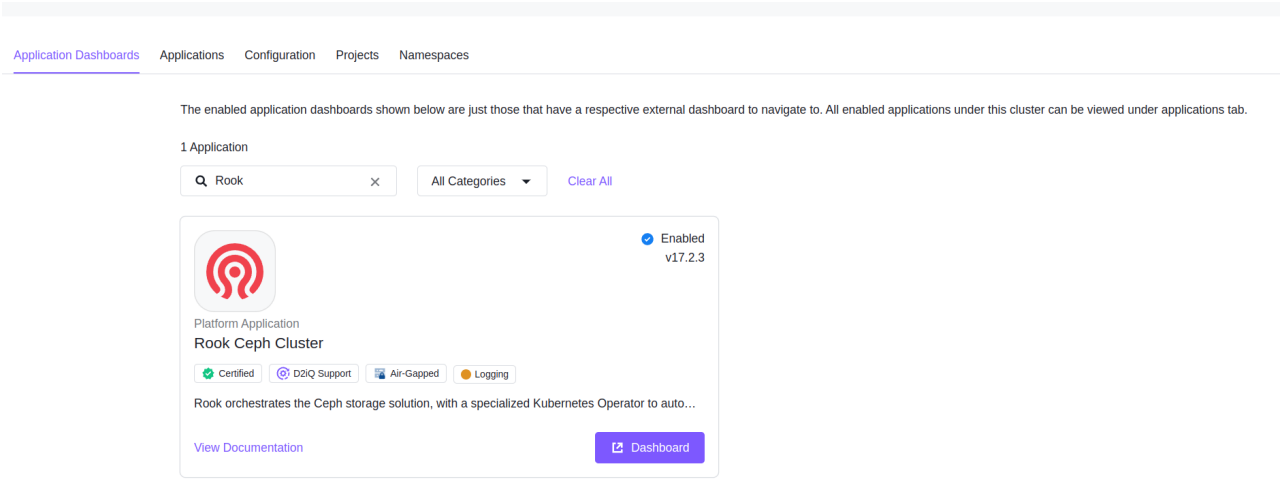
Rook Ceph in DKP - Prerequisites (see page 895)

Rook Ceph Dashboard (see page 900)

6.13.1.3 Rook Ceph Dashboard

6.13.1.3.1 Rook Ceph Dashboard Login

Rook Ceph Cluster provides a Dashboard which can be accessed from the UI in the Application Dashboards tab inside Kommander.



5 Rook Ceph Dashboard Link in the UI

The dashboard can be used to view current cluster health and logs.

6.13.1.3.1.1 How to Access the Rook Ceph Dashboard

1. Go to the applications dashboard
2. Select the Dashboard button
3. Username is `admin`

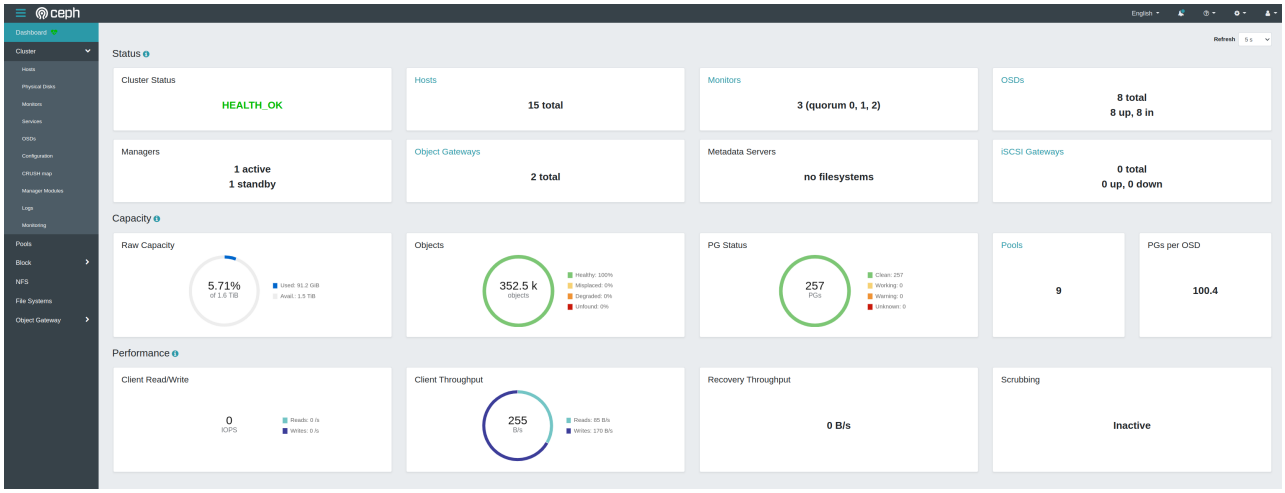
628 https://www.rook.io/docs/rook/v1.11/Storage-Configuration/Advanced/ceph-osd-mgmt/

629 https://www.rook.io/docs/rook/v1.11/Storage-Configuration/Advanced/ceph-configuration/?h=expa#auto-expansion-of-osds

4. To retrieve your password, in the command line **and using the kubeconfig of the Kubernetes cluster you have Rook Ceph deployed to**, run the following command:
   NOTE: Set the NAMESPACE variable according to your environment (`kommander` on management cluster or workspace namespace on attached clusters and managed clusters).

   ```
   kubectl get secret -n ${NAMESPACE} rook-ceph-dashboard-password -ogo-
   template="{{.data.password|base64decode}}"
   ```

5. Copy the password and paste it into the UI to access the dashboard. After successful login, a `Healthy` Rook Ceph Cluster dashboard will look similar to:



6 Rook Ceph Cluster Dashboard

6.13.2 BYOS (Bring Your Own Storage) to DKP Clusters

Ceph can be used as the CSI Provisioner in some environments. For environments where Ceph was installed before installing DKP, you can reuse your existing Ceph installation to satisfy the storage requirements of DKP Applications. Hello

> This guide assumes you have a Ceph cluster that is not managed by DKP.
> Refer to Rook Ceph Configuration (see page 896) for information on how to configure the Ceph instance installed by DKP for use by DKP platform applications.

6.13.2.1 Disable DKP Managed Ceph

Disable `rook-ceph` in your installer config since the default config of DKP has already installed a Ceph Cluster.

Disable `rook-ceph` in the installer config to prevent DKP from installing a Ceph cluster:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  rook-ceph:
    enabled: false
  rook-ceph-cluster:
    enabled: false
  ...
...
```

The DKP instances of `velero` and `grafana-loki` rely on the storage provided by Ceph. Before installing the Kommander component of DKP, be sure to configure appropriate Ceph resources for their usage as detailed in the next section.

6.13.2.2 Creating DKP Compatible Ceph Resources

This section walks you through the creation of `CephObjectStore` and then a set of `ObjectBucketClaim` s, which can be consumed by either `velero` and `grafana-loki` .

Typically, Ceph is installed in the `rook-ceph` namespace, which is the default namespace if you have followed the Quickstart - Rook Ceph Documentation[630] guide.

> This guide assumes your Ceph instance is installed in the `rook-ceph` namespace. Configure the variable `CEPH_NAMESPACE` in subsequent steps as it is applicable to your environment.

6.13.2.2.1 Creating `CephObjectStore`

There are two ways to install Ceph:

- Using Helm Charts (see page 902)
- Directly applying Kubernetes manifests (see page 904)

6.13.2.2.1.1 Helm Chart Values

This section is relevant if you have installed Ceph using `helm install` or some other managed Helm resource mechanism.

630 https://www.rook.io/docs/rook/v1.10/Getting-Started/quickstart/#create-a-ceph-cluster

If you have applied any configuration overrides to your Rook Ceph operator, ensure it was deployed with `currentNamespaceOnly` set to false[631] (It is the default value, so unless you have applied any overrides, it will be `false` by default). This ensures that the Ceph Operator in the `rook-ceph` namespace is able to monitor and manage resources in other namespaces such as `kommander`.

Ensure the following configuration[632] for `rook-ceph` helm chart[633]:

```
# This is the default value, so need to overwrite if you are just using the defaults
as-is
currentNamespaceOnly: false
```

You must enable the following configuration overrides[634] for the `rook-ceph-cluster` helm chart[635]:

```
cephObjectStores:
  - name: dkp-object-store
    # see https://github.com/rook/rook/blob/master/Documentation/CRDs/Object-Storage/
ceph-object-store-crd.md#object-store-settings for available configuration
    spec:
      metadataPool:
        # The failure domain: osd/host/(region or zone if available) - technically
also any type in the crush map
        failureDomain: osd
        # Must use replicated pool ONLY. Erasure coding is not supported.
        replicated:
          size: 3
      dataPool:
        # The failure domain: osd/host/(region or zone if available) - technically
also any type in the crush map
        failureDomain: osd
        # Data pool can use either replication OR erasure coding. Consider the
following example scenarios:
        # Erasure Coding is used here with 3 data chunks and 1 parity chunks which
assumes 4 OSDs exist.
        # Configure this according to your CephCluster specification.
        erasureCoded:
          dataChunks: 3
          codingChunks: 1
      preservePoolsOnDelete: false
      gateway:
        port: 80
        instances: 2
        priorityClassName: system-cluster-critical
        resources:
```

631 https://www.rook.io/docs/rook/v1.10/Helm-Charts/operator-chart/
632 https://www.rook.io/docs/rook/v1.10/Helm-Charts/operator-chart/#configuration
633 https://www.rook.io/docs/rook/v1.10/Helm-Charts/operator-chart/#release
634 https://www.rook.io/docs/rook/v1.10/Helm-Charts/ceph-cluster-chart/#ceph-object-stores
635 https://www.rook.io/docs/rook/v1.10/Helm-Charts/ceph-cluster-chart/#release

```
        limits:
          cpu: "750m"
          memory: "1Gi"
        requests:
          cpu: "250m"
          memory: "500Mi"
    healthCheck:
      bucket:
        interval: 60s
  storageClass:
    enabled: true
    name: dkp-object-store
    reclaimPolicy: Delete
```

6.13.2.2.1.2 Managing resources directly

1. Set a variable to refer to the namespace the `AppDeployment`s are created in.

   NOTE: This is the `kommander` namespace on the management cluster or `Workspace` namespace on all other clusters.

   ```
   export CEPH_NAMESPACE=rook-ceph
   export NAMESPACE=kommander
   ```

2. Create `CephObjectStore` in the same namespace as the `CephCluster`:

   ```
   cat <<EOF | kubectl apply -f -
   apiVersion: ceph.rook.io/v1
   kind: CephObjectStore
   metadata:
     name: dkp-object-store
     namespace: ${CEPH_NAMESPACE}
   spec:
     metadataPool:
       # The failure domain: osd/host/(region or zone if available) - technically,
   any type in the crush map
       failureDomain: osd
       # Must use replicated pool ONLY. Erasure coding is not supported.
       replicated:
         size: 3
     dataPool:
       # The failure domain: osd/host/(region or zone if available) - technically,
   any type in the crush map
       failureDomain: osd
       # Data pool can use either replication OR erasure coding. Consider the
   following example scenarios:
       # Erasure Coding is used here with 3 data chunks and 1 parity chunks which
   assumes 4 OSDs exist.
   ```

```
    # Configure this according to your CephCluster specification.
    erasureCoded:
      dataChunks: 3
      codingChunks: 1
  preservePoolsOnDelete: false
  gateway:
    port: 80
    instances: 2
    priorityClassName: system-cluster-critical
    resources:
      limits:
        cpu: "750m"
        memory: "1Gi"
      requests:
        cpu: "250m"
        memory: "500Mi"
  healthCheck:
    bucket:
      interval: 60s
EOF
```

3. Wait for the `CephObjectStore` to be `Connected`:

```
$ kubectl get cephobjectstore -A
NAMESPACE   NAME                PHASE
rook-ceph   dkp-object-store    Progressing
...
...
rook-ceph   dkp-object-store    Connected
```

4. Create a `StorageClass` to consume the object storage:

```
cat <<EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: dkp-object-store
parameters:
  objectStoreName: dkp-object-store
  objectStoreNamespace: ${CEPH_NAMESPACE}
provisioner: ${CEPH_NAMESPACE}.ceph.rook.io/bucket
reclaimPolicy: Delete
volumeBindingMode: Immediate
EOF
```

6.13.2.2.2 Creating `ObjectBucketClaim`s

1. Once the Object Store is `Connected`, create the `ObjectBucketClaim` in the same namespace as `velero` and `grafana-loki`.
   This results in the creation of `ObjectBucket`, which creates `Secret`s that are consumed by `velero` and `grafana-loki`.
   a. For `grafana-loki`:

```
cat <<EOF | kubectl apply -f -
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: dkp-loki
  namespace: ${NAMESPACE}
spec:
  additionalConfig:
    maxSize: 80G
  bucketName: dkp-loki
  storageClassName: dkp-object-store
EOF
```

   b. For `velero`:

```
cat <<EOF | kubectl apply -f -
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: dkp-velero
  namespace: ${NAMESPACE}
spec:
  additionalConfig:
    maxSize: 10G
  bucketName: dkp-velero
  storageClassName: dkp-object-store
EOF
```

2. Wait for the `ObjectBucket`s to be `Bound` by executing the following command:

```
kubectl get objectbucketclaim -n${NAMESPACE} -ocustom-
columns='NAME:.metadata.name,PHASE:.status.phase'
```

   which should display something similar to:

```
NAME         PHASE
dkp-loki     Bound
dkp-velero   Bound
```

6.13.2.3 Configure Loki to use S3 Compatible Storage

If you wish to use your own storage in DKP that is S3 compatible, create a secret that contains your AWS secret credentials.

```
apiVersion: v1
data:
  AWS_ACCESS_KEY_ID: base64EncodedValue
  AWS_SECRET_ACCESS_KEY: base64EncodedValue
kind: Secret
metadata:
  name: dkp-loki #If you want to configure a custom name here also use it in the step
below
  namespace: kommander
```

6.13.2.4 Overriding `velero` and `grafana-loki` Configuration

Once all the buckets are in the `Bound` state, DKP applications are now ready to be installed with the following configuration overrides populated in the installer config:

```
cat <<EOF | kubectl apply -f -
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  grafana-loki:
    enabled: true
    values: |
      loki:
        structuredConfig:
          storage_config:
            aws:
              s3: "http://rook-ceph-rgw-dkp-object-store.${CEPH_NAMESPACE}.svc:80/
dkp-loki"
      ingester:
        extraEnvFrom:
          - secretRef:
              name: dkp-loki # Optional: This is the default value
      querier:
        extraEnvFrom:
          - secretRef:
```

```
            name: dkp-loki # Optional: This is the default value
      queryFrontend:
        extraEnvFrom:
          - secretRef:
              name: dkp-loki # Optional: This is the default value
      compactor:
        extraEnvFrom:
          - secretRef:
              name: dkp-loki # Optional: This is the default value
      ruler:
        extraEnvFrom:
          - secretRef:
              name: dkp-loki # Optional: This is the default value
      distributor:
        extraEnvFrom:
          - secretRef:
              name: dkp-loki # Optional: This is the default value
  velero:
    enabled: true
    values: |
      configuration:
        provider: "aws"
        backupStorageLocation:
          bucket: dkp-velero
          config:
            region: dkp-object-store
            s3Url: http://rook-ceph-rgw-dkp-object-store.${CEPH_NAMESPACE}.svc:80/
      credentials:
        # This secret is owned by the ObjectBucketClaim. A ConfigMap and a Secret
with same name as bucket are created.
        extraSecretRef: dkp-velero
EOF
```

This installer config can be merged with your installer config with any other relevant configuration before installing DKP.

6.13.2.5 Overriding `project-grafana-loki` Configuration

When installing project level grafana loki, its configuration needs to be overridden in a similar manner to workspace level grafana loki, so that the project logs can be persisted in Ceph storage.

The following overrides need to be applied to `project-grafana-loki` :

```
loki:
  structuredConfig:
    storage_config:
      aws:
        s3: "http://rook-ceph-rgw-dkp-object-store.${CEPH_NAMESPACE}.svc:80/dkp-loki"
```

These overrides can be applied from the UI directly while substituting the `${CEPH_NAMESPACE}` appropriately.

If you are using using CLI, follow these steps:

1. Set `NAMESPACE` to project namespace and `CEPH_NAMESPACE` to Ceph install namespace:

```
export CEPH_NAMESPACE=rook-ceph
export NAMESPACE=my-project
```

2. Create a `ConfigMap` to apply the configuration overrides:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
data:
  values.yaml: |
    loki:
      structuredConfig:
        storage_config:
          aws:
            s3: "http://rook-ceph-rgw-dkp-object-store.$
{CEPH_NAMESPACE}.svc:80/proj-loki-${NAMESPACE}"
kind: ConfigMap
metadata:
  name: project-grafana-loki-ceph
  namespace: ${NAMESPACE}
EOF
```

3. Create the `AppDeployment` with a reference to the above `ConfigMap`:

   NOTE: The `clusterSelector` can be adjusted according to your needs.

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: project-grafana-loki
  namespace: ${NAMESPACE}
spec:
  appRef:
    kind: ClusterApp
    name: project-grafana-loki-0.48.6
  clusterSelector: {}
  configOverrides:
    name: project-grafana-loki-ceph
EOF
```

The project level Grafana Loki creates an `ObjectBucketClaim` and assumes that the Ceph operator is montitoring the project namespace, so there is no need to create `ObjectBucketClaim` manually.

6.14 DKP Troubleshooting

The following pages provide the steps to troubleshoot a Kubernetes cluster installation.

- Generate a Support Bundle (see page 910)
- Custom Collectors (see page 914)

6.14.1 Generate a Support Bundle

Follow these instructions to generate a support bundle with data collected for the last 48 hours of the life of the cluster.

6.14.1.1 Prerequisites

Before generating a support bundle, verify that you have:

- An AMD64-based Linux or macOS machine with a supported version of the operating system.
- A running Kubernetes cluster.
- Access to the DKP CLI (see page 1436).

6.14.1.2 Create a Diagnostic Bundle

`dkp diagnose` was developed by D2iQ and builds on the open source `troubleshoot.sh` project.

The command `dkp diagnose` is based on version `0.13.16` of `troubleshoot.sh` with custom modifications. The D2iQ fork is open source and available from on this public GitHub repository[636].

`dkp diagnose` supports multiple support bundle collectors[637] and can be configured as a `SupportBundle` Kubernetes resource in a yaml file.

The following list is the minimum set of resources that is required to debug a cluster, but can be further customized.

The bundle uses the following collectors:

636 https://github.com/mesosphere/troubleshoot
637 https://troubleshoot.sh/docs/collect/all/

- clusterInfo[638] collects basic information about the cluster
- clusterResources[639] collects a subset of available resources in the cluster
- configMap[640] collects the values of Kubernetes ConfigMaps
- secrets[641] collects the values of Kubernetes ConfigMaps
- execCopyFromHost (see page 0) runs a container on each node on the cluster and copies the created data
- allLogs (see page 914) is capable of collecting logs from all containers on the cluster

6.14.1.3 Generate a Support Bundle

> The command `dkp diagnose` uses the same Kubernetes configuration as `kubectl`. `dkp diagnose` can also be pointed at a specific configuration by using the `--kubeconfig` parameter.

To generate the support bundle, perform the following steps:

1. Run the `dkp diagnose` command by running the default collectors configuration.

   ```
   dkp diagnose
   ```

   The output looks similar to this:

   ```
   Collecting support bundle ...

   support-bundle-2021-08-13T14_44_23.tar.gz
   ```

2. To view the bundle contents, extract the bundle (replacing `support-bundle-2021-08-13T14_44_23.tar.gz` with the location from the previous step):

   ```
   tar -xzvf support-bundle-2021-08-13T14_44_23.tar.gz
   ```

3. A new directory named `support-bundle-<date-created>` is created. This directory contains the files specified:

638 https://troubleshoot.sh/docs/collect/cluster-info/
639 https://troubleshoot.sh/docs/collect/cluster-resources/
640 https://troubleshoot.sh/docs/collect/configmap/
641 https://troubleshoot.sh/docs/collect/secret/

```
ls support-bundle-2021-08-13T14_44_23
```

The output looks similar to this:

```
cluster-info  cluster-resources  configmaps  node-diagnostics  pod-logs
secrets  version.yaml
```

6.14.1.3.1 Collect Information from a Bootstrap Cluster

In the case where your bootstrap cluster has not yet pivoted towards your Konvoy cluster, you can collect log information from that bootstrap cluster as well, and there are a preconfigured set of relevant collectors. Specify an additional bootstrap cluster kubeconfig using the `--bootstrap-kubeconfig` parameter to activate bootstrap cluster diagnostics. You will receive an additional support bundle named `bootstrap-support-bundle-<date created>`.

Note that the bootstrap cluster diagnostics are independent of the configuration of the “main” or Konvoy cluster diagnostics. We run a static collector set that collects the following bootstrap cluster information:

- ClusterInfo
- ClusterResources
- AllLogs
- ConfigMaps
- Secrets

1. Run the `dkp diagnose` command with bootstrap bundle configuration.

```
dkp diagnose bundle.yaml
```

6.14.1.3.2 Customizations

To print the default collectors configuration, run the following command:

```
dkp diagnose default-config > bundle.yaml
```

Edit the file to make appropriate modifications.

By default, `dkp diagnose` does not require that you supply a configuration. You can print the default bundle by running `dkp diagnose default-config`.

6.14.1.4 SSH Fallback

In some cases the Kubernetes API is not available for the cluster. In those cases you can collect node level information using SSH access to the diagnosed nodes. Be aware that not all clusters have SSH access configured. If they do not then access using SSH fallback is not possible.

To get node level information from your cluster using SSH access, perform the following steps:

1. Enter the following command:

```
dkp diagnose ssh <path/to/ansible-inventory.yaml>
```

The `ansible-inventory.yaml` file specifies the nodes to access for data collection.

This collector does not use the full Ansible `inventory.yaml` format only a limited subset to describe the infrastructure.

Only the following attributes of the `ansible-inventory.yaml` are supported. All other group definitions are ignored.

- Support for `all` shared variables.
- Support for `hosts` key in `all` groups.
- Supported behavioral inventory is limited to:
    - `ansible_host`
    - `ansible_port`
    - `ansible_user`
    - `ansible_ssh_private_key_file`

  The following is an example `inventory.yaml` file:

```
all:
  vars:
    ansible_user: centos
  hosts:
    host-1:
      ansible_host: 192.168.10.1
    host-2:
      ansible_host: 192.168.10.22
      ansible_port: 2222
```

More information on these Ansible parameters can be found in the Ansible user guide[642].

> All other group definitions in the `inventory.yaml` file are ignored.

Refer to the following example file:

```
all:
  vars:
    ansible_user: centos
  hosts:
    host-1:
      ansible_host: 192.168.10.1
    host-2:
      ansible_host: 192.168.10.22
      ansible_port: 2222
```

The fallback collector runs a bash script over SSH and copies the collected data. The format of the created bundle matches that of `dkp diagnose` collector generated bundles.

```
    node-diagnostics/<HOSTNAME_PORT>/data/
        - dmesg
        - ....
```

Redactors are supported and are in the same format as the main `dkp diagnose` command. Per node collection timeouts are supported using the `--timeout` parameter.

See also: dkp-cli (see page 1436)

6.14.2 Custom Collectors

For creating diagnostic bundles, D2iQ is using a customized version of `troubleshoot.sh` integrated into `dkp-diagnose`.

6.14.2.1 Customizations

To meet the specific needs of diagnosing DKP 2 clusters we have developed custom collectors and modified the behavior of upstream collectors. Go to our repository for more information on the details[643] of the changes.

642 https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#connecting-to-hosts-behavioral-inventory-parameters
643 https://github.com/mesosphere/troubleshoot/blob/v0.13-d2iq/README.d2iq.md

6.14.2.1.1 `ExecCopyFromHost` Collector

This is a new collector created specifically for gathering host level information from cluster nodes. The collector allows you to run a provided container image in a privileged mode, as a root user, with additional Linux capabilities and with the host filesystem mounted in the container.

You can collect host level information other than copying host level files. (This is already possible with the `CopyFromHost` collector.) Like the `CopyFromHost` collector, this collector runs as a Kubernetes `DaemonSet` executed on all nodes in the system. The data produced by the container are copied from a pre-defined directory into the diagnostics bundle under each node name. The name of the parent directory, in the diagnostics bundle, is determined by the name of the collector specified in its configuration.

The data written into the diagnostics bundle follows this format:

```
<collector-name> / <node-name> / data / (file1|file2|...)
```

The following is a sample configuration file:

```
spec:
  collectors:
    - execCopyFromHost:
        name: node-diagnostics
        image: mesosphere/dkp-diagnostics-node-collector:latest
        timeout: 30s
        command:
          - "/bin/bash"
          - "-c"
          - "/diagnostics/container.sh --hostroot /host --hostpath ${PATH} --
outputroot /output"
        workingDir: "/diagnostics"
        includeControlPlane: true
        privileged: true
        capabilities:
          - AUDIT_CONTROL
          - AUDIT_READ
          - BLOCK_SUSPEND
          - BPF
          - CHECKPOINT_RESTORE
          - DAC_READ_SEARCH
          - IPC_LOCK
          - IPC_OWNER
          - LEASE
          - LINUX_IMMUTABLE
          - MAC_ADMIN
          - MAC_OVERRIDE
          - NET_ADMIN
          - NET_BROADCAST
          - PERFMON
```

```
          - SYS_ADMIN
          - SYS_BOOT
          - SYS_MODULE
          - SYS_NICE
          - SYS_PACCT
          - SYS_PTRACE
          - SYS_RAWIO
          - SYS_RESOURCE
          - SYS_TIME
          - SYS_TTY_CONFIG
          - SYSLOG
          - WAKE_ALARM
        extractArchive: true
```

The following is an example of the data produced by running this collector:

```
├── node-diagnostics
│   ├── troubleshoot-control-plane
│   │   └── data
│   │       ├── certs_expiration_kubeadm
│   │       ├── containerd_config.toml
                    ...
│   │       └── whoami_validate
│   └── troubleshoot-worker
│       └── data
│           ├── containerd_config.toml
│           ├── containers_crictl
                    ...
│           └── whoami_validate
```

In the event that an error occurs while collecting node diagnostics, the `node-diagnostics/<node>/pod-collector.json` file contains the serialized JSON representations of the running pod. This helps debug the reasons for the collection failure. The `node-diagnostics/<node>/pod-collector.log` file contains stdout from the collector container that runs the diagnostics script. In addition, the command may also produce certain `-error.txt` files. `file-copy-error.txt` and `pod-collector-files-copy-error.txt` are two file examples. These files contain error messages generated while trying to fetch log files from the collector.

When using this collector for node level information you must run additional docker containers and must have the following docker images:

- `mesosphere/pause-alpine:3.2`
- `mesosphere/dkp-diagnostics-node-collector:$(dkp-diagnose version)`

For more information on the configuration options see the `ExecCopyFromHost` in the `pkg/apis/troubleshoot/v1beta2/exec_copy_from_host.go` file.

6.14.2.1.2 `AllLogs` Collector

This collector gathers pod logs from specified namespaces or from all namespaces if none are specified. You can collect logs of all the pods from all the namespaces. The pod logs are collected under the `allPodLogs` directory.

The data written into the diagnostics bundle follows this format:

```
<collector-name> / <namespace-name> / <pod-name> - (container1|container2|...)
```

The following is a sample configuration file to collect logs from all the pods from all the namespaces:

```
spec:
  collectors:
    - allLogs:
        namespaces:
          - "*"
```

The following is a sample configuration file to collect logs from all the pods from specific namespaces:

```
spec:
  collectors:
    - allLogs:
        namespaces:
          - default
          - dev
          - prod
```

The following is an example of the data produced by running this collector:

```
├── node-diagnostics
│   ├── troubleshoot-control-plane
│   │   └── data
│   │       ├── certs_expiration_kubeadm
│   │       ├── containerd_config.toml
│   │                ...
│   │       └── whoami_validate
│   └── troubleshoot-worker
│       └── data
│           ├── containerd_config.toml
│           ├── containers_crictl
│                    ...
│           └── whoami_validate
```

In the event that an error occurs while collecting node diagnostics, the `node-diagnostics/<node>/pod-collector.json` file contains the serialized JSON representations of the running pod. This helps debug the reasons for the collection failure. The `node-diagnostics/<node>/pod-collector.log` file contains stdout from the collector container that runs the diagnostics script.

When using this collector for node level information you must run additional docker containers and must have the following docker images:

- `mesosphere/pause-alpine:3.2`
- `mesosphere/dkp-diagnostics-node-collector:$(dkp-diagnose version)`

For more information on the configuration options see the `ExecCopyFromHost` in the `pkg/apis/troubleshoot/v1beta2/exec_copy_from_host.go` file.

6.14.2.1.3 Collect from all Namespaces for `ConfigMap` and `Secret` Collector

Support for collecting from all namespaces for `ConfigMap` and `Secret` collector.

In the original collectors `namespace` there is a required parameter. This adds support for collecting from all namespaces by not setting the `namespace` (or setting it to `""`). Note: To collect all config maps / secrets an empty selector must be used (`selector: [""]`).

6.14.2.1.4 Support for Optional support-bundle Name Prefix

When generating a support bundle, you need naming defaults to provide deterministic bundle identifiers. This feature is especially useful for our convenience extension of providing diagnostics for both, a bootstrap, Konvoy, or other K8s cluster. Using an empty prefix keeps the original naming convention.

6.14.2.1.5 `ClusterResources` Collector

Another customization is added to collect custom resource definitions and all custom resources in the cluster.

7 Additional Infrastructure Customized Configuration

The Konvoy component of DKP can be customized for different environments and infrastructures depending on your network technology choices. If you have already installed using Additional Infrastructure Customized Configuration (see page 919) instructions, you may find helpful tools in this section. However, if you have not already installed DKP with the Basic Install instructions, find your infrastructure and begin the process of custom DKP installation in this area.See the following sections for more information on custom settings:

7.1 Universal Configurations for all Infrastructure Providers

7.2 AWS Infrastructure

7.3 EKS Infrastructure

- Delete EKS Cluster from UI (see page 1032)

7.4 Azure Infrastructure

- Azure Prerequisites (see page 1033)
- Azure using Konvoy Image Builder (see page 1036)
- Azure Bootstrap (see page 1039)
- Create a New Custom Azure Cluster (see page 1041)
- Explore new Azure Cluster (see page 1050)
- Azure Make new Cluster Self-Managed (see page 1054)
- Azure Certificate Renewal (see page 1057)
- Azure Replace a Node (see page 1059)
- Azure Delete Cluster (see page 1062)
- Create a new Azure Cluster Using the DKP UI (see page 1066)

7.5 AKS Infrastructure

- Create a New AKS Cluster (see page 1068)
- Explore New AKS Cluster (see page 1072)
- Delete AKS Cluster (see page 1075)
- Create a new AKS Cluster via UI (see page 1077)

7.6 Pre-provisioned Infrastructure

- Advanced Workflow (see page 1079)
- Pre-provisioned Prerequisite Configurations (see page 1079)
- Pre-provisioned Air-gapped Define Environment (see page 1082)
- Pre-provisioned Set Infrastructure (see page 1087)
- Pre-provisioned Define Control Plane Endpoint (see page 1090)
- Pre-provisioned Create Secrets and Overrides (see page 1091)
- Pre-provisioned Bootstrap Cluster (see page 1097)
- Pre-provisioned Create a New Cluster (see page 1098)
- Pre-provisioned Azure only Configurations (see page 1106)
- Pre-provisioned Modify the Calico Installation (see page 1110)
- Pre-provisioned Built-in Virtual IP (see page 1114)
- Provision on the Flatcar Linux OS (see page 1115)
- Pre-provisioned Use HTTP Proxy (see page 1115)
- Pre-provisioned Use Alternate Pod or Service Subnets (see page 1117)
- Pre-provisioned Make Cluster Self-managed (see page 1118)
- Pre-provisioned Configure MetalLB (see page 1121)
- Pre-provisioned Create and Delete Node Pools (see page 1123)
- Pre-provisioned Add Nodes to Existing Node Pool (see page 1125)

- GPU Nodepools in a Pre-provisioned Environment (see page 1127)
- Pre-provisioned Delete Cluster (see page 1129)

7.7 vSphere Infrastructure

- vSphere Prerequisites (see page 1133)
- Create a Base OS image in vSphere (see page 1138)
- Create a VM Template (see page 1140)
- vSphere Bootstrap (see page 1142)
- Create new vSphere Cluster (see page 1145)
- Explore a vSphere Cluster (see page 1152)
- Make vSphere Cluster Self-Managed (see page 1156)
- Configure MetalLB for a vSphere infrastructure (see page 1159)
- Install vSphere Air-Gapped (see page 1162)
- vSphere Certificate Renewal (see page 1182)
- Delete vSphere Cluster (see page 1184)
- Manage vSphere Node Pools (see page 1188)

7.8 GCP Infrastructure

- GCP Prerequisites (see page 1199)
- GCP Konvoy Image Builder (see page 1201)
- Bootstrap GCP (see page 1205)
- Create a New GCP Cluster (see page 1206)
- Explore the GCP Cluster (see page 1210)
- Make the New GCP Cluster Self-Managed (see page 1213)
- Manage GCP Node Pools (see page 1216)
- Delete a GCP Cluster (see page 1223)

7.9 Universal Configurations for all Infrastructure Providers

Several areas of DKP configuration are common amongst all amongst all infrastructure providers. Some of the universal configurations are described in this section, and some pages include links to expanded information for these topics.

- Configuring an HTTP/HTTPS Proxy (see page 922)
- Working with Load Balancers (see page 928)
- Registry and Registry Mirrors (see page 929)
- Subnets and Pods (see page 929)

7.9.1 Alternative Mirror

If you need to configure a private registry using an alternative mirror, see this page (see page 1350) for details.

7.9.2 Additional Configurations

More information regarding global configurations or customization of specific components can be found in the Additional Configurations (see page 1282) section of the documentation.

- Konvoy Image Builder (see page 1282)
- FIPS 140-2 Compliance (see page 1342)
- Local Registry Tools (see page 1349)
- Air-gapped Seed the Registry (see page 1350)
- Configure the Control Plane (see page 1351)
- Update Cluster Nodepools (see page 1359)

7.9.3 Configuring an HTTP/HTTPS Proxy

When creating a DKP cluster in environments that use an HTTP/HTTPS proxy, you must provide proxy details. The proxy values are strings that list a set of proxy servers, URLs, or wildcard addresses that is specific to your environment.

When creating a DKP cluster in a proxied environment, you need to specify proxy settings for the:

- Bootstrap cluster
- CAPI components
- DKP Kommander component

When you create a DKP cluster using the `--self-managed` flag, the bootstrap cluster and CAPI components are created for you automatically, and use the HTTP and HTTPS proxy settings you specify in the `dkp create cluster <provider>...` command.

You can also create the bootstrap cluster and CAPI components manually, using the appropriate commands, `dkp create bootstrap` and `dkp create capi-components` respectively, combined with the command line flags to include your HTTP/S proxy information.

You can also specify HTTP/S proxy information in an override file when using Konvoy Image Builder (KIB)[644].

Without these values provided as part of the relevant `dkp create` command, DKP cannot create the requisite parts of your new cluster correctly. This is true of both management and managed clusters[645] alike.

> ⚠ For DKP installation, you should create the bootstrap cluster from within the same network in which the new cluster will run. Using a bootstrap cluster on a laptop with different proxy settings, for example, or residing in a different network, could cause problems.

You can define HTTP/HTTP proxy information using the steps in these pages:

644 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29918909/Konvoy+Image+Builder
645 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/168788158/Cluster+Types+and+Concepts

7.9.3.1 Configure the Bootstrap Cluster HTTP/HTTPS Proxy Settings

When creating a bootstrap cluster, you must locate the device used to create the bootstrap in the same proxied environment in which the workload cluster will run. D2iQ does not recommend creating a bootstrap cluster from outside a proxied environment.

When you first install DKP, the API server doesn't exist yet in the bootstrap environment, because the API server is created *during* cluster creation. To create a bootstrap server in a proxied environment, you need to include the following flags:

- `--http-proxy <<http proxy list>>`
- `--https-proxy <<https proxy list>>`
- `--no-proxy <<no proxy list>>`

The following is an example `dkp create bootstrap` command's syntax with the HTTP proxy settings included:

```
dkp create bootstrap --http-proxy <<http proxy list>> --https-proxy <<https proxy
list>> --no-proxy <<no proxy list>>
```

7.9.3.1.1 Create a Bootstrap Cluster with HTTP Proxy Settings

Note that the delimiter between each proxy value within a flag is a comma (,) with no space character following it. The flags can include a mix of IP addresses and domain names.

1. If an HTTP proxy is required, locate the values to use for the `http_proxy` , `https_proxy` , and `no_proxy` flags. They will be built into the bootstrap cluster during cluster creation.
2. Create a bootstrap cluster using this command syntax, in addition to any other flags you may need:

   ```
   dkp create bootstrap --kubeconfig $HOME/.kube/config \
     --http-proxy <string> \
     --https-proxy <string> \
     --no-proxy <string>
   ```

   This code sample shows the command with example values for the proxy settings:

   ```
   dkp create bootstrap \
      --http-proxy 10.0.0.15:3128 \
      --https-proxy  10.0.0.15:3128 \
      --no-proxy 127.0.0.1,192.168.0.0/16,10.0.0.0/16,10.96.0.0/12,169.254.169.254,1
   69.254.0.0/24,localhost,kubernetes,kubernetes.default,kubernetes.default.svc,ku
   bernetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.clu
   ster,.svc.cluster.local,.svc.cluster.local.,kubecost-prometheus-
   ```

```
server.kommander,logging-operator-logging-
fluentd.kommander.svc.cluster.local,elb.amazonaws.com
```

7.9.3.2 Create CAPI Components with HTTP/HTTPS Proxy Settings

Creating CAPI components for a DKP cluster from the command line requires HTTP/HTTPS proxy information, if your environment is proxied.

If you created a cluster without using the `--self-managed flag`, the cluster will not have any of the CAPI controllers or the cert-manager component. This means that the cluster will be managed from the context of the cluster from which it was created such as the bootstrap cluster. However, you can transform the cluster to a self-managed cluster by performing the commands:
`dkp create capi-components --kubeconfig=<newcluster>`

and…
`dkp move --to-kubeconfig=<newcluster>`

This combination of actions is sometimes called a **pivot**.

When creating the CAPI components for a proxied environment using the DKP command line interface, you must include the following flags :

- `--http-proxy <<http proxy list>>`
- `--https-proxy <<https proxy list>>`
- `--no-proxy <<no proxy list>>`

The following is an example `dkp create capi-components` command's syntax with the HTTP proxy settings included:

```
dkp create capi-components --http-proxy <<http proxy list>> --https-proxy <<https
proxy list>> --no-proxy <<no proxy list>>
```

7.9.3.2.1 Create CAPI Components with HTTP Proxy Settings

Note that the delimiter between each proxy value within a flag is a comma (,) with no space character following it. The flags can include a mix of IP addresses and domain names.

1. If an HTTP proxy is required, locate the values to use for the `http_proxy`, `https_proxy`, and `no_proxy` flags. They will be built into the CAPI components during their creation.
2. Create CAPI components using this command syntax, in addition to any other flags you may need:

```
dkp create capi-components --kubeconfig $HOME/.kube/config \
  --http-proxy <string> \
  --https-proxy <string> \
  --no-proxy <string>
```

This code sample shows the command with example values for the proxy settings:

```
dkp create capi-components \
   --http-proxy 10.0.0.15:3128 \
   --https-proxy  10.0.0.15:3128 \
   --no-proxy 127.0.0.1,192.168.0.0/16,10.0.0.0/16,10.96.0.0/12,169.254.169.254,1
69.254.0.0/24,localhost,kubernetes,kubernetes.default,kubernetes.default.svc,ku
bernetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.clu
ster,.svc.cluster.local,.svc.cluster.local.,kubecost-prometheus-
server.kommander,logging-operator-logging-
fluentd.kommander.svc.cluster.local,elb.amazonaws.com
```

7.9.3.3 Create a Cluster with HTTP/HTTPS Proxy

During cluster creation, you may need to configure the control plane and worker nodes to use an HTTP proxy. This can occur during installation of the Konvoy component of DKP, or when creating a managed[646] cluster.

If you require HTTP proxy configurations, you can apply them during the `create` operation by adding the appropriate flags to the `create cluster` command example below:

Proxy configuration	Flag
HTTP proxy for control plane machines	`--control-plane-http-proxy string`
HTTPS proxy for control plane machines	`--control-plane-https-proxy string`
No Proxy list for control plane machines	`--control-plane-no-proxy strings`
HTTP proxy for worker machines	`--worker-http-proxy string`
HTTPS proxy for worker machines	`--worker-https-proxy string`
No Proxy list for worker machines	`--worker-no-proxy strings`

646 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273319974

You must apply the same configuration to any custom machine images built with the Konvoy Image Builder (KIB) by using an HTTP override[647] file. For more information, refer to Use Override Files with Konvoy Image Builder[648] section of the documentation.

7.9.3.3.1 Configure the Control plane and Worker Nodes to Use HTTP/S proxy

This method for configuring the HTTP proxy values uses environment variables. (You are not required to use this method.)

Review this sample code for configuring environment variables for the control plane and worker nodes, taking into account the list of considerations that follows the sample.

```
export CONTROL_PLANE_HTTP_PROXY=http://example.org:8080
export CONTROL_PLANE_HTTPS_PROXY=http://example.org:8080
export
CONTROL_PLANE_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1,10.96
.0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.svc,kubernete
s.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.cluster,.svc.clu
ster.local,169.254.169.254,.elb.amazonaws.com"

export WORKER_HTTP_PROXY=http://example.org:8080
export WORKER_HTTPS_PROXY=http://example.org:8080
export
WORKER_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1,10.96.0.0/12
,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.defau
lt.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.cluster,.svc.cluster.lo
cal,169.254.169.254,.elb.amazonaws.com"
```

HTTP proxy configuration considerations to ensure the core components work correctly

- Replace `example.org,example.com,example.net` with your internal addresses
- `localhost` and `127.0.0.1` addresses should not use the proxy
- `10.96.0.0/12` is the default Kubernetes service subnet
- `192.168.0.0/16` is the default Kubernetes pod subnet
- `kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local` is the internal Kubernetes kube-apiserver service
- The entries `.svc,.svc.cluster,.svc.cluster.local` are the internal Kubernetes services
- Auto-IP addresses `169.254.169.254` for any cloud provider

647 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919002/Custom+Override+Files
648 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29918966/Use+Override+Files+with+Konvoy+Image+Builder

7.9.3.3.1.1 Create a Cluster Using the Configured HTTP Proxy Variables

The following is an example of a `dkp create cluster...` command that uses the values set in the environment variables from the code sample above. Use the appropriate infrastructure provider name in line 1 from the choices listed:

```
dkp create cluster [aws, azure, gcp, preprovisoned, vsphere] \
 --cluster-name ${CLUSTER_NAME} \
 --control-plane-http-proxy="${CONTROL_PLANE_HTTP_PROXY}" \
 --control-plane-https-proxy="${CONTROL_PLANE_HTTPS_PROXY}" \
 --control-plane-no-proxy="${CONTROL_PLANE_NO_PROXY}" \
 --worker-http-proxy="${WORKER_HTTP_PROXY}" \
 --worker-https-proxy="${WORKER_HTTPS_PROXY}" \
 --worker-no-proxy="${WORKER_NO_PROXY}"
```

7.9.3.4 Configure an HTTP/HTTPS Proxy for the DKP Kommander Component

After the cluster is running in the Konvoy component, you need to configure the `NO_PROXY` variable for each provider.

For example, in addition to the values for AWS, you need the following settings:

- The default VPC CIDR range of `10.0.0.0/16`
- `kube-apiserver` internal/external ELB address

- The `NO_PROXY` variable contains the Kubernetes Services CIDR. This example uses the default CIDR, `10.96.0.0/12`. If your cluster's CIDR is different, update the value in the `NO_PROXY` field.

Set the `httpProxy` and `httpsProxy` environment variables to the address of the HTTP and HTTPS proxy servers, respectively. (Frequently, environments use the same values for both.) Set the `noProxy` environment variable to the addresses that should be accessed directly, and not through the proxy.

For the Kommander component of DKP, refer to more HTTP Proxy information in Installing Kommander with an HTTP Proxy[649].

649 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29892224/Installing+Kommander+with+an+HTTP+Proxy

7.9.4 Working with Load Balancers

7.9.4.1 Load Balancer

In a Kubernetes cluster, depending on the flow of traffic direction, there are two kinds of load balancing:

- Internal load balancing for the traffic within a Kubernetes cluster
- External load balancing for the traffic coming from outside the cluster

7.9.4.1.1 External Load Balancer

DKP includes a load balancing solution for the supported cloud infrastructure providers (see page 62) and for pre-provisioned environments. For more information, see Load Balancing for external traffic (see page 854) in DKP.

If you want to use a **non-DKP load balancer** (for example, as an alternative to MetalLB in pre-provisioned environments), DKP supports setting up an **external load balancer**.

When enabled, the external load balancer routes incoming traffic requests to a single point of entry in your cluster. Users and services can then access the **DKP UI** through an established IP or DNS address.

7.9.4.2 Select your Connection Mechanism

A virtual IP is the address that the client uses to connect to the service. A load balancer is the device that distributes the client connections to the backend servers. Before you create a new DKP cluster, choose an external load balancer(LB) or virtual IP.

- **External load balancer**

It is recommended that an external load balancer be the control plane endpoint. To distribute request load among the control plane machines, configure the load balancer to send requests to all the control plane machines. Configure the load balancer to send requests only to control plane machines that are responding to API requests.

- **Built-in virtual IP (option for Pre-provisioned or vSphere)**

If an external load balancer is not available, use the built-in virtual IP (see page 921). The virtual IP is *not* a load balancer; it does not distribute request load among the control plane machines. However, if the machine receiving requests does not respond to them, the virtual IP automatically moves to another machine.

7.9.4.3 External Load Balancer for the Kommander Component of DKP

If you want to use a **non-DKP load balancer** (for example, as an alternative to MetalLB in pre-provisioned environments), DKP supports setting up an **external load balancer**.

When enabled, the external load balancer routes incoming traffic requests to a single point of entry in your cluster. Users and services can then access the **DKP UI** through an established IP or DNS address. Refer to Install Kommander with an External Load Balancer (see page 1242) page of documentation for further details.

7.9.5 Registry and Registry Mirrors

If you need to set up a private registry with a registry mirror, see this page[650] for details on using that flag.

Container registries are collections of container repositories, and can also offer API paths and access rules.

Container repositories are a collection of related container images. The container image has everything a piece of software may need to run, including code, resources, and tools. Container repositories store container images for setup and deployment, and you use the repositories to manage, pull, and push images during cluster operations.

DKP supports operation with several local registry tools[651].

7.9.6 Subnets and Pods

Some subnets are reserved by Kubernetes and can prevent proper cluster deployment if you unknowingly configure DKP so that the Node subnet collides with either the Pod or Service subnet.

> ⚠ Ensure your subnets do not overlap with your host subnet because they cannot be changed after cluster creation. If you need to change the Kubernetes subnets, you must do this at cluster creation.

The default subnets used in DKP are:

```
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 192.168.0.0/16
    services:
      cidrBlocks:
      - 10.96.0.0/12
```

In Konvoy, the default pod subnet is 192.168.0.0/16, and the default service subnet is 10.96.0.0/12. If you wish to change the subnets you can do so with the following steps:

1. Generate the YAML manifests for the cluster using the `--dry-run` and `-o yaml` flags, along with the desired `dkp cluster create` command:

   ```
   dkp create cluster preprovisioned --cluster-name ${CLUSTER_NAME} --control-
   plane-endpoint-host <control plane endpoint host> --control-plane-endpoint-port
   ```

650 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/178618416/Use+a+Registry+Mirror
651 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/210829371/Registry+Mirror+Tools

```
<control plane endpoint port, if different than 6443> --dry-run -o yaml >
cluster.yaml
```

2. To modify the service subnet, add or edit the `spec.clusterNetwork.services.cidrBlocks` field of the `Cluster` object:

```
kind: Cluster
spec:
  clusterNetwork:
    services:
      cidrBlocks:
      - 10.0.0.0/12
```

3. To modify the pod subnet, edit the `Cluster` and calico-cni `ConfigMap` resources:

   Cluster: Add or edit the `spec.clusterNetwork.pods.cidrBlocks` field:

```
kind: Cluster
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 172.16.0.0/16
```

   ConfigMap: Edit the `data."custom-resources.yaml".spec.calicoNetwork.ipPools.cidr` field with your desired pod subnet:

```
apiVersion: v1
data:
  custom-resources.yaml: |
    apiVersion: operator.tigera.io/v1
    kind: Installation
    metadata:
      name: default
    spec:
      # Configures Calico networking.
      calicoNetwork:
        # Note: The ipPools section cannot be modified post-install.
        ipPools:
        - blockSize: 26
          cidr: 172.16.0.0/16
kind: ConfigMap
metadata:
  name: calico-cni-<cluter-name>
```

When you provision the cluster, the configured pod and service subnets will be applied.

7.10 AWS Infrastructure

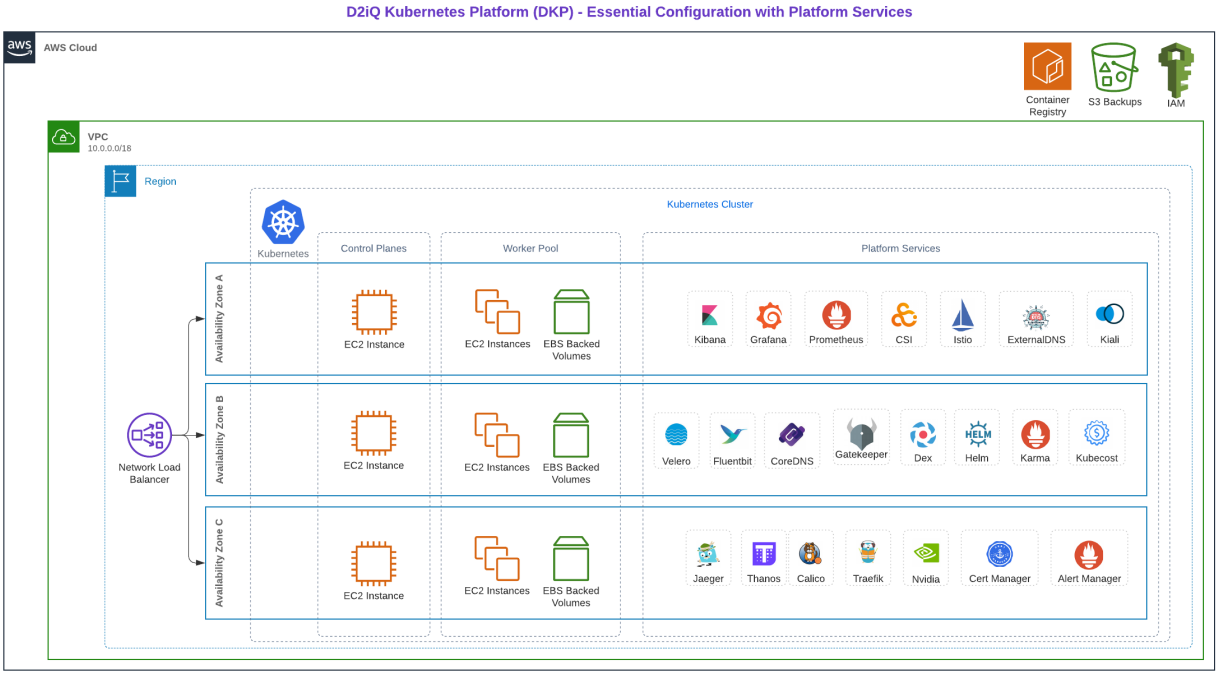
7.10.1 Configuration Types

When installing DKP on AWS infrastructure, you can choose from multiple configuration types. The different types of AWS configuration types supported in DKP are listed below.

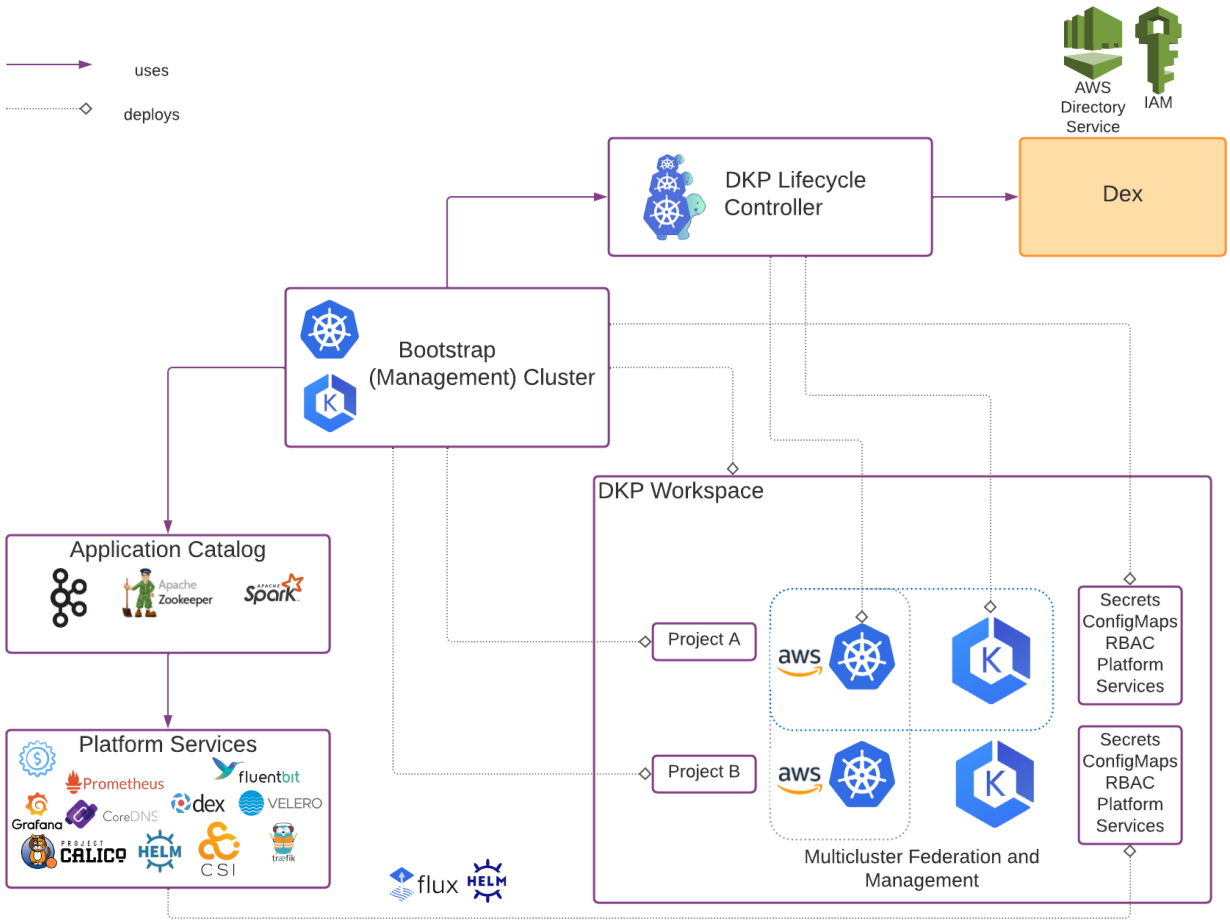
7.10.2 AWS Diagrams

The following diagrams show the two different ways you can implement DKP Essential and DKP Enterprise on AWS.

This diagram shows the granular detail of a single Kubernetes cluster running in AWS Cloud:



This diagram shows a higher-level view of DKP Enterprise, and assumes a multi-cluster environment, where each cluster might look like the single cluster example above:



7.10.3 AWS Pricing Considerations

Deploying AWS services can incur costs to your organization, depending on how and what you deploy. For more information, see the AWS Pricing Calculator[652].

7.10.4 AWS Service Limits

When using DKP on AWS, you need to be aware of the possibility of errors due to AWS service limits. For more information, see the AWS Service Limits[653].

652 https://calculator.aws/#/
653 https://aws.amazon.com/premiumsupport/knowledge-center/manage-service-limits/

7.10.5 AWS Prerequisites

7.10.5.1 Konvoy Prerequisites

Before you begin using Konvoy, you must have:

- An x86_64-based Linux or macOS machine.
- The `dkp` binary for Linux, or macOS.
- A Container engine/runtime installed is required to install DKP:
    - Version Docker®[654] container engine version 18.09.2 or higher installed for Linux or MacOS - On macOS, Docker runs in a virtual machine which needs configured with at least 8 GB of memory.
    - Version 4.0 of Podman[655] or higher for Linux. Host requirements found here: Host Requirements[656]
- kubectl[657] for interacting with the running cluster.
- A valid AWS account with credentials configured[658].
- For air-gapped environment:
    - Linux machine (bastion) that has access to the existing VPC.
    - The `dkp` binary on the bastion.
    - kubectl[659] for interacting with the running cluster on the bastion.
    - An existing local registry (see page 1349).
    - Ability to download artifacts from the internet and then copy those onto your bootstrap machine.
    - An AWS Air-Gapped Machine Image (see page 1292)

On macOS, Docker runs in a virtual machine. Configure this virtual machine with at least 8GB of memory.

7.10.5.2 Control Plane Nodes

You should have at least three control plane nodes. Each control plane node should have at least:

654 https://docs.docker.com/get-docker/
655 https://podman.io/getting-started/installation
656 https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
657 https://kubernetes.io/docs/tasks/tools/#kubectl
658 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html
659 https://kubernetes.io/docs/tasks/tools/#kubectl

- 4 cores
- 16 GiB memory
- Approximately 80 GiB of free space for the volume used for /var/lib/kubelet and /var/lib/containerd.
- Disk usage must be below 85% on the root volume.

DKP on AWS defaults to deploying an `m5.xlarge` instance with an 80GiB root volume for control plane nodes, which meets the above requirements.

7.10.5.3 Worker Nodes

You should have at least four worker nodes. The specific number of worker nodes required for your environment can vary depending on the cluster workload and size of the nodes. Each worker node should have at least:

- 8 cores
- 32 GiB memory
- Around 80 GiB of free space for the volume used for /var/lib/kubelet and /var/lib/containerd.
- Disk usage must be below 85% on the root volume.

DKP on AWS defaults to deploying a `m5.2xlarge` instance with an 80GiB root volume for worker nodes, which meets the above requirements.

If you use these instructions to create a cluster on AWS using the DKP default settings without any edits to configuration files or additional flags, your cluster is deployed on an Ubuntu 20.04 operating system image[660] with 3 control plane nodes, and 4 worker nodes which match the requirements above.

Using these default images work, but due to missing optimizations, the created cluster will have certain limits. We suggest using Konvoy Image Builder to create a custom AMI[661] to take advantage of enhanced cluster operations.

7.10.5.4 AWS Prerequisites

Before you begin using Konvoy with AWS, you must:

- Create Minimal Permissions and Role to Create Clusters (see page 940)
- Create Cluster IAM Policies, Roles, and Artifacts (see page 946)
- Export the AWS region where you want to deploy the cluster:

```
export AWS_REGION=us-west-2
```

660 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29917260/Supported+Infrastructure+Operating+Systems
661 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/143891417

- Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

```
export AWS_PROFILE=<profile>
```

7.10.6 AWS Konvoy Image Builder

You must have at least one image before creating a new cluster. As long as you have an image, this step in your configuration is not required each time since that image can be used to spin up a new cluster. However, if you need different images for different environments or providers, you will need to create a new custom image.

The following section describes how to use Konvoy Image Builder (KIB) with Amazon Web Services (AWS). There are two options:

Konvoy Image Builder (see page 1282) has a more detailed section in the documentation if you need to refer there for compatible versions with DKP and other specific information.

7.10.6.1 Learn how to build a custom AMI for use with DKP

This procedure describes how to use the Konvoy Image Builder (see page 1282) (KIB) to create a Cluster API[662] compliant Amazon Machine Image (AMI). KIB uses variable overrides (see page 1330) to specify base image and container images to use in your new AMI.

The default AWS image is not recommended for use in production. We suggest using Konvoy Image Builder to create a custom AMI to take advantage of enhanced cluster operations. Explore the KIB with AWS (see page 1285) topics for more options.

7.10.6.2 Prerequisites

Before you begin, you must:

- Check the DKP Supported Kubernetes Versions (see page 1543).
- Check the Supported Infrastructure Operating Systems (see page 62)
- Download the KIB (see page 0) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
- Create a working `Docker` setup.

662 https://cluster-api.sigs.k8s.io/

- Ensure you have met the minimal set of permissions from the AWS Image Builder Book[663].
- A Minimal IAM Permissions for KIB (see page 1285) to create an Image for an AWS account using Konvoy Image Builder.

7.10.6.3 Extract KIB Bundle

Extract the bundle and `cd` into the extracted `konvoy-image-bundle-$VERSION` folder. The bundled version of `konvoy-image` contains an embedded `docker` image that contains all the requirements for building.

> The `konvoy-image` binary and all supporting folders are also extracted. When run, `konvoy-image` bind mounts the current working directory (`${PWD}`) into the container to be used.

- Set environment variables for AWS access[664]. The following variables must be set using your credentials including required IAM (see page 1285):

```
export AWS_ACCESS_KEY_ID
export AWS_SECRET_ACCESS_KEY
export AWS_DEFAULT_REGION
```

- If you have an override file (see page 1330) to configure specific attributes of your AMI file, add it.

7.10.6.4 1. Create a Custom AMI

Depending on which version of DKP (see page 1282) you are running, steps and flags will be different. To deploy in a region where CAPI images are not provided, you need to use KIB to create your own image for the region. For a list of supported AWS regions, refer to the Published AMI[665]information from AWS.

7.10.6.4.1 Execute the following to begin image creation:

Run the `konvoy-image` command to build and validate the image.

```
konvoy-image build aws images/ami/centos-79.yaml
```

By default it builds in the `us-west-2` region. to specify another region set the `--region` flag:

663 https://image-builder.sigs.k8s.io/capi/providers/aws.html#required-permissions-to-build-the-aws-amis
664 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-envvars.html
665 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html

```
konvoy-image build aws --region us-east-1 images/ami/centos-79.yaml
```

Ensure you have named the correct YAML file for your OS in the `konvoy-image build` command.

Once KIB provisions the image successfully, the `ami` id is printed and written to the `manifest.json` file. This file has an `artifact_id` field whose value provides the name of the AMI ID as shown in the example below:

```
{
      "name": "rhel-7.9-fips",
      "builder_type": "amazon-ebs",
      "build_time": 1659486130,
      "files": null,
      "artifact_id": "us-west-2:ami-0f2ef742482e1b829",
      "packer_run_uuid": "0ca500d9-a5f0-815c-6f12-aceb4d46645b",
      "custom_data": {
        "containerd_version": "",
        "distribution": "RHEL",
        "distribution_version": "7.9",
        "kubernetes_cni_version": "",
        "kubernetes_version": "1.24.5+fips.0"
      }
    }
```

7.10.6.5 2. Air-gapped AMI

7.10.6.5.1 Create an Image using Konvoy Image Builder (KIB) for use in an air-gapped cluster

7.10.6.6 Prerequisites

- Before you begin, you must:
  - Check the DKP Supported Kubernetes Versions (see page 1543).
  - Check the Supported Infrastructure Operating Systems (see page 62)
  - Download the KIB (see page 0) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
  - Create a working Docker or other registry setup.

- Ensure you have met the minimal set of permissions from the AWS Image Builder Book[666].
- A Minimal IAM Permissions for KIB (see page 1285) to create an Image for an AWS account using Konvoy Image Builder.

The default AWS image is not recommended for use in production. We suggest using Konvoy Image Builder to create an AWS Air-gapped AMI to take advantage of enhanced cluster operations. Explore the KIB with AWS (see page 1285) topics for more options.

Using KIB (see page 1282), you can build an AMI without requiring access to the internet by providing an additional `--override` flag.

1. Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz` , extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz && cd dkp-v2.5.2/kib
   ```

2. Follow the instructions below to build an AMI.

Depending on which version of DKP (see page 1282) you are running, steps and flags will be different. To deploy in a region where CAPI images are not provided, you need to use KIB to create your own image for the region. For a list of supported AWS regions, refer to the Published AMI[667]information from AWS.

7.10.6.6.1 Execute the following to begin image creation:

Run the `konvoy-image` command to build and validate the image.

```
konvoy-image build aws images/ami/centos-79.yaml --overrides overrides/offline.yaml
```

By default it builds in the `us-west-2` region. to specify another region set the `--region` flag:

```
konvoy-image build aws --region us-east-1 images/ami/centos-79.yaml --overrides
overrides/offline.yaml
```

When the command is complete the `ami` id is printed and written to `./manifest.json` .

7.10.6.7 Next Step:

Minimal Permissions and Role to Create Clusters (see page 940)

666 https://image-builder.sigs.k8s.io/capi/providers/aws.html#required-permissions-to-build-the-aws-amis
667 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html

7.10.7 Minimal Permissions and Role to Create Clusters

Configure IAM Prerequisites before starting a cluster

This section guides you in creating and using a minimally-scoped policy to create DKP clusters on an AWS account.

7.10.7.1 Prerequisites

Before applying the IAM Policies, verify the following:

- You have a valid AWS account with credentials configured[668] that can manage CloudFormation Stacks, IAM Policies, IAM Roles, and IAM Instance Profiles.
- The AWS CLI utility[669] is installed.

7.10.7.2 Minimal Permissions

The following is an AWSCloudformation stack that creates:

- A policy named `dkp-bootstrapper-policy` that enumerates the minimal permissions for a user that can create dkp aws clusters.
- A role named `dkp-bootstrapper-role` that uses the `dkp-bootstrapper-policy` with a trust policy to allow IAM users and ec2 instances from `MYAWSACCOUNTID` to use the role via STS.
- An instance profile `DKPBootstrapInstanceProfile` that wraps the `dkp-bootstrapper-role` to be used by ec2 instances.

7.10.7.3 Create Resources in Cloudformation Stack

To create the resources in the cloudformation stack:

1. Copy the following contents into a file:

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  AWSIAMInstanceProfileDKPBootstrapper:
    Properties:
      InstanceProfileName: DKPBootstrapInstanceProfile
      Roles:
      - Ref: DKPBootstrapRole
    Type: AWS::IAM::InstanceProfile
  AWSIAMManagedPolicyDKPBootstrapper:
```

668 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html
669 https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html

```
Properties:
  Description: Minimal policy to create dkp clusters in AWS
  ManagedPolicyName: dkp-bootstrapper-policy
  PolicyDocument:
    Statement:
    - Action:
      - ec2:AllocateAddress
      - ec2:AssociateRouteTable
      - ec2:AttachInternetGateway
      - ec2:AuthorizeSecurityGroupIngress
      - ec2:CreateInternetGateway
      - ec2:CreateNatGateway
      - ec2:CreateRoute
      - ec2:CreateRouteTable
      - ec2:CreateSecurityGroup
      - ec2:CreateSubnet
      - ec2:CreateTags
      - ec2:CreateVpc
      - ec2:ModifyVpcAttribute
      - ec2:DeleteInternetGateway
      - ec2:DeleteNatGateway
      - ec2:DeleteRouteTable
      - ec2:DeleteSecurityGroup
      - ec2:DeleteSubnet
      - ec2:DeleteTags
      - ec2:DeleteVpc
      - ec2:DescribeAccountAttributes
      - ec2:DescribeAddresses
      - ec2:DescribeAvailabilityZones
      - ec2:DescribeInstances
      - ec2:DescribeInternetGateways
      - ec2:DescribeImages
      - ec2:DescribeNatGateways
      - ec2:DescribeNetworkInterfaces
      - ec2:DescribeNetworkInterfaceAttribute
      - ec2:DescribeRouteTables
      - ec2:DescribeSecurityGroups
      - ec2:DescribeSubnets
      - ec2:DescribeVpcs
      - ec2:DescribeVpcAttribute
      - ec2:DescribeVolumes
      - ec2:DetachInternetGateway
      - ec2:DisassociateRouteTable
      - ec2:DisassociateAddress
      - ec2:ModifyInstanceAttribute
      - ec2:ModifyNetworkInterfaceAttribute
      - ec2:ModifySubnetAttribute
      - ec2:ReleaseAddress
      - ec2:RevokeSecurityGroupIngress
      - ec2:RunInstances
      - ec2:TerminateInstances
      - tag:GetResources
```

```
          - elasticloadbalancing:AddTags
          - elasticloadbalancing:CreateLoadBalancer
          - elasticloadbalancing:ConfigureHealthCheck
          - elasticloadbalancing:DeleteLoadBalancer
          - elasticloadbalancing:DescribeLoadBalancers
          - elasticloadbalancing:DescribeLoadBalancerAttributes
          - elasticloadbalancing:ApplySecurityGroupsToLoadBalancer
          - elasticloadbalancing:DescribeTags
          - elasticloadbalancing:ModifyLoadBalancerAttributes
          - elasticloadbalancing:RegisterInstancesWithLoadBalancer
          - elasticloadbalancing:DeregisterInstancesFromLoadBalancer
          - elasticloadbalancing:RemoveTags
          - autoscaling:DescribeAutoScalingGroups
          - autoscaling:DescribeInstanceRefreshes
          - ec2:CreateLaunchTemplate
          - ec2:CreateLaunchTemplateVersion
          - ec2:DescribeLaunchTemplates
          - ec2:DescribeLaunchTemplateVersions
          - ec2:DeleteLaunchTemplate
          - ec2:DeleteLaunchTemplateVersions
          - ec2:DescribeKeyPairs
          Effect: Allow
          Resource:
          - '*'
        - Action:
          - autoscaling:CreateAutoScalingGroup
          - autoscaling:UpdateAutoScalingGroup
          - autoscaling:CreateOrUpdateTags
          - autoscaling:StartInstanceRefresh
          - autoscaling:DeleteAutoScalingGroup
          - autoscaling:DeleteTags
          Effect: Allow
          Resource:
          - arn:*:autoscaling:*:*:autoScalingGroup:*:autoScalingGroupName/*
        - Action:
          - iam:CreateServiceLinkedRole
          Condition:
            StringLike:
              iam:AWSServiceName: autoscaling.amazonaws.com
          Effect: Allow
          Resource:
          - arn:*:iam::*:role/aws-service-role/autoscaling.amazonaws.com/
AWSServiceRoleForAutoScaling
        - Action:
          - iam:CreateServiceLinkedRole
          Condition:
            StringLike:
              iam:AWSServiceName: elasticloadbalancing.amazonaws.com
          Effect: Allow
          Resource:
          - arn:*:iam::*:role/aws-service-role/
elasticloadbalancing.amazonaws.com/AWSServiceRoleForElasticLoadBalancing
```

```
        - Action:
          - iam:CreateServiceLinkedRole
          Condition:
            StringLike:
              iam:AWSServiceName: spot.amazonaws.com
          Effect: Allow
          Resource:
          - arn:*:iam::*:role/aws-service-role/spot.amazonaws.com/
AWSServiceRoleForEC2Spot
        - Action:
          - iam:PassRole
          Effect: Allow
          Resource:
          - arn:*:iam::*:role/*.cluster-api-provider-aws.sigs.k8s.io
        - Action:
          - secretsmanager:CreateSecret
          - secretsmanager:DeleteSecret
          - secretsmanager:TagResource
          Effect: Allow
          Resource:
          - arn:*:secretsmanager:*:*:secret:aws.cluster.x-k8s.io/*
        Version: 2012-10-17
      Roles:
      - Ref: DKPBootstrapRole
    Type: AWS::IAM::ManagedPolicy
  DKPBootstrapRole:
    Properties:
      AssumeRolePolicyDocument:
        Statement:
        - Action:
          - sts:AssumeRole
          Effect: Allow
          Principal:
            Service:
            - ec2.amazonaws.com
        - Action:
            - sts:AssumeRole
          Effect: Allow
          Principal:
                AWS: arn:aws:iam::MYAWSACCOUNT:root
        Version: 2012-10-17
      RoleName: dkp-bootstrapper-role
    Type: AWS::IAM::Role
```

2. Replace the following with the correct values:
   a. `MYFILENAME.yaml` - give your file a meaningful name.
   b. `MYSTACKNAME` - give your cloudformation stack a meaningful name.
   c. `MYAWSACCOUNT` - replace with an AWS Account ID number such as: `111122223333`
3. Run the following command to create the stack :

```
aws cloudformation create-stack --template-body=file://MYFILENAME.yaml --stack-
name=MYSTACKNAME --capabilities CAPABILITY_NAMED_IAM
```

7.10.7.4 Leverage the Role

Use temporary User Access Keys via STS.

The created `dkp-bootstrapper-role` can be assumed by IAM users for temporary credentials via STS by running the command below:

```
aws sts assume-role --role-arn arn:aws:iam::MYAWSACCOUNT:role/dkp-bootstrapper-role
--role-session-name EXAMPLE
```

Which returns something similar to this:

```
{
    "Credentials": {
        "AccessKeyId": "ASIA6RTF53ZH5B52EVM5",
        "SecretAccessKey": "BSssyvSsdfJY74jubsadfdsafdsaH7x1L+8Vk/",
        "SessionToken": "IQoJb3JpZ2z5cyChb9PtJvP0S6KAi",
        "Expiration": "2022-07-14T20:19:13+00:00"
    },
    "AssumedRoleUser": {
        "AssumedRoleId": "ASIA6RTF53ZH5B52EVM5:test",
        "Arn": "arn:aws:sts::MYAWSACCOUNTID:assumed-role/dkp-bootstrapper-role/test"
    }
}
```

And then `export` the following environment variables with the results:

```
export AWS_ACCESS_KEY_ID=(.Credentials.AccessKeyId)

export AWS_SECRET_ACCESS_KEY=(.Credentials.SecretAccessKey)

export AWS_SESSION_TOKEN=(.Credentials.SessionToken)
```

These credentials are short lived and would need to be updated in the bootstrap cluster

7.10.7.5 Use EC2 Instance Profiles

The created `dkp-bootstrapper-role` can be assumed by an ec2 instance a user would run `dkp create cluster` commands from. To do this, specify the IAM Instance Profile `DKPBootstrapInstanceProfile` on creation.

7.10.7.6 Use Access Keys

AWS administrators can attach the `dkp-bootstrapper-policy` to an existing IAM user[670] and authenticate with Access Keys[671] on the work station they would run `dkp create cluster` commands from by exporting the following environment variables with the appropriate values for the IAM user.

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_DEFAULT_REGION=us-west-2
```

In regards to Access Keys usage, a system administrator should always consider AWS's Best practices[672].

> EKS Minimal Permissions that govern cluster creation are found here: Minimal User Permission for EKS Cluster Creation (see page 1007). The CloudFormation stack on that page adds a policy named `eks-bootstrapper` to manage EKS cluster to the `dkp-bootstrapper-role` created by the CloudFormation stack on this page.

If your organization uses encrypted AMI's (https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIEncryption.html), then you will need to add additional permissions to the control plane policy to allow access to the Amazon Key Management Services. See the following documentation for information on the necessary policies you may need: AWS Key Policies[673].

7.10.7.7 Next Step:

Cluster IAM Policies, Roles, and Artifacts (see page 946)

670 https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_manage-attach-detach.html
671 https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html
672 https://docs.aws.amazon.com/general/latest/gr/aws-access-keys-best-practices.html
673 https://docs.aws.amazon.com/kms/latest/developerguide/key-policy-default.html#key-policy-service-integration

7.10.8 Cluster IAM Policies, Roles, and Artifacts

This guides a DKP user in creating IAM Policies and Instance Profiles used by the cluster's control plane and worker nodes using the provided AWS CloudFormation Stack.

7.10.8.1 Prerequisites

Before applying the IAM Policies, verify the following:

- You have a valid AWS account with credentials configured[674] that can manage CloudFormation Stacks, IAM Policies, IAM Roles, and IAM Instance Profiles.
- You have the AWS CLI utility installed[675].

Below is information regarding the set up for policies and roles. After reading the information for each of these areas, you will find the CloudFormation Stack that creates these policies and roles in the IAM Artifacts drop-down section below:

Policies

1. `AWSIAMManagedPolicyCloudProviderControlPlane` enumerates the Actions required by the workload cluster control plane machines. It is attached to the `AWSIAMRoleControlPlane` Role.
2. `AWSIAMManagedPolicyCloudProviderNodes` enumerates the Actions required by the workload cluster worker machines. It is attached to the `AWSIAMRoleNodes` Role.
3. `AWSIAMManagedPolicyControllers` enumerates the Actions required by the workload cluster worker machines. It is attached to the `AWSIAMRoleControlPlane` Role.

Roles

1. `AWSIAMRoleControlPlane` is the Role associated with the `AWSIAMInstanceProfileControlPlane` Instance Profile.
2. `AWSIAMRoleNodes` is the Role associated with the `AWSIAMInstanceProfileNodes` Instance Profile.

For more information on learning how to grant cluster access to IAM users and roles, see the official AWS Documentation[676].

IAM Artifacts

7.10.8.2 Instance Profiles

1. `AWSIAMInstanceProfileControlPlane`, assigned to workload cluster control plane machines.

674 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html
675 https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html
676 https://docs.aws.amazon.com/eks/latest/userguide/add-user-role.html

If the name is changed from the default, used below, it must be passed to `dkp create cluster` with the `--control-plane-iam-instance-profile` flag.

1. `AWSIAMInstanceProfileNodes`, assigned to workload cluster worker machines.

If the name is changed from the default, used below, it must be passed to `dkp create cluster` with the `--worker-iam-instance-profile` flag.

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  AWSIAMInstanceProfileControlPlane:
    Properties:
      InstanceProfileName: control-plane.cluster-api-provider-aws.sigs.k8s.io
      Roles:
      - Ref: AWSIAMRoleControlPlane
    Type: AWS::IAM::InstanceProfile
  AWSIAMInstanceProfileNodes:
    Properties:
      InstanceProfileName: nodes.cluster-api-provider-aws.sigs.k8s.io
      Roles:
      - Ref: AWSIAMRoleNodes
    Type: AWS::IAM::InstanceProfile
  AWSIAMManagedPolicyCloudProviderControlPlane:
    Properties:
      Description: For the Kubernetes Cloud Provider AWS Control Plane
      ManagedPolicyName: control-plane.cluster-api-provider-aws.sigs.k8s.io
      PolicyDocument:
        Statement:
        - Action:
          - autoscaling:DescribeAutoScalingGroups
          - autoscaling:DescribeLaunchConfigurations
          - autoscaling:DescribeTags
          - ec2:DescribeInstances
          - ec2:DescribeImages
          - ec2:DescribeRegions
          - ec2:DescribeRouteTables
          - ec2:DescribeSecurityGroups
          - ec2:DescribeSubnets
          - ec2:DescribeVolumes
          - ec2:CreateSecurityGroup
          - ec2:CreateTags
          - ec2:CreateVolume
```

```
        - ec2:ModifyInstanceAttribute
        - ec2:ModifyVolume
        - ec2:AttachVolume
        - ec2:AuthorizeSecurityGroupIngress
        - ec2:CreateRoute
        - ec2:DeleteRoute
        - ec2:DeleteSecurityGroup
        - ec2:DeleteVolume
        - ec2:DetachVolume
        - ec2:RevokeSecurityGroupIngress
        - ec2:DescribeVpcs
        - elasticloadbalancing:AddTags
        - elasticloadbalancing:AttachLoadBalancerToSubnets
        - elasticloadbalancing:ApplySecurityGroupsToLoadBalancer
        - elasticloadbalancing:CreateLoadBalancer
        - elasticloadbalancing:CreateLoadBalancerPolicy
        - elasticloadbalancing:CreateLoadBalancerListeners
        - elasticloadbalancing:ConfigureHealthCheck
        - elasticloadbalancing:DeleteLoadBalancer
        - elasticloadbalancing:DeleteLoadBalancerListeners
        - elasticloadbalancing:DescribeLoadBalancers
        - elasticloadbalancing:DescribeLoadBalancerAttributes
        - elasticloadbalancing:DetachLoadBalancerFromSubnets
        - elasticloadbalancing:DeregisterInstancesFromLoadBalancer
        - elasticloadbalancing:ModifyLoadBalancerAttributes
        - elasticloadbalancing:RegisterInstancesWithLoadBalancer
        - elasticloadbalancing:SetLoadBalancerPoliciesForBackendServer
        - elasticloadbalancing:AddTags
        - elasticloadbalancing:CreateListener
        - elasticloadbalancing:CreateTargetGroup
        - elasticloadbalancing:DeleteListener
        - elasticloadbalancing:DeleteTargetGroup
        - elasticloadbalancing:DescribeListeners
        - elasticloadbalancing:DescribeLoadBalancerPolicies
        - elasticloadbalancing:DescribeTargetGroups
        - elasticloadbalancing:DescribeTargetHealth
        - elasticloadbalancing:ModifyListener
        - elasticloadbalancing:ModifyTargetGroup
        - elasticloadbalancing:RegisterTargets
        - elasticloadbalancing:SetLoadBalancerPoliciesOfListener
        - iam:CreateServiceLinkedRole
        - kms:DescribeKey
        - kms:CreateGrant
        Effect: Allow
        Resource:
        - '*'
      Version: 2012-10-17
    Roles:
    - Ref: AWSIAMRoleControlPlane
  Type: AWS::IAM::ManagedPolicy
AWSIAMManagedPolicyCloudProviderNodes:
  Properties:
```

```
    Description: For the Kubernetes Cloud Provider AWS nodes
    ManagedPolicyName: nodes.cluster-api-provider-aws.sigs.k8s.io
    PolicyDocument:
      Statement:
      - Action:
        - ec2:DescribeInstances
        - ec2:DescribeRegions
        - ecr:GetAuthorizationToken
        - ecr:BatchCheckLayerAvailability
        - ecr:GetDownloadUrlForLayer
        - ecr:GetRepositoryPolicy
        - ecr:DescribeRepositories
        - ecr:ListImages
        - ecr:BatchGetImage
        Effect: Allow
        Resource:
        - '*'
      - Action:
        - secretsmanager:DeleteSecret
        - secretsmanager:GetSecretValue
        Effect: Allow
        Resource:
        - arn:*:secretsmanager:*:*:secret:aws.cluster.x-k8s.io/*
      - Action:
        - ssm:UpdateInstanceInformation
        - ssmmessages:CreateControlChannel
        - ssmmessages:CreateDataChannel
        - ssmmessages:OpenControlChannel
        - ssmmessages:OpenDataChannel
        - s3:GetEncryptionConfiguration
        Effect: Allow
        Resource:
        - '*'
      Version: 2012-10-17
    Roles:
    - Ref: AWSIAMRoleControlPlane
    - Ref: AWSIAMRoleNodes
  Type: AWS::IAM::ManagedPolicy
AWSIAMManagedPolicyControllers:
  Properties:
    Description: For the Kubernetes Cluster API Provider AWS Controllers
    ManagedPolicyName: controllers.cluster-api-provider-aws.sigs.k8s.io
    PolicyDocument:
      Statement:
      - Action:
        - ec2:AllocateAddress
        - ec2:AssociateRouteTable
        - ec2:AttachInternetGateway
        - ec2:AuthorizeSecurityGroupIngress
        - ec2:CreateInternetGateway
        - ec2:CreateNatGateway
        - ec2:CreateRoute
```

```
- ec2:CreateRouteTable
- ec2:CreateSecurityGroup
- ec2:CreateSubnet
- ec2:CreateTags
- ec2:CreateVpc
- ec2:ModifyVpcAttribute
- ec2:DeleteInternetGateway
- ec2:DeleteNatGateway
- ec2:DeleteRouteTable
- ec2:DeleteSecurityGroup
- ec2:DeleteSubnet
- ec2:DeleteTags
- ec2:DeleteVpc
- ec2:DescribeAccountAttributes
- ec2:DescribeAddresses
- ec2:DescribeAvailabilityZones
- ec2:DescribeInstances
- ec2:DescribeInternetGateways
- ec2:DescribeImages
- ec2:DescribeNatGateways
- ec2:DescribeNetworkInterfaces
- ec2:DescribeNetworkInterfaceAttribute
- ec2:DescribeRouteTables
- ec2:DescribeSecurityGroups
- ec2:DescribeSubnets
- ec2:DescribeVpcs
- ec2:DescribeVpcAttribute
- ec2:DescribeVolumes
- ec2:DetachInternetGateway
- ec2:DisassociateRouteTable
- ec2:DisassociateAddress
- ec2:ModifyInstanceAttribute
- ec2:ModifyNetworkInterfaceAttribute
- ec2:ModifySubnetAttribute
- ec2:ReleaseAddress
- ec2:RevokeSecurityGroupIngress
- ec2:RunInstances
- ec2:TerminateInstances
- tag:GetResources
- elasticloadbalancing:AddTags
- elasticloadbalancing:CreateLoadBalancer
- elasticloadbalancing:ConfigureHealthCheck
- elasticloadbalancing:DeleteLoadBalancer
- elasticloadbalancing:DescribeLoadBalancers
- elasticloadbalancing:DescribeLoadBalancerAttributes
- elasticloadbalancing:ApplySecurityGroupsToLoadBalancer
- elasticloadbalancing:DescribeTags
- elasticloadbalancing:ModifyLoadBalancerAttributes
- elasticloadbalancing:RegisterInstancesWithLoadBalancer
- elasticloadbalancing:DeregisterInstancesFromLoadBalancer
- elasticloadbalancing:RemoveTags
- autoscaling:DescribeAutoScalingGroups
```

```
          - autoscaling:DescribeInstanceRefreshes
          - ec2:CreateLaunchTemplate
          - ec2:CreateLaunchTemplateVersion
          - ec2:DescribeLaunchTemplates
          - ec2:DescribeLaunchTemplateVersions
          - ec2:DeleteLaunchTemplate
          - ec2:DeleteLaunchTemplateVersions
          Effect: Allow
          Resource:
          - '*'
        - Action:
          - autoscaling:CreateAutoScalingGroup
          - autoscaling:UpdateAutoScalingGroup
          - autoscaling:CreateOrUpdateTags
          - autoscaling:StartInstanceRefresh
          - autoscaling:DeleteAutoScalingGroup
          - autoscaling:DeleteTags
          Effect: Allow
          Resource:
          - arn:*:autoscaling:*:*:autoScalingGroup:*:autoScalingGroupName/*
        - Action:
          - iam:CreateServiceLinkedRole
          Condition:
            StringLike:
              iam:AWSServiceName: autoscaling.amazonaws.com
          Effect: Allow
          Resource:
          - arn:*:iam::*:role/aws-service-role/autoscaling.amazonaws.com/
AWSServiceRoleForAutoScaling
        - Action:
          - iam:CreateServiceLinkedRole
          Condition:
            StringLike:
              iam:AWSServiceName: elasticloadbalancing.amazonaws.com
          Effect: Allow
          Resource:
          - arn:*:iam::*:role/aws-service-role/elasticloadbalancing.amazonaws.com/
AWSServiceRoleForElasticLoadBalancing
        - Action:
          - iam:CreateServiceLinkedRole
          Condition:
            StringLike:
              iam:AWSServiceName: spot.amazonaws.com
          Effect: Allow
          Resource:
          - arn:*:iam::*:role/aws-service-role/spot.amazonaws.com/
AWSServiceRoleForEC2Spot
        - Action:
          - iam:PassRole
          Effect: Allow
          Resource:
          - arn:*:iam::*:role/*.cluster-api-provider-aws.sigs.k8s.io
```

```
        - Action:
          - secretsmanager:CreateSecret
          - secretsmanager:DeleteSecret
          - secretsmanager:TagResource
          Effect: Allow
          Resource:
          - arn:*:secretsmanager:*:*:secret:aws.cluster.x-k8s.io/*
        Version: 2012-10-17
      Roles:
      - Ref: AWSIAMRoleControlPlane
    Type: AWS::IAM::ManagedPolicy
  AWSIAMRoleControlPlane:
    Properties:
      AssumeRolePolicyDocument:
        Statement:
        - Action:
          - sts:AssumeRole
          Effect: Allow
          Principal:
            Service:
            - ec2.amazonaws.com
        Version: 2012-10-17
      RoleName: control-plane.cluster-api-provider-aws.sigs.k8s.io
    Type: AWS::IAM::Role
  AWSIAMRoleNodes:
    Properties:
      AssumeRolePolicyDocument:
        Statement:
        - Action:
          - sts:AssumeRole
          Effect: Allow
          Principal:
            Service:
            - ec2.amazonaws.com
        Version: 2012-10-17
      RoleName: nodes.cluster-api-provider-aws.sigs.k8s.io
    Type: AWS::IAM::Role
```

To create the resources in the cloudformation stack copy the contents above into a file replacing `MYFILENAME.yaml` and `MYSTACKNAME` with the intended values. Then execute the following command:

```
 aws cloudformation create-stack --template-body=file://MYFILENAME.yaml --stack-
name=MYSTACKNAME --capabilities  CAPABILITY_NAMED_IAM
```

If your organization uses encrypted AMIs[677], then you will need to add additional permissions to the control plane policy `control-plane.cluster-api-provider-aws.sigs.k8s.io` to allow access to the Amazon Key Management Services. The code snippet shows how to add a particular key ARN that is used to encrypt and decrypt AMIs.

```
---
Action:
- kms:CreateGrant
- kms:DescribeKey
- kms:Encrypt
- kms:Decrypt
- kms:ReEncrypt*
- kms:GenerateDataKey*
Resource:
- arn:aws:kms:us-west-2:111122223333:key/key-arn
Effect: Allow
```

EKS IAM Policies and Instance Profiles that govern who has access to the cluster are found here: EKS Cluster IAM Permissions and Roles (see page 1010). If attaching an EKS cluster, that CloudStack Formation will also need to be run to create the ARN of the bootstrapper role.

7.10.8.3 Next Steps:

AWS Cluster Creation Choices (see page 953)

7.10.9 AWS Cluster Creation Choices

Create a new AWS Kubernetes cluster in your AWS infrastructure with the following choices:

- Custom AMI in Cluster Creation (see page 954)
- AWS Non-air-gapped Create a Custom Cluster (see page 955)
- AWS Air-gapped Create a Custom Cluster (see page 969)
- Explore New AWS Cluster (see page 976)
- Make the AWS Cluster Self-Managed (see page 976)
- Creating DKP Clusters on AWS in the UI (see page 980)

677 https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIEncryption.html

7.10.9.1 Custom AMI in Cluster Creation

7.10.9.1.1 Launch a DKP Cluster with a Custom AMI

To use the built `ami` with DKP, specify it with the `--ami` flag when calling cluster create.

```
dkp create cluster aws --cluster-name=$(whoami)-aws-cluster --ami ami-0123456789
```

7.10.9.1.2 Launch a DKP Cluster with Custom AMI Lookup

By default `konvoy-image` will name the AMI in such a way that `dkp` can discover the latest AMI for a base OS and Kubernetes version. To create a cluster that will use the latest AMI, specify the `--ami-format`, `--ami-base-os` and `--ami-owner` flags:

```
dkp create cluster aws --cluster-name=$(whoami)-aws-cluster --ami-format "konvoy-ami-
{{.BaseOS}}-?{{.K8sVersion}}-*" --ami-base-os centos-7 --ami-owner 123456789012
```

7.10.9.1.3 Using Custom Source AMIs

When using KIB for building machine images to Amazon, the default source AMIs that we provide are modeled by looking up an AMI based on the owner. Then we apply a filter for that operating system and version.

You can view an example of that with the provided `centos-79.yaml` snippet below:

```
download_images: true

packer:
  ami_filter_name: "CentOS Linux 7"
  ami_filter_owners: "125523088429"
  distribution: "CentOS"
  distribution_version: "7.9"
  source_ami: ""
  ssh_username: "centos"
  root_device_name: "/dev/sda1"
...
```

At times, a particular upstream AMI may not be available in your region or something could be renamed. Other times you want to provide a custom AMI. If this is the case, you will want to edit or create your own YAML file that looks up based on the `source_ami` field. For example, you can select images that are otherwise deprecated.

Once you select the source AMI that you want, you can declare that when running your build command:

```
konvoy-image build aws path/to/ami/centos-79.yaml --source-ami ami-0123456789
```

Alternatively, if you want to add it to your YAML file, or make your own file, you can do that as well. You add that AMI ID into the `source_ami` in the YAML file:

```
download_images: true

packer:
  ami_filter_name: ""
  ami_filter_owners: ""
  distribution: "CentOS"
  distribution_version: "7.9"
  source_ami: "ami-123456789"
  ssh_username: "centos"
  root_device_name: "/dev/sda1"
...
```

When you're done selecting your `source_ami`, you can build your KIB image as you would normally:

```
konvoy-image build aws path/to/ami/centos-79.yaml
```

7.10.9.2 AWS Non-air-gapped Create a Custom Cluster

This section provides instructions to install DKP in an AWS non-air-gapped environment with custom settings. First you create an Bootstrap Cluster (see page 956) and then move the CAPI resources to the workload cluster and delete the bootstrap cluster.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

7.10.9.2.1 AWS Prerequisites

Before you begin using Konvoy with AWS, you must:

1. Follow the steps to create a Minimal Permissions and Role to Create Clusters (see page 940)..
2. Create Cluster IAM Policies and Roles (see page 946).
3. Export the AWS region where you want to deploy the cluster:

```
export AWS_REGION=us-west-2
```

4. Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

```
export AWS_PROFILE=<profile>
```

7.10.9.2.2 Next Step:

Bootstrap AWS (see page 956)

7.10.9.2.3 AWS Bootstrap Cluster

To create Kubernetes clusters, Konvoy uses Cluster API[678] (CAPI) controllers. These controllers run on a Kubernetes cluster. To get started, you need a *bootstrap* cluster. By default, Konvoy creates a bootstrap cluster for you in a Docker container using the Kubernetes-in-Docker (KIND[679]) tool.

7.10.9.2.3.1 Prerequisites

Before you begin, you must:

- Complete the steps in Prerequisites (see page 119).
- Ensure the `dkp` binary can be found in your $PATH.

7.10.9.2.3.2 Bootstrap Cluster Lifecycle Services

1. If an HTTP proxy is required for the bootstrap cluster, set the local `http_proxy`, `https_proxy`, and `no_proxy` environment variables. They are copied into the bootstrap cluster.
2. Create a bootstrap cluster:

   ```
   dkp create bootstrap --kubeconfig $HOME/.kube/config
   ```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

   The output resembles:

678 https://cluster-api.sigs.k8s.io/
679 https://github.com/kubernetes-sigs/kind

```
✓ Creating a bootstrap cluster
✓ Initializing new CAPI components
```

Konvoy creates a bootstrap cluster using KIND[680] as a library. Konvoy then deploys the following Cluster API[681] providers on the cluster:

- Core Provider[682]
- AWS Infrastructure Provider[683]
- Kubeadm Bootstrap Provider[684]
- Kubeadm ControlPlane Provider[685]

Konvoy waits until the controller-manager and webhook deployments of these providers are ready. List these deployments using this command:

```
kubectl get --all-namespaces deployments -l=clusterctl.cluster.x-k8s.io
```

```
NAMESPACE                           NAME
READY   UP-TO-DATE   AVAILABLE   AGE
capa-system                         capa-controller-manager
1/1     1            1           2m8s
capi-kubeadm-bootstrap-system       capi-kubeadm-bootstrap-controller-manager
1/1     1            1           2m10s
capi-kubeadm-control-plane-system   capi-kubeadm-control-plane-controller-
manager   1/1     1            1           2m10s
capi-system                         capi-controller-manager
1/1     1            1           2m11s
cappp-system                        cappp-controller-manager
1/1     1            1           2m6s
capv-system                         capv-controller-manager
1/1     1            1           2m5s
capz-system                         capz-controller-manager
1/1     1            1           2m7s
cert-manager                        cert-manager
1/1     1            1           2m21s
cert-manager                        cert-manager-cainjector
1/1     1            1           2m21s
cert-manager                        cert-manager-webhook
1/1     1            1           2m21s
```

680 https://github.com/kubernetes-sigs/kind
681 https://cluster-api.sigs.k8s.io/
682 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/
683 https://github.com/kubernetes-sigs/cluster-api-provider-aws
684 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/bootstrap/kubeadm
685 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/controlplane/kubeadm

7.10.9.2.3.3 Next Step:

Create a New Customized AWS Cluster (see page 958)

7.10.9.2.4 Create a New Customized AWS Cluster

7.10.9.2.4.1 Prerequisites

Before you begin, make sure you have created a Bootstrap (see page 956) cluster.

7.10.9.2.4.2 Name Your Cluster

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable:

   ```
   export CLUSTER_NAME=aws-example
   ```

> The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[686] for more naming information.

7.10.9.2.4.3 Tips and Tricks

Below are a few ways to customize your setup. If you prefer to do a basic setup, skip Tips and Tricks and proceed to Create a New AWS Cluster (see page 960) section.

1. To get a list of names used in your AWS account, use the `aws` CLI[687]. After downloading, use the following command:

   ```
   aws ec2 describe-vpcs --filter "Name=tag-key,Values=kubernetes.io/cluster" --
   query "Vpcs[*].Tags[?Key=='kubernetes.io/cluster'].Value | sort(@[*][0])"
   ```

686 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/
687 https://aws.amazon.com/cli/

```
    "alex-aws-cluster-afe98",
    "sam-aws-cluster-8if9q"
```

2. (Optional) To create a cluster name that is unique, use the following command:

```
export CLUSTER_NAME=aws-example-$(LC_CTYPE=C tr -dc 'a-z0-9' </dev/urandom |
fold -w 5 | head -n1)
echo $CLUSTER_NAME
```

```
aws-example-pf4a3
```

This will create a unique name every time you run it, so use it with forethought.

3. To use a custom AMI when creating your cluster, you must create that AMI using KIB (see page 1288) first. Then perform the export and name the custom AMI for use in the command `dkp create cluster` after this step:

```
export AWS_AMI_ID=ami-<ami-id-here>
```

4. Set the environment variable for the AMI you choose during the `dkp create cluster` command. This will output the generated manifest into a new file:

```
dkp create cluster aws --cluster-name=${CLUSTER_NAME} \
--ami=${AWS_AMI_ID} \
--dry-run \
--output=yaml \
> ${CLUSTER_NAME}.yaml
```

5. Ensure your subnets (see page 929) do not overlap with your host subnet because they cannot be changed after cluster creation. If you need to change the kubernetes subnets, you must do this at cluster creation. The default subnets used in DKP are:

```
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 192.168.0.0/16
    services:
      cidrBlocks:
      - 10.96.0.0/12
```

6. (Optional) Alternatively, you can create individual files with different smaller manifests for ease in editing using the `--output-directory` flag. This will create multiple files in the specified directory which must already exist:

```
dkp create cluster aws --cluster-name=${CLUSTER_NAME} \
--ami=${AWS_AMI_ID} \
--dry-run \
--output=yaml \
--output-directory=<existing-directory>
```

For more information regarding this flag or others, please refer to the CLI section of the documentation for the dkp create cluster (see page 1454) command and select your provider.

7.10.9.2.4.4 Create a New AWS Kubernetes Cluster

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

1. Ensure your AWS credentials are up to date. If you are using Static Credentials, use the following command to refresh the credentials. Otherwise, proceed to step 2:

```
dkp update bootstrap credentials aws
```

2. Generate the Kubernetes cluster objects. The following example shows a common configuration. See dkp create cluster aws (see page 1456) reference for the full list of cluster creation options:

   NOTE: To increase Docker Hub's rate limit[688] use your Docker Hub credentials when creating the cluster, by setting the following flags `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username=<your-username> --registry-mirror-password=<your-password>` on the `dkp create cluster` command.

688 https://docs.docker.com/docker-hub/download-rate-limit/

```
dkp create cluster aws --cluster-name=${CLUSTER_NAME} \
--dry-run \
--output=yaml \
> ${CLUSTER_NAME}.yaml
```

```
Generating cluster resources
```

Refer to the Tips and Tricks section for more information on how to use optional flags such as the `--output-directory` flag.

3. (Optional) To configure the Control Plane and Worker nodes to use an HTTP proxy:

```
export CONTROL_PLANE_HTTP_PROXY=http://example.org:8080
export CONTROL_PLANE_HTTPS_PROXY=http://example.org:8080
export
CONTROL_PLANE_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1
,10.96.0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.s
vc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.sv
c.cluster,.svc.cluster.local,169.254.169.254,.elb.amazonaws.com"

export WORKER_HTTP_PROXY=http://example.org:8080
export WORKER_HTTPS_PROXY=http://example.org:8080
export
WORKER_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1,10.96.
0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.svc,kube
rnetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.clust
er,.svc.cluster.local,169.254.169.254,.elb.amazonaws.com"
```

- Replace `example.org,example.com,example.net` with you internal addresses
- `localhost` and `127.0.0.1` addresses should not use the proxy
- `10.96.0.0/12` is the default Kubernetes service subnet
- `192.168.0.0/16` is the default Kubernetes pod subnet
- `kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local` is the internal Kubernetes kube-apiserver service
- `.svc,.svc.cluster,.svc.cluster.local` is the internal Kubernetes services

- `169.254.169.254` is the AWS metadata server
- `.elb.amazonaws.com` is for the worker nodes to allow them to communicate directly to the kube-apiserver ELB

4. (Optional) Create a Kubernetes cluster with HTTP proxy configured. This step assumes you did not already create a cluster in the previous steps:

   NOTE: To increase Docker Hub's rate limit[689] use your Docker Hub credentials when creating the cluster, by setting flags `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username=<your-username> --registry-mirror-password=<your-password>` when running `dkp create cluster`.

   ```
   dkp create cluster aws --cluster-name=${CLUSTER_NAME} \
   --control-plane-http-proxy="${CONTROL_PLANE_HTTP_PROXY}" \
   --control-plane-https-proxy="${CONTROL_PLANE_HTTPS_PROXY}" \
   --control-plane-no-proxy="${CONTROL_PLANE_NO_PROXY}" \
   --worker-http-proxy="${WORKER_HTTP_PROXY}" \
   --worker-https-proxy="${WORKER_HTTPS_PROXY}" \
   --worker-no-proxy="${WORKER_NO_PROXY}" \
   --dry-run \
   --output=yaml \
   > ${CLUSTER_NAME}.yaml
   ```

5. Inspect or edit the cluster objects:
   NOTE: Familiarize yourself with Cluster API before editing the cluster objects as edits can prevent the cluster from deploying successfully.

   The objects are Custom Resources[690] defined by Cluster API components, and they belong in three different categories:

   a. Cluster
      A *Cluster* object has references to the infrastructure-specific and control plane objects. Because this is an AWS cluster, there is an *AWSCluster* object that describes the infrastructure-specific cluster properties. Here, this means the AWS region, the VPC ID, subnet IDs, and security group rules required by the Pod network implementation.
   b. Control Plane
      A *KubeadmControlPlane* object describes the control plane, which is the group of machines that run the Kubernetes control plane components, which include the etcd distributed database, the API server, the core controllers, and the scheduler. The object describes the configuration for these components. The object also has a reference to an infrastructure-specific object that describes the properties of all control plane machines. Here, it references

689 https://docs.docker.com/docker-hub/download-rate-limit/
690 https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/

an *AWSMachineTemplate* object, which describes the instance type, the type of disk used, and the size of the disk, among other properties.

c. Node Pool
A Node Pool is a collection of machines with identical properties. For example, a cluster might have one Node Pool with large memory capacity, another Node Pool with GPU support. Each Node Pool is described by three objects: The MachinePool references an object that describes the configuration of Kubernetes components (for example, kubelet) deployed on each node pool machine, and an infrastructure-specific object that describes the properties of all node pool machines. Here, it references a *KubeadmConfigTemplate*, and an *AWSMachineTemplate* object, which describes the instance type, the type of disk used, the size of the disk, among other properties.

For in-depth documentation about the objects, read Concepts[691] in the Cluster API Book.

6. Modify Control Plane Audit logs settings using the information contained in the page Configuring the Control Plane (see page 1351).

7. Create the cluster from the objects. A warning will appear in the console if the resource already exists and will require you to remove the resource or update your YAML.

```
kubectl create -f ${CLUSTER_NAME}.yaml
```

NOTE: If you used the `--output-directory` flag in your `dkp create .. --dry-run` step above, create the cluster from the objects you created by specifying the directory:

```
kubectl create -f <existing-directory>/
```

Output will be similar to output shown below:

```
cluster.cluster.x-k8s.io/aws-example created
awscluster.infrastructure.cluster.x-k8s.io/aws-example created
kubeadmcontrolplane.controlplane.cluster.x-k8s.io/aws-example-control-plane
created
awsmachinetemplate.infrastructure.cluster.x-k8s.io/aws-example-control-plane
created
secret/aws-example-etcd-encryption-config created
machinedeployment.cluster.x-k8s.io/aws-example-md-0 created
awsmachinetemplate.infrastructure.cluster.x-k8s.io/aws-example-md-0 created
kubeadmconfigtemplate.bootstrap.cluster.x-k8s.io/aws-example-md-0 created
clusterresourceset.addons.cluster.x-k8s.io/calico-cni-installation-aws-example
created
configmap/calico-cni-installation-aws-example created
configmap/tigera-operator-aws-example created
clusterresourceset.addons.cluster.x-k8s.io/aws-ebs-csi-aws-example created
configmap/aws-ebs-csi-aws-example created
```

691 https://cluster-api.sigs.k8s.io/user/concepts.html

```
clusterresourceset.addons.cluster.x-k8s.io/cluster-autoscaler-aws-example
created
configmap/cluster-autoscaler-aws-example created
clusterresourceset.addons.cluster.x-k8s.io/node-feature-discovery-aws-example
created
configmap/node-feature-discovery-aws-example created
clusterresourceset.addons.cluster.x-k8s.io/nvidia-feature-discovery-aws-example
created
configmap/nvidia-feature-discovery-aws-example created
```

8. Wait for the cluster control-plane to be ready:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=20m
```

```
cluster.cluster.x-k8s.io/aws-example condition met
```

The `READY` status becomes `True` after the cluster control-plane becomes ready in one of the following steps.

9. After the objects are created on the API server, the Cluster API controllers reconcile them. They create infrastructure and machines. As they progress, they update the Status of each object. Konvoy provides a command to describe the current status of the cluster:

```
dkp describe cluster -c ${CLUSTER_NAME}
```

```
NAME                                                          READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/aws-example                                          True
60s
├──ClusterInfrastructure - AWSCluster/aws-example             True
5m23s
├──ControlPlane - KubeadmControlPlane/aws-example-control-plane  True
60s
│  ├──Machine/aws-example-control-plane-55jh4                 True
4m59s
│  ├──Machine/aws-example-control-plane-6sn97                 True
2m49s
│  └──Machine/aws-example-control-plane-nx9v5                 True
66s
└──Workers
```

```
   └──MachineDeployment/aws-example-md-0                          True
117s
     ├──Machine/aws-example-md-0-cb9c9bbf7-hcl8z                  True
3m1s
     ├──Machine/aws-example-md-0-cb9c9bbf7-rtdqw                  True
3m2s
     ├──Machine/aws-example-md-0-cb9c9bbf7-t894m                  True
3m1s
     └──Machine/aws-example-md-0-cb9c9bbf7-td29r                  True
3m1s
```

10. As they progress, the controllers also create Events. List the Events using this command:

```
kubectl get events | grep ${CLUSTER_NAME}
```

For brevity, the example uses `grep`. It is also possible to use separate commands to get Events for specific objects. For example, `kubectl get events --field-selector involvedObject.kind="AWSCluster"` and `kubectl get events --field-selector involvedObject.kind="AWSMachine"`.

```
7m26s       Normal    SuccessfulSetNodeRef                          machine/
aws-example-control-plane-2wb9q      ip-10-0-182-218.us-west-2.compute.internal
11m         Normal    SuccessfulCreate
awsmachine/aws-example-control-plane-vcjkr   Created new control-plane instance
with id "i-0dde024e80ae3de7a"
11m         Normal    SuccessfulAttachControlPlaneELB
awsmachine/aws-example-control-plane-vcjkr   Control plane instance
"i-0dde024e80ae3de7a" is registered with load balancer
7m25s       Normal    SuccessfulDeleteEncryptedBootstrapDataSecrets
awsmachine/aws-example-control-plane-vcjkr   AWS Secret entries containing
userdata deleted
7m6s        Normal    FailedDescribeInstances
awsmachinepool/aws-example-mp-0              No Auto Scaling Groups with aws-
example-mp-0 found
7m3s        Warning   FailedLaunchTemplateReconcile
awsmachinepool/aws-example-mp-0              Failed to reconcile launch
template: ValidationError: AutoScalingGroup name not found - AutoScalingGroup
aws-example-mp-0 not found
7m1s        Warning   FailedLaunchTemplateReconcile
awsmachinepool/aws-example-mp-0              Failed to reconcile launch
template: ValidationError: AutoScalingGroup name not found - AutoScalingGroup
aws-example-mp-0 not found
6m59s       Warning   FailedLaunchTemplateReconcile
awsmachinepool/aws-example-mp-0              Failed to reconcile launch
```

```
template: ValidationError: AutoScalingGroup name not found - AutoScalingGroup
aws-example-mp-0 not found
6m57s       Warning   FailedLaunchTemplateReconcile
awsmachinepool/aws-example-mp-0              Failed to reconcile launch
template: ValidationError: AutoScalingGroup name not found - AutoScalingGroup
aws-example-mp-0 not found
6m55s       Warning   FailedLaunchTemplateReconcile
awsmachinepool/aws-example-mp-0              Failed to reconcile launch
template: ValidationError: AutoScalingGroup name not found - AutoScalingGroup
aws-example-mp-0 not found
6m53s       Warning   FailedLaunchTemplateReconcile
awsmachinepool/aws-example-mp-0              Failed to reconcile launch
template: ValidationError: AutoScalingGroup name not found - AutoScalingGroup
aws-example-mp-0 not found
6m51s       Warning   FailedLaunchTemplateReconcile
awsmachinepool/aws-example-mp-0              Failed to reconcile launch
template: ValidationError: AutoScalingGroup name not found - AutoScalingGroup
aws-example-mp-0 not found
6m49s       Warning   FailedLaunchTemplateReconcile
awsmachinepool/aws-example-mp-0              Failed to reconcile launch
template: ValidationError: AutoScalingGroup name not found - AutoScalingGroup
aws-example-mp-0 not found
6m47s       Warning   FailedLaunchTemplateReconcile
awsmachinepool/aws-example-mp-0              Failed to reconcile launch
template: ValidationError: AutoScalingGroup name not found - AutoScalingGroup
aws-example-mp-0 not found
74s         Warning   FailedLaunchTemplateReconcile
awsmachinepool/aws-example-mp-0              (combined from similar events):
Failed to reconcile launch template: ValidationError: AutoScalingGroup name not
found - AutoScalingGroup aws-example-mp-0 not found
16m         Normal    SuccessfulCreateVPC
awscluster/aws-example                       Created new managed VPC
"vpc-032fff0fe06a85035"
16m         Normal    SuccessfulSetVPCAttributes
awscluster/aws-example                       Set managed VPC attributes for
"vpc-032fff0fe06a85035"
16m         Normal    SuccessfulCreateSubnet
awscluster/aws-example                       Created new managed Subnet
"subnet-0677a4fbd7d170dfe"
16m         Normal    SuccessfulModifySubnetAttributes
awscluster/aws-example                       Modified managed Subnet
"subnet-0677a4fbd7d170dfe" attributes
16m         Normal    SuccessfulCreateSubnet
awscluster/aws-example                       Created new managed Subnet
"subnet-04fc9deb4fa9f8333"
16m         Normal    SuccessfulCreateSubnet
awscluster/aws-example                       Created new managed Subnet
"subnet-0a266c15dd211ce6c"
16m         Normal    SuccessfulModifySubnetAttributes
awscluster/aws-example                       Modified managed Subnet
"subnet-0a266c15dd211ce6c" attributes
```

```
16m         Normal    SuccessfulCreateSubnet
awscluster/aws-example                      Created new managed Subnet
"subnet-06269d5b52d50840f"
16m         Normal    SuccessfulCreateSubnet
awscluster/aws-example                      Created new managed Subnet
"subnet-0fc41ffef7dceface"
16m         Normal    SuccessfulModifySubnetAttributes
awscluster/aws-example                      Modified managed Subnet
"subnet-0fc41ffef7dceface" attributes
16m         Normal    SuccessfulCreateSubnet
awscluster/aws-example                      Created new managed Subnet
"subnet-0725068cca16ad9f9"
16m         Normal    SuccessfulCreateInternetGateway
awscluster/aws-example                      Created new managed Internet
Gateway "igw-07cd7ad3e6c7c1ca7"
16m         Normal    SuccessfulAttachInternetGateway
awscluster/aws-example                      Internet Gateway
"igw-07cd7ad3e6c7c1ca7" attached to VPC "vpc-032fff0fe06a85035"
16m         Normal    SuccessfulCreateNATGateway
awscluster/aws-example                      Created new NAT Gateway
"nat-0a0cf17d29150cf9a"
16m         Normal    SuccessfulCreateNATGateway
awscluster/aws-example                      Created new NAT Gateway
"nat-065e5e383e6f23320"
16m         Normal    SuccessfulCreateNATGateway
awscluster/aws-example                      Created new NAT Gateway
"nat-01c4a6fed2a31ed4c"
13m         Normal    SuccessfulCreateRouteTable
awscluster/aws-example                      Created managed RouteTable
"rtb-09f4e2eecb7462d22"
13m         Normal    SuccessfulCreateRoute
awscluster/aws-example                      Created route {
13m         Normal    SuccessfulAssociateRouteTable
awscluster/aws-example                      Associated managed RouteTable
"rtb-09f4e2eecb7462d22" with subnet "subnet-0677a4fbd7d170dfe"
13m         Normal    SuccessfulCreateRouteTable
awscluster/aws-example                      Created managed RouteTable
"rtb-0007b98b36f37d1e4"
13m         Normal    SuccessfulCreateRoute
awscluster/aws-example                      Created route {
13m         Normal    SuccessfulAssociateRouteTable
awscluster/aws-example                      Associated managed RouteTable
"rtb-0007b98b36f37d1e4" with subnet "subnet-04fc9deb4fa9f8333"
13m         Normal    SuccessfulCreateRouteTable
awscluster/aws-example                      Created managed RouteTable
"rtb-079a1d7d3667c2525"
13m         Normal    SuccessfulCreateRoute
awscluster/aws-example                      Created route {
13m         Normal    SuccessfulAssociateRouteTable
awscluster/aws-example                      Associated managed RouteTable
"rtb-079a1d7d3667c2525" with subnet "subnet-0a266c15dd211ce6c"
```

```
13m         Normal    SuccessfulCreateRouteTable
awscluster/aws-example                      Created managed RouteTable
"rtb-0e5ebc8ec29848a17"
13m         Normal    SuccessfulCreateRoute
awscluster/aws-example                      Created route {
13m         Normal    SuccessfulAssociateRouteTable
awscluster/aws-example                      Associated managed RouteTable
"rtb-0e5ebc8ec29848a17" with subnet "subnet-06269d5b52d50840f"
13m         Normal    SuccessfulCreateRouteTable
awscluster/aws-example                      Created managed RouteTable
"rtb-087f0c400675c4bce"
13m         Normal    SuccessfulCreateRoute
awscluster/aws-example                      Created route {
13m         Normal    SuccessfulAssociateRouteTable
awscluster/aws-example                      Associated managed RouteTable
"rtb-087f0c400675c4bce" with subnet "subnet-0fc41ffef7dceface"
13m         Normal    SuccessfulCreateRouteTable
awscluster/aws-example                      Created managed RouteTable
"rtb-05a05080bbb3cead9"
13m         Normal    SuccessfulCreateRoute
awscluster/aws-example                      Created route {
13m         Normal    SuccessfulAssociateRouteTable
awscluster/aws-example                      Associated managed RouteTable
"rtb-05a05080bbb3cead9" with subnet "subnet-0725068cca16ad9f9"
13m         Normal    SuccessfulCreateSecurityGroup
awscluster/aws-example                      Created managed SecurityGroup
"sg-0379bf77211472854" for Role "bastion"
13m         Normal    SuccessfulCreateSecurityGroup
awscluster/aws-example                      Created managed SecurityGroup
"sg-0a4e0635f68a2f57d" for Role "apiserver-lb"
13m         Normal    SuccessfulCreateSecurityGroup
awscluster/aws-example                      Created managed SecurityGroup
"sg-022da9dfc21ef3d5e" for Role "lb"
13m         Normal    SuccessfulCreateSecurityGroup
awscluster/aws-example                      Created managed SecurityGroup
"sg-00db2e847c0b49d6e" for Role "controlplane"
13m         Normal    SuccessfulCreateSecurityGroup
awscluster/aws-example                      Created managed SecurityGroup
"sg-01fe3426404f94708" for Role "node"
13m         Normal    SuccessfulAuthorizeSecurityGroupIngressRules
awscluster/aws-example                      Authorized security group ingress
rules [protocol=tcp/range=[22-22]/description=SSH] for SecurityGroup
"sg-0379bf77211472854"
13m         Normal    SuccessfulAuthorizeSecurityGroupIngressRules
awscluster/aws-example                      Authorized security group ingress
rules [protocol=tcp/range=[6443-6443]/description=Kubernetes API] for
 SecurityGroup "sg-0a4e0635f68a2f57d"
13m         Normal    SuccessfulAuthorizeSecurityGroupIngressRules
awscluster/aws-example                      Authorized security group ingress
rules [protocol=tcp/range=[5473-5473]/description=typha (calico) protocol=tcp/
range=[179-179]/description=bgp (calico) protocol=4/range=[-1-65535]/
description=IP-in-IP (calico) protocol=tcp/range=[22-22]/description=SSH
protocol=tcp/range=[6443-6443]/description=Kubernetes API protocol=tcp/range=[2
```

```
379-2379]/description=etcd protocol=tcp/range=[2380-2380]/description=etcd
peer] for SecurityGroup "sg-00db2e847c0b49d6e"
13m         Normal    SuccessfulAuthorizeSecurityGroupIngressRules
awscluster/aws-example                      Authorized security group ingress
rules [protocol=tcp/range=[5473-5473]/description=typha (calico) protocol=tcp/
range=[179-179]/description=bgp (calico) protocol=4/range=[-1-65535]/
description=IP-in-IP (calico) protocol=tcp/range=[22-22]/description=SSH
protocol=tcp/range=[30000-32767]/description=Node Port Services protocol=tcp/
range=[10250-10250]/description=Kubelet API] for SecurityGroup
"sg-01fe3426404f94708"
```

7.10.9.2.4.5 Known Limitations

Be aware of these limitations in the current release of Konvoy.

- The Konvoy version used to create a bootstrap cluster must match the Konvoy version used to create a workload cluster.
- Konvoy supports deploying one workload cluster.
- Konvoy generates a set of objects for one Node Pool.
- Konvoy does not validate edits to cluster objects.

7.10.9.3 AWS Air-gapped Create a Custom Cluster

This section provides the instructions to install DKP in an AWS air-gapped environment with custom settings. First you create an Air-gapped Bootstrap Cluster (see page 970) and then move the CAPI resources to the workload cluster and delete the bootstrap cluster.

If not already done, refer to Get Started (see page 93) section of the documentation for:

- Resource Requirements (see page 110)
- Install Overview (see page 117)
- Prerequisites for Install (see page 119)

7.10.9.3.1 AWS Prerequisites

Before you begin using Konvoy with AWS, you must:

1. Follow the steps to create a Minimal Permissions and Role to Create Clusters (see page 940)..
2. Create Cluster IAM Policies and Roles (see page 946).
3. Export the AWS region where you want to deploy the cluster:

```
export AWS_REGION=us-west-2
```

4. Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

```
export AWS_PROFILE=<profile>
```

For an air-gapped environment, ensure you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz` , so you can extract the tarball on next page.

7.10.9.3.2 Next Step:

AWS Air-gapped Bootstrap (see page 970)

7.10.9.3.3 AWS Air-gapped Bootstrap

7.10.9.3.3.1 Bootstrap a kind cluster and CAPI controllers

Konvoy deploys all cluster lifecycle services to a bootstrap cluster, which deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster, after which the workload cluster manages its own lifecycle.

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.5.1_linux_amd64.tar.gz` , extract the tarball to a local directory:

```
tar -xzvf dkp-air-gapped-bundle_v2.5.1_linux_amd64.tar.gz && cd dkp-v2.5.1
```

2. Set an environment variable with your registry address with this command:

```
export REGISTRY_ADDRESS=<registry-address>:<registry-port>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

3. Seed the registry by running the following command to load the air-gapped image bundle into your private registry:

```
dkp push image-bundle --image-bundle ./container-images/konvoy-image-bundle-
v2.6.0.tar --to-registry $REGISTRY_ADDRESS --to-registry-username
$REGISTRY_USERNAME --to-registry-password $REGISTRY_PASSWORD
```

4. Load the bootstrap image on your bastion machine.

```
docker load -i konvoy-bootstrap-image-v2.5.1.tar
```

```
podman load -i konvoy-bootstrap-image-v2.5.1.tar
```

5. Create a bootstrap cluster:

```
dkp create bootstrap --kubeconfig $HOME/.kube/config
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

6. (Optional) Refresh the credentials used by the AWS provider at any time, using the command:

```
dkp update bootstrap credentials aws
```

7.10.9.3.3.2 Next Step:

Create a New Air-gapped AWS Cluster (see page 971)

7.10.9.3.4 Create a New Air-gapped AWS Cluster

7.10.9.3.4.1 Prerequisites

Before you begin, make sure you have created a Bootstrap (see page 970) cluster.

> ⚠ DKP uses `localvolumeprovisioner` as the default storage provider (see page 103). However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[692] compatible storage that is suitable for production.

You can choose from any of the storage options[693] available for Kubernetes. To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your

692 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
693 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

newly created StorageClass to be the default by following the commands in the Kubernetes documentation called Changing the Default Storage Class[694].

7.10.9.3.4.2 Create a New Cluster

Create a new AWS Kubernetes cluster in an Air-gapped environment in your AWS infrastructure. When you use existing infrastructure, DKP does *not* create, modify, or delete the following AWS resources:

- Internet Gateways
- NAT Gateways
- Routing tables
- Subnets
- VPC
- VPC Endpoints (for subnets without NAT Gateways)

An AWS subnet has Network ACLs that can control traffic in and out of the subnet. DKP does not modify the Network ACLs of an existing subnet. DKP uses Security Groups to control traffic. If a Network ACL denies traffic that is allowed by DKP-managed Security Groups, the cluster may not work correctly.

1. Set the environment variable to the name you assigned this cluster:

```
export CLUSTER_NAME=aws-example
```

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[695] for more naming information.

2. Export variables for the existing infrastructure details:

```
export AWS_VPC_ID=<vpc-...>
export AWS_SUBNET_IDS=<subnet-...,subnet-...,subnet-...>
export AWS_ADDITIONAL_SECURITY_GROUPS=<sg-...>
export AWS_AMI_ID=<ami-...>
```

694 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/
695 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

- `AWS_VPC_ID` : the VPC ID where the cluster will be created. The VPC requires the `ec2` , `elasticloadbalancing` , `secretsmanager` and `autoscaling` VPC endpoints to be already present.
- `AWS_SUBNET_IDS` : a comma-separated list of one or more private Subnet IDs with each one in a different Availability Zone. The cluster control-plane and worker nodes will automatically be spread across these Subnets.
- `AWS_ADDITIONAL_SECURITY_GROUPS` : a comma-seperated list of one or more Security Groups IDs to use in addition to the ones automatically created by CAPA[696].
- `AWS_AMI_ID` : the AMI ID to use for control-plane and worker nodes. The AMI must be created by the konvoy-image-builder (see page 1282).

⚠ **IMPORTANT**: Ensure your subnets (see page 929) do not overlap with your host subnet because they cannot be changed after cluster creation. If you need to change the kubernetes subnets, you must do this at cluster creation. The default subnets used in DKP are:

```
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 192.168.0.0/16
    services:
      cidrBlocks:
      - 10.96.0.0/12
```

⚠ **IMPORTANT:** You must tag the subnets as described below to allow for Kubernetes to create ELBs for services of type `LoadBalancer` in those subnets. If the subnets are not tagged, they will not receive an ELB and the following error displays: `Error syncing load balancer, failed to ensure load balancer; could not find any suitable subnets for creating the ELB.` .

The tags should be set as follows, where `<CLUSTER_NAME>` corresponds to the name set in `CLUSTER_NAME` environment variable:

```
kubernetes.io/cluster = <CLUSTER_NAME>
kubernetes.io/cluster/<CLUSTER_NAME> = owned
kubernetes.io/role/internal-elb = 1
```

⚠ If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1349) page for more information.

696 https://github.com/kubernetes-sigs/cluster-api-provider-aws

3. Configure your cluster to use an existing Docker registry as a mirror when attempting to pull images. The example below is for AWS ECR:

   ⚠ **IMPORTANT:** The AMI must be created with Konvoy Image Builder (see page 1282) in order to use the registry mirror feature.

   ```
   export --registry-mirror-url=<YOUR_ECR_URL>
   ```

   - `CONTAINER_REGISTRY_URL` : the address of an existing Docker registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
   - NOTE: Other local registries may use the options below:
     - JFrog - `CONTAINER_REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
     - `CONTAINER_REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
     - `CONTAINER_REGISTRY_PASSWORD` : optional if username is not set.

4. Create a Kubernetes cluster. The following example shows a common configuration. See dkp create cluster aws (see page 1456) reference for the full list of cluster creation options:

   ```
   dkp create cluster aws --cluster-name=${CLUSTER_NAME} \
   --vpc-id=${AWS_VPC_ID} \
   --ami=${AWS_AMI_ID} \
   --subnet-ids=${AWS_SUBNET_IDS} \
   --internal-load-balancer=true \
   --additional-security-group-ids=${AWS_ADDITIONAL_SECURITY_GROUPS} \
   --registry-mirror-url=<YOUR_ECR_URL>
   ```

5. (Optional) The Control Plane and Worker nodes can be configured to use an HTTP proxy:

   ```
   export CONTROL_PLANE_HTTP_PROXY=http://example.org:8080
   export CONTROL_PLANE_HTTPS_PROXY=http://example.org:8080
   export
   CONTROL_PLANE_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1
   ,10.96.0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.s
   vc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.sv
   c.cluster,.svc.cluster.local,169.254.169.254,.elb.amazonaws.com"

   export WORKER_HTTP_PROXY=http://example.org:8080
   export WORKER_HTTPS_PROXY=http://example.org:8080
   ```

```
export
WORKER_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1,10.96.
0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.svc,kube
rnetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.clust
er,.svc.cluster.local,169.254.169.254,.elb.amazonaws.com"
```

- Replace `example.org,example.com,example.net` with you internal addresses
- `localhost` and `127.0.0.1` addresses should not use the proxy
- `10.96.0.0/12` is the default Kubernetes service subnet
- `192.168.0.0/16` is the default Kubernetes pod subnet
- `kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local` is the internal Kubernetes kube-apiserver service
- `.svc,.svc.cluster,.svc.cluster.local` is the internal Kubernetes services
- `169.254.169.254` is the AWS metadata server
- `.elb.amazonaws.com` is for the worker nodes to allow them to communicate directly to the kube-apiserver ELB

6. (Optional) Create a Kubernetes cluster with HTTP proxy configured. This step assumes you did not already create a cluster in the previous steps:

```
dkp create cluster aws --cluster-name=${CLUSTER_NAME} \
--vpc-id=${AWS_VPC_ID} \
--ami=${AWS_AMI_ID} \
--subnet-ids=${AWS_SUBNET_IDS} \
--internal-load-balancer=true \
--additional-security-group-ids=${AWS_ADDITIONAL_SECURITY_GROUPS} \
--registry-mirror-url=<YOUR_ECR_URL>
--control-plane-http-proxy="${CONTROL_PLANE_HTTP_PROXY}" \
--control-plane-https-proxy="${CONTROL_PLANE_HTTPS_PROXY}" \
--control-plane-no-proxy="${CONTROL_PLANE_NO_PROXY}" \
--worker-http-proxy="${WORKER_HTTP_PROXY}" \
--worker-https-proxy="${WORKER_HTTPS_PROXY}" \
--worker-no-proxy="${WORKER_NO_PROXY}"
```

7. Inspect the created cluster resources:

```
kubectl get clusters,kubeadmcontrolplanes,machinedeployments
```

8. Wait for the cluster control-plane to be ready:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=60m
```

Then, Explore New AWS Cluster (see page 976)

7.10.9.4 Explore New AWS Cluster

This guide explains how to use the command line to interact with your newly deployed Kubernetes cluster.

Follow these steps:

1. Fetch the kubeconfig file with the command:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

2. List the Nodes with the command:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf get nodes
```

   Note: wait for the Status to move to `Ready` while `calico-node` pods are being deployed.

3. List the Pods with the command:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf get pods -A
```

7.10.9.5 Make the AWS Cluster Self-Managed

Konvoy deploys all cluster lifecycle services to a bootstrap cluster, which then deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster, which makes the workload cluster self-managed. This section describes how to make a workload cluster self-managed.

This page contains instructions on how to make your cluster self-managed. This is necessary if there is only one cluster in your environment, or if this cluster should become the Management cluster in a multi-cluster environment.

If you already have a self-managed or Management cluster in your environment, skip this page.

7.10.9.5.1 Make the New Kubernetes Cluster Manage Itself

1. Deploy cluster lifecycle services on the workload cluster:

```
dkp create capi-components --kubeconfig ${CLUSTER_NAME}.conf
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

The output resembles:

```
✓ Initializing new CAPI components
```

2. Move the Cluster API objects from the bootstrap to the workload cluster:

   The cluster lifecycle services on the workload cluster are ready, but the workload cluster configuration is on the bootstrap cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the bootstrap to the workload cluster. This process is also called a Pivot[697].

   ```
   dkp move capi-resources --to-kubeconfig ${CLUSTER_NAME}.conf
   ```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

   The output resembles:

   ```
    ✓ Moving cluster resources
   You can now view resources in the moved cluster by using the --kubeconfig flag
   with kubectl. For example: kubectl --kubeconfig=aws-example.conf get nodes
   ```

   NOTE: To ensure only one set of cluster lifecycle services manages the workload cluster, Konvoy first pauses reconciliation of the objects on the bootstrap cluster, then creates the objects on the workload cluster. As Konvoy copies the objects, the cluster lifecycle services on the workload cluster reconcile the objects. The workload cluster becomes self-managed after Konvoy creates all the objects. If it fails, the `move` command can be safely retried.

3. Wait for the cluster control-plane to be ready:

697 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf wait --for=condition=ControlPlaneRead
y "clusters/${CLUSTER_NAME}" --timeout=20m
```

```
cluster.cluster.x-k8s.io/aws-example condition met
```

4. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

   NOTE: After moving the cluster lifecycle services to the workload cluster, remember to use Konvoy with the workload cluster kubeconfig.

```
dkp describe cluster --kubeconfig ${CLUSTER_NAME}.conf -c ${CLUSTER_NAME}
```

```
NAME                                                              READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/aws-example                                              True
109s
├──ClusterInfrastructure - AWSCluster/aws-example                 True
112s
├──ControlPlane - KubeadmControlPlane/aws-example-control-plane  True
109s
│  ├──Machine/aws-example-control-plane-55jh4                     True
111s
│  ├──Machine/aws-example-control-plane-6sn97                     True
111s
│  └──Machine/aws-example-control-plane-nx9v5                     True
110s
└──Workers
   └──MachineDeployment/aws-example-md-0                          True
114s
     ├──Machine/aws-example-md-0-cb9c9bbf7-hcl8z                  True
111s
     ├──Machine/aws-example-md-0-cb9c9bbf7-rtdqw                  True
111s
     ├──Machine/aws-example-md-0-cb9c9bbf7-t894m                  True
111s
     └──Machine/aws-example-md-0-cb9c9bbf7-td29r                  True
111s
```

5. Remove the bootstrap cluster, as the workload cluster is now self-managed:

```
dkp delete bootstrap
```

```
✓ Deleting bootstrap cluster
```

7.10.9.5.2 Install and Log in to the Kommander UI

You can now proceed to installing the UI with Kommander (see page 1227) and applications. After installation, you will be able to log in (see page 1280) to the UI to explore it.

7.10.9.5.3 DKP Install Configuration

You can configure (see page 1227) the Kommander component of DKP during the initial installation, and also post-installation using the Kommander CLI.

The following explains how to run DKP on top of an air-gapped DKP cluster (see page 971) installation with catalog applications.

Depending on your configuration, there are three different ways you can install DKP to an air-gapped environment.
Ensure you follow the correct procedure for your configuration and license type, and ignore the other sections that do not pertain to your environment:

Essential:

- Kommander in an Air-gapped Environment (see page 1260)
- Kommander in Air-gapped with DKP Insights (see page 1262)

Enterprise:

- Install Air-gapped Kommander with DKP Catalog Applications (see page 1264)
- Install Air-gapped Kommander with DKP Insights and DKP Catalog Applications (see page 1267)

7.10.9.5.4 Known Limitations

- Before making a workload cluster self-managed, be sure that its control plane nodes have sufficient permissions for running Cluster API controllers. See IAM Policy Configuration (see page 946).

- Konvoy supports moving only one set of cluster objects from the bootstrap cluster to the workload cluster, or vice-versa.
- Konvoy only supports moving all namespaces in the cluster; Konvoy does not support migration of individual namespaces.

7.10.9.6 Creating DKP Clusters on AWS in the UI

A guide for creating DKP clusters on AWS console

7.10.9.6.1 Before you begin

- Configure an AWS Infrastructure (see page 931), or add credentials using AWS role credentials (see page 531).

7.10.9.6.2 Simplified Cluster Creation on AWS in the UI

1. In the selected workspace Dashboard, select the **Add Cluster** option in the **Actions** dropdown menu at the top right.
2. On the Add Cluster page, select the **Create DKP Cluster**.
3. Provide some basic cluster details within the form:
    - **Workspace**: The workspace where this cluster belongs (if within the Global workspace).
    - **Kubernetes Version**: The initial version of Kubernetes to install on the cluster.
    - **Name**: A valid Kubernetes name for the cluster.
    - **Add Labels**: By default, your cluster has labels that reflect the infrastructure provider provisioning. For example, your AWS cluster may have a label for the data center region and `provider: aws`. Cluster labels are matched to the selectors created for Projects (see page 608). Changing a cluster label may add or remove the cluster from projects.
4. Select the pre-configured AWS Infrastructure (see page 931), or AWS role credentials (see page 531) to display the remaining options specific to AWS.
    - **Region**: Select a data center region or specify a custom region.
    - **Configure Node Pools**: Specify pools of nodes, their machine types, quantity, and the IAM instance profile.
        - **Machine Type**: Machine instance type.
        - **Quantity**: Number of nodes. The control plane must be an odd number.
        - **IAM instance profile**: Name of the IAM instance profile to assign to the machines.
    - **AMI Image Lookup**: You can also specify the base OS, lookup format, and owner ID of the AMI Image, or the AMI ID as part of each pool. Selecting ‘Show Advanced’ within the Configure Node Pools section will show these options.
        - **Base OS**: Base OS for Lookup search.

        - **Lookup Format**: Lookup Format string to generate AMI search name from.
        - **Owner ID**: Owner ID for AMI Lookup search.
        - **AMI ID**: AMI ID to use for all nodes.
    - **Worker Availability Zone:** Worker Zones for a region (worker nodes only).
    - **Add Infrastructure Provider Tags**: Specify tags applied on all resources created in your infrastructure for this cluster. Different infrastructure providers have varying restrictions on the usable tags. See the AWS Tags User Guide[698] for more information on using tags in AWS.
5. Select **Create** to begin provisioning the cluster. This step may take a few minutes, taking time for the cluster to be ready and fully deploy its components. The cluster automatically tries to join, and should resolve once it is fully-provisioned.

7.10.10 Delete an AWS Cluster

> A self-managed workload cluster cannot delete itself. If your workload cluster is self-managed, you must create a bootstrap cluster and move the cluster lifecycle services to the bootstrap cluster before deleting the workload cluster.

If you did not make your workload cluster self-managed, as described in Make New Cluster Self-Managed (see page 976), see Delete the workload cluster (see page 983).

Follow these steps to prepare to delete an AWS Cluster.

1. Make sure your AWS credentials are up to date. Refresh the credentials using this command:

   ```
   dkp update bootstrap credentials aws --kubeconfig $HOME/.kube/config
   ```

2. Create a bootstrap cluster:

   The bootstrap cluster will host the Cluster API controllers that reconcile the cluster objects marked for deletion:

   NOTE: To avoid using the wrong kubeconfig, the following steps use explicit kubeconfig paths and contexts.

   ```
   dkp create bootstrap --kubeconfig $HOME/.kube/config --with-aws-bootstrap-credentials=true
   ```

698 https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Using_Tags.html

```
 ✓ Creating a bootstrap cluster
 ✓ Initializing new CAPI components
```

3. Move the Cluster API objects from the workload to the bootstrap cluster: The cluster lifecycle services on the bootstrap cluster are ready, but the workload cluster configuration is on the workload cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the workload to the bootstrap cluster. This process is also called a Pivot[699].

```
dkp move capi-resources \
    --from-kubeconfig ${CLUSTER_NAME}.conf \
    --from-context ${CLUSTER_NAME}-admin@${CLUSTER_NAME} \
    --to-kubeconfig $HOME/.kube/config \
    --to-context kind-konvoy-capi-bootstrapper
```

```
✓ Moving cluster resources
You can now view resources in the moved cluster by using the --kubeconfig flag
with kubectl. For example: kubectl --kubeconfig $HOME/.kube/config get nodes
```

4. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

```
dkp describe cluster --kubeconfig $HOME/.kube/config -c ${CLUSTER_NAME}
```

```
NAME                                                          READY  SEVERITY
REASON  SINCE  MESSAGE
Cluster/aws-example                                           True
91s
├──ClusterInfrastructure - AWSCluster/aws-example              True
103s
├──ControlPlane - KubeadmControlPlane/aws-example-control-plane  True
91s
│  ├──Machine/aws-example-control-plane-55jh4                  True
102s
│  ├──Machine/aws-example-control-plane-6sn97                  True
102s
│  └──Machine/aws-example-control-plane-nx9v5                  True
102s
└──Workers
```

699 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

```
   └──MachineDeployment/aws-example-md-0                          True
108s
     ├──Machine/aws-example-md-0-cb9c9bbf7-hcl8z                  True
102s
     ├──Machine/aws-example-md-0-cb9c9bbf7-rtdqw                  True
102s
     ├──Machine/aws-example-md-0-cb9c9bbf7-td29r                  True
102s
     └──Machine/aws-example-md-0-cb9c9bbf7-w64kg                  True
102s
```

NOTE: After moving the cluster lifecycle services to the workload cluster, remember to use DKP with the workload cluster kubeconfig and use DKP with the bootstrap cluster to delete the workload cluster.

5. Wait for the cluster control-plane to be ready:

```
kubectl --kubeconfig $HOME/.kube/config wait --for=condition=controlplaneready
"clusters/${CLUSTER_NAME}" --timeout=20m
```

```
cluster.cluster.x-k8s.io/aws-example condition met
```

7.10.10.1 Delete the Workload Cluster

1. Make sure your AWS credentials are up to date. Refresh the credentials using this command:

```
dkp update bootstrap credentials aws --kubeconfig $HOME/.kube/config
```

NOTE: Persistent Volumes (PVs) are not deleted automatically by design in order to preserve your data. However, they take up storage space if not deleted. You must delete PVs manually. Information for backup of a cluster and PVs is on the page in documentation called Back up your Cluster's Applications and Persistent Volumes (see page 736) .

2. To delete a cluster, you would use `dkp delete cluster` and pass in the name of the cluster you are trying to delete with `--cluster-name` flag. You would use `kubectl get clusters` to get those details (`--cluster-name` and `--namespace`) of the Kubernetes cluster to delete it.
NOTE: Do not use `dkp get clusters` since that gets you Kommander cluster details rather than Konvoy kubernetes cluster details.

```
kubectl get clusters
```

3. Delete the Kubernetes cluster and wait a few minutes:

   NOTE: Before deleting the cluster, dkp deletes all Services of type LoadBalancer on the cluster. Each Service is backed by an AWS Classic ELB. Deleting the Service deletes the ELB that backs it. To skip this step, use the flag `--delete-kubernetes-resources=false`. Do not skip this step if the VPC is managed by DKP. When DKP deletes the cluster, it deletes the VPC. If the VPC has any AWS Classic ELBs, AWS does not allow the VPC to be deleted, and DKP cannot delete the cluster.

   ```
   dkp delete cluster --cluster-name=${CLUSTER_NAME} --kubeconfig $HOME/.kube/
   config
   ```

   ```
   ✓ Deleting Services with type LoadBalancer for Cluster default/azure-example
   ✓ Deleting ClusterResourceSets for Cluster default/azure-example
   ✓ Deleting cluster resources
   ✓ Waiting for cluster to be fully deleted
   Deleted default/azure-example cluster
   ```

   After the workload cluster is deleted, delete the bootstrap cluster.

7.10.10.2 Delete the Bootstrap Cluster

After you have moved the workload resources back to a bootstrap cluster and deleted the workload cluster, you no longer need the bootstrap cluster. You can safely delete the bootstrap cluster with these steps:

1. Make sure your AWS credentials are up to date. Refresh the credentials using this command:

   ```
   dkp update bootstrap credentials aws --kubeconfig $HOME/.kube/config
   ```

2. Delete the bootstrap cluster:

   ```
   dkp delete bootstrap --kubeconfig $HOME/.kube/config
   ```

   ```
   ✓ Deleting bootstrap cluster
   ```

7.10.10.2.1 Air-gapped:

1. Make sure your AWS credentials are up to date. Refresh the credentials using this command:

```
dkp update bootstrap credentials aws --kubeconfig $HOME/.kube/config
```

2. Delete the provisioned Kubernetes cluster and wait a few minutes:

```
dkp delete cluster --cluster-name=${CLUSTER_NAME}
```

3. Delete the `kind` Kubernetes cluster:

```
dkp delete bootstrap
```

7.10.10.3 Next Step:

Once your cluster is built in the Konvoy component of DKP for your infrastructure/environment, you will install the Kommander (see page 1227) component of DKP to see your dashboard and continue customization.

7.10.10.4 Known Limitations

- The Konvoy version used to create the workload cluster must match the Konvoy version used to delete the workload cluster.

7.10.11 Multiple AWS Accounts

Leverage Multiple AWS Accounts for Kubernetes Cluster Deployments

7.10.11.1 Objective

You can leverage multiple AWS accounts in your organization to meet specific business purposes, reflect your organizational structure, or implement a multi-tenancy strategy. Specific scenarios include:

- Implementing isolation between environment tiers such as development, testing, acceptance, and production.
- Implementing separation of concerns between management clusters, and workload clusters.
- Reducing the impact of security events and incidents.

For additional benefits of using multiple AWS accounts, refer to the following white paper[700].

700 https://docs.aws.amazon.com/whitepapers/latest/organizing-your-aws-environment/benefits-of-using-multiple-aws-accounts.html

This document describes how to leverage the D2iQ Kubernetes Platform (DKP) to deploy a management cluster, and multiple workload clusters, leveraging multiple AWS accounts.

7.10.11.2 Assumptions

This guide assumes you have some understanding of Cluster API concepts and basic DKP provisioning workflows on AWS.

Cluster API Concepts - cluster API concepts[701]

AWS Install Options (see page 261)

7.10.11.3 Glossary

- **Management cluster** - The cluster that runs in AWS and is used to create target clusters in different AWS accounts.
- **Target account** - The account where the target cluster is created.
- **Source account** - The AWS account where the CAPA controllers for the management cluster runs.

7.10.11.4 Prerequisites

Before you begin deploying DKP on AWS, you configure the prerequisites for the environment you use either non-air-gapped or air-gapped.

Prerequisites for Install (see page 119)

7.10.11.5 Deploy DKP on AWS

1. Deploy a management cluster in your AWS source account.
   AWS: create Kubernetes AWS cluster (see page 958)
2. Configure a trusted relationship between source and target accounts and create a management cluster:

7.10.11.5.1 Step 1:

DKP leverages the Cluster API provider for AWS (CAPA) to provision Kubernetes clusters in a declarative way. Customers declare the desired state of the cluster through a cluster configuration YAML file which is generated using:

(AWS)

```
dkp create cluster aws --cluster-name=${CLUSTER_NAME} \
--dry-run \
--output=yaml \
```

701 https://cluster-api.sigs.k8s.io/user/concepts.html

```
> ${CLUSTER_NAME}.yaml
```

7.10.11.5.2 Step 2:

Configure a trust relationship between the source and target accounts.

Follow all the prerequisite steps in both the source and target accounts

1. Create all policies and roles in management and workload accounts a. The prerequisite IAM policies for DKP are documented here: Configure AWS IAM policies (see page 946).
2. Establish a trust relationship in workload account for the management account.

   a. Go to your target (workload) account b. Search for the role control-plane.cluster-api-provider-aws.sigs.k8s.io c. Navigate to the Trust Relationship tab and select Edit Trust Relationship d. Add the following relationship:

   ```
   {
     "Effect": "Allow",
     "Principal": {
        "AWS": "arn:aws:iam::${mgmt-aws-account}:role/control-plane.cluster-api-
   provider-aws.sigs.k8s.io"
     },
     "Action": "sts:AssumeRole"
   }
   ```

3. Give permission to role in the source (management cluster) account to call the `sts:AssumeRole API` a. Log in to the source AWS account and attach the following inline policy to control-plane.cluster-api-provider-aws.sigs.k8s.io role:

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
        {
          "Effect": "Allow",
          "Action": "sts:AssumeRole",
          "Resource": [
   "arn:aws:iam::${workload-aws-account}:role/control-plane.cluster-api-provider-
   aws.sigs.k8s.io"
          ]
        }
     ]
   }
   ```

4. Modify the management cluster configuration file and update the AWSCluster object with following details:

```
apiVersion: infrastructure.cluster.x-k8s.io/v1alpha3
kind: AWSCluster
metadata:
spec:
  identityRef:
     kind: AWSClusterRoleIdentity
     name: cross-account-role
…
…
---
apiVersion: infrastructure.cluster.x-k8s.io/v1alpha3
kind: AWSClusterRoleIdentity
metadata:
  name: cross-account-role
spec:
  allowedNamespaces: {}
  roleARN: "arn:aws:iam::${workload-aws-account}:role/control-plane.cluster-
api-provider-aws.sigs.k8s.io"
  sourceIdentityRef:
    kind: AWSClusterControllerIdentity
    name: default
```

After performing the above steps, your Management cluster will be configured to create new managed clusters in the target AWS workload account.

7.10.11.6 Next Steps:

AWS Install Options (see page 261)

7.10.12 GPUs in an AWS environment

7.10.12.1 Understanding GPUs

Before working with GPUs, ensure you are familiar with the following:

- Install GPU support on supported distributions on AWS
- GPU node labeling specifications described in Configure Konvoy Automatic GPU Node Labels (see page 990)

Kommander also accesses Kommander GPU configuration (see page 866) resources.

For using GPUs in an air-gapped on-premises environment, D2iQ recommends setting up Pod Disruption Budget[702] before Update Cluster Nodepools (see page 1359).

7.10.12.2 Install GPU Support for Supported Distributions on AWS

Using the Konvoy Image Builder (see page 1282), you can build an image that has support to use NVIDIA GPU hardware to support GPU workloads. DKP supported NVIDIA driver[703] version is 470.x.

1. In your `overrides/nvidia.yaml` file add the following to enable GPU builds. You can also access and use the overrides repo.[704]

   ```
   gpu:
       type:
       - nvidia
   ```

   Build your image using the following Konvoy image builder commands:

   ```
   konvoy-image build --region us-west-2 --source-ami=ami-12345abcdef images/ami/
   centos-7.yaml --overrides overrides/nvidia.yaml
   ```

   By default, your image builds in the `us-west-2` region. To specify another region, set the `--region` flag:

   ```
   konvoy-image build --region us-east-1 --overrides override-source-ami.yaml
   images/ami/<Your OS>.yaml
   ```

Ensure that an AMI file is available for your OS selection.

1. When the command is complete the `ami` id is printed and written to `./manifest.json`.

702 https://kubernetes.io/docs/concepts/workloads/pods/disruptions/
703 https://www.nvidia.com/Download/Find.aspx
704 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

To use the built `ami` with Konvoy Image Builder, specify it with the `--ami` flag when calling cluster create.

```
dkp create cluster aws --cluster-name=$(whoami)-aws-cluster --region us-west-2
 --ami <ami>
```

Additional helpful information can be found in the NVIDIA Device Plug-in[705] for Kubernetes instructions and the Installation Guide of Supported Platforms[706].

See also: NVIDIA documentation[707]

7.10.12.3 Configure Konvoy Automatic GPU Node Labels

When using GPU nodes, it is important they have the proper label identifying them as Nvidia GPU nodes. Node feature discovery (NFD), by default labels PCI hardware as:

```
"feature.node.kubernetes.io/pci-<device label>.present": "true"
```

where `<device label>` is by default as defined in this topic[708]:

```
< class > _ < vendor >
```

However, because there is a wide variety in devices and their assigned PCI classes, you may find that the labels assigned to your GPU nodes do not always properly identify them as containing an Nvidia GPU.

If the default detection does not work, you can manually change the daemonset that the GPU operator creates by running the following command:

```
          nodeSelector:
            feature.node.kubernetes.io/pci-< class > _ < vendor>.present: "true"
```

where `class` is any 4 digit number starting with `03xy` and the vendor for Nvidia is `10de`. If this is already deployed, you can always change the `daemonset` and change the `nodeSelector` field so that it deploys to the right nodes.

7.10.12.4 Update NVIDIA GPU Clusters

Upgrading a node pool involves draining the existing nodes in the node pool and replacing them with new nodes. In order to ensure minimum downtime and maintain high availability of the critical application

705 https://github.com/NVIDIA/k8s-device-plugin/blob/master/README.md
706 https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html
707 https://nvidia.custhelp.com/app/answers/detail/a_id/131/kw/driver%20installation%20docs/related/1
708 https://kubernetes-sigs.github.io/node-feature-discovery/v0.7/get-started/features.html#pci

workloads during the upgrade process, we recommend deploying Pod Disruption Budget (Disruptions[709]) for your critical applications.

The Pod Disruption Budget will prevent any impact on critical applications as a result of misconfiguration or failures during the upgrade process.

7.10.12.4.1 Prerequisites:

- Deploy Pod Disruption Budget[710] (PDB)
- Konvoy Image Builder (see page 1282) (KIB)

7.10.12.4.2 Steps:

1. Deploy Pod Disruption Budget for your critical applications. If your application can tolerate only one replica to be unavailable at a time, then you can set Pod disruption budget as shown in the following example. The example below is for NVIDIA GPU node pools, but the process is the same for all node pools.

> Repeat this step for each additional node pool.

Create the file: `pod-disruption-budget-nvidia.yaml`

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: nvidia-critical-app
spec:
  maxUnavailable: 1
  selector:
    matchLabels:
      app: nvidia-critical-app
```

Apply the YAML file above using the following command:

```
kubectl create -f pod-disruption-budget-nvidia.yaml
```

2. Prepare OS image for your node pool using Konvoy Image Builder (see page 1282).

709 https://kubernetes.io/docs/concepts/workloads/pods/disruptions/
710 https://kubernetes.io/docs/concepts/workloads/pods/disruptions/

7.10.13 Manage AWS Node Pools

Node pools are part of a cluster and managed as a group, and you can use a node pool to manage a group of machines using the same common properties. When Konvoy creates a new default cluster, there is one node pool for the worker nodes and all nodes in that new node pool have the same configuration. You can create additional node pools for more specialized hardware or configuration. For example, if you want to tune your memory usage on a cluster where you need maximum memory for some machines and minimal memory on other machines, you would create a new node pool with those specific resource needs.

DKP implements node pools using Cluster API MachineDeployments[711]. For more information on node pools, see these sections:

- Create AWS Node Pools (see page 992)
- List AWS Node Pools (see page 993)
- Scale AWS Node Pools (see page 994)
- Delete AWS Node Pools (see page 997)
- AWS Cluster Autoscaler (see page 997)

7.10.13.1 Create AWS Node Pools

Creating a node pool is useful when you need to run workloads that require machines with specific resources, such as a GPU, additional memory, or specialized network or storage hardware.

7.10.13.1.1 Prepare the environment

1. Set the environment variable to the name you assigned this cluster.

   ```
   export CLUSTER_NAME=aws-example
   ```

   See AWS Install Options (see page 261) for information on naming your cluster.

2. If your workload cluster is self-managed, as described in Make the New Cluster Self-Managed (see page 976), configure `kubectl` to use the kubeconfig for the cluster.

   ```
   export KUBECONFIG=${CLUSTER_NAME}.conf
   ```

3. Define your node pool name.

   ```
   export NODEPOOL_NAME=example
   ```

711 https://cluster-api.sigs.k8s.io/developer/architecture/controllers/machine-deployment.html

7.10.13.1.2 Create an AWS node pool

Availability zones (AZs) are isolated locations within data center regions from which public cloud services originate and operate. Because all the nodes in a node pool are deployed in a single Availability Zone, you may wish to create additional node pools is to ensure your cluster has nodes deployed in multiple Availability Zones.

> By default, the first Availability Zone in the region will be used for the Nodes in the node pool, set `--availability-zone` to create the Nodes in a different Availability Zone.

Create a new AWS node pool with 3 replicas using this command:

```
dkp create nodepool aws ${NODEPOOL_NAME} \
    --cluster-name=${CLUSTER_NAME} \
    --replicas=3
```

This example uses default values for brevity. Use flags to define custom instance types, AMIs, and other properties.

Advanced users can use a combination of the `--dry-run` and `--output=yaml` or `--output-directory=<existing-directory>` flags to get a complete set of node pool objects to modify locally or store in version control.

> For more information regarding this flag or others, please refer to the DKP CLI Create (see page 1472) section of the documentation for either cluster or nodepool and select your provider.

7.10.13.2 List AWS Node Pools

List node pools for a cluster

Use this command to list the node pools of a given cluster. This returns specific properties of each node pool so that you can see the name of the MachineDeployments.

To list all node pools for a managed cluster, run:

```
dkp get nodepools --cluster-name=${CLUSTER_NAME} --kubeconfig=${CLUSTER_NAME}.conf
```

The expected output is similar to the following example, indicating the desired size of the node pool, the number of replicas ready in the node pool, and the Kubernetes version those nodes are running:

```
NODEPOOL                        DESIRED             READY
KUBERNETES VERSION
example                         3                   3                   v1.24.6
aws-example-md-0                4                   4                   v1.24.6
```

7.10.13.2.1 Attached Clusters

Before running the command to list the attached clusters, ensure that you know the following:

- KUBECONFIG (see page 99) for the management cluster (see page 97) - In order to find the KUBECONFIG for a cluster from the UI, refer to this section in the documentation: Access a Managed or Attached Cluster (see page 731)
- The CLUSTER_NAME of the attached cluster
- The NAMESPACE of the attached cluster

To list all node pools for an attached cluster, run:

```
dkp get nodepools --cluster-name=${ATTACHED_CLUSTER_NAME} --kubeconfig=$
{CLUSTER_NAME}.conf -n ${ATTACHED_CLUSTER_NAMESPACE}
```

The expected output is similar to below:

```
NODEPOOL                    DESIRED             READY               KUBERNETES
VERSION
aws-attached-md-0           4                   4                   v1.24.6
```

7.10.13.3 Scale AWS Node Pools

Scale node pools in a cluster

While you can run Cluster Autoscaler (see page 997), you can also manually scale your node pools up or down when you need more finite control over your environment. For example, if you require 10 machines to run a process, you can manually set the scaling to run those 10 machines only. However, if also using the Cluster Autoscaler, you must stay within your minimum and maximum bounds.

7.10.13.3.1 Scaling Up Node Pools

To scale up a node pool in a cluster, run:

```
dkp scale nodepools ${NODEPOOL_NAME} --replicas=5 --cluster-name=${CLUSTER_NAME}
```

OR for attached clusters:

```
dkp scale nodepools ${ATTACHED_NODEPOOL_NAME} --replicas=5 --cluster-name=$
{ATTACHED_CLUSTER_NAME} --kubeconfig=${CLUSTER_NAME}.conf -n $
{ATTACHED_CLUSTER_WORKSPACE}
```

Your output should be similar to this example, indicating the scaling is in progress:

```
✓ Scaling node pool example to 5 replicas
```

After a few minutes, you can list the node pools to:

```
dkp get nodepools --cluster-name=${CLUSTER_NAME} --kubeconfig=${CLUSTER_NAME}.conf
```

Your output should be similar to this example, with the number of DESIRED and READY replicas increased to 5:

```
NODEPOOL                          DESIRED              READY
KUBERNETES VERSION
aws-example-md-0                   5                    5                    v1.25.4

aws-attached-md-0                  5                    5                    v1.25.4
```

7.10.13.3.2 Scaling Down Node Pools

To scale down a node pool, run:

```
dkp scale nodepools ${NODEPOOL_NAME} --replicas=4 --cluster-name=${CLUSTER_NAME}
```

OR for attached clusters:

```
dkp scale nodepools ${ATTACHED_NODEPOOL_NAME} --replicas=4 --cluster-name=$
{ATTACHED_CLUSTER_NAME} --kubeconfig=${CLUSTER_NAME}.conf -n $
{ATTACHED_CLUSTER_WORKSPACE}
```

```
✓ Scaling node pool example to 4 replicas
```

After a few minutes, you can list the node pools using this command:

```
dkp get nodepools --cluster-name=${CLUSTER_NAME} --kubeconfig=${CLUSTER_NAME}.conf
```

Your output should be similar to this example, with the number of DESIRED and READY replicas decreased to 4:

```
NODEPOOL                          DESIRED              READY
KUBERNETES VERSION
example                           4                    4                    v1.25.4
aws-example-md-0                  4                    4                    v1.25.4
```

In a default cluster, the nodes to delete are selected at random. This behavior is controlled by CAPI's delete policy[712]. However, when using the DKP CLI to scale down a node pool, it is also possible to specify the Kubernetes Nodes you want to delete.

To do this, set the flag `--nodes-to-delete` with a list of nodes as below. This adds an annotation `cluster.x-k8s.io/delete-machine=yes` to the matching Machine object that contains `status.NodeRef` with the node names from `--nodes-to-delete`.

```
dkp scale nodepools ${NODEPOOL_NAME} --replicas=3 --nodes-to-delete=<> --cluster-
name=${CLUSTER_NAME}
```

```
✓ Scaling node pool example to 3 replicas
```

7.10.13.3.3 Scaling Node Pools When Using Cluster Autoscaler

If you configured the cluster autoscaler (see page 997) for the `demo-cluster-md-0` node pool, the value of `--replicas` must be within the minimum and maximum bounds.

For example, assuming you have these annotations:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment ${NODEPOOL_NAME}
cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size=2
kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment ${NODEPOOL_NAME}
cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size=6
```

Try to scale the node pool to 7 replicas with the command:

```
dkp scale nodepools ${NODEPOOL_NAME} --replicas=7 -c demo-cluster
```

Which results in an error similar to:

```
✗ Scaling node pool example to 7 replicas
```

712 https://github.com/kubernetes-sigs/cluster-api/blob/v0.4.0/api/v1alpha4/machineset_types.go#L85-L105

```
failed to scale nodepool: scaling MachineDeployment is forbidden: desired replicas 7
 is greater than the configured max size annotation cluster.x-k8s.io/cluster-api-
autoscaler-node-group-max-size: 6
```

Similarly, scaling down to a number of replicas less than the configured `min-size` also returns an error.

7.10.13.4 Delete AWS Node Pools

Delete node pools in a cluster

Deleting a node pool deletes the Kubernetes nodes and the underlying infrastructure. All nodes will be drained prior to deletion and the pods running on those nodes will be rescheduled.

To delete a node pool from a managed cluster, run:

```
dkp delete nodepool ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME}
```

Here, `example` is the node pool to be deleted.

The expected output will be similar to the following example, indicating the node pool is being deleted:

```
✓ Deleting default/example nodepool resources
```

Deleting an invalid node pool results in output similar to this example:

```
dkp delete nodepool ${CLUSTER_NAME}-md-invalid --cluster-name=${CLUSTER_NAME}
```

```
MachineDeployments or MachinePools.infrastructure.cluster.x-k8s.io "no
MachineDeployments or MachinePools found for cluster aws-example" not found
```

7.10.13.5 AWS Cluster Autoscaler

Configure autoscaler for node pools

Cluster Autoscaler[713] provides the ability to automatically scale-up or scale-down the number of worker nodes in a cluster, based on the number of pending pods to be scheduled. Running the Cluster Autoscaler is optional.

Unlike Horizontal-Pod Autoscaler[714], Cluster Autoscaler does not depend on any Metrics server and does not need Prometheus or any other metrics source.

The Cluster Autoscaler looks at the following annotations on a MachineDeployment to determine its scale-up and scale-down ranges:

713 https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/cloudprovider/clusterapi
714 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-fast-is-hpa-when-combined-with-ca

```
cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size
cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size
```

The full list of command line arguments to the Cluster Autoscaler controller is on the Kubernetes public GitHub repository[715].

For more information about how Cluster Autoscaler works, see these documents:

- What is Cluster Autoscaler[716]
- How does scale-up work[717]
- How does scale-down work[718]
- CAPI Provider for Cluster Autoscaler[719]

7.10.13.5.1 Cluster Autoscaler Prerequisites

Before you begin, you must have:

- A Bootstrap Cluster Lifecycle (see page 956).
- Created a new Kubernetes Cluster (see page 958).
- A Self-Managed Cluster (see page 976).

7.10.13.5.2 Run Cluster Autoscaler

The Cluster Autoscaler controller runs on the workload cluster. Upon creation of the workload cluster, this controller does not have all the objects required to function correctly until after a `dkp move` is issued from the bootstrap cluster.

Run the following steps to enable Cluster Autoscaler:

1. Ensure the Cluster Autoscaler controller is up and running (no restarts and no errors in the logs)

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf logs deployments/cluster-autoscaler
   cluster-autoscaler -n kube-system -f
   ```

2. Enable Cluster Autoscaler by setting the min & max ranges

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment $
   {NODEPOOL_NAME} cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size=2
   kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment $
   {NODEPOOL_NAME} cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size=6
   ```

715 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#what-are-the-parameters-to-ca
716 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#what-is-cluster-autoscaler
717 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-does-scale-up-work
718 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-does-scale-down-work
719 https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/cloudprovider/clusterapi

3. The Cluster Autoscaler logs will show that the worker nodes are associated with node-groups and that pending pods are being watched.
4. To demonstrate that it is working properly, create a large deployment which will trigger pending pods (For this example we used AWS m5.2xlarge worker nodes. If you have larger worker-nodes, you should scale up the number of replicas accordingly).

```
cat <<EOF | kubectl --kubeconfig=${CLUSTER_NAME}.conf apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: busybox-deployment
  labels:
    app: busybox
spec:
  replicas: 600
  selector:
    matchLabels:
      app: busybox
  template:
    metadata:
      labels:
        app: busybox
    spec:
      containers:
      - name: busybox
        image: busybox:latest
        command:
          - sleep
          - "3600"
        imagePullPolicy: IfNotPresent
      restartPolicy: Always
EOF
```

5. Cluster Autoscaler will scale up the number of Worker Nodes until there are no pending pods.
6. Scale down the number of replicas for `busybox-deployment`.

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf scale --replicas=30 deployment/
busybox-deployment
```

7. Cluster Autoscaler starts to scale down the number of Worker Nodes after the default timeout of 10 minutes.

7.10.14 AWS Certificate Renewal

During cluster creation, Kubernetes establishes a Public Key Infrastructure (PKI) for generating the TLS certificates needed for securing cluster communication for various components such as `etcd`,

`kubernetes-apiserver` and `kube-proxy`. The certificates created by these components have a default expiration of one year and are renewed when an administrator updates the cluster.

Kubernetes provides a facility to renew all certificates automatically during control plane updates. For administrators who need long-running clusters or clusters that are not upgraded often, `dkp` provides automated certificate renewal, without a cluster upgrade.

7.10.14.1 Prerequisites

- This feature requires Python 3.5 or greater to be installed on all control plane hosts.
- Complete the Bootstrap Cluster (see page 956) topic.

7.10.14.2 Create a Cluster with Automated Certificate Renewal

To enable the automated certificate renewal, create a Konvoy cluster using the `certificate-renew-interval` flag:

```
dkp create cluster aws --certificate-renew-interval=60 --cluster-name=long-running
```

The `certificate-renew-interval` is the number of days after which Kubernetes-managed PKI certificates will be renewed. For example, an `certificate-renew-interval` value of 60 means the certificates will be renewed every 60 days.

7.10.14.3 Technical Details

The following manifests are modified on the control plane hosts, and are located at `/etc/kubernetes/manifests`. Modifications to these files requires SUDO access.

```
kube-controller-manager.yaml
kube-apiserver.yaml
kube-scheduler.yaml
kube-proxy.yaml
```

The following annotation indicates the time each component was reset:

```
metadata:
  annotations:
    konvoy.d2iq.io/restartedAt: $(date +%s)
```

This only occurs when the PKI certificates are older than the interval given at cluster creation time. This is activated by a `systemd` `timer` called `renew-certs.timer` that triggers an associated `systemd` service called `renew-certs.service` that runs on all of the control plane hosts.

7.10.14.4 Debug

To debug the automatic certificate renewal feature, a cluster administrator can look at several different components to see if the certificates were renewed. For example, an administrator might start with a look at the control plane pod definition to check the last reset time. To determine if a scheduler pod was properly reset, run the command:

```
kubectl get pod -n kube-system kube-scheduler-ip-10-0-xx-xx.us-west-2.compute.internal -o yaml
```

The output of the command will be similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    konvoy.d2iq.io/restartedAt: "1626124940.735733"
```

Administrators who want more details on the execution of the `systemd` service can use `ssh` to connect to the control plane hosts, and then use the `systemctl` and `journalctl` commands that follow to help diagnose potential issues.

To check the status of the timers, when they last ran, and when they are scheduled to run next, use the command:

```
systemctl list-timers
```

To check the status of the `renew-certs` service, use the command:

```
systemctl status renew-certs
```

To get the logs of the last run of the service, use the command:

```
journalctl logs -u renew-certs
```

7.10.15 Configure Infrastructure in UI

This page refers to working inside the AWS Console in order to add Infrastructure Provider.

7.10.15.1 Create Infrastructure Provider

To configure an AWS Provider with a User Role in the AWS Console (UI), perform the following steps:

1. From the top menu bar, select your target workspace.
2. Select **Infrastructure Providers** in the **Administration** section of the sidebar menu.
3. Select the **Add Infrastructure Provider** button.
4. Select the **Amazon Web Services (AWS)** option.
5. Ensure **Role** is selected as the **Authentication Method**.
6. Enter a name for your infrastructure provider. Select a name that matches the AWS user.
7. Enter the **Role ARN**.
8. You can add an **External ID** if you share the Role with a 3rd party. External IDs secure your environment from accidentally used roles. Read more about External IDs[720].
9. Select **Save** to save your provider.

7.10.15.2 Fill out the Add Infrastructure Provider Form

To fill out the Add Infrastructure Provider form in the AWS Console using Static Credentials, perform the following steps:

1. In Kommander, select the Workspace associated with the credentials you are adding.
2. Navigate to **Administration > Infrastructure Providers** and click the **Add Infrastructure Provider** button.
3. Select the Amazon Web Services (AWS) option.
4. Ensure **Static** is selected as the Authentication Method.
5. Enter a name for your infrastructure provider for later reference. Consider choosing a name that matches the AWS user.
6. Fill out the access and secret keys using the keys generated above.
7. Select **Save** to save your provider.

7.10.16 Replace an AWS Node

7.10.16.1 Prerequisites

Before you begin, you must:

720 https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create_for-user_externalid.html

- Create a workload cluster (see page 958).
- Make the new cluster self-managed (see page 976).

7.10.16.2 Replace a Worker Node

In certain situations, you may want to delete a worker node and have Cluster API[721] replace it with a newly provisioned machine.

1. Identify the name of the node to delete.

   List the nodes:

   ```
   kubectl --kubeconfig ${CLUSTER_NAME}.conf get nodes
   ```

   The output from this command resembles the following:

   ```
   NAME                                         STATUS   ROLES
   AGE   VERSION
   ip-10-0-100-85.us-west-2.compute.internal    Ready    <none>
   16m   v1.25.4
   ip-10-0-106-183.us-west-2.compute.internal   Ready    control-plane,master
   15m   v1.25.4
   ip-10-0-158-104.us-west-2.compute.internal   Ready    control-plane,master
   17m   v1.25.4
   ip-10-0-203-138.us-west-2.compute.internal   Ready    control-plane,master
   16m   v1.25.4
   ip-10-0-70-169.us-west-2.compute.internal    Ready    <none>
   16m   v1.25.4
   ip-10-0-77-176.us-west-2.compute.internal    Ready    <none>
   16m   v1.25.4
   ip-10-0-96-61.us-west-2.compute.internal     Ready    <none>
   16m   v1.25.4
   ```

2. Export a variable with the node name to use in the next steps:

   This example uses the name `ip-10-0-100-85.us-west-2.compute.internal`.

   ```
   export NAME_NODE_TO_DELETE="<ip-10-0-100-85.us-west-2.compute.internal>"
   ```

3. Delete the Machine resource

721 https://cluster-api.sigs.k8s.io/

```
export NAME_MACHINE_TO_DELETE=$(kubectl --kubeconfig ${CLUSTER_NAME}.conf get
machine -ojsonpath="{.items[?
(@.status.nodeRef.name==\"$NAME_NODE_TO_DELETE\")].metadata.name}")
kubectl --kubeconfig ${CLUSTER_NAME}.conf delete machine
"$NAME_MACHINE_TO_DELETE"
```

```
machine.cluster.x-k8s.io "aws-example-1-md-0-cb9c9bbf7-t894m" deleted
```

The command will not return immediately. It will return once the Machine resource has been deleted.

A few minutes after the Machine resource is deleted, the corresponding Node resource is also deleted.

4. Observe that the Machine resource is being replaced using this command:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf get machinedeployment
```

```
NAME               CLUSTER       REPLICAS   READY   UPDATED   UNAVAILABLE
PHASE       AGE     VERSION
aws-example-md-0   aws-example   4          3       4         1
ScalingUp   7m53s   v1.25.4
```

In this example, there are two replicas, but only 1 is ready. One replica is unavailable, and the `ScalingUp` phase means a new Machine is being created.

5. Identify the replacement Machine using this command:

```
export NAME_NEW_MACHINE=$(kubectl --kubeconfig ${CLUSTER_NAME}.conf get
machines \
    -l=cluster.x-k8s.io/deployment-name=${CLUSTER_NAME}-md-0 \
    -ojsonpath='{.items[?(@.status.phase=="Running")].metadata.name}{"\n"}')
echo "$NAME_NEW_MACHINE"
```

```
aws-example-md-0-cb9c9bbf7-hcl8z aws-example-md-0-cb9c9bbf7-rtdqw aws-example-
md-0-cb9c9bbf7-td29r aws-example-md-0-cb9c9bbf7-w64kg
```

If the output is empty, the new Machine has probably exited the `Provisioning` phase and entered the `Running` phase.

6. Identify the replacement Node using this command:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf get nodes
```

```
NAME                                          STATUS   ROLES
AGE   VERSION
ip-10-0-106-183.us-west-2.compute.internal    Ready    control-plane,master
20m   v1.25.4
ip-10-0-158-104.us-west-2.compute.internal    Ready    control-plane,master
23m   v1.25.4
ip-10-0-203-138.us-west-2.compute.internal    Ready    control-plane,master
22m   v1.25.4
ip-10-0-70-169.us-west-2.compute.internal     Ready    <none>
22m   v1.25.4
ip-10-0-77-176.us-west-2.compute.internal     Ready    <none>
22m   v1.25.4
ip-10-0-86-58.us-west-2.compute.internal      Ready    <none>
57s   v1.25.4
ip-10-0-96-61.us-west-2.compute.internal      Ready    <none>
22m   v1.25.4
```

If the output is empty, the Node resource is not yet available, or does not yet have the expected annotation. Wait a few minutes, then repeat the command.

7.11 EKS Infrastructure

Enterprise Gov Advanced

When installing DKP on your EKS infrastructure, you can choose from multiple configuration types. The steps for creating and accessing your cluster are listed below:

- EKS Introduction (see page 1006)
- Minimal User Permission for EKS Cluster Creation (see page 1007)
- EKS Cluster IAM Permissions and Roles (see page 1010)
- Create an EKS Cluster from the CLI (see page 1015)
- Create an EKS Cluster from the UI (see page 1022)
- Grant Cluster Access (see page 1024)

7.11.1 EKS Introduction

Enterprise Gov Advanced

DKP brings value to EKS customers by providing all components needed for a production-ready Kubernetes environment. DKP 2.5 provides the capability to provision EKS clusters using the DKP UI, In addition to above, DKP 2.5 also provides the ability to upgrade your EKS clusters using the DKP platform, making it possible to manage the complete lifecycle of EKS clusters from a centralized platform.

DKP adds value to Amazon EKS through:

- **Time to Value** in hours/days to get to production, instead of weeks/months, or even failure. Particularly in complex environments like air-gapped, customers tried various options and even after spending millions did not see success, or saw Day 2 later than they could have. We delivered results in hours/days.
- **Less Risk**
    - **Cloud-Native Expertise** eliminates the lack of skills issue. Our industry-leading expertise closes skill gaps on the customer side, avoids costly mistakes, transfers skills, and improves project success rates while shortening timelines.
    - **Simplicity** mitigates operational complexity. We focus on a great user experience and automate the hard parts of cloud-native operations to get customers to Day 2 faster and meet all Day 2 operational challenges. This frees up time on the customer side to build what differentiates them, instead of reinventing the wheel for Kubernetes operations.
    - **Military-Grade Security** alleviates security concerns. The D2iQ Kubernetes Platform can be configured to meet NSA Kubernetes security hardening guidelines. D2iQ Kubernetes Platform and supported add-on components are security scanned and secure out of the box. Encryption of data-at-rest, FIPS compliance, and fully supported air-gapped deployments round out D2iQ offerings.
- **Lower TCO** with operational insights and a simpler platform that curates needed capabilities from Amazon EKS and the open source community that reduces the time and cost of consulting engagements as well as ongoing support costs.
- **Enterprise-grade Kubernetes** - comes with a curated list of Day 2 applications necessary for running Kubernetes in production.
- **One platform for all** - Single platform to manage multiple clusters on any infrastructure cloud, on-premise, and edge.
- **D2IQ GitOps and EKS** - Delivering business value through applications is the primary goal of any Kubernetes cluster. While EKS provides the hosted framework that leads the market, delivering applications to your environment requires a mature and integrated approach. D2IQ DKP provides

workspace and project level constructs to a Kubernetes cluster so that application teams have division of resources, security and cost optimization at the the project and namespace level.
    - Projects deliver applications via the built in GitOps via FluxCD - Just provide a Git repository and DKP does the rest
    - Through integration with Kubecost, DKP monitors utilization of project resources and proves real time reporting for performance and cost optimization
    - Security of projects is defined through DKP integration of customer authentication methods and is reinforced through several layers of application security
- **Cluster Lifecycle Management through CAPI -** Through the use of cluster API, DKP gives customers full lifecycle management of their EKS clusters with the ability to instantiate new EKS clusters through a unified API. This allows administrators to deploy new EKS clusters through code and deliver consistent cluster configurations.
- Time to application value is greatly reduced by minimizing the steps necessary to provision a cluster segment clusters through integrated permissions
- Secure and reliable cluster deployments
- Automatic day 2 operations of EKS clusters (Monitoring, Logging, Central Management, Security, Cost Optimization)
- Day 2 GitOps integration with every EKS cluster

7.11.2 Minimal User Permission for EKS Cluster Creation

The following is a cloudformation stack which adds a policy named `eks-bootstrapper` to manage EKS cluster to the `dkp-bootstrapper-role` created by the cloudformation stack in the Minimal Permissions and Role to Create Cluster (see page 940) section. Consult the Leveraging the Role (see page 944)section for an example of how to use this role and how a system administrator wants to expose using the permissions.

```
AWSTemplateFormatVersion: 2010-09-09
Parameters:
  existingBootstrapperRole:
    Type: CommaDelimitedList
    Description: 'Name of existing minimal role you want to add to add EKS cluster
management permissions to'
    Default: dkp-bootstrapper-role
Resources:
  EKSMinimumPermissions:
    Properties:
      Description: Minimal user policy to manage eks clusters
      ManagedPolicyName: eks-bootstrapper
      PolicyDocument:
        Statement:
          - Action:
              - 'ssm:GetParameter'
            Effect: Allow
            Resource:
              - 'arn:*:ssm:*:*:parameter/aws/service/eks/optimized-ami/*'
```

```
          - Action:
              - 'iam:CreateServiceLinkedRole'
            Condition:
              StringLike:
                'iam:AWSServiceName': eks.amazonaws.com
            Effect: Allow
            Resource:
              - >-
                arn:*:iam::*:role/aws-service-role/eks.amazonaws.com/
AWSServiceRoleForAmazonEKS
          - Action:
              - 'iam:CreateServiceLinkedRole'
            Condition:
              StringLike:
                'iam:AWSServiceName': eks-nodegroup.amazonaws.com
            Effect: Allow
            Resource:
              - >-
                arn:*:iam::*:role/aws-service-role/eks-nodegroup.amazonaws.com/
AWSServiceRoleForAmazonEKSNodegroup
          - Action:
              - 'iam:CreateServiceLinkedRole'
            Condition:
              StringLike:
                'iam:AWSServiceName': eks-fargate.amazonaws.com
            Effect: Allow
            Resource:
              - >-
                arn:aws:iam::*:role/aws-service-role/eks-fargate-pods.amazonaws.com/
AWSServiceRoleForAmazonEKSForFargate
          - Action:
              - 'iam:GetRole'
              - 'iam:ListAttachedRolePolicies'
            Effect: Allow
            Resource:
              - 'arn:*:iam::*:role/*'
          - Action:
              - 'iam:GetPolicy'
            Effect: Allow
            Resource:
              - 'arn:aws:iam::aws:policy/AmazonEKSClusterPolicy'
          - Action:
              - 'eks:DescribeCluster'
              - 'eks:ListClusters'
              - 'eks:CreateCluster'
              - 'eks:TagResource'
              - 'eks:UpdateClusterVersion'
              - 'eks:DeleteCluster'
              - 'eks:UpdateClusterConfig'
              - 'eks:UntagResource'
              - 'eks:UpdateNodegroupVersion'
              - 'eks:DescribeNodegroup'
```

```
            - 'eks:DeleteNodegroup'
            - 'eks:UpdateNodegroupConfig'
            - 'eks:CreateNodegroup'
            - 'eks:AssociateEncryptionConfig'
            - 'eks:ListIdentityProviderConfigs'
            - 'eks:AssociateIdentityProviderConfig'
            - 'eks:DescribeIdentityProviderConfig'
            - 'eks:DisassociateIdentityProviderConfig'
          Effect: Allow
          Resource:
            - 'arn:*:eks:*:*:cluster/*'
            - 'arn:*:eks:*:*:nodegroup/*/*/*'
        - Action:
            - 'ec2:AssociateVpcCidrBlock'
            - 'ec2:DisassociateVpcCidrBlock'
            - 'eks:ListAddons'
            - 'eks:CreateAddon'
            - 'eks:DescribeAddonVersions'
            - 'eks:DescribeAddon'
            - 'eks:DeleteAddon'
            - 'eks:UpdateAddon'
            - 'eks:TagResource'
            - 'eks:DescribeFargateProfile'
            - 'eks:CreateFargateProfile'
            - 'eks:DeleteFargateProfile'
          Effect: Allow
          Resource:
            - '*'
        - Action:
            - 'iam:PassRole'
          Condition:
            StringEquals:
              'iam:PassedToService': eks.amazonaws.com
          Effect: Allow
          Resource:
            - '*'
        - Action:
            - 'kms:CreateGrant'
            - 'kms:DescribeKey'
          Condition:
            'ForAnyValue:StringLike':
              'kms:ResourceAliases': alias/cluster-api-provider-aws-*
          Effect: Allow
          Resource:
            - '*'
      Version: 2012-10-17
    Roles: !Ref existingBootstrapperRole
  Type: 'AWS::IAM::ManagedPolicy'
```

If your role is not named `dkp-bootstrapper-role` change the parameter on line 6 of the file.

To create the resources in the cloudformation stack, copy the contents above into a file. Before executing the following command, replace `MYFILENAME.yaml` and `MYSTACKNAME` with the intended values for your system:

```
aws cloudformation create-stack --template-body=file://MYFILENAME.yaml --stack-
name=MYSTACKNAME --capabilities CAPABILITY_NAMED_IAM
```

7.11.2.1 Next Step:

EKS: Cluster IAM Policies and Roles (see page 355)

7.11.3 EKS Cluster IAM Permissions and Roles

This section guides a DKP user in creating IAM Policies and Instance Profiles that governs who has access to the cluster. The IAM Role is used by the cluster's control plane and worker nodes using the provided AWS CloudFormation Stack specific to EKS. This CloudFormation Stack has additional permissions that are used to delegate access roles for other users.

7.11.3.1 Prerequisites from AWS:

- The user you delegate from your role must have a minimum set of permissions, see User Roles and Instance Profiles (see page 940) page for AWS.
- Create the Cluster IAM Policies (see page 946) in your AWS account.

7.11.3.2 EKS IAM Artifacts

7.11.3.2.1 Policies

- `controllers-eks.cluster-api-provider-aws.sigs.k8s.io` - enumerates the Actions required by the workload cluster to create and modify EKS clusters in the user's AWS Account. It is attached to the existing `control-plane.cluster-api-provider-aws.sigs.k8s.io` role
- `eks-nodes.cluster-api-provider-aws.sigs.k8s.io` - enumerates the Actions required by the EKS workload cluster's worker machines. It is attached to the existing `nodes.cluster-api-provider-aws.sigs.k8s.io`

7.11.3.2.2 Roles

- `eks-controlplane.cluster-api-provider-aws.sigs.k8s.io` - is the Role associated with EKS cluster control planes

> NOTE: `control-plane.cluster-api-provider-aws.sigs.k8s.io` and `nodes.cluster-api-provider-aws.sigs.k8s.io` roles were created by Cluster IAM Policies and Roles (see page 946) in AWS.

Below is a CloudFormation stack[722] that includes IAM policies and roles required to setup EKS Clusters:

```
AWSTemplateFormatVersion: 2010-09-09
Parameters:
  existingControlPlaneRole:
    Type: CommaDelimitedList
    Description: 'Names of existing Control Plane Role you want to add to the newly
created EKS Managed Policy for AWS cluster API controllers'
    Default: control-plane.cluster-api-provider-aws.sigs.k8s.io
  existingNodeRole:
    Type: CommaDelimitedList
    Description: 'ARN of the Nodes Managed Policy to add to the role for nodes'
    Default: nodes.cluster-api-provider-aws.sigs.k8s.io
Resources:
  AWSIAMManagedPolicyControllersEKS:
    Properties:
      Description: For the Kubernetes Cluster API Provider AWS Controllers
      ManagedPolicyName: controllers-eks.cluster-api-provider-aws.sigs.k8s.io
      PolicyDocument:
        Statement:
          - Action:
              - 'ssm:GetParameter'
            Effect: Allow
            Resource:
              - 'arn:*:ssm:*:*:parameter/aws/service/eks/optimized-ami/*'
          - Action:
              - 'iam:CreateServiceLinkedRole'
            Condition:
              StringLike:
                'iam:AWSServiceName': eks.amazonaws.com
            Effect: Allow
            Resource:
              - >-
```

722 https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html

```
                arn:*:iam::*:role/aws-service-role/eks.amazonaws.com/
AWSServiceRoleForAmazonEKS
          - Action:
              - 'iam:CreateServiceLinkedRole'
            Condition:
              StringLike:
                'iam:AWSServiceName': eks-nodegroup.amazonaws.com
            Effect: Allow
            Resource:
              - >-
                arn:*:iam::*:role/aws-service-role/eks-nodegroup.amazonaws.com/
AWSServiceRoleForAmazonEKSNodegroup
          - Action:
              - 'iam:CreateServiceLinkedRole'
            Condition:
              StringLike:
                'iam:AWSServiceName': eks-fargate.amazonaws.com
            Effect: Allow
            Resource:
              - >-
                arn:aws:iam::*:role/aws-service-role/eks-fargate-pods.amazonaws.com/
AWSServiceRoleForAmazonEKSForFargate
          - Action:
              - 'iam:GetRole'
              - 'iam:ListAttachedRolePolicies'
            Effect: Allow
            Resource:
              - 'arn:*:iam::*:role/*'
          - Action:
              - 'iam:GetPolicy'
            Effect: Allow
            Resource:
              - 'arn:aws:iam::aws:policy/AmazonEKSClusterPolicy'
          - Action:
              - 'eks:DescribeCluster'
              - 'eks:ListClusters'
              - 'eks:CreateCluster'
              - 'eks:TagResource'
              - 'eks:UpdateClusterVersion'
              - 'eks:DeleteCluster'
              - 'eks:UpdateClusterConfig'
              - 'eks:UntagResource'
              - 'eks:UpdateNodegroupVersion'
              - 'eks:DescribeNodegroup'
              - 'eks:DeleteNodegroup'
              - 'eks:UpdateNodegroupConfig'
              - 'eks:CreateNodegroup'
              - 'eks:AssociateEncryptionConfig'
              - 'eks:ListIdentityProviderConfigs'
              - 'eks:AssociateIdentityProviderConfig'
              - 'eks:DescribeIdentityProviderConfig'
              - 'eks:DisassociateIdentityProviderConfig'
```

```
            Effect: Allow
            Resource:
              - 'arn:*:eks:*:*:cluster/*'
              - 'arn:*:eks:*:*:nodegroup/*/*/*'
          - Action:
              - 'ec2:AssociateVpcCidrBlock'
              - 'ec2:DisassociateVpcCidrBlock'
              - 'eks:ListAddons'
              - 'eks:CreateAddon'
              - 'eks:DescribeAddonVersions'
              - 'eks:DescribeAddon'
              - 'eks:DeleteAddon'
              - 'eks:UpdateAddon'
              - 'eks:TagResource'
              - 'eks:DescribeFargateProfile'
              - 'eks:CreateFargateProfile'
              - 'eks:DeleteFargateProfile'
            Effect: Allow
            Resource:
              - '*'
          - Action:
              - 'iam:PassRole'
            Condition:
              StringEquals:
                'iam:PassedToService': eks.amazonaws.com
            Effect: Allow
            Resource:
              - '*'
          - Action:
              - 'kms:CreateGrant'
              - 'kms:DescribeKey'
            Condition:
              'ForAnyValue:StringLike':
                'kms:ResourceAliases': alias/cluster-api-provider-aws-*
            Effect: Allow
            Resource:
              - '*'
        Version: 2012-10-17
      Roles: !Ref existingControlPlaneRole
    Type: 'AWS::IAM::ManagedPolicy'
  AWSIAMManagedEKSNodesPolicy:
    Properties:
      Description: Additional Policies to nodes role to work for EKS
      ManagedPolicyName: eks-nodes.cluster-api-provider-aws.sigs.k8s.io
      PolicyDocument:
        Statement:
          - Action:
              - "ec2:AssignPrivateIpAddresses"
              - "ec2:AttachNetworkInterface"
              - "ec2:CreateNetworkInterface"
              - "ec2:DeleteNetworkInterface"
              - "ec2:DescribeInstances"
```

```
            - "ec2:DescribeTags"
            - "ec2:DescribeNetworkInterfaces"
            - "ec2:DescribeInstanceTypes"
            - "ec2:DetachNetworkInterface"
            - "ec2:ModifyNetworkInterfaceAttribute"
            - "ec2:UnassignPrivateIpAddresses"
          Effect: Allow
          Resource:
            - '*'
        - Action:
            - ec2:CreateTags
          Effect: Allow
          Resource:
            - arn:aws:ec2:*:*:network-interface/*
        - Action:
            - "ec2:DescribeInstances"
            - "ec2:DescribeInstanceTypes"
            - "ec2:DescribeRouteTables"
            - "ec2:DescribeSecurityGroups"
            - "ec2:DescribeSubnets"
            - "ec2:DescribeVolumes"
            - "ec2:DescribeVolumesModifications"
            - "ec2:DescribeVpcs"
            - "eks:DescribeCluster"
          Effect: Allow
          Resource:
            - '*'
      Version: 2012-10-17
    Roles: !Ref existingNodeRole
  Type: 'AWS::IAM::ManagedPolicy'
AWSIAMRoleEKSControlPlane:
  Properties:
    AssumeRolePolicyDocument:
      Statement:
        - Action:
            - 'sts:AssumeRole'
          Effect: Allow
          Principal:
            Service:
              - eks.amazonaws.com
      Version: 2012-10-17
    ManagedPolicyArns:
      - 'arn:aws:iam::aws:policy/AmazonEKSClusterPolicy'
    RoleName: eks-controlplane.cluster-api-provider-aws.sigs.k8s.io
  Type: 'AWS::IAM::Role'
```

To create the resources in the CloudFormation stack, copy the contents above into a file and execute the following command after replacing `MYFILENAME.yaml` and `MYSTACKNAME` with the intended values:

```
 aws cloudformation create-stack --template-body=file://MYFILENAME.yaml --stack-
name=MYSTACKNAME --capabilities  CAPABILITY_NAMED_IAM
```

7.11.3.2.3 Add EKS CSI Policy

AWS CloudFormation does not support attaching an existing IAM Policy to an existing IAM Role. Add the necessary IAM policy to your worker instance profile using the `aws` CLI:

```
aws iam attach-role-policy --role-name nodes.cluster-api-provider-aws.sigs.k8s.io --
policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy
```

7.11.4 Create an EKS Cluster from the CLI

Enterprise Gov Advanced

Ensure that the `KUBECONFIG` environment variable is set to the self-managed cluster by running `export KUBECONFIG={SELF_MANAGED_AWS_CLUSTER}.conf`

7.11.4.1 Create a New EKS Kubernetes Cluster

By default, the control-plane Nodes will be created in 3 different Availability Zones. However, the default worker Nodes will reside in a single zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Set the environment variable to the name you assigned this cluster.

   ```
   export CLUSTER_NAME=eks-example
   ```

2. Make sure your AWS credentials are up-to-date. Refresh the credentials command is only necessary if you are using Static Credentials (see page 0) (Access Keys) otherwise, if you are using role-based authentication on a bastion host, proceed to step 3:

```
dkp update bootstrap credentials aws
```

3. Create the cluster:

```
dkp create cluster eks \
  --cluster-name=${CLUSTER_NAME} \
  --additional-tags=owner=$(whoami)
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).
The output displays similar to this:

```
Generating cluster resources
cluster.cluster.x-k8s.io/eks-example created
awsmanagedcontrolplane.controlplane.cluster.x-k8s.io/eks-example-control-plane
 created
machinedeployment.cluster.x-k8s.io/eks-example-md-0 created
awsmachinetemplate.infrastructure.cluster.x-k8s.io/eks-example-md-0 created
eksconfigtemplate.bootstrap.cluster.x-k8s.io/eks-example-md-0 created
clusterresourceset.addons.cluster.x-k8s.io/calico-cni-installation-eks-example
 created
configmap/calico-cni-installation-eks-example created
configmap/tigera-operator-eks-example created
clusterresourceset.addons.cluster.x-k8s.io/cluster-autoscaler-eks-example
 created
configmap/cluster-autoscaler-eks-example created
clusterresourceset.addons.cluster.x-k8s.io/node-feature-discovery-eks-example
 created
configmap/node-feature-discovery-eks-example created
clusterresourceset.addons.cluster.x-k8s.io/nvidia-feature-discovery-eks-example
 created
configmap/nvidia-feature-discovery-eks-example created
```

4. Inspect or edit the cluster objects:

   Use your favorite editor.

   NOTE: Editing the cluster objects requires some understanding of Cluster API. Edits can prevent the cluster from deploying successfully.

The objects are Custom Resources[723] defined by Cluster API components, and they belong in three different categories:

a. **Cluster**
   A *Cluster* object has references to the infrastructure-specific and control plane objects. Because this is an AWS cluster, there is an *AWSCluster* object that describes the infrastructure-specific cluster properties. Here, this means the AWS region, the VPC ID, subnet IDs, and security group rules required by the Pod network implementation.
b. **Control Plane**
   A *AWSManagedControlPlane* object describes the control plane, which is the group of machines that run the Kubernetes control plane components, which include the etcd distributed database, the API server, the core controllers, and the scheduler. The object describes the configuration for these components. The object also has a reference to an infrastructure-specific object that describes the properties of all control plane machines.
c. **Node Pool**
   A Node Pool is a collection of machines with identical properties. For example, a cluster might have one Node Pool with large memory capacity, another Node Pool with GPU support. Each Node Pool is described by three objects: The MachinePool references an object that describes the configuration of Kubernetes components (for example, kubelet) deployed on each node pool machine, and an infrastructure-specific object that describes the properties of all node pool machines. Here, it references a *KubeadmConfigTemplate*, and an *AWSMachineTemplate* object, which describes the instance type, the type of disk used, the size of the disk, among other properties.

For in-depth documentation about the objects, read Concepts[724] in the Cluster API Book.

5. Wait for the cluster control-plane to be ready:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=20m
```

```
cluster.cluster.x-k8s.io/eks-example condition met
```

The `READY` status will become `True` after the cluster control-plane becomes ready in one of the following steps.

6. Once the objects are created on the API server, the Cluster API controllers reconcile them. They create infrastructure and machines. As they progress, they update the Status of each object. Konvoy provides a command to describe the current status of the cluster:

```
dkp describe cluster -c ${CLUSTER_NAME}
```

723 https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/
724 https://cluster-api.sigs.k8s.io/user/concepts.html

```
NAME                                                                READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/eks-example                                                 True
10m
├──ControlPlane - AWSManagedControlPlane/eks-example-control-plane  True
10m
└──Workers
   └──MachineDeployment/eks-example-md-0                            True
26s
     ├──Machine/eks-example-md-0-78fcd7c7b7-66ntt                   True
84s
     ├──Machine/eks-example-md-0-78fcd7c7b7-b9qmc                   True
84s
     ├──Machine/eks-example-md-0-78fcd7c7b7-v5vfq                   True
84s
     └──Machine/eks-example-md-0-78fcd7c7b7-zl6m2                   True
84s
```

7. As they progress, the controllers also create Events. List the Events using this command:

```
kubectl get events | grep ${CLUSTER_NAME}
```

For brevity, the example uses `grep`. It is also possible to use separate commands to get Events for specific objects. For example, `kubectl get events --field-selector involvedObject.kind="AWSCluster"` and `kubectl get events --field-selector involvedObject.kind="AWSMachine"`.

```
46m         Normal   SuccessfulCreateVPC
awsmanagedcontrolplane/eks-example-control-plane   Created new managed VPC
"vpc-05e775702092abf09"
46m         Normal   SuccessfulSetVPCAttributes
awsmanagedcontrolplane/eks-example-control-plane   Set managed VPC attributes
for "vpc-05e775702092abf09"
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-0419dd3f2dfd95ff8"
46m         Normal   SuccessfulModifySubnetAttributes
awsmanagedcontrolplane/eks-example-control-plane   Modified managed Subnet
"subnet-0419dd3f2dfd95ff8" attributes
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-0e724b128e3113e47"
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-06b2b31ea6a8d3962"
```

```
46m         Normal   SuccessfulModifySubnetAttributes
awsmanagedcontrolplane/eks-example-control-plane   Modified managed Subnet
"subnet-06b2b31ea6a8d3962" attributes
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-0626ce238be32bf98"
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-0f53cf59f83177800"
46m         Normal   SuccessfulModifySubnetAttributes
awsmanagedcontrolplane/eks-example-control-plane   Modified managed Subnet
"subnet-0f53cf59f83177800" attributes
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-0878478f6bbf153b2"
46m         Normal   SuccessfulCreateInternetGateway
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Internet
Gateway "igw-09fb52653949d4579"
46m         Normal   SuccessfulAttachInternetGateway
awsmanagedcontrolplane/eks-example-control-plane   Internet Gateway
"igw-09fb52653949d4579" attached to VPC "vpc-05e775702092abf09"
46m         Normal   SuccessfulCreateNATGateway
awsmanagedcontrolplane/eks-example-control-plane   Created new NAT Gateway
"nat-06356aac28079952d"
46m         Normal   SuccessfulCreateNATGateway
awsmanagedcontrolplane/eks-example-control-plane   Created new NAT Gateway
"nat-0429d1cd9d956bf35"
46m         Normal   SuccessfulCreateNATGateway
awsmanagedcontrolplane/eks-example-control-plane   Created new NAT Gateway
"nat-059246bcc9d4e88e7"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-01689c719c484fd3c"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-01689c719c484fd3c" with subnet "subnet-0419dd3f2dfd95ff8"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-065af81b9752eeb69"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-065af81b9752eeb69" with subnet "subnet-0e724b128e3113e47"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-03eeff810a89afc98"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
```

```
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-03eeff810a89afc98" with subnet "subnet-06b2b31ea6a8d3962"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-0fab36f8751fdee73"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-0fab36f8751fdee73" with subnet "subnet-0626ce238be32bf98"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-0e5c9c7bbc3740a0f"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-0e5c9c7bbc3740a0f" with subnet "subnet-0f53cf59f83177800"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-0bf58eb5f73c387af"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-0bf58eb5f73c387af" with subnet "subnet-0878478f6bbf153b2"
46m         Normal   SuccessfulCreateSecurityGroup
awsmanagedcontrolplane/eks-example-control-plane   Created managed
SecurityGroup "sg-0b045c998a120a1b2" for Role "node-eks-additional"
46m         Normal   InitiatedCreateEKSControlPlane
awsmanagedcontrolplane/eks-example-control-plane   Initiated creation of a new
 EKS control plane default_eks-example-control-plane
37m         Normal   SuccessfulCreateEKSControlPlane
awsmanagedcontrolplane/eks-example-control-plane   Created new EKS control
plane default_eks-example-control-plane
37m         Normal   SucessfulCreateKubeconfig
awsmanagedcontrolplane/eks-example-control-plane   Created kubeconfig for
 cluster "eks-example"
37m         Normal   SucessfulCreateUserKubeconfig
awsmanagedcontrolplane/eks-example-control-plane   Created user kubeconfig for
 cluster "eks-example"
27m         Normal   SuccessfulCreate
awsmachine/eks-example-md-0-4t9nc                  Created new node instance
with id "i-0aecc1897c93df740"
26m         Normal   SuccessfulDeleteEncryptedBootstrapDataSecrets
awsmachine/eks-example-md-0-4t9nc                  AWS Secret entries
containing userdata deleted
26m         Normal   SuccessfulSetNodeRef                            machine/
eks-example-md-0-78fcd7c7b7-fn7x9          ip-10-0-88-24.us-west-2.compute.inte
rnal
```

```
26m         Normal   SuccessfulSetNodeRef                            machine/
eks-example-md-0-78fcd7c7b7-g64nv          ip-10-0-110-219.us-west-2.compute.in
ternal
26m         Normal   SuccessfulSetNodeRef                            machine/
eks-example-md-0-78fcd7c7b7-gwc5j          ip-10-0-101-161.us-west-2.compute.in
ternal
26m         Normal   SuccessfulSetNodeRef                            machine/
eks-example-md-0-78fcd7c7b7-j58s4          ip-10-0-127-49.us-west-2.compute.int
ernal
46m         Normal   SuccessfulCreate
machineset/eks-example-md-0-78fcd7c7b7             Created machine "eks-
example-md-0-78fcd7c7b7-fn7x9"
46m         Normal   SuccessfulCreate
machineset/eks-example-md-0-78fcd7c7b7             Created machine "eks-
example-md-0-78fcd7c7b7-g64nv"
46m         Normal   SuccessfulCreate
machineset/eks-example-md-0-78fcd7c7b7             Created machine "eks-
example-md-0-78fcd7c7b7-j58s4"
46m         Normal   SuccessfulCreate
machineset/eks-example-md-0-78fcd7c7b7             Created machine "eks-
example-md-0-78fcd7c7b7-gwc5j"
27m         Normal   SuccessfulCreate
awsmachine/eks-example-md-0-7whkv                  Created new node instance
with id "i-06dfc0466b8f26695"
26m         Normal   SuccessfulDeleteEncryptedBootstrapDataSecrets
awsmachine/eks-example-md-0-7whkv                  AWS Secret entries
containing userdata deleted
27m         Normal   SuccessfulCreate
awsmachine/eks-example-md-0-ttgzv                  Created new node instance
with id "i-0544fce0350fd41fb"
26m         Normal   SuccessfulDeleteEncryptedBootstrapDataSecrets
awsmachine/eks-example-md-0-ttgzv                  AWS Secret entries
containing userdata deleted
27m         Normal   SuccessfulCreate
awsmachine/eks-example-md-0-v2hrf                  Created new node instance
with id "i-0498906edde162e59"
26m         Normal   SuccessfulDeleteEncryptedBootstrapDataSecrets
awsmachine/eks-example-md-0-v2hrf                  AWS Secret entries
containing userdata deleted
46m         Normal   SuccessfulCreate
machinedeployment/eks-example-md-0                 Created MachineSet "eks-
example-md-0-78fcd7c7b7"
```

7.11.4.2 Known Limitations

Be aware of these limitations in the current release of Konvoy.

- The Konvoy version used to create a workload cluster must match the Konvoy version used to delete a workload cluster.
- EKS clusters cannot be Self-managed.
- Konvoy supports deploying one workload cluster.
- Konvoy generates a set of objects for one Node Pool.
- Konvoy does not validate edits to cluster objects.

7.11.5 Create an EKS Cluster from the UI

Enterprise Gov Advanced

DKP UI allows you to quickly and easily provision a Cluster from your browser.

7.11.5.1 Create an AWS Infrastructure Provider

Before you create a Cluster, you first need to create an AWS infrastructure provider to hold your AWS/EKS Credentials:

1. Get the AWS RoleARN (see page 940).

   ```
   aws iam get-role --role-name <role-name> --query 'Role.[RoleName, Arn]' --
   output text
   ```

2. Select **Infrastructure Providers** from the Dashboard menu.
3. Select **Add Infrastructure Provider**.
4. Choose a workspace. If you are already in a workspace, the provider is automatically created in that workspace.
5. Ensure you select **Amazon Web Services**.
6. Add a **Name** for your Infrastructure Provider and include the **Role ARN** from Step 1 above.
7. Select **Save**.

> ⚠ If you choose to, you can use static credentials. However, this method is not as secure so it is not recommended.

7.11.5.2 Provision an EKS Cluster

Follow these steps to provision the EKS cluster:

1. From the top menu bar, select your target workspace.
2. Select **Clusters > Add Cluster**.
   This begins the provisioning workflow.
3. Choose **Create Cluster**.
4. Enter the **Cluster Name**.
5. Select **EKS** from the **Choose Infrastructure** choices.
6. If available, choose a **Kubernetes Version**. Otherwise, the default Kubernetes version (see page 1543) installs.
7. Select a data center **region** or specify a custom **region**.
8. Edit your worker **Node Pools** as necessary. You can choose the **Number of Nodes**, the **Machine Type**, and our **IAM Instance Profile**. For the worker pool, you can also choose a **Worker Availability Zone**.
9. Add any additional **Labels** or **Infrastructure Provider Tags** as necessary.
10. Validate your inputs, and then select **Create**.

You are redirected to the **Clusters** page, where you see your **Cluster** in the **Provisioning** status. Hover over the status to view the details.

After about 15 minutes, your **Cluster** should be in the **Provisioned** status.

See AWS RoleARN[725] for more information from the AWS site.

7.11.5.3 Access EKS Cluster

After the cluster is successfully attached(managed), you can retrieve a custom `kubeconfig` file from the UI using your Kommander administrator credentials.

7.11.5.4 IAM User and Role Access for EKS Clusters

When creating an EKS cluster through the UI, the `kubeconfig` that is returned using the download `kubeconfig` button allows access for 15 minutes. To follow best practices for AWS security, you should configure accessing the EKS cluster using IAM role or user based authentication. This allows account administrators to monitor all actions made.

To enable IAM based cluster access, follow the steps below:

1. Download the `kubeconfig` by selecting the Download `kubeconfig` button on the top section of the UI.
2. Using that `kubeconfig`, edit the config map with a command similar to the one below:

```
kubectl --kubeconfig=MYCLUSTER.conf edit cm -n kube-system aws-auth
```

725 https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_identifiers.html#identifiers-arns

3. Modify the `mapRoles` and `mapUsers` objects according to the permissions as needed. The example below is mapping the `arn:aws:iam::MYAWSACCOUNTID:role/PowerUser` role to `systems:masters` on the Kubernetes cluster:

```
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws:iam::MYAWSACCOUNTID:role/nodes.cluster-api-provider-
aws.sigs.k8s.io
      username: system:node:{{EC2PrivateDNSName}}
    - groups:
      - system:masters
      rolearn: arn:aws:iam::MYAWSACCOUNTID:role/PowerUser
      username: admin
kind: ConfigMap
```

   For more information, consult the Enabling IAM user and role access[726] to you cluster guide and the Kubernetes RBAC guide[727].

4. From your management cluster run the following command to fetch a kubeconfig that uses IAM based permissions by running:

```
dkp get kubeconfig -c ${EKS_CLUSTER_NAME} -n ${KOMMANDER_WORKSPACE_NAMESPACE}
>> ${EKS_CLUSTER_NAME}.conf
```

7.11.6 Grant Cluster Access

Enterprise Gov Advanced

7.11.6.1 How to Grant EKS Cluster Access

You can access your cluster using AWS IAM roles in the dashboard. When you create an EKS cluster, the IAM entity is granted `system:masters` permissions in Kubernetes Role Based Access Control (RBAC) configuration.[728]

726 https://docs.aws.amazon.com/eks/latest/userguide/add-user-role.html
727 https://kubernetes.io/docs/reference/access-authn-authz/rbac/
728 https://kubernetes.io/docs/reference/access-authn-authz/rbac/

More information about the configuration of the EKS control plane can be found on the EKS Cluster IAM Policies and Roles (see page 1010) page.

If the EKS cluster was created as a cluster using a self-managed AWS cluster that uses IAM Instance Profiles, you will need to modify the `IAMAuthenticatorConfig` field in the `AWSManagedControlPlane` API object to allow other IAM entities to access the EKS workload cluster. Follow the steps below:

1. Run the following command with your `KUBECONFIG` configured to select the self-managed cluster previously used to create the workload EKS cluster. Ensure you substitute `${CLUSTER_NAME}` and `${CLUSTER_NAMESPACE}` with their corresponding values for your cluster.

   ```
   kubectl edit awsmanagedcontrolplane ${CLUSTER_NAME}-control-plane -n $
   {CLUSTER_NAMESPACE}
   ```

2. Edit the `IamAuthenticatorConfig` field with the IAM Role to the corresponding Kubernetes Role. In this example, the IAM role `arn:aws:iam::111122223333:role/PowerUser` is granted the cluster role `system:masters`. Note that this example uses example AWS resource ARNs[729], so these values should be substituted for real values in the corresponding AWS account.

   ```
   iamAuthenticatorConfig:
       mapRoles:
         - groups:
           - system:bootstrappers
           - system:nodes
           rolearn: arn:aws:iam::111122223333:role/my-node-role
           username: system:node:{{EC2PrivateDNSName}}
         - groups:
           - system:masters
           rolearn: arn:aws:iam::111122223333:role/PowerUser
           username: admin
   ```

For further instructions on changing or assigning `roles` or `clusterroles` to which you can map IAM users or roles, see Amazon Enabling IAM access to your cluster[730].

7.11.7 Explore EKS Cluster

Enterprise Gov Advanced

729 https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html
730 https://docs.aws.amazon.com/eks/latest/userguide/add-user-role.html

This guide explains how to use the command line to interact with your newly deployed Kubernetes cluster.

Before you start, make sure you have created a workload cluster, as described in Create a New Cluster (see page 1015).

7.11.7.1 Explore the new Kubernetes cluster

Follow these steps:

1. Get a kubeconfig file for the workload cluster:

   When the workload cluster is created, the cluster lifecycle services generate a kubeconfig file for the workload cluster, and write it to a *Secret*. The kubeconfig file is scoped to the cluster administrator.

   Get the kubeconfig from the *Secret*, and write it to a file, using this command:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

2. List the Nodes using this command:

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf get nodes
   ```

   ```
   NAME                                          STATUS   ROLES    AGE   VERSION
   ip-10-0-122-211.us-west-2.compute.internal    Ready    <none>   35m   v1.23.9-
   eks-ba74326
   ip-10-0-127-74.us-west-2.compute.internal     Ready    <none>   35m   v1.23.9-
   eks-ba74326
   ip-10-0-71-155.us-west-2.compute.internal     Ready    <none>   35m   v1.23.9-
   eks-ba74326
   ip-10-0-93-47.us-west-2.compute.internal      Ready    <none>   35m   v1.23.9-
   eks-ba74326
   ```

   NOTE: It may take a few minutes for the Status to move to `Ready` while the Pod network is deployed. The Nodes' Status should change to Ready soon after the `calico-node` DaemonSet Pods are Ready.

3. List the Pods using this command:

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf get --all-namespaces pods
   ```

```
NAMESPACE                NAME                                        READY
STATUS     RESTARTS      AGE
calico-system            calico-kube-controllers-7d6749878f-ccsx9    1/1
Running    0             34m
calico-system            calico-node-2r6l8                           1/1
Running    0             34m
calico-system            calico-node-5pdlb                           1/1
Running    0             34m
calico-system            calico-node-n24hh                           1/1
Running    0             34m
calico-system            calico-node-qrh7p                           1/1
Running    0             34m
calico-system            calico-typha-7bbcb87696-7pk45               1/1
Running    0             34m
calico-system            calico-typha-7bbcb87696-t4c8r               1/1
Running    0             34m
calico-system            csi-node-driver-bz48k                       2/2
Running    0             34m
calico-system            csi-node-driver-k5mmk                       2/2
Running    0             34m
calico-system            csi-node-driver-nvcck                       2/2
Running    0             34m
calico-system            csi-node-driver-x4xnh                       2/2
Running    0             34m
kube-system              aws-node-2xp86                              1/1
Running    0             35m
kube-system              aws-node-5f2kx                              1/1
Running    0             35m
kube-system              aws-node-6lzm7                              1/1
Running    0             35m
kube-system              aws-node-pz8c6                              1/1
Running    0             35m
kube-system              cluster-autoscaler-789d86b489-sz9x2         0/1
Init:0/1   0             36m
kube-system              coredns-57ff979f67-pk5cg                    1/1
Running    0             75m
kube-system              coredns-57ff979f67-sf2j9                    1/1
Running    0             75m
kube-system              ebs-csi-controller-5f6bd5d6dc-bplwm         6/6
Running    0             36m
kube-system              ebs-csi-controller-5f6bd5d6dc-dpjt7         6/6
Running    0             36m
kube-system              ebs-csi-node-7hmm5                          3/3
Running    0             35m
kube-system              ebs-csi-node-l4vfh                          3/3
Running    0             35m
kube-system              ebs-csi-node-mfr7c                          3/3
Running    0             35m
kube-system              ebs-csi-node-v8krq                          3/3
Running    0             35m
```

```
kube-system                 kube-proxy-7fc5x                                  1/1
Running     0               35m
kube-system                 kube-proxy-vvkmk                                  1/1
Running     0               35m
kube-system                 kube-proxy-x6hcc                                  1/1
Running     0               35m
kube-system                 kube-proxy-x8frb                                  1/1
Running     0               35m
kube-system                 snapshot-controller-8ff89f489-4cfxv               1/1
Running     0               36m
kube-system                 snapshot-controller-8ff89f489-78gg8               1/1
Running     0               36m
node-feature-discovery      node-feature-discovery-master-7d5985467-52fcn     1/1
Running     0               36m
node-feature-discovery      node-feature-discovery-worker-88hr7               1/1
Running     0               34m
node-feature-discovery      node-feature-discovery-worker-h95nq               1/1
Running     0               35m
node-feature-discovery      node-feature-discovery-worker-lfghg               1/1
Running     0               34m
node-feature-discovery      node-feature-discovery-worker-prc8p               1/1
Running     0               35m
tigera-operator             tigera-operator-6dcd98c8ff-k97hq                  1/1
Running     0               36m
```

7.11.8 Manage EKS Nodepools

Enterprise Gov Advanced

Ensure that the `KUBECONFIG` environment variable is set to the self-managed cluster by running `export KUBECONFIG={SELF_MANAGED_AWS_CLUSTER}.conf`

Node pools are part of a cluster and managed as a group, and can be used to manage a group of machines using common properties. New default clusters created by Konvoy contain one node pool of worker nodes that have the same configuration.

You can create additional node pools for specialized hardware or other configurations. For example, if you want to tune your memory usage on a cluster where you need maximum memory for some machines and minimal memory on others, you can create a new node pool with those specific resource needs.

NOTE: Konvoy implements node pools using Cluster API MachineDeployments[731].

7.11.8.1 Create a node pool

Availability zones (AZs) are isolated locations within data center regions from which public cloud services originate and operate. Because all the nodes in a node pool are deployed in a single Availability Zone, you may wish to create additional node pools is to ensure your cluster has nodes deployed in multiple Availability Zones.

By default, the first Availability Zone[732] in the region is used for the nodes in the node pool. To create the nodes in a different Availability Zone set the appropriate `--availability-zone`.

To create a new EKS node pool with 3 replicas, run:

```
dkp create nodepool eks ${NODEPOOL_NAME} \
    --cluster-name=${CLUSTER_NAME} \
    --replicas=3
```

```
machinedeployment.cluster.x-k8s.io/example created
awsmachinetemplate.infrastructure.cluster.x-k8s.io/example created
eksconfigtemplate.bootstrap.cluster.x-k8s.io/example created
 ✓ Creating default/example nodepool resources
```

Advanced users can use a combination of the `--dry-run` and `--output=yaml` flags to get a complete set of node pool objects to modify locally or store in version control.

7.11.8.2 Scaling Up Node Pools

To scale up a node pool in a cluster, run:

```
dkp scale nodepools ${NODEPOOL_NAME} --replicas=5 --cluster-name=${CLUSTER_NAME}
```

Your output should be similar to this example, indicating the scaling is in progress:

731 https://cluster-api.sigs.k8s.io/developer/architecture/controllers/machine-deployment.html
732 https://aws.amazon.com/about-aws/global-infrastructure/regions_az/

```
✓ Scaling node pool example to 5 replicas
```

After a few minutes, you can list the node pools to:

```
dkp get nodepools --cluster-name=${CLUSTER_NAME}
```

Your output should be similar to this example, with the number of DESIRED and READY replicas increased to 5:

```
NODEPOOL                        DESIRED             READY
KUBERNETES VERSION
example                         5                   5                   v1.23.6
eks-example-md-0                4                   4                   v1.23.6
```

7.11.8.3 Delete EKS Node Pools

Delete node pools in a cluster

Deleting a node pool deletes the Kubernetes nodes and the underlying infrastructure. All nodes are drained prior to deletion and the pods running on those nodes are rescheduled.

To delete a node pool from a managed cluster, run:

```
dkp delete nodepool ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME}
```

Here, `example` is the node pool to be deleted.

The expected output will be similar to the following example, indicating the node pool is being deleted:

```
✓ Deleting default/example nodepool resources
```

Deleting an invalid node pool results in output similar to this example:

```
dkp delete nodepool ${CLUSTER_NAME}-md-invalid --cluster-name=${CLUSTER_NAME}
```

```
MachineDeployments or MachinePools.infrastructure.cluster.x-k8s.io "no
MachineDeployments or MachinePools found for cluster eks-example" not found
```

7.11.9 Delete EKS Cluster from CLI

Enterprise Gov Advanced

Ensure that the `KUBECONFIG` environment variable is set to the self-managed cluster by running `export KUBECONFIG={SELF_MANAGED_AWS_CLUSTER}.conf`

7.11.9.1 Delete the EKS cluster

Follow these steps:

1. Ensure your AWS credentials are up to date. If you are using user profiles, then refresh the credentials using the command below. Otherwise, proceed to step 2.

   ```
   dkp update bootstrap credentials aws
   ```

2. Delete the Kubernetes cluster and wait a few minutes:

   Before deleting the cluster, dkp deletes all Services of type LoadBalancer on the cluster. Each Service is backed by an AWS Classic ELB. Deleting the Service deletes the ELB that backs it. To skip this step, use the flag `--delete-kubernetes-resources=false`.

   NOTE: Do not skip this step if the VPC is managed by DKP. When DKP deletes the cluster, it deletes the VPC. If the VPC has any EKS Classic ELBs, EKS does not allow the VPC to be deleted, and DKP cannot delete the cluster.

   ```
   dkp delete cluster --cluster-name=${CLUSTER_NAME}
   ```

   ```
    ✓ Deleting Services with type LoadBalancer for Cluster default/eks-example
    ✓ Deleting ClusterResourceSets for Cluster default/eks-example
    ✓ Deleting cluster resources
    ✓ Waiting for cluster to be fully deleted
   Deleted default/eks-example cluster
   ```

7.11.9.2 Next Step:

Once your cluster is built in the Konvoy component of DKP for your infrastructure/environment, you will install the Kommander (see page 1227) component of DKP to see your dashboard and continue customization.

7.11.9.3 Known Limitations

NOTE: Be aware of these limitations in the current release of DKP.

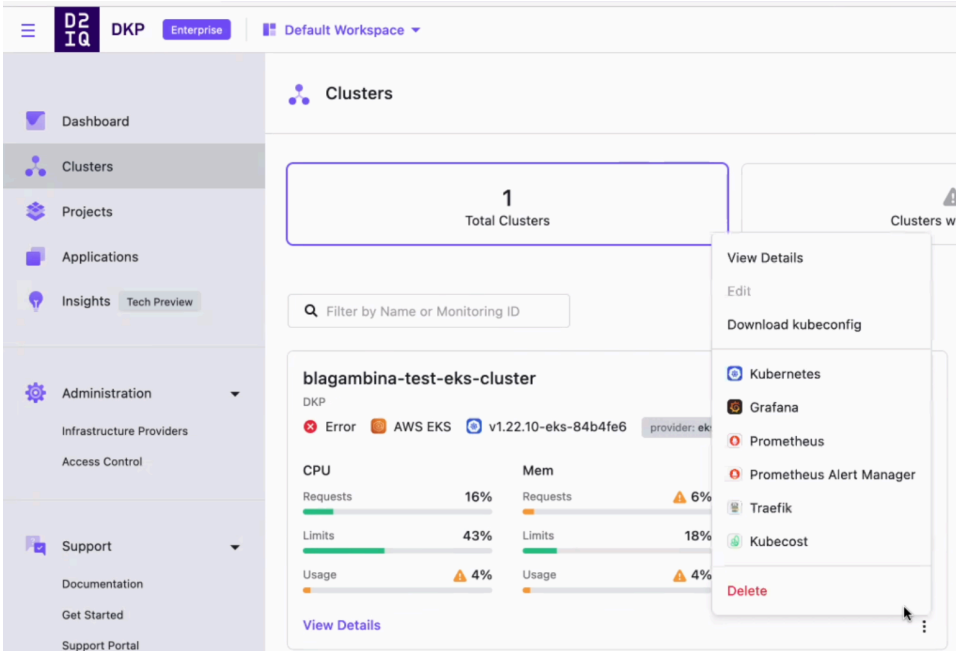
- The DKP version used to create the workload cluster must match the DKP version used to delete the workload cluster.

7.11.10 Delete EKS Cluster from UI

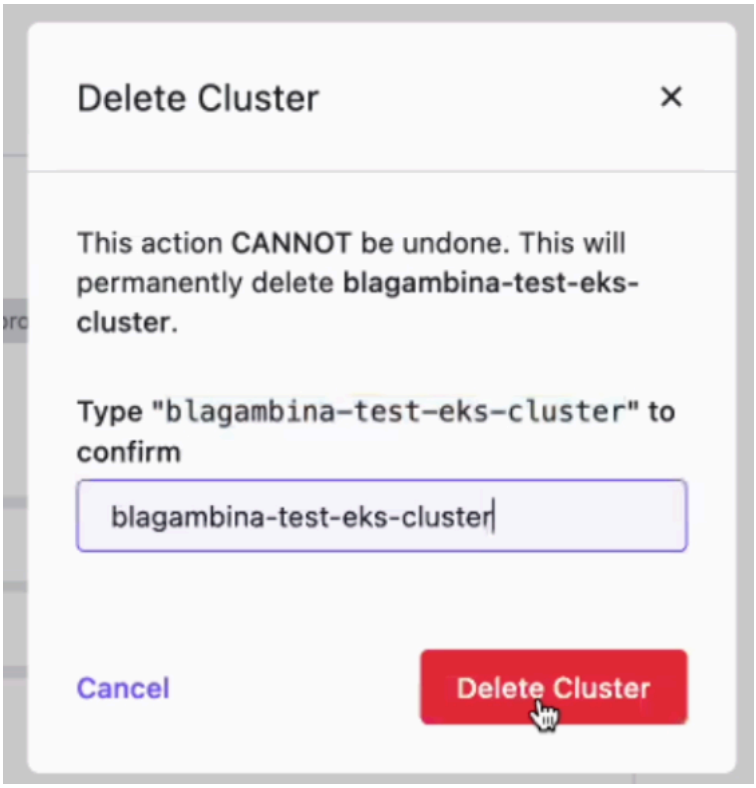
Enterprise Gov Advanced

In order to delete a cluster in the UI, you must first have created a cluster (see page 1022) and have permissions to delete. Otherwise, an administrator privilege could delete a cluster as well.

1. Open the dashboard and select Clusters in the left menu.
2. Select the cluster you wish to delete and click the dotted icon in the bottom right corner.
3. Then select Delete in red.



4. When the next screen appears, copy the name of your cluster and paste it into the empty box.
5. Now execute the deletion using Delete Cluster button.



6. You will see the status as “Deleting” in the top left corner of the cluster you selected for deletion.

This removes the cluster from the DKP UI. For a generic overview of deleting clusters within the UI as well as troubleshooting, see the Disconnect or Delete Clusters (see page 770) instructions.

7.12 Azure Infrastructure

This section describes how you create a DKP cluster in Azure.

7.12.1 Azure Prerequisites

Prepare your machine and environment to run DKP

7.12.1.1 DKP Prerequisites

Before you begin using DKP you must have:

- An x86_64-based Linux or macOS machine.
- Download (see page 71) the `dkp` binary for Linux, or macOS. To check which version of DKP you installed for compatibility reasons, run the `dkp version -h` command (dkp version (see page 1534)).
- A Container engine/runtime installed is required to install DKP:
    - Version Docker®[733] container engine version 18.09.2 or higher installed for Linux or MacOS - On macOS, Docker runs in a virtual machine which needs configured with at least 8 GB of memory.
    - Version 4.0 of Podman[734] or higher for Linux. Host requirements found here: Host Requirements[735]
- kubectl[736] for interacting with the running cluster.
- Azure CLI[737].
- A valid Azure account with credentials configured[738].
- Create a custom Azure image using KIB (see page 1282).

On macOS, Docker runs in a virtual machine. Configure this virtual machine with at least 8GB of memory.

7.12.1.1.1 Control plane nodes

You should have at least three control plane nodes. Each control plane node should have at least:

- 4 cores
- 16 GiB memory
- Approximately 80 GiB of free space for the volume used for `/var/lib/kubelet` and `/var/lib/containerd`.
- Disk usage must be below 85% on the root volume.

DKP on Azure defaults to deploying a `Standard_D4s_v3` virtual machine with an 128 GiB volume for the OS and an 80GiB volume for etcd storage, which meets the above requirements.

733 https://docs.docker.com/get-docker/
734 https://podman.io/getting-started/installation
735 https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
736 https://kubernetes.io/docs/tasks/tools/#kubectl
737 https://docs.microsoft.com/en-us/cli/azure/install-azure-cli
738 https://github.com/kubernetes-sigs/cluster-api-provider-azure/blob/master/docs/book/src/topics/getting-started.md#prerequisites

7.12.1.1.2 Worker nodes

You should have at least four worker nodes. The specific number of worker nodes required for your environment can vary depending on the cluster workload and size of the nodes. Each worker node should have at least:

- 8 cores
- 32 GiB memory
- Around 80 GiB of free space for the volume used for `/var/lib/kubelet` and `/var/lib/containerd`.
- Disk usage must be below 85% on the root volume.

DKP on Azure defaults to deploying a `Standard_D8s_v3` virtual machine with an 80 GiB volume for the OS, which meets the above requirements.

If you use these instructions to create a cluster on Azure using the DKP default settings without any edits to configuration files or additional flags, your cluster is deployed on an Ubuntu 20.04 operating system image (see page 62) with 3 control plane nodes, and 4 worker nodes which match the requirements above.

7.12.1.2 Azure Prerequisites

Before you begin using Konvoy with Azure, you must:

1. Log in to Azure:

```
az login
```

```
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "id": "b1234567-abcd-11a1-a0a0-1234a5678b90",
    "isDefault": true,
    "managedByTenants": [],
    "name": "Mesosphere Developer Subscription",
    "state": "Enabled",
    "tenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "user": {
      "name": "user@azuremesosphere.onmicrosoft.com",
      "type": "user"
    }
  }
]
```

2. Create an Azure Service Principal (SP) by running the following command:

If an SP with the name exists, this command will rotate the password.

```
az ad sp create-for-rbac --role contributor --name "$(whoami)-konvoy" --scopes=/
subscriptions/$(az account show --query id -o tsv)
```

```
{
  "appId": "7654321a-1a23-567b-b789-0987b6543a21",
  "displayName": "azure-cli-2021-03-09-23-17-06",
  "password": "Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C",
  "tenant": "a1234567-b132-1234-1a11-1234a5678b90"
}
```

3. Set the required environment variables:

```
export AZURE_SUBSCRIPTION_ID="<id>"       # b1234567-abcd-11a1-a0a0-1234a5678b90
export AZURE_TENANT_ID="<tenant>"         # a1234567-b132-1234-1a11-1234a5678b90
export AZURE_CLIENT_ID="<appId>"          # 7654321a-1a23-567b-b789-0987b6543a21
export AZURE_CLIENT_SECRET="<password>"   # Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C
```

4. Base64 encode the same environment variables:

```
export AZURE_SUBSCRIPTION_ID_B64="$(echo -n "${AZURE_SUBSCRIPTION_ID}" | base64 | tr
-d '\n')"
export AZURE_TENANT_ID_B64="$(echo -n "${AZURE_TENANT_ID}" | base64 | tr -d '\n')"
export AZURE_CLIENT_ID_B64="$(echo -n "${AZURE_CLIENT_ID}" | base64 | tr -d '\n')"
export AZURE_CLIENT_SECRET_B64="$(echo -n "${AZURE_CLIENT_SECRET}" | base64 | tr -d
'\n')"
```

When you completed, move on to the Bootstrap (see page 1039) section.

7.12.2 Azure using Konvoy Image Builder

This procedure describes how to use the Konvoy Image Builder (see page 1282) (KIB) to create a Cluster API[739] compliant Azure Virtual Machine (VM) Image. The VM Image contains the base operating system you specify and all the necessary Kubernetes components. The Konvoy Image Builder uses variable `overrides` to specify the base image and container images to use in your new Azure VM image.

739 https://cluster-api.sigs.k8s.io/

The default Azure image is not recommended for use in production. We suggest using KIB for Azure to build the image in order to take advantage of enhanced cluster operations. To explore more information on this topic refer to the Azure Infrastructure (see page 1033).

7.12.2.1 Prerequisites

Before you begin, you must:

- Check the DKP 2.5.0 Supported Kubernetes Versions (see page 1543)
- Check the Supported Infrastructure Operating Systems (see page 62)
- Download the Konvoy Image Builder (see page 1282) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
- Create a working `Docker` setup

7.12.2.2 Extract the KIB Bundle

Extract the bundle and `cd` into the extracted `konvoy-image-bundle-$VERSION_$OS` folder. The bundled version of `konvoy-image` contains an embedded `docker` image that contains all the requirements for building.

The `konvoy-image` binary and all supporting folders are also extracted. When extracted, `konvoy-image` bind mounts the current working directory (`${PWD}`) into the container to be used.

7.12.2.3 Configure Azure Prerequisites

If you have already followed the Azure Prerequisites (see page 1033) topic steps, then the environment variables needed by KIB (`[AZURE_CLIENT_SECRET` , `AZURE_CLIENT_ID` , `AZURE_TENANT_ID` , `AZURE_SUBSCRIPTION_ID]`) are set and do not need repeated if you are still working in the same window.

If you have not executed the Azure Prerequisite (see page 1033) steps, they are listed below.

1. Sign in to Azure:

```
az login
```

```
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "id": "b1234567-abcd-11a1-a0a0-1234a5678b90",
    "isDefault": true,
    "managedByTenants": [],
    "name": "Mesosphere Developer Subscription",
    "state": "Enabled",
    "tenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "user": {
      "name": "user@azuremesosphere.onmicrosoft.com",
      "type": "user"
    }
  }
]
```

2. Create an Azure Service Principal (SP) by running the following command:

   If an SP with the name exists, this command will rotate the password.

```
az ad sp create-for-rbac --role contributor --name "$(whoami)-konvoy" --
scopes=/subscriptions/$(az account show --query id -o tsv) --query
"{ client_id: appId, client_secret: password, tenant_id: tenant }"
```

```
{
  "client_id": "7654321a-1a23-567b-b789-0987b6543a21",
  "client_secret": "Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C",
  "tenant_id": "a1234567-b132-1234-1a11-1234a5678b90"
}
```

3. Set the `AZURE_CLIENT_SECRET` environment variable:

```
export AZURE_CLIENT_SECRET="<azure_client_secret>" #
Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C
export AZURE_CLIENT_ID="<client_id>"               # 7654321a-1a23-567b-
b789-0987b6543a21
export AZURE_TENANT_ID="<tenant_id>"               # a1234567-b132-1234-1a11-12
34a5678b90
export AZURE_SUBSCRIPTION_ID="<subscription_id>"   # b1234567-abcd-11a1-
a0a0-1234a5678b90
```

4. Ensure you have an override file (see page 1330) to configure specific attributes[740] of your Azure image.

7.12.2.4 Next Step:

Azure Bootstrap (see page 1039)

7.12.3 Azure Bootstrap

7.12.3.1 Prepare to deploy Kubernetes clusters

To create Kubernetes clusters, Konvoy uses Cluster API[741] (CAPI) controllers. These controllers run on a Kubernetes cluster. To get started, you need a *bootstrap* cluster. By default, Konvoy creates a bootstrap cluster for you in a Docker container using the Kubernetes-in-Docker (KIND[742]) tool.

7.12.3.2 Prerequisites

Before you begin, you must:

- Complete the steps in Prerequisites (see page 1033).
- Ensure the `dkp` binary can be found in your $PATH.

7.12.3.3 Bootstrap Cluster Lifecycle Services

1. If an HTTP proxy is required for the bootstrap cluster, set the local `http_proxy`, `https_proxy`, and `no_proxy` environment variables. They are copied into the bootstrap cluster.
2. Create a bootstrap cluster:

   ```
   dkp create bootstrap --kubeconfig $HOME/.kube/config
   ```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

   The output resembles:

740 https://github.com/mesosphere/konvoy-image-builder/tree/7447074a6d910e71ad2e61fc4a12820d073ae5ae/images/azure
741 https://cluster-api.sigs.k8s.io/
742 https://github.com/kubernetes-sigs/kind

```
✓ Creating a bootstrap cluster
✓ Initializing new CAPI components
```

Konvoy creates a bootstrap cluster using KIND[743] as a library. Konvoy then deploys the following Cluster API[744] providers on the cluster:

- Core Provider[745]
- Azure Infrastructure Provider[746]
- kubeadm Bootstrap Provider[747]
- kubeadm ControlPlane Provider[748]

Konvoy waits until the controller-manager and webhook deployments of these providers are ready. List these deployments using this command:

```
kubectl get --all-namespaces deployments -l=clusterctl.cluster.x-k8s.io
```

```
NAMESPACE                           NAME
READY   UP-TO-DATE   AVAILABLE   AGE
capa-system                         capa-controller-manager
1/1     1            1           69s
capi-kubeadm-bootstrap-system       capi-kubeadm-bootstrap-controller-manager
1/1     1            1           71s
capi-kubeadm-control-plane-system   capi-kubeadm-control-plane-controller-
manager   1/1     1            1           70s
capi-system                         capi-controller-manager
1/1     1            1           73s
cappp-system                        cappp-controller-manager
1/1     1            1           66s
capv-system                         capv-controller-manager
1/1     1            1           65s
capz-system                         capz-controller-manager
1/1     1            1           67s
cert-manager                        cert-manager
1/1     1            1           16m
cert-manager                        cert-manager-cainjector
1/1     1            1           16m
```

743 https://github.com/kubernetes-sigs/kind
744 https://cluster-api.sigs.k8s.io/
745 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/
746 https://github.com/kubernetes-sigs/cluster-api-provider-azure
747 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/bootstrap/kubeadm
748 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/controlplane/kubeadm

```
cert-manager                         cert-manager-webhook
1/1     1            1           16m
```

7.12.3.4 (Optional) Create identity secret for Azure

If your bootstrap cluster resides on a Virtual machine inside Azure, create an identity secret that uses the cappz-controller:

```
export AZURE_CLUSTER_IDENTITY_SECRET_NAME="cluster-identity-secret"
export CLUSTER_IDENTITY_NAME="cluster-identity"
export AZURE_CLUSTER_IDENTITY_SECRET_NAMESPACE="default"

kubectl create secret generic "${AZURE_CLUSTER_IDENTITY_SECRET_NAME}" --from-
literal=clientSecret="${AZURE_CLIENT_SECRET}"
```

7.12.3.5 Next Step:

Create a New Custom Azure Cluster (see page 1041)

7.12.4 Create a New Custom Azure Cluster

7.12.4.1 Prerequisites

- Before you begin, make sure you have created a Bootstrap (see page 1039) cluster.

7.12.4.2 Name your cluster

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<azure-example>
```

7.12.4.3 Tips and Tricks

Below are a few ways to customize your setup which are optional. If you prefer to do a basic setup, skip Tips and Tricks and proceed to Create a New Azure Cluster (see page 1043) section.

Important to remember related to the options below: `--compute-gallery-id` image will be in the format `--compute-gallery-id /subscriptions/<subscription id>/resourceGroups/<resource group name>/providers/Microsoft.Compute/galleries/<gallery name>/images/<image definition name>/versions/<version id>`

- Option 1: To create a cluster name that is unique, use the following command. This creates a unique name every time you run it, so use it with forethought:

```
export CLUSTER_NAME=azure-example-$(LC_CTYPE=C tr -dc 'a-z0-9' </dev/urandom |
fold -w 5 | head -n1)
echo $CLUSTER_NAME
```

```
azure-example-pf4a3
```

- Option 2: To use a custom Azure Image when creating your cluster, you must create that Azure Image using KIB (see page 1295) first.

```
dkp create cluster azure --cluster-name=${CLUSTER_NAME} \
--compute-gallery-id "<Managed Image Shared Image Gallery Id>"
--dry-run \
--output=yaml \
> ${CLUSTER_NAME}.yaml
```

- Option 3: To create individual files with different smaller manifests for ease in editing, you can add the `--output-directory` flag. This will create multiple files in the specified directory which must already exist.

```
dkp create cluster azure --cluster-name=${CLUSTER_NAME} \
--compute-gallery-id "<Managed Image Shared Image Gallery Id>"
--dry-run \
--output=yaml \
--output-directory=<existing-directory>
```

- Option 4: To use a custom DNS on Azure, you need a DNS name in your control. Then create a DKP cluster using the standard method described below with the `--self-managed` flag. Once the resource group has been created, you can create your hosted zone with the command below:

```
az network dns zone create --resource-group "d2iq-professional-services" --name
```

  You no longer need to create a cluster issuer. There are several documents that explain custom DNS (see page 1230) in the Kommander component.
- Option 5: To allow DKP to create a cluster with Marketplace based images such as for Rocky Linux, the following flags are available. If these fields were specified in the override file (see page 1330) during image creation (see page 1295), the flags must be used in cluster creation:

- `--plan-offer`, `--plan-publisher` and `--plan-sku`

```
--plan-offer rockylinux-9
--plan-publisher erockyenterprisesoftwarefoundationinc1653071250513
--plan-sku rockylinux-9
```

If you see a similar error to "Creating a virtual machine from Marketplace image or a custom image sourced from a Marketplace image requires Plan information in the request." when creating a cluster, you must also set the following flags `--plan-offer`, `--plan-publisher`, `--plan-sku`. For example when creating a cluster with Rocky Linux VMs, add the following flags to your `dkp create cluster azure` command:

- `--plan-offer`, `--plan-publisher` and `--plan-sku`

For more information regarding this flag or others, please refer to the CLI for the dkp create cluster (see page 1454) section of the documentation and select your provider.

7.12.4.4 Create a new Azure Kubernetes cluster

Availability zones (AZs) are isolated locations within data center regions from which public cloud services originate and operate. Because all the nodes in a node pool are deployed in a single Availability Zone, you may wish to create additional node pools to ensure your cluster has nodes deployed in multiple Availability Zones.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command. See Microsoft's documentation for more information on Availability Options for Azure VM[749].

749 https://docs.microsoft.com/en-us/azure/virtual-machines/manage-availability#configure-multiple-virtual-machines-in-an-availability-set-for-redundancy

If you are using Azure as a pre-provisoined environment: DKP uses `localvolumeprovisioner` as the default storage provider (see page 103) if creating a pre-provisioned Azure cluster (see page 1106). However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[750] compatible storage that is suitable for production.
You can choose from any of the storage options[751] available for Kubernetes. To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your newly created StorageClass to be the default by following the commands in the Kubernetes documentation called Changing the Default Storage Class[752].
The below cluster create directions instead describes how to create a cluster using Azure as the infrastructure provider provisioning clusters, which uses Azure Disks Container Storage Interface as the default StorageClass.

1. Generate the Kubernetes cluster objects. The following example shows a common configuration. See dkp create cluster azure (see page 1459) reference for the full list of cluster creation options.

```
dkp create cluster azure --cluster-name=${CLUSTER_NAME} \
--dry-run \
--output=yaml \
> ${CLUSTER_NAME}.yaml
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

```
Generating cluster resources
```

Refer to the Tips and Tricks section for more information on how to use optional flags such as the `--output-directory` flag.

2. (Optional) To configure the Control Plane and Worker nodes to use an HTTP proxy:

```
export CONTROL_PLANE_HTTP_PROXY=http://example.org:8080
```

750 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
751 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
752 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

```
export CONTROL_PLANE_HTTPS_PROXY=http://example.org:8080
export
CONTROL_PLANE_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1
,10.96.0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.s
vc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.sv
c.cluster,.svc.cluster.local,169.254.169.254,.cloudapp.azure.com"

export WORKER_HTTP_PROXY=http://example.org:8080
export WORKER_HTTPS_PROXY=http://example.org:8080
export
WORKER_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1,10.96.
0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.svc,kube
rnetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.clust
er,.svc.cluster.local,169.254.169.254,.cloudapp.azure.com"
```

- Replace `example.org,example.com,example.net` with your internal addresses
- `localhost` and `127.0.0.1` addresses should not use the proxy
- `10.96.0.0/12` is the default Kubernetes service subnet
- `192.168.0.0/16` is the default Kubernetes pod subnet
- `kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local` is the internal Kubernetes kube-apiserver service
- `.svc,.svc.cluster,.svc.cluster.local` is the internal Kubernetes services
- `169.254.169.254` is the Azure metadata server
- `.cloudapp.azure.com` is for the worker nodes to allow them to communicate directly to the kube-apiserver load balancer

3. (Optional) Create a Kubernetes cluster with HTTP proxy configured. This step assumes you did not already create a cluster in the previous steps:

```
dkp create cluster azure --cluster-name=${CLUSTER_NAME} \
--control-plane-http-proxy="${CONTROL_PLANE_HTTP_PROXY}" \
--control-plane-https-proxy="${CONTROL_PLANE_HTTPS_PROXY}" \
--control-plane-no-proxy="${CONTROL_PLANE_NO_PROXY}" \
--worker-http-proxy="${WORKER_HTTP_PROXY}" \
--worker-https-proxy="${WORKER_HTTPS_PROXY}" \
--worker-no-proxy="${WORKER_NO_PROXY}" \
--dry-run \
--output=yaml \
> ${CLUSTER_NAME}.yaml
```

4. Inspect or edit the cluster objects:

   NOTE: Familiarize yourself with Cluster API before editing the cluster objects as edits can prevent the cluster from deploying successfully.

The objects are Custom Resources[753] defined by Cluster API components, and they belong in three different categories:

a. Cluster
   A *Cluster* object has references to the infrastructure-specific and control plane objects. Because this is an Azure cluster, there is an *AzureCluster* object that describes the infrastructure-specific cluster properties. Here, this means the Azure region, the VPC ID, subnet IDs, and security group rules required by the Pod network implementation.
b. Control Plane
   A *KubeadmControlPlane* object describes the control plane, which is the group of machines that run the Kubernetes control plane components, which include the etcd distributed database, the API server, the core controllers, and the scheduler. The object describes the configuration for these components. The object also has a reference to an infrastructure-specific object that describes the properties of all control plane machines. Here, it references an *AzureMachineTemplate* object, which describes the instance type, the type of disk used, and the size of the disk, among other properties.
c. Node Pool
   A Node Pool is a collection of machines with identical properties. For example, a cluster might have one Node Pool with large memory capacity, another Node Pool with GPU support. Each Node Pool is described by three objects: The MachinePool references an object that describes the configuration of Kubernetes components (for example, kubelet) deployed on each node pool machine, and an infrastructure-specific object that describes the properties of all node pool machines. Here, it references a *KubeadmConfigTemplate*, and an *AzureMachineTemplate* object, which describes the instance type, the type of disk used, the size of the disk, among other properties.

For in-depth documentation about the objects, read Concepts[754] in the Cluster API Book.

5. Modify Control Plane Audit logs settings using the information contained in the page Configuring the Control Plane (see page 1351).

6. Create the cluster from the objects. A warning will appear in the console if the resource already exists and will require you to remove the resource or update your YAML.

```
kubectl create -f ${CLUSTER_NAME}.yaml
```

NOTE: If you used the `--output-directory` flag in your `dkp create .. --dry-run` step above, create the cluster from the objects you created by specifying the directory:

```
kubectl create -f <existing-directory>/
```

Output will be similar to output below:

753 https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/
754 https://cluster-api.sigs.k8s.io/user/concepts.html

```
cluster.cluster.x-k8s.io/azure-example created
azurecluster.infrastructure.cluster.x-k8s.io/azure-example created
kubeadmcontrolplane.controlplane.cluster.x-k8s.io/azure-example-control-plane
created
azuremachinetemplate.infrastructure.cluster.x-k8s.io/azure-example-control-
plane created
secret/azure-example-etcd-encryption-config created
machinedeployment.cluster.x-k8s.io/azure-example-md-0 created
azuremachinetemplate.infrastructure.cluster.x-k8s.io/azure-example-md-0 created
kubeadmconfigtemplate.bootstrap.cluster.x-k8s.io/azure-example-md-0 created
clusterresourceset.addons.cluster.x-k8s.io/calico-cni-installation-azure-
example created
configmap/calico-cni-installation-azure-example created
configmap/tigera-operator-azure-example created
clusterresourceset.addons.cluster.x-k8s.io/azure-disk-csi-azure-example created
configmap/azure-disk-csi-azure-example created
clusterresourceset.addons.cluster.x-k8s.io/cluster-autoscaler-azure-example
created
configmap/cluster-autoscaler-azure-example created
clusterresourceset.addons.cluster.x-k8s.io/node-feature-discovery-azure-example
created
configmap/node-feature-discovery-azure-example created
clusterresourceset.addons.cluster.x-k8s.io/nvidia-feature-discovery-azure-
example created
configmap/nvidia-feature-discovery-azure-example created
```

7. Wait for the cluster control-plane to be ready:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=20m
```

```
cluster.cluster.x-k8s.io/azure-example condition met
```

8. After the objects are created on the API server, the Cluster API controllers reconcile them. They create infrastructure and machines. As they progress, they update the Status of each object. Konvoy provides a command to describe the current status of the cluster:

```
dkp describe cluster -c ${CLUSTER_NAME}
```

```
NAME                                                          READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/azure-example                                         True
3m4s
├──ClusterInfrastructure - AzureCluster/azure-example          True
8m26s
```

```
├──ControlPlane - KubeadmControlPlane/azure-example-control-plane  True
3m4s
│  ├──Machine/azure-example-control-plane-l8j9r                    True
3m9s
│  ├──Machine/azure-example-control-plane-slprd                    True
7m17s
│  └──Machine/azure-example-control-plane-xhxxg                    True
5m9s
└──Workers
   └──MachineDeployment/azure-example-md-0                         True
4m31s
     ├──Machine/azure-example-md-0-d67567c8b-2674r                 True
5m19s
     ├──Machine/azure-example-md-0-d67567c8b-mbmhk                 True
5m17s
     ├──Machine/azure-example-md-0-d67567c8b-pzg8k                 True
5m17s
     └──Machine/azure-example-md-0-d67567c8b-z8km9                 True
5m17s
```

9. As they progress, the controllers also create Events. List the Events using this command:

```
kubectl get events | grep ${CLUSTER_NAME}
```

For brevity, the example uses `grep`. It is also possible to use separate commands to get Events for specific objects. For example, `kubectl get events --field-selector involvedObject.kind="AzureCluster"` and `kubectl get events --field-selector involvedObject.kind="AzureMachine"`.

```
15m         Normal    AzureClusterObjectNotFound                 azurecluster
AzureCluster object default/azure-example not found
15m         Normal    AzureManagedControlPlaneObjectNotFound
azuremanagedcontrolplane                              AzureManagedControlPlane
object default/azure-example not found
15m         Normal    AzureClusterObjectNotFound                 azurecluster
AzureCluster.infrastructure.cluster.x-k8s.io "azure-example" not found
8m22s       Normal    SuccessfulSetNodeRef                       machine/
azure-example-control-plane-bmc9b          azure-example-control-plane-fdvnm
10m         Normal    Machine controller dependency not yet met   azuremachine/
azure-example-control-plane-fdvnm     Machine Controller has not yet set
OwnerRef
12m         Normal    SuccessfulSetNodeRef                       machine/
azure-example-control-plane-msftd          azure-example-control-plane-z9q45
10m         Normal    SuccessfulSetNodeRef                       machine/
azure-example-control-plane-nrvff          azure-example-control-plane-vmqwx
12m         Normal    Machine controller dependency not yet met   azuremachine/
azure-example-control-plane-vmqwx     Machine Controller has not yet set
OwnerRef
```

```
14m         Normal    Machine controller dependency not yet met   azuremachine/
azure-example-control-plane-z9q45     Machine Controller has not yet set
OwnerRef
14m         Warning   VMIdentityNone
azuremachinetemplate/azure-example-control-plane   You are using Service
Principal authentication for Cloud Provider Azure which is less secure than
Managed Identity. Your Service Principal credentials will be written to a file
on the disk of each VM in order to be accessible by Cloud Provider. To learn
more, see https://capz.sigs.k8s.io/topics/identities-use-cases.html#azure-host-
identity
12m         Warning   ControlPlaneUnhealthy
kubeadmcontrolplane/azure-example-control-plane    Waiting for control plane to
pass preflight checks to continue reconciliation: [machine azure-example-
control-plane-msftd does not have APIServerPodHealthy condition, machine azure-
example-control-plane-msftd does not have ControllerManagerPodHealthy
condition, machine azure-example-control-plane-msftd does not have
SchedulerPodHealthy condition, machine azure-example-control-plane-msftd does
not have EtcdPodHealthy condition, machine azure-example-control-plane-msftd
does not have EtcdMemberHealthy condition]
11m         Warning   ControlPlaneUnhealthy
kubeadmcontrolplane/azure-example-control-plane    Waiting for control plane to
pass preflight checks to continue reconciliation: [machine azure-example-
control-plane-nrvff does not have APIServerPodHealthy condition, machine azure-
example-control-plane-nrvff does not have ControllerManagerPodHealthy
condition, machine azure-example-control-plane-nrvff does not have
SchedulerPodHealthy condition, machine azure-example-control-plane-nrvff does
not have EtcdPodHealthy condition, machine azure-example-control-plane-nrvff
does not have EtcdMemberHealthy condition]
9m52s       Normal    SuccessfulSetNodeRef                          machine/
azure-example-md-0-84bd8b5f5b-b8cnq        azure-example-md-0-bsc82
9m53s       Normal    SuccessfulSetNodeRef                          machine/
azure-example-md-0-84bd8b5f5b-j8ldg        azure-example-md-0-mjcbn
9m52s       Normal    SuccessfulSetNodeRef                          machine/
azure-example-md-0-84bd8b5f5b-lx89f        azure-example-md-0-pmq8f
10m         Normal    SuccessfulSetNodeRef                          machine/
azure-example-md-0-84bd8b5f5b-pcv7q        azure-example-md-0-vzprf
15m         Normal    SuccessfulCreate                              machineset/
azure-example-md-0-84bd8b5f5b           Created machine "azure-example-
md-0-84bd8b5f5b-j8ldg"
15m         Normal    SuccessfulCreate                              machineset/
azure-example-md-0-84bd8b5f5b           Created machine "azure-example-
md-0-84bd8b5f5b-lx89f"
15m         Normal    SuccessfulCreate                              machineset/
azure-example-md-0-84bd8b5f5b           Created machine "azure-example-
md-0-84bd8b5f5b-pcv7q"
15m         Normal    SuccessfulCreate                              machineset/
azure-example-md-0-84bd8b5f5b           Created machine "azure-example-
md-0-84bd8b5f5b-b8cnq"
15m         Normal    Machine controller dependency not yet met   azuremachine/
azure-example-md-0-bsc82              Machine Controller has not yet set
OwnerRef
```

```
15m         Normal    Machine controller dependency not yet met   azuremachine/
azure-example-md-0-mjcbn              Machine Controller has not yet set
OwnerRef
15m         Normal    Machine controller dependency not yet met   azuremachine/
azure-example-md-0-pmq8f              Machine Controller has not yet set
OwnerRef
```

If changing the Calico encapsulation (see page 1110), D2iQ recommends changing it after cluster creation, but before production.

7.12.4.5 Known Limitations

Be aware of these limitations in the current release of Konvoy.

The Konvoy version used to create a bootstrap cluster must match the Konvoy version used to create a workload cluster.

- Konvoy supports deploying one workload cluster.
- Konvoy generates a set of objects for one Node Pool.
- Konvoy does not validate edits to cluster objects.

Next, you can Explore the New Cluster (see page 1050) or Make it Self-managed (see page 1054).

7.12.5 Explore new Azure Cluster

7.12.5.1 Learn to interact with your Kubernetes cluster

This guide explains how to use the command line to interact with your newly deployed Kubernetes cluster.

Before you start, make sure you have created a workload cluster, as described in Create a New Cluster (see page 1041).

7.12.5.2 Explore the new Kubernetes cluster

1. Get a kubeconfig file for the workload cluster:

   When the workload cluster is created, the cluster lifecycle services generate a kubeconfig file for the workload cluster, and write it to a *Secret*. The kubeconfig file is scoped to the cluster administrator.

Get the kubeconfig from the *Secret*, and write it to a file, using this command:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

2. List the Nodes using this command:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf get nodes
```

```
NAME                                STATUS   ROLES                  AGE
VERSION
azure-example-control-plane-7ffnl   Ready    control-plane,master   6m18s
v1.25.4
azure-example-control-plane-l4bv8   Ready    control-plane,master   14m
v1.25.4
azure-example-control-plane-n4g4l   Ready    control-plane,master   18m
v1.25.4
azure-example-md-0-mpctb            Ready    <none>                 15m     v1.2
5.4
azure-example-md-0-qglp9            Ready    <none>                 15m     v1.2
5.4
azure-example-md-0-sgrd6            Ready    <none>                 16m     v1.2
5.4
azure-example-md-0-wzbkl            Ready    <none>                 16m     v1.2
5.4
```

NOTE: It may take a few minutes for the Status to move to `Ready` while the Pod network is deployed. The Nodes' Status should change to Ready soon after the `calico-node` DaemonSet Pods are Ready.

3. List the Pods using this command:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf get --all-namespaces pods
```

```
NAMESPACE                           NAME
READY   STATUS    RESTARTS      AGE
calico-system                       calico-kube-controllers-57fbd7bd59-v4tss
1/1     Running   0             19m
calico-system                       calico-node-59llv
1/1     Running   0             17m
calico-system                       calico-node-7t7wj
1/1     Running   0             16m
calico-system                       calico-node-pf8q8
1/1     Running   0             17m
calico-system                       calico-node-sh2b7
1/1     Running   0             8m17s
calico-system                       calico-node-tmxl5
1/1     Running   0             19m
calico-system                       calico-node-vt5fh
1/1     Running   0             18m
calico-system                       calico-node-whfs8
1/1     Running   0             18m
calico-system                       calico-typha-797c9666d5-5w99r
1/1     Running   0             19m
calico-system                       calico-typha-797c9666d5-hj6mj
1/1     Running   0             18m
calico-system                       calico-typha-797c9666d5-s7rc6
1/1     Running   0             17m
capa-system                         capa-controller-manager-74fffb5676-ch6xd
1/1     Running   0             11m
capi-kubeadm-bootstrap-system       capi-kubeadm-bootstrap-controller-
manager-867759cc67-vg4lh       1/1     Running   0             15m
capi-kubeadm-control-plane-system   capi-kubeadm-control-plane-controller-
manager-5df55579c4-pc8x9   1/1     Running   1 (11m ago)   15m
capi-system                         capi-controller-manager-79cc58bf5f-xsp9t
1/1     Running   0             15m
cappp-system                        cappp-controller-manager-85b5c77497-8ss8r
1/1     Running   0             14m
capv-system                         capv-controller-manager-7bf4d8b66-6x2mx
1/1     Running   0             14m
capz-system                         capz-controller-manager-5d4c6468bf-wfhcc
1/1     Running   0             14m
capz-system                         capz-nmi-2cbrg
1/1     Running   0             14m
capz-system                         capz-nmi-8dllm
1/1     Running   0             14m
capz-system                         capz-nmi-95dfk
1/1     Running   0             14m
capz-system                         capz-nmi-rtnd4
1/1     Running   0             14m
cert-manager                        cert-manager-848f547974-gjc5p
1/1     Running   1 (10m ago)   15m
cert-manager                        cert-manager-cainjector-54f4cc6b5-rnh4f
1/1     Running   0             15m
```

```
cert-manager                        cert-manager-webhook-7c9588c76-rn2sd
1/1     Running   0             15m
kube-system                         cluster-autoscaler-68c759fbf6-6vg5r
1/1     Running   1 (11m ago)   20m
kube-system                         coredns-78fcd69978-6gx44
1/1     Running   0             20m
kube-system                         coredns-78fcd69978-gr5q7
1/1     Running   0             20m
kube-system                         csi-azuredisk-controller-c8fb44c8b-jhmfz
6/6     Running   5 (11m ago)   20m
kube-system                         csi-azuredisk-controller-c8fb44c8b-lpbbs
6/6     Running   0             20m
kube-system                         csi-azuredisk-node-2g7vw
3/3     Running   0             8m17s
kube-system                         csi-azuredisk-node-6rdqc
3/3     Running   0             18m
kube-system                         csi-azuredisk-node-99c6q
3/3     Running   0             17m
kube-system                         csi-azuredisk-node-9b4ms
3/3     Running   0             17m
kube-system                         csi-azuredisk-node-mz5pr
3/3     Running   0             18m
kube-system                         csi-azuredisk-node-r2t99
3/3     Running   0             16m
kube-system                         csi-azuredisk-node-t7gfs
3/3     Running   0             20m
kube-system                         etcd-azure-example-control-plane-7ffnl
1/1     Running   0             8m15s
kube-system                         etcd-azure-example-control-plane-l4bv8
1/1     Running   0             16m
kube-system                         etcd-azure-example-control-plane-n4g4l
1/1     Running   0             19m
kube-system                         kube-apiserver-azure-example-control-plane-7ffnl
1/1     Running   0             8m16s
kube-system                         kube-apiserver-azure-example-control-plane-l4bv8
1/1     Running   0             16m
kube-system                         kube-apiserver-azure-example-control-plane-n4g4l
1/1     Running   0             19m
kube-system                         kube-controller-manager-azure-example-control-
plane-7ffnl         1/1     Running   0             8m17s
kube-system                         kube-controller-manager-azure-example-control-
plane-l4bv8         1/1     Running   0             16m
kube-system                         kube-controller-manager-azure-example-control-
plane-n4g4l         1/1     Running   1 (17m ago)   19m
kube-system                         kube-proxy-82zdl
1/1     Running   0             8m17s
kube-system                         kube-proxy-fd9f9
1/1     Running   0             18m
kube-system                         kube-proxy-l6lgc
1/1     Running   0             17m
kube-system                         kube-proxy-lzswh
1/1     Running   0             16m
```

```
kube-system                         kube-proxy-ndfmt
1/1     Running   0             20m
kube-system                         kube-proxy-nxlp9
1/1     Running   0             18m
kube-system                         kube-proxy-v9sxp
1/1     Running   0             17m
kube-system                         kube-scheduler-azure-example-control-plane-7ffnl
1/1     Running   0             8m16s
kube-system                         kube-scheduler-azure-example-control-plane-l4bv8
1/1     Running   0             16m
kube-system                         kube-scheduler-azure-example-control-plane-n4g4l
1/1     Running   1 (17m ago)   19m
node-feature-discovery              node-feature-discovery-master-84c67dcbb6-d2gm7
1/1     Running   0             20m
node-feature-discovery              node-feature-discovery-worker-drgf6
1/1     Running   0             17m
node-feature-discovery              node-feature-discovery-worker-hcz6k
1/1     Running   0             17m
node-feature-discovery              node-feature-discovery-worker-pgbcd
1/1     Running   0             16m
node-feature-discovery              node-feature-discovery-worker-vhj96
1/1     Running   0             16m
tigera-operator                     tigera-operator-d499f5c8f-jnj8b
1/1     Running   1 (18m ago)   19m
```

7.12.6 Azure Make new Cluster Self-Managed

Konvoy deploys all cluster lifecycle services to a bootstrap cluster, which then deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster, which makes the workload cluster self-managed. This section describes how to make a workload cluster self-managed.

This page contains instructions on how to make your cluster self-managed. This is necessary if there is only one cluster in your environment, or if this cluster should become the Management cluster in a multi-cluster environment.

If you already have a self-managed or Management cluster in your environment, skip this page.

7.12.6.1 Make the new Kubernetes cluster manage itself

1. Deploy cluster lifecycle services on the workload cluster:

```
dkp create capi-components --kubeconfig ${CLUSTER_NAME}.conf
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

The output resembles:

```
✓ Initializing new CAPI components
```

2. Move the Cluster API objects from the bootstrap to the workload cluster:

   The cluster lifecycle services on the workload cluster are ready, but the workload cluster configuration is on the bootstrap cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the bootstrap to the workload cluster. This process is also called a Pivot[755].

   ```
   dkp move capi-resources --to-kubeconfig ${CLUSTER_NAME}.conf
   ```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

   The output resembles:

   ```
   ✓ Moving cluster resources
   You can now view resources in the moved cluster by using the --kubeconfig flag
   with kubectl. For example: kubectl --kubeconfig=azure-example.conf get nodes
   ```

To ensure only one set of cluster lifecycle services manages the workload cluster, Konvoy first pauses reconciliation of the objects on the bootstrap cluster, then creates the objects on the workload cluster. As Konvoy copies the objects, the cluster lifecycle services on the workload cluster reconcile the objects. The workload cluster becomes self-managed after Konvoy creates all the objects. If it fails, the `move` command can be safely retried.

755 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

3. Wait for the cluster control-plane to be ready:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf wait --for=condition=ControlPlaneReady
"clusters/${CLUSTER_NAME}" --timeout=20m
```

```
cluster.cluster.x-k8s.io/azure-example condition met
```

4. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

> After moving the cluster lifecycle services to the workload cluster, remember to use Konvoy with the workload cluster kubeconfig.

```
dkp describe cluster --kubeconfig ${CLUSTER_NAME}.conf -c ${CLUSTER_NAME}
```

```
NAME                                                                READY  SEVERITY
REASON  SINCE  MESSAGE
Cluster/azure-example                                               True
55s
├──ClusterInfrastructure - AzureCluster/azure-example               True
67s
├──ControlPlane - KubeadmControlPlane/azure-example-control-plane   True
55s
│  ├──Machine/azure-example-control-plane-67f47                     True
58s
│  ├──Machine/azure-example-control-plane-7pllh                     True
65s
│  └──Machine/azure-example-control-plane-jtfgv                     True
65s
└──Workers
   └──MachineDeployment/azure-example-md-0                          True
67s
     ├──Machine/azure-example-md-0-f9cb9c79b-6nsb9                  True
59s
     ├──Machine/azure-example-md-0-f9cb9c79b-jxwl6                  True
58s
     ├──Machine/azure-example-md-0-f9cb9c79b-ktg7z                  True
59s
     └──Machine/azure-example-md-0-f9cb9c79b-nxcm2                  True
66s
```

5. Remove the bootstrap cluster, as the workload cluster is now self-managed:

```
dkp delete bootstrap --kubeconfig $HOME/.kube/config
```

```
✓ Deleting bootstrap cluster
```

7.12.6.2 Known Limitations

- Konvoy supports moving only one set of cluster objects from the bootstrap cluster to the workload cluster, or vice-versa.
- Konvoy only supports moving all namespaces in the cluster; Konvoy does not support migration of individual namespaces.

7.12.7 Azure Certificate Renewal

7.12.7.1 Configure Automated Renewal for Managed Kubernetes PKI Certficates

During cluster creation, Kubernetes establishes a Public Key Infrastructure (PKI) for generating the TLS certificates needed for securing cluster communication for various components such as `etcd`, `kubernetes-apiserver` and `kube-proxy`. The certificates created by these components have a default expiration of one year and are renewed when an administrator updates the cluster.

Kubernetes provides a facility to renew all certificates automatically during control plane updates. For administrators who need long-running clusters or clusters that are not upgraded often, `dkp` provides automated certificate renewal, without a cluster upgrade.

7.12.7.2 Requirements

This feature requires Python 3.5 or greater to be installed on all control plane hosts.

7.12.7.3 Prerequisites

Prerequisite:

- Complete the bootstrap Cluster Lifecycle (see page 1039) topic.

7.12.7.3.1 Create a cluster with automated certificate renewal

To enable the automated certificate renewal, create a Konvoy cluster using the `certificate-renew-interval` flag:

```
dkp create cluster azure --certificate-renew-interval=60 --cluster-name=long-running
```

The `certificate-renew-interval` is the number of days after which Kubernetes-managed PKI certificates will be renewed. For example, a `certificate-renew-interval` value of 60 means the certificates will be renewed every 60 days.

7.12.7.3.2 Technical details

The following manifests are modified on the control plane hosts, and are located at `/etc/kubernetes/manifests`. Modifications to these files requires SUDO access.

```
kube-controller-manager.yaml
kube-apiserver.yaml
kube-scheduler.yaml
kube-proxy.yaml
```

The following annotation indicates the time each component was reset:

```
metadata:
  annotations:
    konvoy.d2iq.io/restartedAt: $(date +%s)
```

This only occurs when the PKI certificates are older than the interval given at cluster creation time. This is activated by a `systemd` `timer` called `renew-certs.timer` that triggers an associated `systemd` service called `renew-certs.service` that runs on all of the control plane hosts.

7.12.7.3.3 Debugging

To debug the automatic certificate renewal feature, a cluster administrator can look at several different components to see if the certificates were renewed. For example, an administrator might start with a look at the control plane pod definition to check the last reset time. To determine if a scheduler pod was properly reset, run the command:

```
kubectl get pod -n kube-system kube-scheduler-ip-10-0-xx-xx.us-west-2.compute.internal -o yaml
```

The output of the command will be similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
```

```
    konvoy.d2iq.io/restartedAt: "1626124940.735733"
```

Administrators who want more details on the execution of the `systemd` service can use `ssh` to connect to the control plane hosts, and then use the `systemctl` and `journalctl` commands that follow to help diagnose potential issues.

To check the status of the timers, when they last ran, and when they are scheduled to run next, use the command:

```
systemctl list-timers
```

To check the status of the `renew-certs` service, use the command:

```
systemctl status renew-certs
```

To get the logs of the last run of the service, use the command:

```
journalctl logs -u renew-certs
```

7.12.8 Azure Replace a Node

7.12.8.1 Replace a worker node

7.12.8.2 Prerequisites

Before you begin, you must:

- Create a workload cluster (see page 1041).
- Make the new cluster self-managed (see page 1054).

7.12.8.3 Replace a worker node

In certain situations, you may want to delete a worker node and have Cluster API[756] replace it with a newly provisioned machine.

1. Identify the name of the node to delete.

   List the nodes:

   ```
   kubectl --kubeconfig ${CLUSTER_NAME}.conf get nodes
   ```

756 https://cluster-api.sigs.k8s.io/

The output from this command resembles the following:

```
NAME                                 STATUS   ROLES                  AGE
VERSION
azure-example-control-plane-ckwm4    Ready    control-plane,master   35m
v1.25.4
azure-example-control-plane-d4fdf    Ready    control-plane,master   31m
v1.25.4
azure-example-control-plane-qrvm9    Ready    control-plane,master   33m
v1.25.4
azure-example-md-0-4w7gq             Ready    <none>                 33m   v1.25
.4
azure-example-md-0-6gb9k             Ready    <none>                 33m   v1.25
.4
azure-example-md-0-p2n8c             Ready    <none>                 11m   v1.25
.4
azure-example-md-0-s5zbh             Ready    <none>                 33m   v1.25
.4
```

2. Export a variable with the node name to use in the next steps:

   This example uses the name `azure-example-control-plane-ckwm4`.

   ```
   export NAME_NODE_TO_DELETE="<azure-example-control-plane-ckwm4>"
   ```

3. Delete the Machine resource

   ```
   export NAME_MACHINE_TO_DELETE=$(kubectl --kubeconfig ${CLUSTER_NAME}.conf get
   machine -ojsonpath="{.items[?
   (@.status.nodeRef.name==\"$NAME_NODE_TO_DELETE\")].metadata.name}")
   kubectl --kubeconfig ${CLUSTER_NAME}.conf delete machine
   "$NAME_MACHINE_TO_DELETE"
   ```

   ```
   machine.cluster.x-k8s.io "azure-example-control-plane-slprd" deleted
   ```

   The command will not return immediately. It will return once the Machine resource has been deleted.

   A few minutes after the Machine resource is deleted, the corresponding Node resource is also deleted.

4. Observe that the Machine resource is being replaced using this command:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf get machinedeployment
```

```
NAME                  CLUSTER         REPLICAS   READY   UPDATED   UNAVAILABLE
PHASE       AGE     VERSION
azure-example-md-0   azure-example   4          3       4         1
ScalingUp   7m30s   v1.25.4
long-running-md-0    long-running    4          4       4         0
Running     7m28s   v1.25.4
```

   In this example, there are two replicas, but only 1 is ready. One replica is unavailable, and the `ScalingUp` phase means a new Machine is being created.

5. Identify the replacement Machine using this command:

```
export NAME_NEW_MACHINE=$(kubectl --kubeconfig ${CLUSTER_NAME}.conf get
machines \
    -l=cluster.x-k8s.io/deployment-name=${CLUSTER_NAME}-md-0 \
    -ojsonpath='{.items[?(@.status.phase=="Running")].metadata.name}{"\n"}')
echo $NAME_NEW_MACHINE
```

```
azure-example-md-0-d67567c8b-2674r azure-example-md-0-d67567c8b-n276j azure-
example-md-0-d67567c8b-pzg8k azure-example-md-0-d67567c8b-z8km9
```

   If the output is empty, the new Machine has probably exited the `Provisioning` phase and entered the `Running` phase.

6. Identify the replacement Node using this command:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf get nodes
```

```
NAME                                STATUS   ROLES                  AGE
VERSION
```

```
azure-example-control-plane-d4fdf   Ready    control-plane,master   43m
v1.25.4
azure-example-control-plane-qrvm9   Ready    control-plane,master   45m
v1.25.4
azure-example-control-plane-tz56m   Ready    control-plane,master   8m22s
v1.25.4
azure-example-md-0-4w7gq            Ready    <none>                 45m     v1.2
5.4
azure-example-md-0-6gb9k            Ready    <none>                 45m     v1.2
5.4
azure-example-md-0-p2n8c            Ready    <none>                 22m     v1.2
5.4
azure-example-md-0-s5zbh            Ready    <none>                 45m     v1.2
5.4
```

If the output is empty, the Node resource is not yet available, or does not yet have the expected annotation. Wait a few minutes, then repeat the command.

7.12.9 Azure Delete Cluster

7.12.9.1 Delete the Kubernetes cluster and clean up your environment

7.12.9.2 Prepare to Delete a Workload Cluster

NOTE: A self-managed workload cluster cannot delete itself. If your workload cluster is self-managed, you must create a bootstrap cluster and move the cluster lifecycle services to the bootstrap cluster before deleting the workload cluster.

If you did not make your workload cluster self-managed, as described in Make New Cluster Self-Managed (see page 1054), proceed to Delete the workload cluster (see page 1065) section below.

1. Create a bootstrap cluster:

   The bootstrap cluster will host the Cluster API controllers that reconcile the cluster objects marked for deletion:

NOTE: To avoid using the wrong kubeconfig, the following steps use explicit kubeconfig paths and contexts.

```
dkp create bootstrap --kubeconfig $HOME/.kube/config
```

```
✓ Creating a bootstrap cluster
✓ Initializing new CAPI components
```

2. Move the Cluster API objects from the workload to the bootstrap cluster: The cluster lifecycle services on the bootstrap cluster are ready, but the workload cluster configuration is on the workload cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the workload to the bootstrap cluster. This process is also called a Pivot[757].

```
dkp move capi-resources \
    --from-kubeconfig ${CLUSTER_NAME}.conf \
    --from-context ${CLUSTER_NAME}-admin@${CLUSTER_NAME} \
    --to-kubeconfig $HOME/.kube/config \
    --to-context kind-konvoy-capi-bootstrapper
```

```
✓ Moving cluster resources
```

3. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

```
dkp describe cluster --kubeconfig $HOME/.kube/config -c ${CLUSTER_NAME}
```

```
NAME                                                        READY  SEVERITY
REASON  SINCE  MESSAGE
```

757 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

```
Cluster/azure-example                                                  True
15s
├──ClusterInfrastructure - AzureCluster/azure-example                   True
29s
├──ControlPlane - KubeadmControlPlane/azure-example-control-plane       True
15s
│  ├──Machine/azure-example-control-plane-gvj5d                         True
22s
│  ├──Machine/azure-example-control-plane-l8j9r                         True
23s
│  └──Machine/azure-example-control-plane-xhxxg                         True
23s
└──Workers
   └──MachineDeployment/azure-example-md-0                              True
35s
     ├──Machine/azure-example-md-0-d67567c8b-2674r                      True
24s
     ├──Machine/azure-example-md-0-d67567c8b-n276j                      True
25s
     ├──Machine/azure-example-md-0-d67567c8b-pzg8k                      True
23s
     └──Machine/azure-example-md-0-d67567c8b-z8km9                      True
24s
```

NOTE: After moving the cluster lifecycle services to the workload cluster, remember to use `dkp` with the workload cluster `kubeconfig`.

1. Wait for the cluster control-plane to be ready:

```
kubectl --kubeconfig $HOME/.kube/config wait --for=condition=controlplaneready
"clusters/${CLUSTER_NAME}" --timeout=60m
```

```
cluster.cluster.x-k8s.io/azure-example condition met
```

Persistent Volumes (PVs) are not deleted automatically by design in order to preserve your data. However, they take up storage space if not deleted. You must delete PVs manually. Information for backup of a cluster and PVs is on the page in documentation called Back up your Cluster's Applications and Persistent Volumes (see page 736).

7.12.9.3 Delete the Workload Cluster

1. Make sure your Azure credentials are up to date. Refresh the credentials using this command:

```
dkp update bootstrap credentials azure --kubeconfig $HOME/.kube/config
```

   a. Use `dkp` with the bootstrap cluster to delete the workload cluster.

2. To delete a cluster, you would use `dkp delete cluster` and pass in the name of the cluster you are trying to delete with `--cluster-name` flag. You would use `kubectl get clusters` to get those details (`--cluster-name` and `--namespace`) of the Kubernetes cluster to delete it.
   NOTE: Do not use `dkp get clusters` since that gets you Kommander cluster details rather than Konvoy kubernetes cluster details.

```
kubectl get clusters
```

3. Delete the Kubernetes cluster and wait a few minutes:

   NOTE: Before deleting the cluster, dkp deletes all Services of type LoadBalancer on the cluster. To skip this step, use the flag `--delete-kubernetes-resources=false` .

```
dkp delete cluster --cluster-name=${CLUSTER_NAME} --kubeconfig $HOME/.kube/
config
```

```
✓ Deleting Services with type LoadBalancer for Cluster default/azure-example
✓ Deleting ClusterResourceSets for Cluster default/azure-example
✓ Deleting cluster resources
✓ Waiting for cluster to be fully deleted
Deleted default/azure-example cluster
```

   After the workload cluster is deleted, delete the bootstrap cluster.

7.12.9.4 Delete the Bootstrap Cluster

```
dkp delete bootstrap --kubeconfig $HOME/.kube/config
```

```
✓ Deleting bootstrap cluster
```

7.12.9.5 Next Step:

Once your cluster is built in the Konvoy component of DKP for your infrastructure/environment, you will install the Kommander (see page 1227) component of DKP to see your dashboard and continue customization.

7.12.9.6 Known Limitations

NOTE: Be aware of these limitations in the current release of Konvoy.

- The Konvoy version used to create the workload cluster must match the Konvoy version used to delete the workload cluster.

7.12.10 Create a new Azure Cluster Using the DKP UI

Enterprise Gov Advanced

DKP UI allows you to provision an Azure cluster from your browser.

7.12.10.1 Prerequisites

7.12.10.1.1 Create an Azure Infrastructure Provider

Before you provision a cluster via the UI, first create an Azure infrastructure provider to hold your Azure credentials:

1. Log in to the Azure command line:

   ```
   az login
   ```

2. Create an Azure Service Principal (SP) by running the following command:

```
az ad sp create-for-rbac --role contributor --name "$(whoami)-konvoy" --
scopes=/subscriptions/$(az account show --query id -o tsv)
```

3. Select **Infrastructure Providers** from the Dashboard menu.
4. Select **Add Infrastructure Provider**.
5. Workspaces (see page 573) If you are already in a workspace, the provider is automatically created in that workspace.
6. Select **Microsoft Azure**.
7. Add a **Name** for your Infrastructure Provider.
8. Take the `id` output from the log in command above and put it into the **Subscription ID** field.
9. Take the `tenant` used in Step 2 and put it into the **Tenant ID** field.
10. Take the `appId` used in Step 2 and put it into the **Client ID** field.
11. Take the `password` used in Step 2 and put it into the **Client Secret** field.
12. Select **Save**.

7.12.10.2 Provision an Azure Cluster

Follow these steps to provision an Azure cluster:

1. From the top menu bar, select your target workspace.
2. Select **Clusters > Add Cluster**.
3. Choose **Create Cluster**.
4. Enter the **Cluster Name**.
5. From **Select Infrastructure Provider**, choose the provider created in the prerequisites section.
6. If available, choose a **Kubernetes Version**. Otherwise, the DKP 2.5.0 Supported Kubernetes Versions (see page 1543) installs.
7. Select a data center **location** or specify a custom **location**.
8. Edit your worker **Node Pools** as necessary. You can choose the **Number of Nodes**, the **Machine Type**, and for the worker nodes you can choose a **Worker Availability Zone**.
9. Add any additional **Labels** or **Infrastructure Provider Tags** as necessary.
10. Review your inputs to ensure they meet the predefined criteria, and select **Create**.

It can take up to 15 minutes for your cluster to appear in the **Provisioned** status.

You are then redirected to the **Clusters** page, where you'll see your new cluster in the **Provisioning** status. Hover over the status to view the details.

7.13 AKS Infrastructure

Enterprise Gov Advanced

When installing DKP on Azure Kubernetes Service (**AKS**) infrastructure, you can choose from multiple configuration types. The different types of AKS configuration types supported in DKP are covered in this section.

- Create a New AKS Cluster (see page 1068)
- Explore New AKS Cluster (see page 1072)
- Delete AKS Cluster (see page 1075)
- Create a new AKS Cluster via UI (see page 1077)

7.13.1 Create a New AKS Cluster

Enterprise Gov Advanced

7.13.1.1 Use DKP to create a new AKS cluster

Ensure that the `KUBECONFIG` environment variable is set to the self-managed cluster by running `export KUBECONFIG=${SELF_MANAGED_AZURE_CLUSTER}.conf`

7.13.1.2 Name Your Cluster

Give your cluster a unique name suitable for your environment.

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[758] for more naming information.

758 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

7.13.1.3 Create a New AKS Kubernetes Cluster

1. Set the environment variable to a name for this cluster.

```
export CLUSTER_NAME=<aks-example>
```

2. Check to see what version of Kubernetes is available in your region. When deploying with AKS, you need to declare the version of Kubernetes you wish to use by running the following command, substituting `<your-location>` for the Azure region you're deploying to:

```
az aks get-versions -o table --location <your-location>
```

3. Set the version of Kubernetes you've chosen:
NOTE: Using Kubernetes v1.25.4 is recommended, but if it is not available, choose an available 1.25.x version. The version listed in the command is an example.

```
export KUBERNETES_VERSION=1.25.4
```

4. Create the cluster:

```
dkp create cluster aks --cluster-name=${CLUSTER_NAME} --additional-tags=owner=$
(whoami) --kubernetes-version=${KUBERNETES_VERSION}
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).
The output displays similar to this:

```
Generating cluster resources
cluster.cluster.x-k8s.io/aks-example created
azuremanagedcontrolplane.infrastructure.cluster.x-k8s.io/aks-example created
azuremanagedcluster.infrastructure.cluster.x-k8s.io/aks-example created
machinepool.cluster.x-k8s.io/aks-example created
azuremanagedmachinepool.infrastructure.cluster.x-k8s.io/cp6dsz8 created
machinepool.cluster.x-k8s.io/aks-example-md-0 created
azuremanagedmachinepool.infrastructure.cluster.x-k8s.io/mp6gglj created
clusterresourceset.addons.cluster.x-k8s.io/cluster-autoscaler-aks-example
created
configmap/cluster-autoscaler-aks-example created
clusterresourceset.addons.cluster.x-k8s.io/node-feature-discovery-aks-example
created
configmap/node-feature-discovery-aks-example created
```

```
clusterresourceset.addons.cluster.x-k8s.io/nvidia-feature-discovery-aks-example
created
configmap/nvidia-feature-discovery-aks-example created
```

7.13.1.4 Inspecting or Editing the Cluster Objects

Use your favorite editor.

> **NOTE:** Editing the cluster objects requires some understanding of Cluster API. Edits can prevent the cluster from deploying successfully.

The objects are Custom Resources[759] defined by Cluster API components, and they belong in three different categories:

- Cluster
  A *Cluster* object has references to the infrastructure-specific and control plane objects.
- Control Plane
- Node Pool
  A Node Pool is a collection of machines with identical properties. For example, a cluster might have one Node Pool with large memory capacity, another Node Pool with GPU support. Each Node Pool is described by three objects: The MachinePool references an object that describes the configuration of Kubernetes components (for example, kubelet) deployed on each node pool machine, and an infrastructure-specific object that describes the properties of all node pool machines. Here, it references a *KubeadmConfigTemplate*.

For in-depth documentation about the objects, read Concepts[760] in the Cluster API Book.

1. Wait for the cluster control-plane to be ready:

   ```
   kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
   timeout=20m
   ```

   ```
   cluster.cluster.x-k8s.io/aks-example condition met
   ```

   The `READY` status will become `True` after the cluster control-plane becomes ready.

759 https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/
760 https://cluster-api.sigs.k8s.io/user/concepts.html

2. Once the objects are created on the API server, the Cluster API controllers reconcile them. They create infrastructure and machines. As they progress, they update the Status of each object. DKP provides a command to describe the current status of the cluster:

```
dkp describe cluster -c ${CLUSTER_NAME}
```

```
NAME                                                          READY  SEVERITY
REASON  SINCE  MESSAGE
Cluster/aks-example                                           True
48m
├──ClusterInfrastructure - AzureManagedCluster/aks-example
└──ControlPlane - AzureManagedControlPlane/aks-example
```

3. As they progress, the controllers also create Events. List the Events using this command:

```
kubectl get events | grep ${CLUSTER_NAME}
```

For brevity, the example uses `grep`. It is also possible to use separate commands to get Events for specific objects. For example, `kubectl get events --field-selector involvedObject.kind="AKSCluster"` and `kubectl get events --field-selector involvedObject.kind="AKSMachine"`.

```
48m         Normal    SuccessfulSetNodeRefs               machinepool/aks-
example-md-0                  [{Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000000 UID:e3c30389-660d-46f5-b9d7-219f80b5674d APIVersion:
ResourceVersion: FieldPath:} {Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000001 UID:300d71a0-f3a7-4c29-9ff1-1995ffb9cfd3 APIVersion:
ResourceVersion: FieldPath:} {Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000002 UID:8eae2b39-a415-425d-8417-d915a0b2fa52 APIVersion:
ResourceVersion: FieldPath:} {Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000003 UID:3e860b88-f1a4-44d1-b674-a54fad599a9d APIVersion:
ResourceVersion: FieldPath:}]
6m4s        Normal    AzureManagedControlPlane available
azuremanagedcontrolplane/aks-example          successfully reconciled
48m         Normal    SuccessfulSetNodeRefs               machinepool/aks-
example                       [{Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000000 UID:e3c30389-660d-46f5-b9d7-219f80b5674d APIVersion:
ResourceVersion: FieldPath:} {Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000001 UID:300d71a0-f3a7-4c29-9ff1-1995ffb9cfd3 APIVersion:
ResourceVersion: FieldPath:} {Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000002 UID:8eae2b39-a415-425d-8417-d915a0b2fa52 APIVersion:
ResourceVersion: FieldPath:}]
```

7.13.1.5 Known Limitations

- The DKP version used to create a workload cluster must match the DKP version used to create a workload cluster.
- DKP supports deploying one workload cluster.
- DKP generates a single nodepool is deployed by default, but you can add additional nodepools.
- DKP does not validate edits to cluster objects.

When complete, you can explore the new cluster (see page 1072).

7.13.2 Explore New AKS Cluster

Enterprise Gov Advanced

7.13.2.1 Learn to interact with your AKS Kubernetes cluster

This guide explains how to use the command line to interact with your newly deployed Kubernetes cluster.

Before you start, make sure you have created a workload cluster, as described in Create a New Cluster (see page 1068).

7.13.2.2 Explore the New AKS Cluster

1. Get a kubeconfig file for the workload cluster:

   When the workload cluster is created, the cluster lifecycle services generate a kubeconfig file for the workload cluster, and write it to a *Secret*. The kubeconfig file is scoped to the cluster administrator.

   Get the kubeconfig from the *Secret*, and write it to a file, using this command:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

2. List the Nodes using this command:

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf get nodes
   ```

   ```
   NAME                                STATUS   ROLES   AGE   VERSION
   aks-cp6dsz8-41174201-vmss000000   Ready    agent   56m   v1.25.4
   ```

```
aks-cp6dsz8-41174201-vmss000001   Ready    agent   55m   v1.25.4
aks-cp6dsz8-41174201-vmss000002   Ready    agent   56m   v1.25.4
aks-mp6gglj-41174201-vmss000000   Ready    agent   55m   v1.25.4
aks-mp6gglj-41174201-vmss000001   Ready    agent   55m   v1.25.4
aks-mp6gglj-41174201-vmss000002   Ready    agent   55m   v1.25.4
aks-mp6gglj-41174201-vmss000003   Ready    agent   56m   v1.25.4
```

NOTE: It may take a few minutes for the Status to move to `Ready` while the Pod network is deployed. The Nodes' Status should change to Ready soon after the `calico-node` DaemonSet Pods are Ready.

3. List the Pods using this command:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf get --all-namespaces pods
```

```
NAMESPACE                NAME                                       READY
STATUS     RESTARTS   AGE
calico-system            calico-kube-controllers-5dcd4b47b5-tgslm   1/1
Running    0          3m58s
calico-system            calico-node-46dj9                          1/1
Running    0          3m58s
calico-system            calico-node-crdgc                          1/1
Running    0          3m58s
calico-system            calico-node-m7s7x                          1/1
Running    0          3m58s
calico-system            calico-node-qfkqc                          1/1
Running    0          3m57s
calico-system            calico-node-sfqfm                          1/1
Running    0          3m57s
calico-system            calico-node-sn67x                          1/1
Running    0          3m53s
calico-system            calico-node-w2pvt                          1/1
Running    0          3m58s
calico-system            calico-typha-6f7f59969c-5z4t5              1/1
Running    0          3m51s
calico-system            calico-typha-6f7f59969c-ddzqb              1/1
Running    0          3m58s
calico-system            calico-typha-6f7f59969c-rr4lj              1/1
Running    0          3m51s
kube-system              azure-ip-masq-agent-4f4v6                  1/1
Running    0          4m11s
```

```
kube-system              azure-ip-masq-agent-5xfh2                        1/1
Running    0          4m11s
kube-system              azure-ip-masq-agent-9hlk8                        1/1
Running    0          4m8s
kube-system              azure-ip-masq-agent-9vsgg                        1/1
Running    0          4m16s
kube-system              azure-ip-masq-agent-b9wjj                        1/1
Running    0          3m57s
kube-system              azure-ip-masq-agent-kpjtl                        1/1
Running    0          3m53s
kube-system              azure-ip-masq-agent-vr7hd                        1/1
Running    0          3m57s
kube-system              cluster-autoscaler-b4789f4bf-qkfk2               0/1
Init:0/1   0          3m28s
kube-system              coredns-845757d86-9jf8b                          1/1
Running    0          5m29s
kube-system              coredns-845757d86-h4xfs                          1/1
Running    0          4m
kube-system              coredns-autoscaler-5f85dc856b-xjb5z              1/1
Running    0          5m23s
kube-system              csi-azuredisk-node-4n4fx                         3/3
Running    0          3m53s
kube-system              csi-azuredisk-node-8pnjj                         3/3
Running    0          3m57s
kube-system              csi-azuredisk-node-sbt6r                         3/3
Running    0          3m57s
kube-system              csi-azuredisk-node-v25wc                         3/3
Running    0          4m16s
kube-system              csi-azuredisk-node-vfbxg                         3/3
Running    0          4m11s
kube-system              csi-azuredisk-node-w5ff5                         3/3
Running    0          4m11s
kube-system              csi-azuredisk-node-zzgqx                         3/3
Running    0          4m8s
kube-system              csi-azurefile-node-2rpcc                         3/3
Running    0          3m57s
kube-system              csi-azurefile-node-4gqkf                         3/3
Running    0          4m11s
kube-system              csi-azurefile-node-f6k8m                         3/3
Running    0          4m16s
kube-system              csi-azurefile-node-k72xq                         3/3
Running    0          4m8s
kube-system              csi-azurefile-node-vx7r4                         3/3
Running    0          3m53s
kube-system              csi-azurefile-node-zc8kr                         3/3
Running    0          4m11s
kube-system              csi-azurefile-node-zkl6b                         3/3
Running    0          3m57s
kube-system              kube-proxy-4fpb6                                 1/1
Running    0          3m53s
kube-system              kube-proxy-6qfbf                                 1/1
Running    0          4m16s
```

```
kube-system                 kube-proxy-6wnt2                                  1/1
Running     0           4m8s
kube-system                 kube-proxy-cspd5                                  1/1
Running     0           3m57s
kube-system                 kube-proxy-nsgq6                                  1/1
Running     0           4m11s
kube-system                 kube-proxy-qz2st                                  1/1
Running     0           4m11s
kube-system                 kube-proxy-zvh9k                                  1/1
Running     0           3m57s
kube-system                 metrics-server-6bc97b47f7-ltkkj                   1/1
Running     0           5m28s
kube-system                 tunnelfront-77d68f78bf-t78ck                      1/1
Running     0           5m23s
node-feature-discovery   node-feature-discovery-master-65dc499cd-fxwb5    1/1
Running     0           3m28s
node-feature-discovery   node-feature-discovery-worker-277xc              1/1
Running     0           3m28s
node-feature-discovery   node-feature-discovery-worker-4dq5k              1/1
Running     0           3m28s
node-feature-discovery   node-feature-discovery-worker-57nb8              1/1
Running     0           3m28s
node-feature-discovery   node-feature-discovery-worker-b4lkl              1/1
Running     0           3m28s
node-feature-discovery   node-feature-discovery-worker-kslst              1/1
Running     0           3m28s
node-feature-discovery   node-feature-discovery-worker-ppjtm              1/1
Running     0           3m28s
node-feature-discovery   node-feature-discovery-worker-x5bgf              1/1
Running     0           3m28s
tigera-operator             tigera-operator-74c4d9cf84-k7css                  1/1
Running     0           5m25s
```

When ready, you can delete the cluster (see page 1075).

7.13.3 Delete AKS Cluster

Enterprise Gov Advanced

Ensure that the `KUBECONFIG` environment variable is set to the self-managed cluster by running `export KUBECONFIG={SELF_MANAGED_AZURE_CLUSTER}.conf`

7.13.3.1 Delete the AKS cluster and clean up your environment

7.13.3.2 Delete the Workload Cluster

1. Delete the Kubernetes cluster and wait a few minutes:

   Before deleting the cluster, DKP deletes all Services of type LoadBalancer on the cluster. Deleting the Service deletes the Azure LoadBalancer that backs it. To skip this step, use the flag `--delete-kubernetes-resources=false`.

> **NOTE:** Do not skip this step if the Azure Network is managed by DKP. When DKP deletes cluster, it deletes the Network.

```
dkp delete cluster --cluster-name=${CLUSTER_NAME}
```

```
 ✓ Deleting Services with type LoadBalancer for Cluster default/aks-example
 ✓ Deleting ClusterResourceSets for Cluster default/aks-example
 ✓ Deleting cluster resources
 ✓ Waiting for cluster to be fully deleted
Deleted default/aks-example cluster
```

7.13.3.3 Next Step:

Once your cluster is built in the Konvoy component of DKP for your infrastructure/environment, you will Install the Kommander (see page 1227) component of DKP to see your dashboard and continue customization.

7.13.3.4 Known Limitations

> **NOTE:** Be aware of these limitations in the current release of DKP.

- The DKP version used to create the workload cluster must match the DKP version used to delete the workload cluster.

7.13.4 Create a new AKS Cluster via UI

Enterprise Gov Advanced

DKP UI allows you to provision an AKS cluster from your browser.

7.13.4.1 Prerequisites

7.13.4.1.1 Create an AKS Infrastructure Provider

Before provisioning a cluster via the UI, first create an AKS infrastructure provider to hold your AKS credentials:

1. Log in to the Azure command line:

   ```
   az login
   ```

2. Create an Azure Service Principal (SP) by running the following command:

   ```
   az ad sp create-for-rbac --role contributor --name "$(whoami)-konvoy" --
   scopes=/subscriptions/$(az account show --query id -o tsv)
   ```

3. Select **Infrastructure Providers** from the Dashboard menu.
4. Select **Add Infrastructure Provider**.
5. Choose a workspace. (see page 573) If you are already in a workspace, the provider is automatically created in that workspace.
6. Select **Microsoft Azure**.
7. Add a **Name** for your Infrastructure Provider.
8. Take the `id` output from the log in command above and put it into the **Subscription ID** field.
9. Take the `tenant` used in Step 2 and put it into the **Tenant ID** field.
10. Take the `appId` used in Step 2 and put it into the **Client ID** field.
11. Take the `password` used in Step 2 and put it into the **Client Secret** field.
12. Select **Save**.

7.13.4.2 Provision an AKS Cluster

Follow these steps to provision an AKS cluster:

1. From the top menu bar, select your target workspace.
2. Select **Clusters > Add Cluster**.
3. Choose **Create Cluster**.
4. Enter the **Cluster Name**.
5. From **Select Infrastructure Provider**, choose the provider created in the prerequisites section.
6. In order to create a **Kubernetes Version**, run the command below in the `az` cli, and then select the version of AKS you want to use.

   ```
   az aks get-versions -o table --location <location>
   ```

7. Select a data center **location** or specify a custom **location**.
8. Edit your worker **Node Pools**, as necessary. You can choose the **Number of Nodes**, the **Machine Type**, and for the worker nodes, you can choose a **Worker Availability Zone**.
9. Add any additional **Labels** or **Infrastructure Provider Tags**, as necessary.
10. Validate your inputs, then select **Create**.

It can take up to 15 minutes for your cluster to appear in the **Provisioned** status.

You are then redirected to the **Clusters** page, where you'll see your new cluster in the **Provisioning** status. Hover over the status to view the details.

7.13.4.3 Access an AKS Cluster

After the cluster is successfully attached (managed), you can retrieve a custom `kubeconfig` file from the UI using your DKP UI administrator credentials.

7.14 Pre-provisioned Infrastructure

7.14.1 Create a Kubernetes cluster on pre-provisioned nodes in a bare metal infrastructure

The following procedure describes creating a DKP cluster on a pre-provisioned infrastructure using SSH.

Completing this procedure results in a Kubernetes cluster that includes a Container Networking Interface (CNI)[761] and a Local Persistence Volume Static Provisioner[762], and that is ready for workload deployment.

Before moving to a production environment, you may want to add applications for logging and monitoring, storage, security, and other functions. You can use DKP to select and deploy applications (see page 564), or deploy your own.

To get started, see these topics.

7.14.2 Advanced Workflow

In the Day 1 - Basic Installations, the installation process for Pre-provisioned on any provider uses a `self-managed` flag. This process eliminates errors of creating a bootstrap and not managing the KUBECONFIG properly to run commands against the correct cluster when you have multiples.

The more advanced workflow for Pre-provisioned, allows for custom YAML files to pass various overrides and other custom configurations during cluster creation. The pages in this section describe these more complex scenarios.

7.14.3 Pre-provisioned Prerequisite Configurations

In order to fulfill all the prerequisites for a successful implementation on a Pre-provisioned environment, there will be infrastructure requirements as well as machine requirements. Please read all the sections on this page to ensure you have met all prerequisites.

761 https://docs.projectcalico.org/
762 https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner

7.14.3.1 Prerequisites for a Pre-provisioned Infrastructure

Before you begin using DKP, you must have:

- An x86_64-based Linux or macOS machine.
- The `dkp` binary for Linux, or macOS.
- kubectl[763] for interacting with the running cluster.
- Pre-provisioned hosts with SSH access enabled.
- An unencrypted SSH private key, whose public key is configured on the above hosts.
- Resource requirements (see page 110)
- Pre-provisioned Override Files (see page 1341)

When air-gapped, you must follow the steps described in the Air-gapped Define Environment (see page 1082) and Docker Registry also as a prerequisite.

DKP uses `localvolumeprovisioner` as the default storage provider (see page 103). However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[764] compatible storage that is suitable for production.

You can choose from any of the storage options[765] available for Kubernetes. To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your newly created StorageClass to be the default by following the commands in the Kubernetes documentation called Changing the Default Storage Class[766].

7.14.3.2 Machine Specifications

7.14.3.2.1 Control plane machines

You should have at least three control plane machines.

Each control plane machine must have:

- 4 cores
- 16 GiB memory
- Approximately 80 GiB of free space for the volume used for `/var/lib/kubelet` and `/var/lib/containerd`.
- 15% free space on the root file system.
- Multiple ports open, as described in DKP Ports (see page 60).

763 https://kubernetes.io/docs/tasks/tools/#kubectl
764 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
765 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
766 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

- `firewalld` systemd service disabled. If it exists and is enabled, use the commands `systemctl stop firewalld` then `systemctl disable firewalld`, so that `firewalld` remains disabled after the machine restarts.
- For a Pre-provisioned environment using Ubuntu 20.04, ensure the machine has the `/run` directory mounted with exec permissions.

> Swap is disabled. The `kubelet` does not have generally-available support for swap. Due to variable commands, refer to your operating system documentation.

7.14.3.2.2 Worker machines

You should have at least four worker machines. The specific number of worker machines required for your environment can vary depending on the cluster workload and size of the machines.

Each worker machine must have:

- 8 cores
- 32 GiB memory
- Around 80 GiB of free space for the volume used for `/var/lib/kubelet` and `/var/lib/containerd`.
- 15% free space on the root file system
- If you plan to use local volume provisioning to provide persistent volumes for your workloads, you must mount at least four volumes to the `/mnt/disks/` mount point on each machine. Each volume must have at least 55 GiB of capacity.
- Ensure your disk meets the resource requirements for Rook Ceph in `Block` mode for ObjectStorageDaemons[767] as specified in the requirements table (see page 896).
- Multiple ports open, as described in DKP Ports (see page 60).
- `firewalld` systemd service disabled. If it exists and is enabled, use the commands `systemctl stop firewalld` then `systemctl disable firewalld`, so that `firewalld` remains disabled after the machine restarts.
- For a Pre-provisioned environment using Ubuntu 20.04, ensure the machine has the `/run` directory mounted with exec permissions.

> Swap is disabled. The `kubelet` does not have generally-available support for swap. Due to variable commands, refer to your operating system documentation.

767 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

7.14.3.3 Next Step:

Pre-provisioned Set Infrastructure (see page 1087)

7.14.4 Pre-provisioned Air-gapped Define Environment

7.14.4.1 Fulfill the prerequisites for using a pre-provisioned infrastructure when Air-Gapped

The instructions below outline how to fulfill the prerequisites for using pre-provisioned infrastructure when using air-gapped. In DKP 2.5.0, there is a new complete DKP air-gapped bundle available to download (see page 71) which contains all the DKP components needed for air-gapped installation. (i.e. `dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz`)

7.14.4.2 Air-Gapped Registry Prerequisites

DKP in an air-gapped environment requires a local container registry of trusted images to enable production level Kubernetes cluster management. In an environment with access to the internet, you retrieve artifacts from specialized repositories dedicated to them such as Docker images contained in DockerHub and Helm Charts that come from a dedicated Helm Chart repository. However, in an air-gapped environment, you need local repositories to store Helm charts, Docker images and other artifacts. Tools such as jFrog, Harbor and Nexus handle multiple types of artifacts in one local repository.

- JFrog Artifactory - If you use Jfrog Artifactory or Jfrog Container Registry, you must update to a new version of the software. Any build newer than version 7.11 will work, as we have confirmed that older versions are not compatible.
- Nexus Registry - If you use Nexus Registry, there was an issue that prevented usage with DKP 2.X and OCI Images, but support for OCI Images was added here in this publicly available Jira ticket: [NEXUS-21087] Support OCI registry format - Sonatype JIRA[768]
- Harbor Registry - Any newer version than **Harbor Registry v2.1.1**-5f52168e will support OCI images.

7.14.4.2.1 Bastion Host

If you have not set up a Bastion Host (see page 1162) yet, refer to that section of the Documentation.

7.14.4.2.2 Load the Bootstrap Image

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz` from the download site (see page 71) mentioned above, extract the tarball to a local directory:

768 https://issues.sonatype.org/browse/NEXUS-21087

```
tar -xzvf dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz && cd dkp-v2.5.2
```

2. Load the bootstrap Docker image on your bastion machine:

```
docker load -i konvoy-bootstrap-image-v2.5.2.tar
```

```
podman load -i konvoy-bootstrap-image-v2.5.2.tar
```

7.14.4.2.3 Copy Air-gapped Artifacts onto Cluster Hosts

Using the Konvoy Image Builder (see page 1282), you can copy the required artifacts (such as charts, java or OS packages like RPM or Deb) onto your cluster hosts.

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz` , extract the tarball to a local directory:

```
tar -xzvf dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz && cd dkp-v2.5.2/kib
```

2. The Kubernetes image bundle will be located in `kib/artifacts/images` and you will want to verify image and artifacts.
   a. Verify the image bundles exist in `kib/artifacts/images` :

```
$ ls kib/artifacts/images/
kubernetes-images-1.25.4-d2iq.1.tar kubernetes-images-1.25.4-d2iq.1-
fips.tar
```

   b. Verify the artifacts for your OS exist in the `artifacts/` directory and export the appropriate variables:

```
$ ls kib/artifacts/
1.25.4_centos_7_x86_64.tar.gz       1.25.4_redhat_8_x86_64_fips.tar.gz
containerd-1.6.17-d2iq.1-rhel-7.9-x86_64.tar.gz   containerd-1.6.17-
d2iq.1-rhel-8.6-x86_64_fips.tar.gz  pip-packages.tar.gz
1.25.4_centos_7_x86_64_fips.tar.gz  1.25.4_rocky_9_x86_64.tar.gz
containerd-1.6.17-d2iq.1-rhel-7.9-x86_64_fips.tar.gz  containerd-1.6.17-
d2iq.1-rocky-9.0-x86_64.tar.gz
1.25.4_redhat_7_x86_64.tar.gz       1.25.4_ubuntu_20_x86_64.tar.gz
containerd-1.6.17-d2iq.1-rhel-8.4-x86_64.tar.gz   containerd-1.6.17-
d2iq.1-rocky-9.1-x86_64.tar.gz
```

```
1.25.4_redhat_7_x86_64_fips.tar.gz  containerd-1.6.17-d2iq.1-centos-7.9-
x86_64.tar.gz       containerd-1.6.17-d2iq.1-rhel-8.4-x86_64_fips.tar.gz
containerd-1.6.17-d2iq.1-ubuntu-20.04-x86_64.tar.gz
1.25.4_redhat_8_x86_64.tar.gz       containerd-1.6.17-d2iq.1-centos-7.9-
x86_64_fips.tar.gz  containerd-1.6.17-d2iq.1-rhel-8.6-x86_64.tar.gz
images
```

c. For example, for RHEL 8.4 you would set:

```
export OS_PACKAGES_BUNDLE=1.25.4_redhat_8_x86_64.tar.gz
export CONTAINERD_BUNDLE=containerd-1.6.17-d2iq.1-rhel-8.4-x86_64.tar.gz
```

3. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
export WORKER_1_ADDRESS="<worker-address-1>"
export WORKER_2_ADDRESS="<worker-address-2>"
export WORKER_3_ADDRESS="<worker-address-3>"
export WORKER_4_ADDRESS="<worker-address-4>"
export SSH_USER="<ssh-user>"
export SSH_PRIVATE_KEY_FILE="<private key file>"
```

`SSH_PRIVATE_KEY_FILE` must be either the name of the SSH private key file in your working directory or an absolute path to the file in your user's home directory.

4. Generate an `inventory.yaml` which is automatically picked up by the `konvoy-image upload` in the next step. This `inventory.yaml` **should exclude any GPU workers**, which will be handled in steps #6-7.

```
cat <<EOF > inventory.yaml
all:
  vars:
    ansible_user: $SSH_USER
    ansible_port: 22
    ansible_ssh_private_key_file: $SSH_PRIVATE_KEY_FILE
  hosts:
    $CONTROL_PLANE_1_ADDRESS:
      ansible_host: $CONTROL_PLANE_1_ADDRESS
    $CONTROL_PLANE_2_ADDRESS:
      ansible_host: $CONTROL_PLANE_2_ADDRESS
    $CONTROL_PLANE_3_ADDRESS:
      ansible_host: $CONTROL_PLANE_3_ADDRESS
    $WORKER_1_ADDRESS:
      ansible_host: $WORKER_1_ADDRESS
```

```
    $WORKER_2_ADDRESS:
      ansible_host: $WORKER_2_ADDRESS
    $WORKER_3_ADDRESS:
      ansible_host: $WORKER_3_ADDRESS
    $WORKER_4_ADDRESS:
      ansible_host: $WORKER_4_ADDRESS
EOF
```

5. Upload the artifacts onto cluster hosts with the following command:

```
konvoy-image upload artifacts \
             --container-images-dir=./artifacts/images/ \
             --os-packages-bundle=./artifacts/$OS_PACKAGES_BUNDLE \
             --containerd-bundle=artifacts/$CONTAINERD_BUNDLE \
             --pip-packages-bundle=./artifacts/pip-packages.tar.gz
```

The `konvoy-image upload artifacts` command copies all OS packages and other artifacts onto each of the machines in your inventory. When you create the cluster, the provisioning process connects to each node and runs commands to install those artifacts and consequently Kubernetes running. KIB uses variable overrides (see page 1330) to specify base image and container images to use in your new machine image. The variable overrides files for NVIDIA and FIPS can be ignored unless adding an overlay feature.

- Use the --overrides flag and reference either `fips.yaml` or `offline-fips.yaml` manifests located in the overrides directory[769] or see these pages in the documentation:
  - FIPS Overrides (see page 1344)
  - Create FIPS 140 Images (see page 1345)

7.14.4.2.4 GPU Only Steps

If the NVIDIA runfile[770] installer has not been downloaded, then retrieve and install the download first by running the following command. The first line in the command below downloads and installs the runfile and the second line places it in the artifacts directory.

769 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides
770 https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html#runfile

- ```
  curl -O https://download.nvidia.com/XFree86/Linux-x86_64/470.82.01/NVIDIA-Linux-x86_64-470.82.01.run
  mv NVIDIA-Linux-x86_64-470.82.01.run artifacts
  ```

6. Create an inventory for GPU Nodes.

```
cat <<EOF >  gpu_inventory.yaml
all:
  vars:
    ansible_port: 22
    ansible_ssh_private_key_file: $SSH_PRIVATE_KEY_FILE
    ansible_user: $SSH_USER

  hosts:
    $GPU_WORKER_1_ADDRESS:
      ansible_host: $GPU_WORKER_1_ADDRESS
EOF
```

7. Upload the artifacts to the gpu nodepool with the `nvidia-runfile` flag

```
konvoy-image upload artifacts --inventory-file=gpu_inventory.yaml \
              --container-images-dir=./artifacts/images/ \
              --os-packages-bundle=./artifacts/$OS_PACKAGES_BUNDLE \
              --containerd-bundle=artifacts/$CONTAINERD_BUNDLE \
              --pip-packages-bundle=./artifacts/pip-packages.tar.gz \
              --nvidia-runfile=./artifacts/NVIDIA-Linux-x86_64-470.82.01.run
```

KIB uses variable overrides (see page 1330) to specify base image and container images to use in your new machine image. The variable overrides files for NVIDIA and FIPS can be ignored unless adding an overlay feature.

- Use the overrides flag (EX: `--overrides overrides/fips.yaml`) and reference either `fips.yaml` or `offline-fips.yaml` manifests located in the overrides directory[771] or see these pages in the documentation:
  - FIPS Overrides (see page 1331)
  - Create FIPS 140 Images (see page 1345)

771 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides
```

7.14.4.2.5 Load your Registry

Before creating a Kubernetes cluster, you need the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1162) and either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1349) page for more information.

1. Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz` , extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz && cd dkp-v2.5.0
   ```

2. Set an environment variable with your registry address with this command:

   ```
   export REGISTRY_ADDRESS=<registry-address>:<registry-port>
   export REGISTRY_USERNAME=<username>
   export REGISTRY_PASSWORD=<password>
   ```

3. Execute the following command to load the air-gapped image bundle into your private registry:

   ```
   dkp push image-bundle --image-bundle ./container-images/konvoy-image-bundle-v2.5.0.tar --to-registry $REGISTRY_ADDRESS --to-registry-username $REGISTRY_USERNAME --to-registry-password $REGISTRY_PASSWORD
   ```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

7.14.4.2.6 Next Step:

Pre-provisioned Set Infrastructure (see page 1087)

7.14.5 Pre-provisioned Set Infrastructure

Define the cluster hosts and infrastructure

Konvoy needs to know how to access your cluster hosts. This is done using inventory resources. For initial cluster creation, you must define a control-plane and at least one worker pool.

7.14.5.1 Define your infrastructure

1. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
export WORKER_1_ADDRESS="<worker-address-1>"
export WORKER_2_ADDRESS="<worker-address-2>"
export WORKER_3_ADDRESS="<worker-address-3>"
export WORKER_4_ADDRESS="<worker-address-4>"
export SSH_USER="<ssh-user>"
export SSH_PRIVATE_KEY_SECRET_NAME="$CLUSTER_NAME-ssh-key"
```

2. Use the following template to help you define your infrastructure. The environment variables that you set in the previous step automatically replace the variable names when the file is created.

```
cat <<EOF > preprovisioned_inventory.yaml
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-control-plane
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    # Create as many of these as needed to match your infrastructure
    # Note that the command line parameter --control-plane-replicas determines
how many control plane nodes will actually be used.
    #
    - address: $CONTROL_PLANE_1_ADDRESS
    - address: $CONTROL_PLANE_2_ADDRESS
    - address: $CONTROL_PLANE_3_ADDRESS
  sshConfig:
    port: 22
    # This is the username used to connect to your infrastructure. This user
must be root or
    # have the ability to use sudo without a password
    user: $SSH_USER
    privateKeyRef:
```

```
      # This is the name of the secret you created in the previous step. It
must exist in the same
      # namespace as this inventory object.
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-md-0
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    - address: $WORKER_1_ADDRESS
    - address: $WORKER_2_ADDRESS
    - address: $WORKER_3_ADDRESS
    - address: $WORKER_4_ADDRESS
  sshConfig:
    port: 22
    user: $SSH_USER
    privateKeyRef:
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
EOF
```

3. To tell the bootstrap cluster which nodes you want to be control plane nodes and which nodes are worker nodes, you use the `kubectl apply` command to apply the file to the bootstrap cluster:

   ```
   kubectl apply -f preprovisioned_inventory.yaml
   ```

   Output:

   ```
   preprovisionedinventory.infrastructure.cluster.konvoy.d2iq.io/preprovisioned-
   example-control-plane created
   preprovisionedinventory.infrastructure.cluster.konvoy.d2iq.io/preprovisioned-
   example-md-0 created
   ```

After defining the infrastructure, Pre-provisioned Define Control Plane Endpoint (see page 1090)

7.14.6 Pre-provisioned Define Control Plane Endpoint

7.14.6.1 Define the Control Plane Endpoint for your cluster

A control plane should have three, five, or seven nodes, so it can remain available if one, two, or three nodes fail. A control plane with one node should not be used in production.

In addition, the control plane should have an endpoint that remains available if some nodes fail.

```
                          -------- cp1.example.com:6443
                          |
      lb.example.com:6443 ---------- cp2.example.com:6443
                          |
                          -------- cp3.example.com:6443
```

In this example, the control plane endpoint host is `lb.example.com`, and the control plane endpoint port is `6443`. The control plane nodes are `cp1.example.com`, `cp2.example.com`, and `cp3.example.com`. The port of each API server is `6443`.

7.14.6.2 External Load Balancer

It is recommended that an external load balancer(LB) be the control plane endpoint. To distribute request load among the control plane machines, configure the load balancer to send requests to all the control plane machines. Configure the load balancer to send requests only to control plane machines that are responding to API requests.

7.14.6.3 Built-in virtual IP

If an external load balancer is not available, use the built-in virtual IP (see page 1098). The virtual IP is *not* a load balancer; it does not distribute request load among the control plane machines. However, if the machine receiving requests does not respond to them, the virtual IP automatically moves to another machine.

7.14.6.4 Single-Node control plane

Do not use a single-node control plane in a production cluster.

A control plane with one node can use its single node as the endpoint, so you will not require an external load balancer, or a built-in virtual IP. At least one control plane node must always be running. Therefore, to upgrade a cluster with one control plane node, a spare machine must be available in the control plane inventory. This machine is used to provision the new node before the old node is deleted.

Modify Control Plane Audit logs settings using the information contained in the page Configuring the Control Plane (see page 1351).

When the API server endpoints are defined, you can create the cluster (see page 1098).

7.14.6.5 Known limitations

Be aware of these limitations in the current release of DKP.

The control plane endpoint port is also used as the API server port on each control plane machine. The default port is 6443. Before you create the cluster, ensure the port is available for use on each control plane machine.

7.14.6.5.1 Next Step:

Pre-provisioned Bootstrap Cluster (see page 1097) unless you need to Create Secrets and Overrides (see page 1091) first.

7.14.7 Pre-provisioned Create Secrets and Overrides

7.14.7.1 Create necessary secrets and overrides for pre-provisioned clusters

DKP requires SSH access to your infrastructure with superuser privileges. You must provide an unencrypted SSH private key to DKP.

Populate this key and create the required secret, on your bootstrap cluster using the following procedure.

7.14.7.2 Create a unique cluster name

Give your cluster a unique name suitable for your environment.

Set the environment variable to be used throughout this procedure:

```
export CLUSTER_NAME=preprovisioned-example
```

(Optional) If you want to create a unique cluster name, use this command. This creates a unique name every time you run it, so use it carefully.

```
export CLUSTER_NAME=preprovisioned-example-$(LC_CTYPE=C tr -dc 'a-z0-9' </dev/urandom
| fold -w 5 | head -n1)
echo $CLUSTER_NAME
```

```
preprovisioned-example-pf4a3
```

7.14.7.3 Create a secret

Create a secret that contains the SSH key with these commands:

```
export SSH_PRIVATE_KEY_FILE="<path-to-ssh-private-key>"
```

```
export SSH_PRIVATE_KEY_SECRET_NAME=$CLUSTER_NAME-ssh-key
```

```
kubectl create secret generic ${SSH_PRIVATE_KEY_SECRET_NAME} --from-file=ssh-
privatekey=${SSH_PRIVATE_KEY_FILE}
kubectl label secret ${SSH_PRIVATE_KEY_SECRET_NAME} clusterctl.cluster.x-k8s.io/move=
```

```
secret/preprovisioned-example-ssh-key created
secret/preprovisioned-example-ssh-key labeled
```

7.14.7.4 Create Overrides

In these steps, you will point your machines at the desired Registry to obtain the Docker images. If your pre-provisioned machines need to have Custom Override Files (see page 1336), create a secret that includes all the overrides you want to provide in one file.

1. Example:
   If you want to provide an override with Docker credentials and a different source for EPEL on a CentOS7 machine, you should create a file like this:

   ```
   cat > overrides.yaml << EOF
   image_registries_with_auth:
   - host: "registry-1.docker.io"
     username: "my-user"
     password: "my-password"
     auth: ""
     identityToken: ""
   ```

```
epel_centos_7_rpm: https://my-rpm-repostory.org/epel/epel-release-
latest-7.noarch.rpm
EOF
```

You can then create the related secret by running the following command:

```
kubectl create secret generic $CLUSTER_NAME-user-overrides --from-
file=overrides.yaml=overrides.yaml
kubectl label secret $CLUSTER_NAME-user-overrides clusterctl.cluster.x-k8s.io/
move=
```

2. Example:
   When using Oracle 7 OS, you may wish to deploy the RHCK kernel instead of the default UEK kernel.
   To do so, add the following text to your `overrides.yaml`:

```
cat > overrides.yaml << EOF
---
oracle_kernel: RHCK
EOF
```

You can then create the related secret by running the following command:

```
kubectl create secret generic $CLUSTER_NAME-user-overrides --from-
file=overrides.yaml=overrides.yaml
kubectl label secret $CLUSTER_NAME-user-overrides clusterctl.cluster.x-k8s.io/
move=
```

7.14.7.5 Next Step:

Pre-provisioned Set Infrastructure (see page 1087)

7.14.7.6 Pre-provisioned FIPS Create Secrets and Overrides

7.14.7.6.1 Create necessary secrets and overrides for pre-provisioned clusters

DKP requires SSH access to your infrastructure with superuser privileges. You must provide an unencrypted SSH private key to DKP.

Populate this key and create the required secret, on your bootstrap cluster using the following procedure.

7.14.7.6.2 Create a unique cluster name

Give your cluster a unique name suitable for your environment.

Set the environment variable to be used throughout this procedure:

```
export CLUSTER_NAME=preprovisioned-example
```

(Optional) If you want to create a unique cluster name, use this command. This creates a unique name every time you run it, so use it carefully.

```
export CLUSTER_NAME=preprovisioned-example-$(LC_CTYPE=C tr -dc 'a-z0-9' </dev/urandom
| fold -w 5 | head -n1)
echo $CLUSTER_NAME
```

```
preprovisioned-example-pf4a3
```

7.14.7.6.3 Create a secret

Create a secret that contains the SSH key with these commands:

```
export SSH_PRIVATE_KEY_FILE="<path-to-ssh-private-key>"
```

```
export SSH_PRIVATE_KEY_SECRET_NAME=$CLUSTER_NAME-ssh-key
```

```
kubectl create secret generic ${SSH_PRIVATE_KEY_SECRET_NAME} --from-file=ssh-
privatekey=${SSH_PRIVATE_KEY_FILE}
kubectl label secret ${SSH_PRIVATE_KEY_SECRET_NAME} clusterctl.cluster.x-k8s.io/move=
```

```
secret/preprovisioned-example-ssh-key created
secret/preprovisioned-example-ssh-key labeled
```

(2.5) Create FIPS 140 Images: Air-gapped Environment

Pre-provisioned FIPS Infrastructure

If you are targeting a Pre-provisioned Installs (see page 125), you can create a FIPS-compliant cluster by doing the following:

1. Create a Pre-provisioned: Bootstrap Cluster (see page 1097)
2. Create a secret on the bootstrap cluster with the contents from `fips.yaml` override file[772] and any other user overrides you wish to provide

```
kubectl create secret generic $CLUSTER_NAME-fips-overrides --from-
file=overrides.yaml=overrides.yaml
kubectl label secret $CLUSTER_NAME-fips-overrides clusterctl.cluster.x-k8s.io/
move=
```

7.14.7.6.4 Create overrides

1. Create a secret that includes the customization Overrides for FIPS compliance:
   Note: Get the latest values for FIPS from the Konvoy Image Builder repo[773].

   ```
   cat > overrides.yaml << EOF
   ---
   k8s_image_registry: docker.io/mesosphere

   fips:
     enabled: true

   build_name_extra: -fips
   kubernetes_build_metadata: fips.0
   default_image_repo: hub.docker.io/mesosphere
   kubernetes_rpm_repository_url: "https://packages.d2iq.com/konvoy/stable/linux/
   repos/el/kubernetes-v{{ kubernetes_version }}-fips/x86_64"
   docker_rpm_repository_url: "\
     https://containerd-fips.s3.us-east-2.amazonaws.com\
     /{{ ansible_distribution_major_version|int }}\
     /x86_64"
   EOF
   ```

2. If your pre-provisioned machines need to have a customization with alternate package libraries, Docker image repos, or other Custom Override Files (see page 1336), **add** more lines to the same Overrides file.

772 https://github.com/mesosphere/konvoy-image-builder/blob/main/overrides/fips.yaml
773 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

a. Example:
If you want to provide an override with Docker credentials and a different source for EPEL on a CentOS7 machine, you should create a file like this:

```
cat > overrides.yaml << EOF
---
# fips configuration
k8s_image_registry: docker.io/mesosphere

fips:
  enabled: true

build_name_extra: -fips
kubernetes_build_metadata: fips.0
default_image_repo: hub.docker.io/mesosphere
kubernetes_rpm_repository_url: "https://packages.d2iq.com/konvoy/stable/
linux/repos/el/kubernetes-v{{ kubernetes_version }}-fips/x86_64"
docker_rpm_repository_url: "\
  https://containerd-fips.s3.us-east-2.amazonaws.com\
  /{{ ansible_distribution_major_version|int }}\
  /x86_64"

# custom configuration
image_registries_with_auth:
- host: "registry-1.docker.io"
  username: "my-user"
  password: "my-password"
  auth: ""
  identityToken: ""

epel_centos_7_rpm: https://my-rpm-repostory.org/epel/epel-release-
latest-7.noarch.rpm
EOF
```

b. Example:
When using Oracle 7 OS, you may wish to deploy the RHCK kernel instead of the default UEK kernel. To do so, add the following text to your `overrides.yaml`:

```
cat > overrides.yaml << EOF
---
# fips configuration
k8s_image_registry: docker.io/mesosphere

fips:
  enabled: true

build_name_extra: -fips
kubernetes_build_metadata: fips.0
```

```
default_image_repo: hub.docker.io/mesosphere
kubernetes_rpm_repository_url: "https://packages.d2iq.com/konvoy/stable/
linux/repos/el/kubernetes-v{{ kubernetes_version }}-fips/x86_64"
docker_rpm_repository_url: "\
  https://containerd-fips.s3.us-east-2.amazonaws.com\
  /{{ ansible_distribution_major_version|int }}\
  /x86_64"

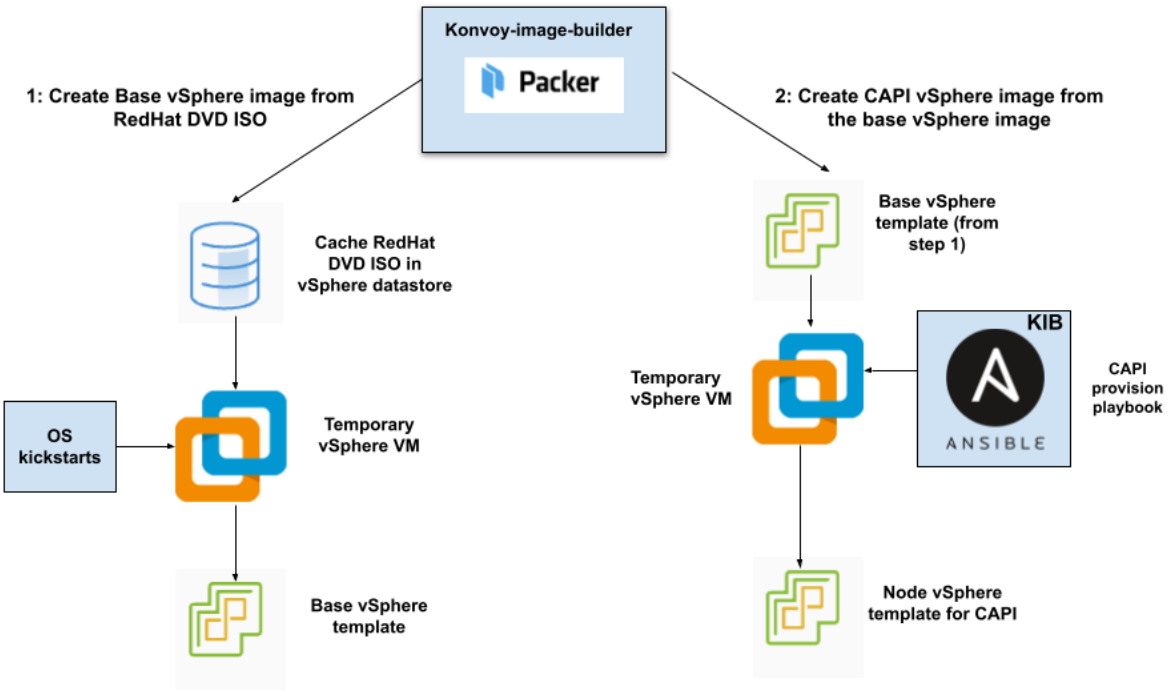
# custom configuration
oracle_kernel: RHCK
EOF
```

3. Create the related secret by running the following command:

```
kubectl create secret generic $CLUSTER_NAME-user-overrides --from-
file=overrides.yaml=overrides.yaml
kubectl label secret $CLUSTER_NAME-user-overrides clusterctl.cluster.x-k8s.io/
move=
```

7.14.8 Pre-provisioned Bootstrap Cluster

A bootstrap cluster refers to a special type of local Kubernetes cluster used to bootstrap other clusters. The bootstrap cluster is required because the controllers that create other Kubernetes clusters require a Kubernetes cluster to run.

Konvoy deploys all cluster lifecycle services to a bootstrap cluster, which deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster. The workload cluster then manages its own lifecycle.

7.14.8.1 Bootstrap a kind cluster and CAPI controllers

Use the following command to create a bootstrap cluster:

```
dkp create bootstrap
```

```
✓ Creating a bootstrap cluster
✓ Initializing new CAPI components
```

7.14.8.2 Next Step:

Pre-provisioned Create a New Cluster (see page 1098)

7.14.9 Pre-provisioned Create a New Cluster

7.14.9.1 Create a Kubernetes cluster using the infrastructure definition

Once you've defined the infrastructure (see page 1087) and control plane endpoints (see page 1090), you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster.

Before you create a new DKP cluster below, you may choose an external load balancer or virtual IP (see page 1098) and use the corresponding `dkp create cluster` command example from that page in the docs from the links below. Other customizations are available, but require different flags during `dkp create cluster` command also. Refer to corresponding section for custom cluster creation:

- Virtual IP (see page 1114)
- Provision on the Flatcar Linux OS (see page 1115)
- Use an HTTP proxy (see page 1115)
- Use alternate pod or service subnets (see page 1117)

DKP uses `localvolumeprovisioner` as the default storage provider (see page 103). However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[774] compatible storage that is suitable for production.

You can choose from any of the storage options[775] available for Kubernetes. To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your newly created StorageClass to be the default by following the commands in the Kubernetes documentation called Changing the Default Storage Class[776].

1. The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory.
    - **NOTE:** When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.
    - **NOTE:** (Optional) If you have overrides for your clusters[777], you must specify the secret as part of the create cluster command. If these are not specified, the overrides for your nodes will not

774 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
775 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
776 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/
777 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29918736

be applied.
`--override-secret-name=$CLUSTER_NAME-user-overrides`

- **NOTE:** To increase Docker Hub's rate limit[778] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.
- **NOTE:** Ensure your subnets (see page 1117) do not overlap with your host subnet because they cannot be changed after cluster creation. If you need to change the kubernetes subnets, you must do this at cluster creation. The default subnets used in DKP are:

```
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 192.168.0.0/16
    services:
      cidrBlocks:
      - 10.96.0.0/12
```

Create cluster command may take a while depending on the size of the cluster:

```
dkp create cluster preprovisioned
--cluster-name ${CLUSTER_NAME}
--control-plane-endpoint-host <control plane endpoint host>
--control-plane-endpoint-port <control plane endpoint port, if different than
6443>
--override-secret-name=$CLUSTER_NAME-user-overrides
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

The output from this command is shortened here for reading clarity, but should begin like this:

```
Generating cluster resources
cluster.cluster.x-k8s.io/preprovisioned-example created
cont.........
```

2. Use the `wait` command to monitor the cluster control-plane readiness:

778 https://docs.docker.com/docker-hub/download-rate-limit/

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

Output:

```
cluster.cluster.x-k8s.io/preprovisioned-example condition met
```

3. If your cluster is air-gapped or you have a local docker registry, you must provide additional arguments when creating the cluster. These tell the cluster where to locate the local Docker registry to use by defining the URL.

```
export DOCKER_REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
export DOCKER_REGISTRY_CA="<path to the CA on the bastion>"
export DOCKER_REGISTRY_USERNAME="<username>"
export DOCKER_REGISTRY_USERNAME="<password>"
```

- `DOCKER_REGISTRY_URL` : the address of an existing Docker registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- `DOCKER_REGISTRY_CA` : (optional) the path on the bastion machine to the Docker registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
- `DOCKER_REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
- `DOCKER_REGISTRY_PASSWORD` : optional if username is not set.

```
dkp create cluster preprovisioned --cluster-name ${CLUSTER_NAME} \
--control-plane-endpoint-host <control plane endpoint host> \
--control-plane-endpoint-port <control plane endpoint port, if different than 6443> \
--registry-mirror-url=${DOCKER_REGISTRY_URL} \
--registry-mirror-cacert=${DOCKER_REGISTRY_CA} \
--registry-mirror-username=${DOCKER_REGISTRY_USERNAME} \
--registry-mirror-password=${DOCKER_REGISTRY_PASSWORD}
```

Depending on the cluster's size, it will take a few minutes to create.

4. After the creation, use this command to get the Kubernetes kubeconfig for the new cluster and begin deploying workloads:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

Azure requires changing the CNI encapsulation type of Calico from the default of IPtoIP to VXlan. If changing the Calico encapsulation (see page 1110), D2iQ recommends changing it after cluster creation, but before production.

7.14.9.2 Audit logs

To modify Control Plane Audit logs settings using the information contained in the page Configuring the Control Plane (see page 1351).

7.14.9.2.1 Further Optional Steps:

- Azure Only Configurations (see page 1106)
- Modify the Calico installation (see page 1110)

7.14.9.2.2 Next Step:

When you complete this procedure, move on to Pre-provisioned Make Cluster Self-managed (see page 1118) to continue the process.

7.14.9.3 Pre-provisioned Air-gapped New Cluster

If your cluster is air-gapped or you have a local docker registry, you must provide additional arguments when creating the cluster. These tell the cluster where to locate the local Docker registry to use by defining the URL.

```
export DOCKER_REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
export DOCKER_REGISTRY_CA="<path to the CA on the bastion>"
export DOCKER_REGISTRY_USERNAME="<username>"
export DOCKER_REGISTRY_USERNAME="<password>"
```

- `DOCKER_REGISTRY_URL` : the address of an existing Docker registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- `DOCKER_REGISTRY_CA` : (optional) the path on the bastion machine to the Docker registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
- `DOCKER_REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
- `DOCKER_REGISTRY_PASSWORD` : optional if username is not set.

Before you create a new DKP cluster below, you may choose an external load balancer or virtual IP and use the corresponding `dkp create cluster` command example from that page in the docs from the links below. Other customizations are available, but require different flags during `dkp create cluster` command also. Refer to corresponding section for custom cluster creation:

- Virtual IP (see page 1114)
- Provision on the Flatcar Linux OS (see page 1101)
- Use an HTTP proxy (see page 1101)
- Use alternate pod or service subnets (see page 1101)
- Alternative Mirror (see page 1350)

7.14.9.3.1 Name Your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[779] for more naming information.

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=preprovisioned-example
```

Before you create a new DKP cluster below, choose an external load balancer or virtual IP (see page 1114) and use the corresponding `dkp create cluster` command.

779 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

7.14.9.3.2 Create an Air-gapped Kubernetes Cluster

Once you've defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster.

NOTE: (Optional) If you have overrides for your clusters (see page 1101), you must specify the secret as part of the create cluster command. If these are not specified, the overrides for your nodes will not be applied.

DKP uses `localvolumeprovisioner` as the default storage provider (see page 103) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[780] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory.

- **NOTE:** When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.
- **NOTE:** (Optional) If you have overrides for your clusters[781], you must specify the secret as part of the create cluster command. If these are not specified, the overrides for your nodes will not be applied. `--override-secret-name=$CLUSTER_NAME-user-overrides`
- **NOTE:** To increase Docker Hub's rate limit[782] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.
- **NOTE:** Ensure your subnets (see page 1117) do not overlap with your host subnet because they cannot be changed after cluster creation. If you need to change the kubernetes subnets, you must do this at cluster creation. The default subnets used in DKP are:

```
spec:
```

780 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
781 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29918736
782 https://docs.docker.com/docker-hub/download-rate-limit/

```
clusterNetwork:
  pods:
    cidrBlocks:
    - 192.168.0.0/16
  services:
    cidrBlocks:
    - 10.96.0.0/12
```

Create cluster command may take a while depending on the size of the cluster:

1. The command below uses the default external load balancer:

   ```
   dkp create cluster preprovisioned --cluster-name ${CLUSTER_NAME}
     --control-plane-endpoint-host <control plane endpoint host>
     --control-plane-endpoint-port <control plane endpoint port, if different than
   6443>
     --pre-provisioned-inventory-file preprovisioned_inventory.yaml
     --ssh-private-key-file <path-to-ssh-private-key>
     --registry-mirror-url=${DOCKER_REGISTRY_URL} \
     --registry-mirror-cacert=${DOCKER_REGISTRY_CA} \
     --registry-mirror-username=${DOCKER_REGISTRY_USERNAME} \
     --registry-mirror-password=${DOCKER_REGISTRY_PASSWORD}
   ```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

2. **Virtual IP ALTERNATIVE** - if you don't have an external LB, and wish to use a VIRTUAL IP provided by kube-vip, specify these flags example below:

   ```
   dkp create cluster preprovisioned \
       --cluster-name ${CLUSTER_NAME} \
       --control-plane-endpoint-host 196.168.1.10 \
       --virtual-ip-interface eth1
   ```

   The output from this command is shortened here for reading clarity, but should start like this:

   ```
   Generating cluster resources
   cluster.cluster.x-k8s.io/preprovisioned-example created
   cont.........
   ```

3. Use the wait command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

Output:

```
cluster.cluster.x-k8s.io/preprovisioned-example condition met
```

NOTE: Depending on the cluster size, it will take a few minutes to create.

When the command completes, you will have a running Kubernetes cluster! Use this command to get the Kubernetes `kubeconfig` for the new cluster and proceed to installing the DKP Kommander UI:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

Azure requires changing the CNI encapsulation type of Calico from the default of IPtoIP to VXlan. If changing the Calico encapsulation (see page 1110), D2iQ recommends changing it after cluster creation, but before production.

7.14.9.3.3 Audit logs

To modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1351).

7.14.9.3.3.1 Further Optional Steps:

- Azure Only Configurations (see page 1106)
- Modify the Calico installation (see page 1110)

7.14.9.3.3.2 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to Verify your Cluster and DKP Installation (see page 1360).

7.14.9.3.3.3 Next Step:

Pre-provisioned Make Cluster Self-managed (see page 1118)

7.14.10 Pre-provisioned Azure only Configurations

After your bootstrap is running and your cluster is created, you will need to install the Azure Disk CSI Driver[783] on your pre-provisioned Azure Kubernetes cluster. The DKP pre-provisioned provider installs by default the storage-local-static-provisioner[784] CSI driver, which is not suitable for production environments. For this reason, it needs to be replaced by the Azure Disk CSI Driver[785].

7.14.10.1 Prerequisites:

Before you begin using DKP you must have:

- An x86_64-based Linux or macOS machine.
- Download (see page 71) the `dkp` binary for Linux, or macOS. To check which version of DKP you installed for compatibility reasons, run the `dkp version -o` command (dkp version (see page 1534)).
- A Container engine/runtime installed is required to install DKP:
  - Version Docker®[786] container engine version 18.09.2 or higher installed for Linux or MacOS - On macOS, Docker runs in a virtual machine which needs configured with at least 8 GB of memory.
  - Version 4.0 of Podman[787] or higher for Linux. Host requirements found here: Host Requirements[788]
- kubectl[789] for interacting with the running cluster.
- Azure CLI[790].
- A valid Azure account with credentials configured[791].
- Create a custom Azure image using KIB (see page 1295).

On macOS, Docker runs in a virtual machine. Configure this virtual machine with at least 8GB of memory.

783 https://github.com/kubernetes-sigs/azuredisk-csi-driver/tree/master
784 https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner
785 https://github.com/kubernetes-sigs/azuredisk-csi-driver/blob/master/docs/install-azuredisk-csi-driver.md
786 https://docs.docker.com/get-docker/
787 https://podman.io/getting-started/installation
788 https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
789 https://kubernetes.io/docs/tasks/tools/#kubectl
790 https://docs.microsoft.com/en-us/cli/azure/install-azure-cli
791 https://github.com/kubernetes-sigs/cluster-api-provider-azure/blob/master/docs/book/src/topics/getting-started.md#prerequisites

7.14.10.2 Set Environment Variables with Credentials:

An Azure Service Principal is needed for deploying resources. To configure your Azure environment[792], follow below:

1. Log in to Azure:

```
az login
```

```
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "id": "b1234567-abcd-11a1-a0a0-1234a5678b90",
    "isDefault": true,
    "managedByTenants": [],
    "name": "Mesosphere Developer Subscription",
    "state": "Enabled",
    "tenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "user": {
      "name": "user@azuremesosphere.onmicrosoft.com",
      "type": "user"
    }
  }
]
```

2. Create an Azure Service Principal (SP) by running the following command:
   Note: If an SP with the name exists, this command will rotate the password.

```
az ad sp create-for-rbac --role contributor --name "$(whoami)-konvoy" --
scopes=/subscriptions/$(az account show --query id -o tsv)
```

```
{
  "appId": "7654321a-1a23-567b-b789-0987b6543a21",
  "displayName": "azure-cli-2021-03-09-23-17-06",
  "password": "Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C",
  "tenant": "a1234567-b132-1234-1a11-1234a5678b90"
}
```

792 https://github.com/kubernetes-sigs/cluster-api-provider-azure/blob/main/docs/book/src/topics/getting-started.md#setting-up-your-azure-environment

For air-gapped environments, you need to create a resource management private link[793] with a private endpoint to ensure the Azure CSI driver will run correctly in further steps. Private link resource can be deployed in different regions from the virtual network and private endpoint.
To set up a private link resource, use the following process.

a. Create the resource management private link using Azure CLI.
b. Create a private link association for the root management group which also references the resource ID for the resource management private link. Provide link association[794] for a management group.
c. Add a private endpoint that references the resource management private link. Create a Private Endpoint using the Azure Documentation[795].

3. Set the required environment variables using that output:

```
export AZURE_SUBSCRIPTION_ID="<id>"       # b1234567-abcd-11a1-
a0a0-1234a5678b90
export AZURE_TENANT_ID="<tenant>"         # a1234567-b132-1234-1a11-1234a5678b9
0
export AZURE_CLIENT_ID="<appId>"          # 7654321a-1a23-567b-
b789-0987b6543a21
export AZURE_CLIENT_SECRET="<password>"   # Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C
export AZURE_RESOURCE_GROUP="<resource group name>" # set to the name of the
resorce group
export AZURE_LOCATION="westus"            # set to the location you are using
```

4. Set your KUBECONFIG environment variable:

```
export kubeconfig=${CLUSTER_NAME}.conf
```

5. Create the Secret with the Azure credentials, this will be used by the Azure CSI driver:
   a. Create an `azure.json` file:

```
cat <<EOF > azure.json
{
  "cloud": "AzurePublicCloud",
  "tenantId": "$AZURE_TENANT_ID",
  "subscriptionId": "$AZURE_SUBSCRIPTION_ID",
  "aadClientId": "$AZURE_CLIENT_ID",
  "aadClientSecret": "$AZURE_CLIENT_SECRET",
  "resourceGroup": "$AZURE_RESOURCE_GROUP",
  "location": "$AZURE_LOCATION"
```

793 https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/create-private-link-access-commands?tabs=azure-cli#create-resource-management-private-link
794 https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/create-private-link-access-commands?tabs=azure-cli#create-private-link-association
795 https://learn.microsoft.com/en-us/azure/private-link/create-private-endpoint-cli?tabs=dynamic-ip

```
}
EOF
```

b. Create the Secret:

```
kubectl create secret generic azure-cloud-provider --namespace=kube-
system --type=Opaque --from-file=cloud-config=azure.json
```

6. Install the Azure Disk CSI driver:

```
$ curl -skSL https://raw.githubusercontent.com/kubernetes-sigs/azuredisk-csi-
driver/v1.26.2/deploy/install-driver.sh | bash -s v1.26.2 snapshot -
```

7. Check the status to see if the driver is ready for use:

```
kubectl -n kube-system get pod -o wide --watch -l app=csi-azuredisk-controller
kubectl -n kube-system get pod -o wide --watch -l app=csi-azuredisk-node
```

8. Now Kubernetes knows that this is Azure disk, and will create clusters on Azure. You are ready to create the StorageClass for the Azure Disk CSI Driver:

```
kubectl create -f https://raw.githubusercontent.com/kubernetes-sigs/azuredisk-
csi-driver/master/deploy/example/storageclass-azuredisk-csi.yaml
```

9. Change the default storage class to this new StorageClass so that every new disk will be created in the Azure environment:

```
kubectl patch sc/localvolumeprovisioner -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class":"false"}}}'
kubectl patch sc/managed-csi -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

10. Verify that the StorageClass chosen is currently the default:

```
kubectl get storageclass
```

For more information about Azure Disk CSI for persistent storage and changing the default StorageClass, refer to that page in the documentation: Default Storage Providers in DKP (see page 103)

7.14.10.3 Next Step:

Pre-provisioned Modify the Calico Installation (see page 1110)

7.14.11 Pre-provisioned Modify the Calico Installation

7.14.11.1 Set the Interface

By default, Calico automatically detects the IP address to use for each node using the `first-found` method[796]. This is not always appropriate for your particular nodes. In that case, you must modify Calico's configuration to use a different method. An alternative is to use the `interface` method by providing the interface ID. Follow the steps outlined in this section to modify Calico's configuration.

Azure does **not** set the interface. Proceed to Change the Encapsulation Type (see page 1112) section below.

In this example, all cluster nodes use `ens192` as the interface name.

1. Get the pods running on your cluster with this command:

```
kubectl get pods -A --kubeconfig ${CLUSTER_NAME}.conf
```

```
NAMESPACE                NAME
READY   STATUS            RESTARTS        AGE
calico-system            calico-kube-controllers-57fbd7bd59-vpn8b
1/1     Running           0               16m
calico-system            calico-node-5tbvl
1/1     Running           0               16m
calico-system            calico-node-nbdwd
1/1     Running           0               4m40s
calico-system            calico-node-twl6b
0/1     PodInitializing   0               9s
calico-system            calico-node-wktkh
1/1     Running           0               5m35s
calico-system            calico-typha-54f46b998d-52pt2
1/1     Running           0               16m
calico-system            calico-typha-54f46b998d-9tzb8
1/1     Running           0               4m31s
default                  cuda-vectoradd
0/1     Pending           0               0s
kube-system              coredns-78fcd69978-frwx4
1/1     Running           0               16m
```

796 https://projectcalico.docs.tigera.io/reference/node/configuration#ip-autodetection-methods

```
kube-system                coredns-78fcd69978-kkf44
1/1     Running            0                16m
kube-system                etcd-ip-10-0-121-16.us-west-2.compute.internal
0/1     Running            0                8s
kube-system                etcd-ip-10-0-46-17.us-west-2.compute.internal
1/1     Running            1                16m
kube-system                etcd-ip-10-0-88-238.us-west-2.compute.internal
1/1     Running            1                5m35s
kube-system                kube-apiserver-ip-10-0-121-16.us-west-2.compute.internal
0/1     Running            6                7s
kube-system                kube-apiserver-ip-10-0-46-17.us-west-2.compute.internal
1/1     Running            1                16m
kube-system                kube-apiserver-ip-10-0-88-238.us-west-2.compute.internal
1/1     Running            1                5m34s
kube-system                kube-controller-manager-ip-10-0-121-16.us-west-2.compute.int
ernal   0/1     Running            0                7s
kube-system                kube-controller-manager-ip-10-0-46-17.us-west-2.compute.inte
rnal    1/1     Running            1 (5m25s ago)    15m
kube-system                kube-controller-manager-ip-10-0-88-238.us-west-2.compute.int
ernal   1/1     Running            0                5m34s
kube-system                kube-proxy-gclmt
1/1     Running            0                16m
kube-system                kube-proxy-gptd4
1/1     Running            0                9s
kube-system                kube-proxy-mwkgl
1/1     Running            0                4m40s
kube-system                kube-proxy-zcqxd
1/1     Running            0                5m35s
kube-system                kube-scheduler-ip-10-0-121-16.us-west-2.compute.internal
0/1     Running            1                7s
kube-system                kube-scheduler-ip-10-0-46-17.us-west-2.compute.internal
1/1     Running            3 (5m25s ago)    16m
kube-system                kube-scheduler-ip-10-0-88-238.us-west-2.compute.internal
1/1     Running            1                5m34s
kube-system                local-volume-provisioner-2mv7z
1/1     Running            0                4m10s
kube-system                local-volume-provisioner-vdcrg
1/1     Running            0                4m53s
kube-system                local-volume-provisioner-wsjrt
1/1     Running            0                16m
node-feature-discovery     node-feature-discovery-master-84c67dcbb6-m78vr
1/1     Running            0                16m
node-feature-discovery     node-feature-discovery-worker-vpvpl
1/1     Running            0                4m10s
tigera-operator            tigera-operator-d499f5c8f-79dc4
1/1     Running            1 (5m24s ago)    16m
```

If a `calico-node` pod is not ready on your cluster, you must edit the `default` Installation resource. To edit the Installation resource, run the command:

```
kubectl edit installation default --kubeconfig ${CLUSTER_NAME}.conf
```

2. Change the value for `spec.calicoNetwork.nodeAddressAutodetectionV4` to `interface: ens192`, and save the resource:

```
spec:
  calicoNetwork:
  ...
    nodeAddressAutodetectionV4:
      interface: ens192
```

3. Save this resource. You may need to delete the node feature discovery worker pod in the `node-feature-discovery` namespace if that pod has failed. After you delete it, Kubernetes replaces the pod as part of its normal reconciliation.

7.14.11.2 Change the Encapsulation Type

Calico can leverage different network encapsulation methods to route traffic for your workloads. Encapsulation is useful when running on top of an underlying network that is not aware of workload IPs. Common examples of this include:

- Public cloud environments where you don't own the hardware.
- AWS across VPC subnet boundaries.
- Environments where you cannot peer Calico over BGP to the underlay or easily configure static routes.

WARNING: Switching encapsulation modes can cause disruption to in-progress connections. You can do this safely when the cluster is first deployed. However, if user workloads are already running on the cluster, plan accordingly for interruption.

7.14.11.3 Provider Specific Settings

The encapsulation type to be used for networking depends on the cloud provider. IP-in-IP is the default encapsulation method for Calico which most providers use, but not Azure.

Azure only supports **VXLAN** encapsulation type. Therefore, if you install on Azure pre-provisioned VMs, you must set the encapsulation mode to VXLAN.

D2iQ recommends changing the encapsulation type after cluster creation but before production using the method below. To change the encapsulation type, follow these steps:

1. First, remove the existing `default-ipv4-ippool` IPPool resource from `kubeconfig`. The resource must be deleted, so that it can re-create after you edit the Installation resource. Execute the command below to delete:

```
kubectl delete ippool default-ipv4-ippool
```

2. Run the following command to edit:

```
kubectl edit installation default --kubeconfig ${CLUSTER_NAME}.conf
```

3. Change the value for encapsulation `- encapsulation:` as shown below:

```
  spec:
  calicoNetwork:
    ipPools:
      - encapsulation: VXLAN
```

7.14.11.3.1 VXLAN

VXLAN is a tunneling protocol that encapsulates layer 2 Ethernet frames in UDP packets, enabling you to create virtualized layer 2 subnets that span Layer 3 networks. It has a slightly larger header than IP-in-IP which creates a slight reduction in performance over IP-in-IP.

7.14.11.3.2 IPIP

IP-in-IP is an IP tunneling protocol that encapsulates one IP packet in another IP packet. An outer packet header is added with the tunnel entry point and the tunnel exit point. The calico implementation of this protocol uses BGP to determine the exit point making this protocol unusable on networks that don't pass BGP.

For more information, see:

- Calico Overlay Networking[797]
- Calico Routing for VXLAN[798]

797 https://docs.projectcalico.org/networking/vxlan-ipip
798 https://joshrosso.com/docs/2020/2020-10-01-calico-routing-modes/

- IP-in-IP RFC 2003[799]
- VXLAN RFC 7348[800]

If using Windows, see this documentation on Calico site regarding limitations: Calico for Windows VXLAN[801]

7.14.12 Pre-provisioned Built-in Virtual IP

As explained in Define the Control Plane Endpoint (see page 129), we recommend using an external load balancer for the control plane endpoint, but provide a built-in virtual IP when an external load balancer is not available. The built-in virtual IP uses the kube-vip[802] project. To use the virtual IP, add these flags to the `create cluster` command:

Virtual IP Configuration	Flag
Network interface to use for Virtual IP. *Must exist on all control plane machines.*	`--virtual-ip-interface string`
IPv4 address. *Reserved for use by the cluster.*	`--control-plane-endpoint string`

7.14.12.1 Virtual IP example

```
dkp create cluster preprovisioned \
    --cluster-name ${CLUSTER_NAME} \
    --control-plane-endpoint-host 196.168.1.10 \
    --virtual-ip-interface eth1
    --pre-provisioned-inventory-file preprovisioned_inventory.yaml
    --ssh-private-key-file <path-to-ssh-private-key>
    --self-managed
```

799 https://datatracker.ietf.org/doc/html/rfc2003
800 https://datatracker.ietf.org/doc/html/rfc7348
801 https://projectcalico.docs.tigera.io/getting-started/windows-calico/limitations#calico-vxlan-networking-limitations
802 https://kube-vip.io/

7.14.13 Provision on the Flatcar Linux OS

When provisioning onto the Flatcar Container Linux distribution, you must instruct the bootstrap cluster to make some changes related to the installation paths. To accomplish this, add the `--os-hint flatcar` flag to the above `create cluster` command.

7.14.13.1 Flatcar Linux example

```
dkp create cluster preprovisioned \
    --cluster-name ${CLUSTER_NAME} \
    --os-hint flatcar
```

For provisioning DKP on Flatcar, DKP configures cluster nodes to use Control Groups (cgroups) version 1[803]. In versions prior to Flatcar 3033.3.x, a restart is required in order to apply the changes to the kernel. For more information, refer to the Flatcar documentation[804].

7.14.14 Pre-provisioned Use HTTP Proxy

If you require HTTP proxy configurations, you can apply them during the `create` operation by adding the appropriate flags to the `create cluster` command:

Proxy configuration	Flag
HTTP proxy for control plane machines	`--control-plane-http-proxy string`
HTTPS proxy for control plane machines	`--control-plane-https-proxy string`
No Proxy list for control plane machines	`--control-plane-no-proxy strings`

803 https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v1/cgroups.html#what-are-cgroups

804 https://www.flatcar.org/docs/latest/container-runtimes/switching-to-unified-cgroups/#starting-new-nodes-with-legacy-cgroups

Proxy configuration	Flag
HTTP proxy for worker machines	`--worker-http-proxy string`
HTTPS proxy for worker machines	`--worker-https-proxy string`
No Proxy list for worker machines	`--worker-no-proxy strings`

You must also add the same configuration as an override (see page 1341). For more information, refer to this documentation (see page 1336).

7.14.14.1 HTTP proxy example

To increase Docker Hub's rate limit[805] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.

```
dkp create cluster preprovisioned \
    --cluster-name ${CLUSTER_NAME} \
    --control-plane-http-proxy http://proxy.example.com:8080 \
    --control-plane-https-proxy https://proxy.example.com:8080 \
    --control-plane-no-proxy
"127.0.0.1,10.96.0.0/12,192.168.0.0/16,kubernetes,kubernetes.default.svc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.cluster,.svc.cluster.local" \
    --worker-http-proxy http://proxy.example.com:8080 \
    --worker-https-proxy https://proxy.example.com:8080 \
    --worker-no-proxy
"127.0.0.1,10.96.0.0/12,192.168.0.0/16,kubernetes,kubernetes.default.svc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.cluster,.svc.cluster.local"
```

805 https://docs.docker.com/docker-hub/download-rate-limit/

7.14.15 Pre-provisioned Use Alternate Pod or Service Subnets

In Konvoy, the default pod subnet is 192.168.0.0/16, and the default service subnet is 10.96.0.0/12. If you wish to change the subnets you can do so with the following steps:

1. Generate the YAML manifests for the cluster using the `--dry-run` and `-o yaml` flags, along with the desired `dkp cluster create` command:

```
dkp create cluster preprovisioned --cluster-name ${CLUSTER_NAME} --control-
plane-endpoint-host <control plane endpoint host> --control-plane-endpoint-port
<control plane endpoint port, if different than 6443> --dry-run -o yaml >
cluster.yaml
```

2. To modify the service subnet, add or edit the `spec.clusterNetwork.services.cidrBlocks` field of the `Cluster` object:

```
kind: Cluster
spec:
  clusterNetwork:
    services:
      cidrBlocks:
      - 10.0.0.0/18
```

3. To modify the pod subnet, edit the `Cluster` and calico-cni `ConfigMap` resources:

   Cluster: Add or edit the `spec.clusterNetwork.pods.cidrBlocks` field:

```
kind: Cluster
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 172.16.0.0/16
```

   ConfigMap: Edit the `data."custom-resources.yaml".spec.calicoNetwork.ipPools.cidr` field with your desired pod subnet:

```
apiVersion: v1
data:
  custom-resources.yaml: |
    apiVersion: operator.tigera.io/v1
    kind: Installation
    metadata:
```

```
      name: default
    spec:
      # Configures Calico networking.
      calicoNetwork:
        # Note: The ipPools section cannot be modified post-install.
        ipPools:
        - blockSize: 26
          cidr: 172.16.0.0/16
kind: ConfigMap
metadata:
  name: calico-cni-<cluter-name>
```

When you provision the cluster, the configured pod and service subnets will be applied.

7.14.16 Pre-provisioned Make Cluster Self-managed

Make the new Kubernetes cluster manage itself

Konvoy deploys all cluster lifecycle services to a bootstrap cluster, which then deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster, which is now self-managed. This guide describes how to make a workload cluster self-managed.

This page contains instructions on how to make your cluster self-managed. This is necessary if there is only one cluster in your environment, or if this cluster should become the Management cluster in a multi-cluster environment.

> If you already have a self-managed or Management cluster in your environment, skip this page.

7.14.16.1 Make the new Kubernetes cluster manage itself

> If you have not already retrieved the kubeconfig after creating the cluster, use this command before proceeding: `dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf`

1. Deploy cluster lifecycle services on the workload cluster:

   ```
   dkp create capi-components --kubeconfig ${CLUSTER_NAME}.conf
   ```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful.

More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).
The output is similar to this:

```
✓ Initializing new CAPI components
```

2. Move the Cluster API objects from the bootstrap to the workload cluster:

   The cluster lifecycle services on the workload cluster are ready, but the workload cluster configuration is on the bootstrap cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the bootstrap to the workload cluster. This process is also called a Pivot[806]. First unset the kubeconfig and then move the CAPI:

   ```
   unset KUBECONFIG
   ```

   ```
   dkp move capi-resources --to-kubeconfig ${CLUSTER_NAME}.conf
   ```

   Output:

   ```
   ✓ Moving cluster resources
   You can now view resources in the moved cluster by using the --kubeconfig flag
   with kubectl. For example: kubectl --kubeconfig=preprovisioned-example.conf get
   nodes
   ```

   NOTE: To ensure only one set of cluster lifecycle services manages the workload cluster, Konvoy first pauses reconciliation of the objects on the bootstrap cluster, then creates the objects on the workload cluster. As Konvoy copies the objects, the cluster lifecycle services on the workload cluster reconcile the objects. The workload cluster becomes self-managed after Konvoy creates all the objects. If it fails, the `move` command can be safely retried.

3. Wait for the cluster control-plane to be ready:

   ```
   kubectl --kubeconfig ${CLUSTER_NAME}.conf wait --for=condition=ControlPlaneRead
   y "clusters/${CLUSTER_NAME}" --timeout=20m
   ```

   ```
   cluster.cluster.x-k8s.io preprovisioned-example condition met
   ```

4. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

806 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

NOTE: After moving the cluster lifecycle services to the workload cluster, remember to use Konvoy with the workload cluster kubeconfig.

```
dkp describe cluster --kubeconfig ${CLUSTER_NAME}.conf -c ${CLUSTER_NAME}
```

```
NAME
READY  SEVERITY  REASON  SINCE  MESSAGE
Cluster/preprovisioned-example                                                 True
2m31s
├──ClusterInfrastructure - PreprovisionedCluster/preprovisioned-example
├──ControlPlane - KubeadmControlPlane/preprovisioned-example-control-plane
True                     2m31s
│  ├──Machine/preprovisioned-example-control-plane-6g6nr
True                     2m33s
│  ├──Machine/preprovisioned-example-control-plane-8lhcv
True                     2m33s
│  └──Machine/preprovisioned-example-control-plane-kk2kg
True                     2m33s
└──Workers
   └──MachineDeployment/preprovisioned-example-md-0
True                     2m34s
     └──Machine/preprovisioned-example-md-0-77f667cd9-tnctd
True                     2m33s
```

5. Remove the bootstrap cluster, as the workload cluster is now self-managed:

```
dkp delete bootstrap
```

```
✓ Deleting bootstrap cluster
```

7.14.16.1.1 Known limitations

Be aware of these limitations in the current release of Konvoy.

- DKP supports moving only one set of cluster objects from the bootstrap cluster to the workload cluster, or vice-versa.

- DKP only supports moving all namespaces in the cluster; DKP does not support migration of individual namespaces.

7.14.17 Pre-provisioned Configure MetalLB

7.14.17.1 Create a MetalLB configmap for your pre-provisioned infrastructure.

Choose one of the following two protocols you want to use to announce service IPs. If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can continue the installation process.

To use MetalLB, create a MetalLB configMap for your Pre-provisioned infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

7.14.17.2 Layer 2 configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to ARP requests on your local network directly, to give the machine's MAC address to clients.

- MetalLB IP address ranges/CIDRs should be within the node's primary network subnet (see page 929).
- MetalLB IP address ranges/CIDRs and node subnet should not conflict with the Kubernetes cluster pod and service subnets.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
```

```
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

7.14.17.3 BGP configuration

For a basic configuration featuring one BGP router and one IP address range, you need 4 pieces of information:

- The router IP address that MetalLB should connect to,
- The router's AS number,
- The AS number MetalLB should use,
- An IP address range expressed as a CIDR prefix.

As an example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
```

```
      addresses:
      - 192.168.10.0/24
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

7.14.18 Pre-provisioned Create and Delete Node Pools

Node pools are part of a cluster and managed as a group, and can be used to manage a group of machines using common properties. New default clusters created by Konvoy contain one node pool of worker nodes that have the same configuration.

You can create additional node pools for specialized hardware or other configurations. For example, if you want to tune your memory usage on a cluster where you need maximum memory for some machines and minimal memory on others, you could create a new node pool with those specific resource needs.

NOTE: Konvoy implements node pools using Cluster API MachineDeployments[807].

7.14.18.1 Create a Pre-provisioned node pool

Follow these steps:

1. Create an inventory object that has the same name as the node pool you're creating, and the details of the pre-provisioned machines that you want to add to it. For example, to create a node pool named `gpu-nodepool` an inventory named `gpu-nodepool` must be present in the same namespace:

```
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: ${MY_NODEPOOL_NAME}
spec:
  hosts:
    - address: ${IP_OF_NODE}
  sshConfig:
    port: 22
    user: ${SSH_USERNAME}
    privateKeyRef:
      name: ${NAME_OF_SSH_SECRET}
```

807 https://cluster-api.sigs.k8s.io/developer/architecture/controllers/machine-deployment.html

```
      namespace: ${NAMESPACE_OF_SSH_SECRET}
```

2. (Optional) If your pre-provisioned machines have overrides (see page 1330), you must create a secret that includes all of the overrides you want to provide in one file. Create an override secret (see page 1091) using the instructions detailed on this page.
3. Once the `PreprovisionedInventory` object and overrides are created, create a node pool:

```
dkp create nodepool preprovisioned -c ${MY_CLUSTER_NAME} ${MY_NODEPOOL_NAME} --
override-secret-name ${MY_OVERRIDE_SECRET}
```

- Advanced users can use a combination of the `--dry-run` and `--output=yaml` or `--output-directory=<existing-directory>` flags to get a complete set of node pool objects to modify locally or store in version control.

For more information regarding this flag or others, please refer to the dkp create nodepool (see page 1472) section of the documentation for either cluster or nodepool and select your provider.

7.14.18.2 Delete a node pool

Deleting a node pool deletes both the Kubernetes nodes and their underlying infrastructure. DKP drains all nodes prior to deletion and reschedules the pods running on those nodes.

To delete a node pool from a managed cluster, run the following command:

```
dkp delete nodepool ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME}
```

The expected output is similar to the following example, indicating the `example` node pool is being deleted:

```
INFO[2021-07-28T17:14:26-07:00] Running nodepool delete command
Nodepool=example clusterName=d2iq-e2e-cluster-1 managementClusterKubeconfig=
namespace=default src="nodepool/delete.go:80"
```

Deleting an invalid node pool results in an output similar to this example command output:

```
dkp delete nodepool ${CLUSTER_NAME}-md-invalid --cluster-name=${CLUSTER_NAME}

INFO[2021-07-28T17:11:44-07:00] Running nodepool delete command
Nodepool=demo-cluster-md-invalid clusterName=d2iq-e2e-cluster-1
 managementClusterKubeconfig= namespace=default src="nodepool/delete.go:80"
```

```
Error: failed to get nodepool with name demo-cluster-md-invalid in namespace default
: failed to get nodepool with name demo-cluster-md-invalid in namespace default :
machinedeployments.cluster.x-k8s.io "demo-cluster-md-invalid" not found
```

7.14.19 Pre-provisioned Add Nodes to Existing Node Pool

This sections covers the prerequisites and procedure you need to scale-up or scale-down nodes in an existing DKP cluster.

7.14.19.1 Prerequisites

- You must have the bootstrap node running with the SSH key/secrets created.
- The export values in the environment variables section should contain the addresses of the nodes that you need to add Pre-provisioned: Define Infrastructure (see page 127).
- Update the `preprovisioned_inventory.yaml` with the new host addresses.
- Run the `kubectl apply` command.

7.14.19.2 Scale up a Cluster Node

Follow these steps:

1. Fetch the existing `preprovisioned_inventory`:

   ```
   $ kubectl get preprovisionedinventory
   ```

2. Edit the `preprovisioned_inventory` to add additional IPs needed for additional worker nodes in the `spec.hosts:` section:

   ```
   $ kubectl edit preprovisionedinventory <preprovisioned_inventory> -n default
   ```

3. Add any additional IPs that you require:

   ```
   spec:
       hosts:
       - address: <worker.ip.add.1>
       - address: <worker.ip.add.2>
   ```

   After you edit `preprovisioned_inventory`, fetch the machine deployment. The naming convention with `md` means that it is for worker machines.
   For example:

```
$ kubectl --kubeconfig ${CLUSTER_NAME}.conf get machinedeployment
NAME                      CLUSTER        AGE     PHASE     REPLICAS   READY
UPDATED   UNAVAILABLE
machinedeployment-md-0    cluster-name   9m10s   Running   4          4       4
```

4. Scale the worker node to the required number. In this example we are scaling from 4 to 6 worker nodes:

```
$ kubectl --kubeconfig ${CLUSTER_NAME}.conf scale --replicas=6 machinedeploymen
t machinedeployment-md-0

machinedeployment.cluster.x-k8s.io/machinedeployment-md-0 scaled
```

5. Monitor the scaling with this command, by adding `-w` option to watch:

```
$ kubectl --kubeconfig ${CLUSTER_NAME}.conf get machinedeployment -w

NAME                      CLUSTER        AGE   PHASE       REPLICAS   READY
UPDATED   UNAVAILABLE
machinedeployment-md-0    cluster-name   20m   ScalingUp   6          4       6
       2
```

6. Also you can check the machine deployment if it is already scaled. The output should resemble this example:

```
$ kubectl --kubeconfig ${CLUSTER_NAME}.conf get machinedeployment

NAME                      CLUSTER        AGE     PHASE     REPLICAS   READY
UPDATED   UNAVAILABLE
machinedeployment-md-0    cluster-name   3h33m   Running   6          6       6
```

7. Alternately, you can use this command and verify the `NODENAME` column and you should see the additional worker nodes added and in `Running` state:

```
$ kubectl --kubeconfig ${CLUSTER_NAME}.conf get machines -o wide

NAME     CLUSTER     AGE     PROVIDERID     PHASE     VERSION     NODENAME
```

7.14.19.3 Scale Down a Cluster Node

Follow these steps

1. Run this command on your worker nodes:

```
kubectl scale machinedeployment <machinedeployment-name> --replicas <new
 number>
```

2. For control plane nodes, execute the following command:

```
kubectl scale kubeadmcontrolplane ${CLUSTER_NAME}-control-plane --replicas <new
 number>
```

7.14.19.3.1 Additional Notes for Scaling Down

It is possible for machines to get stuck in the provisioning stage when you scaling down. You can utilize a delete operation to clear the stale machine deployment:

```
kubectl delete machine ${CLUSTER_NAME}-control-plane-<hash>
```

```
kubectl delete machine <machinedeployment-name>-<hash>
```

7.14.20 GPU Nodepools in a Pre-provisioned Environment

DKP has introduced the `nvidia-runfile` flag for Air-gapped Pre-provisioned environments. If the NVIDIA runfile[808] installer has not been downloaded, then retrieve and install the download first by running the following command. The first line in the command below downloads and installs the runfile and the second line places it in the artifacts directory (you must create an `artifacts` directory if it doesn't already exist).

```
curl -O https://download.nvidia.com/XFree86/Linux-x86_64/470.82.01/NVIDIA-Linux-
x86_64-470.82.01.run
mv NVIDIA-Linux-x86_64-470.82.01.run artifacts
```

DKP supported NVIDIA driver[809] version is 470.x.

808 https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html#runfile
809 https://www.nvidia.com/Download/Find.aspx

1. Create the secret that GPU nodepool would use, this secret is populated from the KIB overrides. In this example we have a file called, `overrides/nvidia.yaml`. It should resemble this:

```
gpu:
  types:
    - nvidia
build_name_extra: "-nvidia"
```

2. Create a secret on the bootstrap cluster that is populated from the above file. We will name it `${CLUSTER_NAME}-user-overrides`

```
kubectl create secret generic ${CLUSTER_NAME}-user-overrides --from-
file=overrides.yaml=overrides/nvidia.yaml
```

3. Create an inventory and nodepool with the instructions below and use the `$CLUSTER_NAME-user-overrides` secret.

Follow these steps:

1. Create an inventory object that has the same name as the node pool you're creating, and the details of the pre-provisioned machines that you want to add to it. For example, to create a node pool named `gpu-nodepool` an inventory named `gpu-nodepool` must be present in the same namespace:

```
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: ${MY_NODEPOOL_NAME}
spec:
  hosts:
    - address: ${IP_OF_NODE}
  sshConfig:
    port: 22
    user: ${SSH_USERNAME}
    privateKeyRef:
      name: ${NAME_OF_SSH_SECRET}
      namespace: ${NAMESPACE_OF_SSH_SECRET}
```

2. (Optional) If your pre-provisioned machines have overrides (see page 1330), you must create a secret that includes all of the overrides you want to provide in one file. Create an override secret (see page 1091) using the instructions detailed on this page.

3. Once the `PreprovisionedInventory` object and overrides are created, create a node pool:

```
dkp create nodepool preprovisioned -c ${MY_CLUSTER_NAME} ${MY_NODEPOOL_NAME} --
override-secret-name ${MY_OVERRIDE_SECRET}
```

- Advanced users can use a combination of the `--dry-run` and `--output=yaml` or `--output-directory=<existing-directory>` flags to get a complete set of node pool objects to modify locally or store in version control.

> For more information regarding this flag or others, please refer to the dkp create nodepool (see page 1472) section of the documentation for either cluster or nodepool and select your provider.

For more information, see:

- Pre-provisioned Create and Delete Node Pools (see page 1123)
- Pre-provisioned Air-gapped Define Environment (see page 1082)

7.14.21 Pre-provisioned Delete Cluster

> **NOTE:** A self-managed workload cluster cannot delete itself. If your workload cluster is self-managed, you must create a bootstrap cluster and move the cluster lifecycle services to the bootstrap cluster before deleting the workload cluster.

If you did not make your workload cluster self-managed, as described in Make New Cluster Self-Managed (see page 1118), see Delete the workload cluster (see page 1131).

7.14.21.1 Prepare to Delete the Pre-provisioned Cluster

Follow these steps:

1. Create a bootstrap cluster:

   The bootstrap cluster will host the Cluster API controllers that reconcile the cluster objects marked for deletion:

   NOTE: To avoid using the wrong kubeconfig, the following steps use explicit kubeconfig paths and contexts.

   ```
   dkp create bootstrap --kubeconfig $HOME/.kube/config
   ```

   ```
   ✓ Creating a bootstrap cluster
   ✓ Initializing new CAPI components
   ```

2. Move the Cluster API objects from the workload to the bootstrap cluster: The cluster lifecycle services on the bootstrap cluster are ready, but the workload cluster configuration is on the workload cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the workload to the bootstrap cluster. This process is also called a Pivot[810].

```
dkp move capi-resources \
    --from-kubeconfig ${CLUSTER_NAME}.conf \
    --from-context ${CLUSTER_NAME}-admin@${CLUSTER_NAME} \
    --to-kubeconfig $HOME/.kube/config \
    --to-context kind-konvoy-capi-bootstrapper
```

```
✓ Moving cluster resources
You can now view resources in the moved cluster by using the --kubeconfig flag
with kubectl. For example: kubectl --kubeconfig $HOME/.kube/config get nodes
```

3. Use the cluster lifecycle services on the workload cluster to check the workload cluster status by running the following command:

```
dkp describe cluster --kubeconfig $HOME/.kube/config -c ${CLUSTER_NAME}
```

```
NAME
READY  SEVERITY  REASON  SINCE  MESSAGE
Cluster/preprovisioned-example                                          True
2m31s
├──ClusterInfrastructure - PreprovisionedCluster/preprovisioned-example
├──ControlPlane - KubeadmControlPlane/preprovisioned-example-control-plane
True                     2m31s
│  ├──Machine/preprovisioned-example-control-plane-6g6nr
True                     2m33s
│  ├──Machine/preprovisioned-example-control-plane-8lhcv
True                     2m33s
│  └──Machine/preprovisioned-example-control-plane-kk2kg
True                     2m33s
└──Workers
   └──MachineDeployment/preprovisioned-example-md-0
True                     2m34s
     └──Machine/preprovisioned-example-md-0-77f667cd9-tnctd
True                     2m33s
```

810 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

NOTE: After moving the cluster lifecycle services to the workload cluster, remember to use dkp with the workload cluster kubeconfig.

4. Wait for the cluster control-plane to be ready. Run the command below and wait for the condition to be met:

```
kubectl --kubeconfig $HOME/.kube/config wait --for=condition=controlplaneready
"clusters/${CLUSTER_NAME}" --timeout=20m
```

```
cluster.cluster.x-k8s.io/preprovisioned-example condition met
```

Persistent Volumes (PVs) are not deleted automatically by design in order to preserve your data. However, they take up storage space if not deleted. You must delete PVs manually. Information for backup of a cluster and PVs is on the page in documentation called Back up your Cluster's Applications and Persistent Volumes (see page 736) .

7.14.21.2 Delete the Workload Cluster

If you have a need to remove the Kubernetes cluster, such as for environment cleanup, use this command to delete the provisioned Kubernetes cluster.

1. To delete a cluster, you would use `dkp delete cluster` and pass in the name of the cluster you are trying to delete with `--cluster-name` flag. You would use `kubectl get clusters` to get those details (`--cluster-name` and `--namespace`) of the Kubernetes cluster to delete it. NOTE: Do not use `dkp get clusters` since that gets you Kommander cluster details rather than Konvoy kubernetes cluster details.

```
kubectl get clusters
```

2. Delete the Kubernetes cluster and wait a few minutes:

   NOTE: Before deleting the cluster, dkp deletes all Services of type LoadBalancer on the cluster. Each Service is backed by an AWS Classic ELB. Deleting the Service deletes the ELB that backs it. To skip this step, use the flag `--delete-kubernetes-resources=false` . Do not skip this step if the VPC is managed by DKP. When DKP deletes the cluster, it deletes the VPC. If the VPC has any AWS Classic ELBs, AWS does not allow the VPC to be deleted, and DKP cannot delete the cluster.

```
dkp delete cluster --cluster-name=${CLUSTER_NAME} --kubeconfig $HOME/.kube/
config
```

```
✓ Deleting Services with type LoadBalancer for Cluster default/preprovisioned-
example
✓ Deleting ClusterResourceSets for Cluster default/preprovisioned-example
✓ Deleting cluster resources
✓ Waiting for cluster to be fully deleted
Deleted default/preprovisioned-example cluster
```

7.14.21.2.1 Delete the Bootstrap Cluster

After you have moved the workload resources back to a bootstrap cluster and deleted the workload cluster, you no longer need the bootstrap cluster. You can safely delete the bootstrap cluster with this command:

1. Use `dkp` with the bootstrap cluster to delete the workload cluster.

Delete the `kind` Kubernetes cluster:

```
dkp delete bootstrap --kubeconfig $HOME/.kube/config
```

```
✓ Deleting bootstrap cluster
```

7.15 vSphere Infrastructure

7.15.1 Creating DKP clusters in a VMware vSphere environment

The overall process for configuring vSphere and DKP together includes the following steps:

1. Configure vSphere to provide the needed elements, described in the Prerequisites (see page 1133).
2. Create a bastion VM host (see page 1162) if you are using an air-gapped environment.
3. Create a base OS image (for use in the OVA package containing the disk images packaged with the OVF).
4. Create a CAPI VM image template that uses the base OS image and adds the needed Kubernetes cluster components.
5. Create a bootstrap cluster.
6. Create a new DKP cluster on vSphere.
7. Make the cluster self-managing.

8. Explore the cluster and perform other functions as needed.

This diagram illustrates the image creation process:



The workflow on the left shows the creation of a base OS image in the vCenter vSphere client using inputs from Packer. The workflow on the right shows how DKP uses that same base OS image to create CAPI-enabled VM images for your cluster.

After creating the base image, the Konvoy Image Builder (see page 1282) uses it to create a CAPI-enabled vSphere template that includes the Kubernetes objects for the cluster. You can use that resulting template with the DKP `create cluster` command to create the VM nodes in your cluster directly on a vCenter server. From that point, you can use DKP to provision and manage your cluster.

To get started, fulfill the prerequisites (see page 1133).

7.15.2 vSphere Prerequisites

7.15.2.1 Prepare your environment to run DKP with VMware vSphere

Fulfilling the prerequisites involves completing these two areas:

- DKP prerequisites
- vSphere prerequisites

7.15.2.2 DKP Prerequisites

Before using DKP to create a vSphere cluster, verify that you have:

- An x86_64-based Linux® or macOS® machine.
- Download DKP binaries and Konvoy Image Builder (KIB) (see page 71) image bundle for Linux or macOS.
- A Container engine/runtime installed is required to install DKP:
  - Version Docker®[811] container engine version 18.09.2 or higher installed for Linux or MacOS - On macOS, Docker runs in a virtual machine which needs configured with at least 8 GB of memory.
  - Version 4.0 of Podman[812] or higher for Linux. Host requirements found here: Host Requirements[813]
- You must have the container engine installed on the host where the DKP Konvoy CLI runs. For example, if you are installing Konvoy on your laptop, ensure the laptop has a supported version of Docker.

On macOS, Docker runs in a virtual machine. Configure this virtual machine with at least 8GB of memory.

- kubectl[814] 1.21.6 for interacting with the running cluster, installed on the host where the DKP Konvoy command line interface (CLI) runs.
- A valid VMware vSphere account with credentials configured.

DKP uses `localvolumeprovisioner` as the default storage provider (see page 103). However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[815] compatible storage that is suitable for production.

811 https://docs.docker.com/get-docker/
812 https://podman.io/getting-started/installation
813 https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
814 https://kubernetes.io/docs/tasks/tools/#kubectl
815 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

You can choose from any of the storage options[816] available for Kubernetes. To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your newly created StorageClass to be the default by following the commands in the Kubernetes documentation called Changing the Default Storage Class[817].

7.15.2.3 VMware vSphere Prerequisites

Before installing, verify that your VMware vSphere Client environment[818] meets the following basic requirements:

- Access to a bastion VM, or other network connected host, running vSphere Client version v6.7.x with Update 3 or later version.
    - You must be able to reach the vSphere API endpoint from where the Konvoy command line interface (CLI) runs.
- vSphere account with credentials configured - this account must have Administrator privileges.
- A RedHat® subscription with user name and password for downloading DVD ISOs.
- For air-gapped environments, a bastion VM host template (see page 117) with access to a configured Docker registry. VMWare site has more information on vSphere Bastion Hosts[819]. The recommended template naming pattern is `../folder-name/dkp-e2e-bastion-template` or similar. Each infrastructure provider has its own set of bastion host instructions. Refer to your provider's site for details - Azure[820], AWS[821], GCP[822], or vSphere[823].
- Valid vSphere values for the following:
    - vCenter API server URL.
    - Datacenter name.
    - Zone name that contains ESXi hosts[824] for your cluster's nodes.
    - Datastore name for the shared storage resource to be used for the VMs in the cluster.
        - Use of PersistentVolumes in your cluster depends on Cloud Native Storage (CNS), available in vSphere v6.7.x with Update 3 and later versions. CNS depends on this shared Datastore's configuration.
    - Datastore URL from the datastore record for the shared datastore you want your cluster to use.

816 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
817 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/
818 https://docs.vmware.com/en/VMware-vSphere/6.7/com.vmware.vsphere.vm_admin.doc/GUID-55238059-912E-411F-A0E9-A7A536972A91.html
819 https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.security.doc/GUID-6975426F-56D0-4FE2-8A58-580B40D2F667.html
820 https://learn.microsoft.com/en-us/azure/bastion/quickstart-host-portal
821 https://aws.amazon.com/solutions/implementations/linux-bastion/
822 https://blogs.vmware.com/cloud/2021/06/02/intro-google-cloud-vmware-engine-bastion-host-access-iap/
823 https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.security.doc/GUID-6975426F-56D0-4FE2-8A58-580B40D2F667.html
824 https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.esxi.install.doc/GUID-B2F01BF5-078A-4C7E-B505-5DFFED0B8C38.html

        - You need this URL value to ensure that the correct Datastore is used when DKP creates VMs for your cluster in vSphere.
    - Folder name.
    - Base template name, such as base-rhel-8, or base-rhel-7.
    - Name of a Virtual Network that has DHCP enabled for both air-gapped and non-air-gapped environments.
    - Resource Pools - at least one resource pool needed, with every host in the pool having access to shared storage, such as VSAN.
        - Each host in the resource pool needs access to shared storage, such as NFS or VSAN, to make use of MachineDeployments and high-availability control planes.

7.15.2.4 Next Steps:

- Non-air-gapped Create a Base OS image in vSphere (see page 1138)
- Air-gapped Create a Base Air-gapped OS VM Image (see page 1165)

7.15.2.5 Minimum User Permissions

7.15.2.5.1 Create minimum required roles for provisioning and installing in vSphere

When a user needs permissions less than Admin, a role must be created. The process for configuring a vSphere role with the least permissions for provisioning nodes and installing includes the following steps:

1. Open a vSphere Client connection to the vCenter Server, described in the Prerequisites (see page 1133).
2. Select Home > Administration > Roles > Add Role.
3. Give the new role a name, then select these Privileges:

Cns		
	☑	Searchable
Datastore		
	☑	Allocate space
	☑	Low level file operations
Host		

	• Configuration	
	☑	Storage partition configuration
Profile-driven storage		
	☑	Profile-driven storage view
Network		
	☑	Assign network
Resource		
	☑	Assign virtual machine to resource pool
Virtual machine		
	• Change Configuration - from the list in that section, select these permissions below:	
	☑	Add new disk
	☑	Add existing disk
	☑	Add or remove device
	☑	Advanced configuration
	☑	Change CPU count
	☑	Change Memory
	☑	Change Settings
	☑	Reload from path
	Edit inventory	

	☑	Create from existing
	☑	Remove
	Interaction	
	☑	Power off
	☑	Power on
	Provisioning	
	☑	Clone template
	☑	Deploy template
	Session	
	☑	ValidateSession

Add the permission at the highest level and set to propagate the permissions.

7.15.2.6 vSphere Storage Options

7.15.2.6.1 Explore storage options and considerations for using DKP with VMware vSphere

The vSphere Container Storage[825] plugin supports shared NFS, vNFS, and vSAN. You need to provision your storage options in vCenter prior to creating a CAPI image (see page 1166) in DKP for use with vSphere.

DKP has integrated the CSI 2.x driver used in vSphere. When creating your DKP cluster, DKP uses whatever configuration you provide for the Datastore name. vSAN is not required. Using NFS can reduce the amount of tagging and permission granting required to configure your cluster.

7.15.3 Create a Base OS image in vSphere

Creating a base OS image from DVD ISO files is a one-time process. Building a base OS image creates a base vSphere template in your vSphere environment. The base OS image is used by Konvoy Image Builder (see page 1282)(KIB) to create a VM template to configure Kubernetes nodes by the DKP vSphere provider.

825 https://docs.vmware.com/en/VMware-vSphere-Container-Storage-Plug-in/2.0/vmware-vsphere-csp-getting-started/GUID-74AF02D7-1562-48BD-A9FE-C81A53342AC3.html

7.15.3.1 Create the Base OS Image

For vSphere, a username and password is populated by `SSH_USERNAME` and the user can use authorization via `SSH_PASSWORD` or `SSH_PRIVATE_KEY_FILE` environment variables and required by default by packer[826]. This user should have administrator privileges. It is possible to configure a custom user and password when building the OS image, however, that requires the Konvoy Image Builder (KIB) configuration to be overridden (see page 1330). DKP advises not to use usernames and passwords for security reasons, but instead to use private and public keys. If that is not possible, passwords should be generated and a minimum of 20 characters long.

While creating the base OS image, it is important to take into consideration the following elements:

- Storage configuration: D2iQ recommends customizing disk partitions and not configuring a SWAP partition.
- Network configuration: as KIB must download and install packages, activating the network is required.
- Connect to Red Hat: if using RHEL, registering with Red Hat is required to configure software repositories and install software packages.
- Software selection: D2iQ recommends choosing **Minimal Install**.
- DKP recommends to install with the packages provided by the operating system package managers. Use the version that corresponds to the major version of your operating system.
- DKP advises not to use usernames and passwords for security reasons, but instead to use private and public keys. If that is not possible, passwords should be generated and a minimum of 20 characters long.

7.15.3.1.1 Disk Size

For each cluster you create using this base OS image, ensure you establish the disk size of the **root file system** based on:

- The minimum DKP storage requirements (see page 110).
- The minimum storage requirements for your organization.

7.15.3.1.1.1 Defaults

Clusters are created with a default disk size of 80 GB.

For clusters created with the default disk size, the base OS image root file system must be exactly 80 GB. The root file system cannot be reduced automatically when a machine first boots.

826 https://github.com/mesosphere/konvoy-image-builder/blob/0523fd2c5e6e1ad1d4962f60a47039aa145a6e42/pkg/packer/manifests/vsphere/packer.pkr.hcl

7.15.3.1.1.2 Customization

You can also specify a custom disk size when you create a cluster (see page 0) (see the flags of the Create new vSphere Cluster (see page 1462) command). This allows you to use one base OS image to create multiple clusters that have different storage requirements.

Before specifying a disk size when you create a cluster, take into account:

1. **For some base OS images, the custom disk size option has no effect on the size of the root file system.** This is because some root file systems, for example, those contained in an LVM Logical Volume, cannot be resized automatically when a machine first boots.
2. **The specified custom disk size must be equal to, or larger than the size of the base OS image root file system.** This is because a root file system cannot be reduced automatically when a machine first boots.

7.15.3.2 Next Step:

Create a VM Template (see page 1140)

7.15.4 Create a VM Template

7.15.4.1 Create a vSphere Template for Your Cluster from a Base OS Image

Using the base OS image created in a previous procedure, DKP creates the new vSphere template directly on the vCenter server.

1. Set the following vSphere environment variables on the bastion VM host:

```
export VSPHERE_SERVER=your_vCenter_APIserver_URL
export VSPHERE_USERNAME=your_vCenter_user_name
export VSPHERE_PASSWORD=your_vCenter_password
```

2. Copy the base OS image file created in the vSphere Client to your desired location on the bastion VM host, and make a note of the path and file name.

3. Create an `image.yaml` file and add the following variables for vSphere. DKP uses this file and these variables as inputs in the next step.
   NOTE: You will need to replace OS name below based on your OS - EX: “rhel-79” to “rhel-86”. See other YAML examples[827] for copy and paste below last step.

```
---
download_images: true
build_name: "rhel-79"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-
init-vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-rhel-7.9" # change default value with your base template name
  vsphere_guest_os_type: "rhel7_64Guest"
  guest_os_type: "rhel7-64"
  # goss params
  distribution: "RHEL"
  distribution_version: "7.9"
# Use following overrides to select the authentication method that can be used
with base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # if set to true, ssh_password and ssh_private_key
will be ignored
```

4. Create a vSphere VM template with your variation of the following command:

```
konvoy-image build images/ova/<image.yaml>
```

   Any additional configurations can be added to this command using `--overrides` flags as shown below:

   a. for FIPS, add this flag - `--overrides overrides/fips.yaml`

827 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

b. for air-gapped, add this flag `--overrides overrides/offline-fips.yaml`

5. The Konvoy Image Builder (see page 1305) (KIB) uses the values in `image.yaml` and the input base OS image to create a vSphere template directly on the vCenter server. This template contains the required artifacts needed to create a Kubernetes cluster.
   When KIB provisions the OS image[828] successfully, it creates a manifest file. The `artifact_id` field of this file contains the name of the AMI ID (AWS), template name (vSphere), or image name (GCP/Azure), for example:

```
{
      "name": "vsphere-clone",
      "builder_type": "vsphere-clone",
      "build_time": 1644985039,
      "files": null,
      "artifact_id": "konvoy-ova-vsphere-rhel-84-1.21.6-1644983717",
      "packer_run_uuid": "260e8110-77f8-ca94-e29e-ac7a2ae779c8",
      "custom_data": {
        "build_date": "2022-02-16T03:55:17Z",
        "build_name": "vsphere-rhel-84",
        "build_timestamp": "1644983717",
        [...]
      }
    }
```

6. Next steps are to deploy a DKP cluster using your vSphere template.

Next, create a Kubernetes Bootstrap Cluster to enable creating your vSphere cluster and moving CAPI objects to it.

YAML Files (see page 1309) found in Konvoy Image Builder section.

7.15.4.2 Next Step:

vSphere Bootstrap (see page 1142)

7.15.5 vSphere Bootstrap

7.15.5.1 Prepare to deploy Kubernetes clusters

To create Kubernetes clusters, Konvoy uses Cluster API[829] (CAPI) controllers, which run on a Kubernetes cluster. To get started creating your vSphere cluster, you need a *bootstrap* cluster. By default, Konvoy creates a bootstrap cluster for you in a Docker container using the Kubernetes-in-Docker (KIND[830]) tool.

828 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova
829 https://cluster-api.sigs.k8s.io/
830 https://github.com/kubernetes-sigs/kind

7.15.5.2 Prerequisites

Before you begin, you must:

- Ensure the `dkp` binary can be found in your $PATH.
- Complete the steps in Create a CAPI VM template (see page 1140)

7.15.5.3 Bootstrap cluster lifecycle services

1. If your environment uses HTTP/HTTPS proxies, you must either set environment variables or include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).
2. Create a bootstrap cluster:

   ```
   dkp create bootstrap --kubeconfig $HOME/.kube/config
   ```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

   The output resembles this example:

   ```
   ✓ Creating a bootstrap cluster
   ✓ Initializing new CAPI components
   ```

   Konvoy creates a bootstrap cluster using KIND[831] as a library. Konvoy then deploys the following Cluster API[832] providers on the cluster:

   - Core Provider[833]
   - vSphere Infrastructure Provider[834]
   - Kubeadm Bootstrap Provider[835]
   - Kubeadm ControlPlane Provider[836]
3. Ensure that the CAPV controllers are present with the command:

831 https://github.com/kubernetes-sigs/kind
832 https://cluster-api.sigs.k8s.io/
833 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/
834 https://github.com/kubernetes-sigs/cluster-api-provider-vsphere
835 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/bootstrap/kubeadm
836 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/controlplane/kubeadm

```
kubectl get pods -n capv-system
```

The output resembles the following:

```
NAME                                        READY   STATUS    RESTARTS   AGE
capv-controller-manager-785c5978f-nnfns     1/1     Running   0          13h
```

4. Konvoy waits until the controller-manager and webhook deployments of these providers are ready. List these deployments using this command:

```
kubectl get --all-namespaces deployments -l=clusterctl.cluster.x-k8s.i
```

The output resembles the following:

```
NAMESPACE                           NAME
READY   UP-TO-DATE   AVAILABLE   AGE
capa-system                         capa-controller-manager
1/1     1            1           22m
capi-kubeadm-bootstrap-system       capi-kubeadm-bootstrap-controller-manager
1/1     1            1           22m
capi-kubeadm-control-plane-system   capi-kubeadm-control-plane-controller-
manager   1/1     1            1           22m
capi-system                         capi-controller-manager
1/1     1            1           22m
cappp-system                        cappp-controller-manager
1/1     1            1           22m
capv-system                         capv-controller-manager
1/1     1            1           22m
capz-system                         capz-controller-manager
1/1     1            1           22m
cert-manager                        cert-manager
1/1     1            1           22m
cert-manager                        cert-manager-cainjector
1/1     1            1           22m
cert-manager                        cert-manager-webhook
1/1     1            1           22m
```

7.15.5.4 Next Step:

Create new vSphere Cluster (see page 1145)

7.15.6 Create new vSphere Cluster

7.15.6.1 Prerequisites

Before you begin, make sure you have created a vSphere Bootstrap (see page 1142) cluster.

7.15.6.2 Name your cluster

1. Give your cluster a unique name suitable for your environment.
2. Set the CLUSTER_NAME environment variable with the command:

```
export CLUSTER_NAME=my-vsphere-cluster
```

7.15.6.3 Create a New vSphere Kubernetes Cluster

Follow these steps:

1. Use the following command to set the environment variables for vSphere:

```
export VSPHERE_SERVER=example.vsphere.url
export VSPHERE_USERNAME=user@example.vsphere.url
export VSPHERE_PASSWORD=example_password
```

2. Ensure your vSphere credentials are up-to-date by refreshing the credentials with the command:

```
dkp update bootstrap credentials vsphere
```

3. Generate the Kubernetes cluster objects by copying and editing this command to include the correct values, including the VM template name you assigned in the previous procedure:

- To increase Dockerhub's rate limit[837] use your Dockerhub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.
- Ensure your subnets (see page 929) do not overlap with your host subnet because they cannot be changed after cluster creation. If you need to change the kubernetes subnets, you must do this at cluster creation. The default subnets used in DKP are:

```
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 192.168.0.0/16
    services:
      cidrBlocks:
      - 10.96.0.0/12
```

- The following `dkp create cluster` example shows a common configuration. See dkp create cluster vsphere (see page 1462) reference for the full list of cluster creation options:

```
dkp create cluster vsphere \
  --cluster-name ${CLUSTER_NAME} \
  --network <NETWORK_NAME> \
  --control-plane-endpoint-host <xxx.yyy.zzz.000> \
  --data-center <DATACENTER_NAME> \
  --data-store <DATASTORE_NAME> \
  --folder <FOLDER_NAME> \
  --server <VCENTER_API_SERVER_URL> \
  --ssh-public-key-file <SSH_PUBLIC_KEY_FILE> \
  --resource-pool <RESOURE_POOL_NAME> \
  --virtual-ip-interface <ip_interface_name> \
  --vm-template <TEMPLATE_NAME>
```

- (Optional) Alternatively, you can create individual files with different smaller manifests for ease in editing using the `--output-directory` flag. This will create multiple files in the specified directory which must already exist:

```
dkp create cluster vsphere --cluster-name=${CLUSTER_NAME} \
--ami=${AWS_AMI_ID} \
```

837 https://docs.docker.com/docker-hub/download-rate-limit/

```
--dry-run \
--output=yaml \
--output-directory=<existing-directory>
```

For more information regarding this flag or others, please refer to the CLI section of the documentation for dkp create cluster (see page 1454) and select your provider.

4. (Optional) To configure the Control Plane and Worker nodes to use an HTTP proxy:

```
export CONTROL_PLANE_HTTP_PROXY=http://example.org:8080
export CONTROL_PLANE_HTTPS_PROXY=http://example.org:8080
export
CONTROL_PLANE_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1,10.96
.0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.svc,kubernete
s.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.cluster,.svc.clu
ster.local,169.254.169.254,.elb.amazonaws.com"

export WORKER_HTTP_PROXY=http://example.org:8080
export WORKER_HTTPS_PROXY=http://example.org:8080
export
WORKER_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1,10.96.0.0/12
,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.defau
lt.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.cluster,.svc.cluster.lo
cal,169.254.169.254,.elb.amazonaws.com"
```

- Replace `example.org,example.com,example.net` with you internal addresses
- `localhost` and `127.0.0.1` addresses should not use the proxy
- `10.96.0.0/12` is the default Kubernetes service subnet
- `192.168.0.0/16` is the default Kubernetes pod subnet
- `kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local` is the internal Kubernetes kube-apiserver service
- `.svc,.svc.cluster,.svc.cluster.local` is the internal Kubernetes services
- `169.254.169.254` is the AWS metadata server
- `.elb.amazonaws.com` is for the worker nodes to allow them to communicate directly to the kube-apiserver ELB

5. (Optional) Create a Kubernetes cluster with HTTP proxy configured. This step assumes you did not already create a cluster in the previous steps:

To increase Dockerhub's rate limit[838] use your Dockerhub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.

```
dkp create cluster vsphere --cluster-name=${CLUSTER_NAME} \
--control-plane-http-proxy="${CONTROL_PLANE_HTTP_PROXY}" \
--control-plane-https-proxy="${CONTROL_PLANE_HTTPS_PROXY}" \
--control-plane-no-proxy="${CONTROL_PLANE_NO_PROXY}" \
--worker-http-proxy="${WORKER_HTTP_PROXY}" \
--worker-https-proxy="${WORKER_HTTPS_PROXY}" \
--worker-no-proxy="${WORKER_NO_PROXY}" \
--dry-run \
--output=yaml \
> ${CLUSTER_NAME}.yaml
```

6. Inspect or edit the cluster objects:

Familiarize yourself with Cluster API before editing the cluster objects as edits can prevent the cluster from deploying successfully.

The objects are Custom Resources[839] defined by Cluster API components, and they belong in three different categories:

1. Cluster

   A *Cluster* object has references to the infrastructure-specific and control plane objects. Because this is a vSphere cluster, there is an object that describes the infrastructure-specific cluster properties.

2. Control plane

   A *KubeadmControlPlane* object describes the control plane, which is the group of machines that run the Kubernetes control plane components, which include the etcd distributed database, the API server, the core controllers, and the scheduler. The object describes the configuration for these components. The object also has a reference to an infrastructure-specific object that describes the properties of all control plane machines. Here, it references an *vSphereMachineTemplate* object.

3. Node pool

838 https://docs.docker.com/docker-hub/download-rate-limit/
839 https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/

A node pool is a collection of machines with identical properties. For example, a cluster might have one node pool with large memory capacity, another node pool with GPU support. Each node pool is described by three objects: The MachinePool references an object that describes the configuration of Kubernetes components (for example, kubelet) deployed on each node pool machine, and an infrastructure-specific object that describes the properties of all node pool machines. Here, it references a *KubeadmConfigTemplate*, and a *vSphereMachineTemplate* object.

For in-depth documentation about the objects, read Concepts[840] in the Cluster API Book.

7. Modify control plane audit logs settings using the information contained in the page Configuring the Control Plane (see page 1351).

8. Create the cluster from the objects. A warning will appear in the console if the resource already exists and will require you to remove the resource or update your YAML.

```
kubectl create -f ${CLUSTER_NAME}.yaml
```

:note: NOTE: If you used the `--output-directory` flag in your `dkp create .. --dry-run` step above, create the cluster from the objects you created by specifying the directory:

```
kubectl create -f <existing-directory>/
```

Output will be similar to below:

```
cluster.cluster.x-k8s.io/vsphere-example created
cluster.infrastructure.cluster.x-k8s.io/vsphere-example created
kubeadmcontrolplane.controlplane.cluster.x-k8s.io/vsphere-example-control-plane
created
machinedeployment.cluster.x-k8s.io/vsphere-example-mp-0 created
kubeadmconfigtemplate.bootstrap.cluster.x-k8s.io/vsphere-example-mp-0 created
```

9. Use the `wait` command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=20m
```

```
cluster.cluster.x-k8s.io/${CLUSTER_NAME} condition met
```

The `READY` status becomes `True` after the cluster control-plane becomes Ready in one of the following steps.

840 https://cluster-api.sigs.k8s.io/user/concepts.html

After DKP creates the objects on the API server, the Cluster API controllers reconcile them, creating infrastructure and machines. As the controllers progress, they update the Status of each object.

10. Run the DKP describe command to monitor the current status of the cluster:

```
dkp describe cluster -c ${CLUSTER_NAME}
```

```
NAME                                                                READY  SEVERITY
REASON  SINCE  MESSAGE
Cluster/d2iq-e2e-cluster_name-1                                     True
13h
├──ClusterInfrastructure - VSphereCluster/d2iq-e2e-cluster_name-1    True
13h
├──ControlPlane - KubeadmControlPlane/d2iq-control-plane            True
13h
│ ├──Machine/d2iq--control-plane-7llgd                              True
13h
│ ├──Machine/d2iq--control-plane-vncbl                              True
13h
│ └──Machine/d2iq--control-plane-wbgrm                              True
13h
└──Workers
    └──MachineDeployment/d2iq--md-0                                 True
13h
    ├──Machine/d2iq--md-0-74c849dc8c-67rv4                          True
13h
    ├──Machine/d2iq--md-0-74c849dc8c-n2skc                          True
13h
    ├──Machine/d2iq--md-0-74c849dc8c-nkftv                          True
13h
    └──Machine/d2iq--md-0-74c849dc8c-sqklv                          True
13h
```

11. Check all machines has `NODE_NAME` assigned

```
kubectl get machines
```

The output appears similar to the following:

```
NAME                                        CLUSTER              NODENAME
PROVIDERID                                        PHASE     AGE   VERSION
```

```
d2iq-e2e-cluster-1-control-plane-7llgd     d2iq-e2e-cluster-1   d2iq-e2e-cluster-1-
control-plane-7llgd     vsphere://421638e2-e776-9af6-f683-5e105de5da5a   Running
13h   v1.22.8
d2iq-e2e-cluster-1-control-plane-vncbl     d2iq-e2e-cluster-1   d2iq-e2e-cluster-1-
control-plane-vncbl     vsphere://42168835-7fef-95c4-3652-ebcad3e10d36   Running
13h   v1.22.8
d2iq-e2e-cluster-1-control-plane-wbgrm     d2iq-e2e-cluster-1   d2iq-e2e-cluster-1-
control-plane-wbgrm     vsphere://421642df-afc4-b6c2-9e61-5b86e7c37eac   Running
13h   v1.22.8
d2iq-e2e-cluster-1-md-0-74c849dc8c-67rv4   d2iq-e2e-cluster-1   d2iq-e2e-cluster-1-
md-0-74c849dc8c-67rv4   vsphere://4216f467-8483-73cb-a8b6-8d6a4a71e4b4   Running
14h   v1.22.8
d2iq-e2e-cluster-1-md-0-74c849dc8c-n2skc   d2iq-e2e-cluster-1   d2iq-e2e-cluster-1-
md-0-74c849dc8c-n2skc   vsphere://42161cde-9904-4dd2-7a3e-cdfc7655f090   Running
14h   v1.22.8
d2iq-e2e-cluster-1-md-0-74c849dc8c-nkftv   d2iq-e2e-cluster-1   d2iq-e2e-cluster-1-
md-0-74c849dc8c-nkftv   vsphere://42163a0d-eb8d-b5a6-82d5-188e24817c00   Running
14h   v1.22.8
d2iq-e2e-cluster-1-md-0-74c849dc8c-sqklv   d2iq-e2e-cluster-1   d2iq-e2e-cluster-1-
md-0-74c849dc8c-sqklv   vsphere://42161dff-92a5-6da9-7ac1-e987e2c8fed2   Running
14h   v1.22.8
```

12. Verify that the kubeadm control plane is ready with the command

```
kubectl get kubeadmcontrolplane
```

The output appears similar to the following:

```
NAME                              CLUSTER              INITIALIZED   API SERVER
AVAILABLE   REPLICAS   READY   UPDATED   UNAVAILABLE   AGE   VERSION
d2iq-e2e-cluster-1-control-plane   d2iq-e2e-cluster-1   true          true
3          3       3         0             14h   v1.22.8
```

13. Describe the kubeadm control plane and check its status and events with the command:

```
kubectl describe kubeadmcontrolplane
```

14. As they progress, the controllers also create Events, which you can list using the command:

```
kubectl get events | grep ${CLUSTER_NAME}
```

For brevity, this example uses `grep`. You can also use separate commands to get Events for specific objects, such as `kubectl get events --field-selector involvedObject.kind="VSphereCluster"` and `kubectl get events --field-selector involvedObject.kind="VSphereMachine"`.

7.15.6.4 Known Limitations

Be aware of these limitations in the current release of DKP Konvoy.

- The DKP Konvoy version used to create a bootstrap cluster must match the DKP Konvoy version used to create a workload cluster.
- DKP Konvoy supports deploying one workload cluster.
- DKP Konvoy generates a set of objects for one Node Pool.
- DKP Konvoy does not validate edits to cluster objects.

The optional next step is to Make the vSphere Cluster Self-managed (see page 1156). The step is optional because, as an example, if you are using an existing, self-managed cluster to create a managed cluster, you would not want the managed cluster to be self-managed.

7.15.6.5 Next Steps:

Explore a vSphere Cluster (see page 1152)

Make vSphere Cluster Self-Managed (see page 1156)

7.15.7 Explore a vSphere Cluster

This guide explains how to use the command line interface to interact with your newly-deployed Kubernetes cluster.

Before you start, make sure you have created a workload cluster (see page 1145) and, if needed, that you have made the cluster self-managing (see page 1156).

7.15.7.1 Get the kubeconfig file for the new Kubernetes cluster

1. Get a kubeconfig file for the workload cluster:

   When the workload cluster is created, the cluster lifecycle services generate a kubeconfig file for the workload cluster, and write it to a *Secret*. The kubeconfig file is scoped to the cluster Administrator.

   Get the kubeconfig from the *Secret*, and write it to a file using this command:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

7.15.7.2 Create a StorageClass with a vSphere datastore

Follow these steps:

1. Access the Datastore tab in the vSphere client and select a datastore by name.
2. Copy the URL of that datastore from the information dialog that displays.
3. Return to the DKP CLI, and delete the existing `StorageClass` with the command:

   ```
   kubectl delete storageclass vsphere-raw-block-sc
   ```

4. Run the following command to create a new StorageClass, supplying the correct values for your environment:

   ```
   cat <<EOF > vsphere-raw-block-sc.yaml
   kind: StorageClass
   apiVersion: storage.k8s.io/v1
   metadata:
     annotations:
       storageclass.kubernetes.io/is-default-class: "true"
     name: vsphere-raw-block-sc
   provisioner: csi.vsphere.vmware.com
   parameters:
     datastoreurl: "<url>"
   volumeBindingMode: WaitForFirstConsumer
   EOF
   ```

7.15.7.3 Explore nodes and pods in the new cluster

1. List the nodes using this command:

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf get nodes
   ```

NOTE: It may take a few minutes for the Status to move to `Ready` while the Pod network is deployed. The Node's Status should change to Ready soon after the `calico-node` DaemonSet Pods are Ready.

The output resembles the following example:

```
NAME                                      STATUS   ROLES                  AGE
VERSION
d2iq-e2e-cluster-1-control-plane-7llgd    Ready    control-plane,master   20h
v1.25.4
d2iq-e2e-cluster-1-control-plane-vncbl    Ready    control-plane,master   19h
v1.25.4
d2iq-e2e-cluster-1-control-plane-wbgrm    Ready    control-plane,master   19h
v1.25.4
d2iq-e2e-cluster-1-md-0-74c849dc8c-67rv4  Ready    <none>                 19h
v1.25.4
d2iq-e2e-cluster-1-md-0-74c849dc8c-n2skc  Ready    <none>                 19h
v1.25.4
d2iq-e2e-cluster-1-md-0-74c849dc8c-nkftv  Ready    <none>                 19h
v1.25.4
d2iq-e2e-cluster-1-md-0-74c849dc8c-sqklv  Ready    <none>                 19h
v1.25.4
```

2. List the pods using this command:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf get --all-namespaces pods
```

The output resembles the following example:

```
NAMESPACE               NAME
READY   STATUS     RESTARTS      AGE
calico-system           calico-kube-controllers-57fbd7bd59-qqd96
1/1     Running    0             20h
calico-system           calico-node-2m524
1/1     Running    3 (19h ago)   19h
calico-system           calico-node-bbhg5
1/1     Running    0             20h
calico-system           calico-node-cc5lf
1/1     Running    2 (19h ago)   19h
calico-system           calico-node-cwg7x
1/1     Running    1 (19h ago)   19h
calico-system           calico-node-d59hn
1/1     Running    1 (19h ago)   19h
calico-system           calico-node-qmmcz
1/1     Running    0             19h
```

```
calico-system            calico-node-wdqhx
1/1     Running    0             19h
calico-system            calico-typha-655489d8cc-b5jnt
1/1     Running    0             20h
calico-system            calico-typha-655489d8cc-q92x9
1/1     Running    0             19h
calico-system            calico-typha-655489d8cc-vjlkx
1/1     Running    0             19h
kube-system              cluster-autoscaler-68c759fbf6-7d2ck
0/1     Init:0/1   0             20h
kube-system              coredns-78fcd69978-qn4qt
1/1     Running    0             20h
kube-system              coredns-78fcd69978-wqpmg
1/1     Running    0             20h
kube-system              etcd-d2iq-e2e-cluster-1-control-plane-7llgd
1/1     Running    0             20h
kube-system              etcd-d2iq-e2e-cluster-1-control-plane-vncbl
1/1     Running    0             19h
kube-system              etcd-d2iq-e2e-cluster-1-control-plane-wbgrm
1/1     Running    0             19h
kube-system              kube-apiserver-d2iq-e2e-cluster-1-control-plane-7llgd
1/1     Running    0             20h
kube-system              kube-apiserver-d2iq-e2e-cluster-1-control-plane-vncbl
1/1     Running    0             19h
kube-system              kube-apiserver-d2iq-e2e-cluster-1-control-plane-wbgrm
1/1     Running    0             19h
kube-system              kube-controller-manager-d2iq-e2e-cluster-1-control-
plane-7llgd   1/1     Running    1 (19h ago)   20h
kube-system              kube-controller-manager-d2iq-e2e-cluster-1-control-
plane-vncbl   1/1     Running    0             19h
kube-system              kube-controller-manager-d2iq-e2e-cluster-1-control-
plane-wbgrm   1/1     Running    0             19h
kube-system              kube-proxy-cpscs
1/1     Running    0             19h
kube-system              kube-proxy-hhmxq
1/1     Running    0             19h
kube-system              kube-proxy-hxhnk
1/1     Running    0             19h
kube-system              kube-proxy-nsrbp
1/1     Running    0             19h
kube-system              kube-proxy-scxfg
1/1     Running    0             20h
kube-system              kube-proxy-tth4k
1/1     Running    0             19h
kube-system              kube-proxy-x2xfx
1/1     Running    0             19h
kube-system              kube-scheduler-d2iq-e2e-cluster-1-control-plane-7llgd
1/1     Running    1 (19h ago)   20h
kube-system              kube-scheduler-d2iq-e2e-cluster-1-control-plane-vncbl
1/1     Running    0             19h
kube-system              kube-scheduler-d2iq-e2e-cluster-1-control-plane-wbgrm
1/1     Running    0             19h
```

```
kube-system                 kube-vip-d2iq-e2e-cluster-1-control-plane-7llgd
1/1     Running    1 (19h ago)   20h
kube-system                 kube-vip-d2iq-e2e-cluster-1-control-plane-vncbl
1/1     Running    0             19h
kube-system                 kube-vip-d2iq-e2e-cluster-1-control-plane-wbgrm
1/1     Running    0             19h
kube-system                 vsphere-cloud-controller-manager-4zj7q
1/1     Running    0             19h
kube-system                 vsphere-cloud-controller-manager-87tgm
1/1     Running    0             19h
kube-system                 vsphere-cloud-controller-manager-xqmn4
1/1     Running    1 (19h ago)   20h
node-feature-discovery      node-feature-discovery-master-84c67dcbb6-txfw9
1/1     Running    0             20h
node-feature-discovery      node-feature-discovery-worker-8tg2l
1/1     Running    3 (19h ago)   19h
node-feature-discovery      node-feature-discovery-worker-c5f6q
1/1     Running    0             19h
node-feature-discovery      node-feature-discovery-worker-fjfkm
1/1     Running    0             19h
node-feature-discovery      node-feature-discovery-worker-x6tz8
1/1     Running    0             19h
tigera-operator             tigera-operator-d499f5c8f-r2srj
1/1     Running    1 (19h ago)   20h
vmware-system-csi           vsphere-csi-controller-7ffd6884cc-d7rql
7/7     Running    5 (19h ago)   20h
vmware-system-csi           vsphere-csi-controller-7ffd6884cc-k82cm
7/7     Running    2 (19h ago)   20h
vmware-system-csi           vsphere-csi-controller-7ffd6884cc-qttkp
7/7     Running    1 (19h ago)   20h
vmware-system-csi           vsphere-csi-node-678hw
3/3     Running    0             19h
vmware-system-csi           vsphere-csi-node-6tbsh
3/3     Running    0             19h
vmware-system-csi           vsphere-csi-node-9htwr
3/3     Running    5 (20h ago)   20h
vmware-system-csi           vsphere-csi-node-g8r6l
3/3     Running    0             19h
vmware-system-csi           vsphere-csi-node-ghmr6
3/3     Running    0             19h
vmware-system-csi           vsphere-csi-node-jhvgm
3/3     Running    0             19h
vmware-system-csi           vsphere-csi-node-rp77r
3/
```

7.15.8 Make vSphere Cluster Self-Managed

Konvoy deploys all cluster lifecycle services to a bootstrap cluster, which then deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster, which makes the workload cluster self-managed. This section describes how to make a workload cluster self-managed.

Before starting, ensure you create a workload cluster as described in Create a New vSphere Cluster or Create a New Air-gapped vSphere Cluster.

This page contains instructions on how to make your cluster self-managed. This is necessary if there is only one cluster in your environment, or if this cluster should become the Management cluster in a multi-cluster environment.

If you already have a self-managed or Management cluster in your environment, skip this page.

7.15.8.1 Make the new Kubernetes cluster manage itself

Follow these steps:

1. Deploy cluster lifecycle services on the workload cluster:

   ```
   dkp create capi-components --kubeconfig ${CLUSTER_NAME}.conf
   ```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).
   The output is similar to this:

   ```
   ✓ Initializing new CAPI components
   ```

2. Move the Cluster API objects from the bootstrap to the workload cluster:

   The cluster lifecycle services on the workload cluster are ready, but the workload cluster configuration is on the bootstrap cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the bootstrap to the workload cluster. This process is also called a Pivot[841].

   ```
   dkp move capi-resources --to-kubeconfig ${CLUSTER_NAME}.conf
   ```

841 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

```
✓ Moving cluster resources
You can now view resources in the moved cluster by using the --kubeconfig flag
with kubectl. For example: kubectl --kubeconfig=vsphere-example.conf get nodes
```

NOTE: To ensure only one set of cluster lifecycle services manages the workload cluster, Konvoy first pauses reconciliation of the objects on the bootstrap cluster, then creates the objects on the workload cluster. As Konvoy copies the objects, the cluster lifecycle services on the workload cluster reconcile the objects. The workload cluster becomes self-managed after Konvoy creates all the objects. If it fails, the move command can be safely retried.

3. Wait for the cluster control-plane to be ready:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf wait --for=condition=ControlPlaneRead
y "clusters/${CLUSTER_NAME}" --timeout=20m
```

```
cluster.cluster.x-k8s.io/vsphere-example condition met
```

4. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

   NOTE: After moving the cluster lifecycle services to the workload cluster, remember to use Konvoy with the workload cluster kubeconfig.

```
dkp describe cluster --kubeconfig ${CLUSTER_NAME}.conf -c ${CLUSTER_NAME}
```

```
NAME                                                                READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/vsphere-example-1                                           True
13h
├──ClusterInfrastructure - VSphereCluster/vsphere-example-1          True
13h
├──ControlPlane - KubeadmControlPlane/vsphere-example-control-plane  True
13h
│  ├──Machine/vsphere-example-control-plane-7llgd                    True
13h
│  ├──Machine/vsphere-example-control-plane-vncbl                    True
13h
│  └──Machine/vsphere-example-control-plane-wbgrm                    True
13h
└──Workers
```

```
     └──MachineDeployment/vsphere-example-md-0                    True
13h
     ├──Machine/vsphere-example-md-0-74c849dc8c-67rv4             True
13h
     ├──Machine/vsphere-example-md-0-74c849dc8c-n2skc             True
13h
     ├──Machine/vsphere-example-md-0-74c849dc8c-nkftv             True
13h
     └──Machine/vsphere-example-md-0-74c849dc8c-sqklv             True
13h
```

5. Remove the bootstrap cluster, if desired, as the workload cluster is now self-managed:

```
dkp delete bootstrap
```

```
✓ Deleting bootstrap cluster
```

7.15.8.2 Known limitations

Be aware of these limitations in the current release of DKP Konvoy.

- Before making a workload cluster self-managed, be sure that its control plane nodes have sufficient permissions for running Cluster API controllers.
- DKP Konvoy supports moving only one set of cluster objects from the bootstrap cluster to the workload cluster, or vice-versa.
- DKP Konvoy only supports moving all namespaces in the cluster; DKP does not support migration of individual namespaces.

7.15.9 Configure MetalLB for a vSphere infrastructure

Create a MetalLB configMap for your vSphere infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP)

Select one of the following procedures to create your MetalLB manifest for further editing and deployment.

7.15.9.1 Layer 2 configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to Address Resolution Protocol (ARP) requests on your local network directly, to give the machine's MAC address to clients.

- MetalLB IP address ranges/CIDRs should be within the node's primary network subnet (see page 929).
- MetalLB IP address ranges/CIDRs and node subnet should not conflict with the Kubernetes cluster pod and service subnets.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

1. Create a `metallb-conf.yaml` file for editing with the command:

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

2. Edit the file to contain values specific to your environment, then run the following `kubectl` command:

```
kubectl apply -f metallb-conf.yaml
```

7.15.9.2 BGP configuration

For a basic configuration featuring one BGP router and one IP address range, you need four pieces of information:

- The router IP address to which MetalLB should connect
- The router's AS number
- The AS number MetalLB should use
- An IP address range expressed as a CIDR prefix

For example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like the code snippet below.

The following example values are generic, enter your specific values into the fields where applicable.

1. Extract the kubeconfig for your cluster and deploy a configMap for MetalLB using the following command:

```
dkp get kubeconfig -c ${DKP_CLUSTER_NAME} > ${DKP_CLUSTER_NAME}.conf
```

2. Create a `metallb-conf.yaml` file for editing with the command:

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

3. Edit the file to contain values specific to your environment, then run the following `kubectl` command:

```
kubectl apply -f metallb-conf.yaml
```

7.15.10 Install vSphere Air-Gapped

7.15.10.1 Create a Kubernetes vSphere cluster in a private network with no access to the Internet (air-gapped)

This section provides the instructions to create a Kubernetes cluster in a private, DHCP-enabled network with no access to the Internet. Complete the list of Prerequisites (see page 1133) before you begin the procedures in this section.

7.15.10.2 Bastion Host

When creating an air-gapped vSphere cluster, the bastion VM hosts the installation of the DKP Konvoy bundles and images, as well as the Docker registry, needed to create and operate your vSphere cluster. The bastion VM must have access to the vSphere API Server (vCenter Server). Ensure the items below are installed and the environment matches the requirements below:

- Create a bastion VM host template for the cluster nodes to use within the air-gapped network. This bastion VM host also needs access to a Docker registry in lieu of an Internet connection for pulling Docker images. The recommended template naming pattern is `../folder-name/dkp-e2e-bastion-template` or similar.
- Find and record the bastion VM's IP or host name.
- Download (see page 71) the following required DKP Konvoy binaries and installation bundles discussed in step 5 below.
- Docker®[842] version 18.09.2 or later installed. You must have Docker installed on the host where the DKP Konvoy CLI runs. For example, if you are installing Konvoy on your laptop, ensure the laptop has a supported version of Docker. On macOS, Docker runs in a virtual machine which you configure with at least 8GB of memory.
- kubectl[843] for interacting with the running cluster, installed on the host where the DKP Konvoy command line interface (CLI) runs.

Depending on your OS, there are various commands for setting up your own bastion host for use with air-gapped vSphere.

This would be a **generic** example for RHEL Bastion nodes:

1. Once `base-rhel-os` boots, open an `ssh` terminal to host and install the tools and packages:

842 https://docs.docker.com/get-docker/
843 https://kubernetes.io/docs/tasks/tools/#kubectl

```
sudo yum install -y yum-utils bzip2 wget
```

2. Install kubectl as mentioned above, below is a RHEL example:

```
cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-\$basearch
enabled=1
gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF
sudo yum install -y kubectl
```

3. Install Docker (Only on Bastion Host) and add the repo for upstream docker:

```
sudo yum-config-manager --add-repo https://download.docker.com/linux/rhel/
docker-ce.repo
```

NOTE: Other Docker repo downloads are available on docker.com[844]: https://download.docker.com/linux/

4. Install Docker:

```
sudo yum install -y docker-ce docker-ce-cli containerd.io
```

5. Create directory for Konvoy Image Builder and DKP CLI:

```
mkdir kib && mkdir dkp
```

6. Get the needed D2iQ Software by downloading the air-gapped bundle: Download (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz`, extract the tarball to a local directory:

```
tar -xzvf dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz && cd dkp-v2.5.0/kib
```

7. Install DKP CLI:

```
cd ..
cd dkp
wget https://downloads.d2iq.com/dkp/v2.5.0/dkp_v2.5.0_linux_amd64.tar.gz
```

844 http://docker.com

8. Set the following vSphere environment variables on the bastion VM host:

```
export VSPHERE_SERVER=your_vCenter_APIserver_URL
export VSPHERE_USERNAME=your_vCenter_user_name
export VSPHERE_PASSWORD=your_vCenter_password
```

9. Set the following environment variables to enable connection to an existing Docker registry. You must create the VM template with the Konvoy Image Builder (see page 1282) to be able to use the registry mirror feature:

```
export DOCKER_REGISTRY_ADDRESS=<https/http>://<registry-address>:<registry-
port>
export DOCKER_REGISTRY_CA=<path to the CA on the bastion host>
```

   - `DOCKER_REGISTRY_ADDRESS` : the address of an existing Docker registry accessible in the vSphere Zone where the new cluster nodes will be configured, to use a mirror registry when pulling images.
   - `DOCKER_REGISTRY_CA` : (optional) the path on the bastion host to the Docker registry CA. Konvoy configures the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the VMs are not already configured to trust this CA.

7.15.10.2.1 More information:

Each infrastructure provider has its own set of bastion host instructions. Refer to your own OS instructions to setup a bastion host like AWS Bastion[845], Azure[846], GCP[847], or vSphere[848].

845 https://aws.amazon.com/solutions/implementations/linux-bastion/
846 https://learn.microsoft.com/en-us/azure/bastion/quickstart-host-portal
847 https://blogs.vmware.com/cloud/2021/06/02/intro-google-cloud-vmware-engine-bastion-host-access-iap/
848 https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.security.doc/GUID-6975426F-56D0-4FE2-8A58-580B40D2F667.html

7.15.10.3 Create a Base Air-gapped OS VM Image

7.15.10.3.1 Create a Base OS VM Image in the VMware vSphere Client (air-gapped)

7.15.10.3.2 Creating a base OS image from DVD ISO files is a one-time process. Building a base OS image creates a base vSphere template in your vSphere environment. The base OS image is used by Konvoy Image Builder (KIB) to create a VM template to configure Kubernetes nodes by the DKP vSphere provider.

7.15.10.3.3 Create the Base OS Image

For vSphere, a username and password is populated by `SSH_USERNAME` and the user can use authorization via `SSH_PASSWORD` or `SSH_PRIVATE_KEY_FILE` environment variables and required by default by packer[849]. This user should have administrator privileges. It is possible to configure a custom user and password when building the OS image, however, that requires the Konvoy Image Builder (KIB) configuration to be overridden (see page 1330). DKP advises not to use usernames and passwords for security reasons, but instead to use private and public keys. If that is not possible, passwords should be generated and a minimum of 20 characters long.

While creating the base OS image, it is important to take into consideration the following elements:

- Storage configuration: D2iQ recommends customizing disk partitions and not configuring a SWAP partition.
- Network configuration: as KIB must download and install packages, activating the network is required.
- Connect to Red Hat: if using RHEL, registering with Red Hat is required to configure software repositories and install software packages.
- Software selection: D2iQ recommends choosing **Minimal Install**.
- DKP recommends to install with the packages provided by the operating system package managers. Use the version that corresponds to the major version of your operating system.
- DKP advises not to use usernames and passwords for security reasons, but instead to use private and public keys. If that is not possible, passwords should be generated and a minimum of 20 characters long.

7.15.10.3.3.1 Disk Size

For each cluster you create using this base OS image, ensure you establish the disk size of the **root file system** based on:

- The minimum DKP storage requirements (see page 110).
- The minimum storage requirements for your organization.

849 https://github.com/mesosphere/konvoy-image-builder/blob/0523fd2c5e6e1ad1d4962f60a47039aa145a6e42/pkg/packer/manifests/vsphere/packer.pkr.hcl

Defaults

Clusters are created with a default disk size of 80 GB.

> ⚠ **For clusters created with the default disk size, the base OS image root file system must be exactly 80 GB.** The root file system cannot be reduced automatically when a machine first boots.

Customization

You can also specify a custom disk size when you create a cluster (see page 0) (see the flags of the Create new vSphere Cluster (see page 1462) command). This allows you to use one base OS image to create multiple clusters that have different storage requirements.

> ⚠ Before specifying a disk size when you create a cluster, take into account:
>
> 1. **For some base OS images, the custom disk size option has no effect on the size of the root file system.** This is because some root file systems, for example, those contained in an LVM Logical Volume, cannot be resized automatically when a machine first boots.
> 2. **The specified custom disk size must be equal to, or larger than the size of the base OS image root file system.** This is because a root file system cannot be reduced automatically when a machine first boots.

7.15.10.3.3.2 Next Step:

Create a CAPI VM Template (see page 1166)

7.15.10.4 Create an Air-gapped CAPI VM Template

You must have at least one image before creating a new cluster. As long as you have an image, this step in your configuration is not required each time since that image can be used to spin up a new cluster. However, if you need different images for different environments or providers, you will need to create a new custom image.

Create a vSphere template for your cluster from a base OS image

Using KIB (see page 1282), you can build an image without requiring access to the internet by providing an additional `--override` flag.

1. Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz`, extract the tarball to a local directory:

```
tar -xzvf dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz && cd dkp-v2.5.0/kib
```

2. Follow the instructions to build a vSphere template below and set the override `--overrides overrides/offline.yaml` flag.

7.15.10.4.1 Create a vSphere Template for Your Cluster from a Base OS Image

Using the base OS image created in a previous procedure, DKP creates the new vSphere template directly on the vCenter server.

1. Set the following vSphere environment variables on the bastion VM host:

```
export VSPHERE_SERVER=your_vCenter_APIserver_URL
export VSPHERE_USERNAME=your_vCenter_user_name
export VSPHERE_PASSWORD=your_vCenter_password
```

2. Copy the base OS image file created in the vSphere Client to your desired location on the bastion VM host, and make a note of the path and file name.
3. Create an `image.yaml` file and add the following variables for vSphere. DKP uses this file and these variables as inputs in the next step.
   NOTE: You will need to replace OS name below based on your OS - EX: “rhel-79” to “rhel-86”. See other YAML examples[850] for copy and paste below last step.

```
---
download_images: true
build_name: "rhel-79"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-
init-vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
```

850 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

```
  template: "base-rhel-7.9" # change default value with your base template name
  vsphere_guest_os_type: "rhel7_64Guest"
  guest_os_type: "rhel7-64"
  # goss params
  distribution: "RHEL"
  distribution_version: "7.9"
# Use following overrides to select the authentication method that can be used
with base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # if set to true, ssh_password and ssh_private_key
will be ignored
```

4. Create a vSphere VM template with your variation of the following command:

   ```
   konvoy-image build images/ova/<image.yaml>
   ```

   Any additional configurations can be added to this command using `--overrides` flags as shown below:

   a. for FIPS, add this flag - `--overrides overrides/fips.yaml`
   b. for air-gapped, add this flag `--overrides overrides/offline-fips.yaml`

5. The Konvoy Image Builder (see page 1305) (KIB) uses the values in `image.yaml` and the input base OS image to create a vSphere template directly on the vCenter server. This template contains the required artifacts needed to create a Kubernetes cluster.
   When KIB provisions the OS image[851] successfully, it creates a manifest file. The `artifact_id` field of this file contains the name of the AMI ID (AWS), template name (vSphere), or image name (GCP/Azure), for example:

   ```
   {
         "name": "vsphere-clone",
         "builder_type": "vsphere-clone",
         "build_time": 1644985039,
         "files": null,
         "artifact_id": "konvoy-ova-vsphere-rhel-84-1.21.6-1644983717",
         "packer_run_uuid": "260e8110-77f8-ca94-e29e-ac7a2ae779c8",
         "custom_data": {
           "build_date": "2022-02-16T03:55:17Z",
           "build_name": "vsphere-rhel-84",
           "build_timestamp": "1644983717",
           [...]
   ```

851 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

```
      }
    }
```

6. Next steps are to deploy a DKP cluster using your vSphere template.

7.15.10.4.2 Next Step:

vSphere Air-gapped Seed Docker Registry (see page 1169)

7.15.10.5 vSphere Air-gapped Seed Registry

Before creating a Kubernetes cluster, you need the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1162) and either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1349) page for more information.

1. Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz` , extract the tarball to a local directory:

```
tar -xzvf dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz && cd dkp-v2.5.0
```

2. Set an environment variable with your registry address with this command:

```
export REGISTRY_ADDRESS=<registry-address>:<registry-port>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

3. Execute the following command to load the air-gapped image bundle into your private registry:

```
dkp push image-bundle --image-bundle ./container-images/konvoy-image-bundle-
v2.5.0.tar --to-registry $REGISTRY_ADDRESS --to-registry-username
$REGISTRY_USERNAME --to-registry-password $REGISTRY_PASSWORD
```

> It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

7.15.10.5.1 Next Step:

vSphere Air-gapped Bootstrap (see page 1170)

7.15.10.6 vSphere Air-gapped Bootstrap

7.15.10.6.1 Prerequisites

Before you perform this procedure, ensure that you have created a CAPI VM template (see page 1166).

7.15.10.6.2 Bootstrap a kind cluster and CAPI controllers

Konvoy deploys all cluster lifecycle services to a bootstrap cluster, which deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster, after which the workload cluster manages its own lifecycle.

1. Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz`, extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz && cd dkp-v2.5.2
   ```

2. Load the bootstrap image on your bastion machine.

   ```
   docker load -i konvoy-bootstrap-image-v2.5.2.tar
   ```

   ```
   podman load -i konvoy-bootstrap-image-v2.5.2.tar
   ```

3. Create a bootstrap cluster:

   ```
   dkp create bootstrap --kubeconfig $HOME/.kube/config
   ```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

The output resembles this example:

```
✓ Creating a bootstrap cluster
✓ Initializing new CAPI components
```

4. Ensure that the CAPV controllers are present with the command:

```
kubectl get pods -n capv-system
```

The output resembles the following:

```
NAME                                       READY   STATUS    RESTARTS   AGE
capv-controller-manager-785c5978f-nnfns    1/1     Running   0          13h
```

7.15.10.6.3 Next Step:

Create a new Air-gapped vSphere Cluster (see page 1171)

7.15.10.7 Create a new Air-gapped vSphere Cluster

7.15.10.7.1 Prerequisites

Before you begin, be sure that you have created a Bootstrap Cluster (see page 1170)

7.15.10.7.2 Create a new vSphere Kubernetes cluster

Use the following steps to create a new, air-gapped vSphere cluster.

1. Configure your cluster to use an existing registry as a mirror when attempting to pull images:

   ⚠ **IMPORTANT:** The image must be created by the konvoy-image-builder[852] project in order to use the registry mirror feature.

```
export REGISTRY_URL=<https/http>://<registry-address>:<registry-port>
export REGISTRY_CA=<path to the CA on the bastion>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

852 https://github.com/mesosphere/konvoy-image-builder

- `REGISTRY_URL` : the address of an existing registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
- `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
- `REGISTRY_PASSWORD` : optional if username is not set.

2. Create a Kubernetes cluster by copying the following command and substituting the valid values for your environment. This example shows a common configuration. See dkp create cluster vsphere (see page 1462) reference for the full list of cluster creation options:
NOTE: Ensure your subnets (see page 929) do not overlap with your host subnet because they cannot be changed after cluster creation. If you need to change the kubernetes subnets, you must do this at cluster creation. The default subnets used in DKP are:

```
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 192.168.0.0/16
    services:
      cidrBlocks:
      - 10.96.0.0/12
```

Create cluster:

```
dkp create cluster vsphere
  --cluster-name ${CLUSTER_NAME} \
  --network <NETWORK_NAME> \
  --control-plane-endpoint-host <CONTROL_PLANE_IP> \
  --data-center <DATACENTER_NAME> \
  --data-store <DATASTORE_NAME> \
  --folder <FOLDER_NAME> \
  --server <VCENTER_API_SERVER_URL> \
  --ssh-public-key-file </path/to/key.pub> \
  --resource-pool <RESOURCE_POOL_NAME> \
  --vm-template konvoy-ova-vsphere-os-release-k8s_release-vsphere-timestamp \
  --virtual-ip-interface <ip_interface_name> \
  --extra-sans "127.0.0.1" \
  --registry-mirror-url=${REGISTRY_URL} \
  --registry-mirror-cacert=${REGISTRY_CA} \
  --registry-mirror-username=${REGISTRY_USERNAME} \
  --registry-mirror-password=${REGISTRY_PASSWORD}
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

3. Inspect the created cluster resources with the command:

```
kubectl get clusters,kubeadmcontrolplanes,machinedeployments
```

4. Use the `wait` command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=20m
```

```
cluster.cluster.x-k8s.io/${CLUSTER_NAME} condition met
```

The `READY` status becomes `True` after the cluster control-plane becomes Ready in one of the following steps.

After DKP creates the objects on the API server, the Cluster API controllers reconcile them, creating infrastructure and machines. As the controllers progress, they update the Status of each object.

5. Run the DKP `describe` command to monitor the current status of the cluster:

```
dkp describe cluster -c ${CLUSTER_NAME}
```

```
NAME                                                          READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/e2e-airgapped-1                                       True
13h
├──ClusterInfrastructure - VSphereCluster/e2e-airgapped-1      True
13h
├──ControlPlane - KubeadmControlPlane/e2e-airgapped-1-control-plane  True
13h
│  ├──Machine/e2e-airgapped-1-control-plane-7llgd              True
13h
│  ├──Machine/e2e-airgapped-1-control-plane-vncbl              True
13h
```

```
│ └──Machine/e2e-airgapped-1-control-plane-wbgrm                    True
13h
└──Workers
   └──MachineDeployment/e2e-airgapped-1-md-0                        True
13h
   ├──Machine/e2e-airgapped-1-md-0-74c849dc8c-67rv4                 True
13h
   ├──Machine/e2e-airgapped-1-md-0-74c849dc8c-n2skc                 True
13h
   ├──Machine/e2e-airgapped-1-md-0-74c849dc8c-nkftv                 True
13h
   └──Machine/e2e-airgapped-1-md-0-74c849dc8c-sqklv                 True
13h
```

6. As they progress, the controllers also create Events, which you can list using the command:

   ```
   kubectl get events | grep ${CLUSTER_NAME}
   ```

   For brevity, this example uses `grep`. You can also use separate commands to get Events for specific objects, such as `kubectl get events --field-selector involvedObject.kind="VSphereCluster"` and `kubectl get events --field-selector involvedObject.kind="VSphereMachine"`.

You can make the cluster self-managed (see page 1156) with the information in the linked page.

7.15.10.7.2.1 Next Step:

You can explore your new cluster (see page 1178).

7.15.10.7.3 Known limitations

NOTE: Be aware of these limitations in the current release of DKP Konvoy.

- The DKP Konvoy version used to create a bootstrap cluster must match the DKP Konvoy version used to create a workload cluster.
- DKP Konvoy supports deploying one workload cluster.
- DKP Konvoy generates a set of objects for one Node Pool.
- DKP Konvoy does not validate edits to cluster objects.

7.15.10.8 Make vSphere Air-gapped Cluster Self-Managed

Konvoy deploys all cluster lifecycle services to a bootstrap cluster, which then deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster, which makes the workload cluster self-managed. This section describes how to make a workload cluster self-managed.

> Before starting, ensure you create a workload cluster as described in Create a New vSphere Cluster or Create a New Air-gapped vSphere Cluster.

This page contains instructions on how to make your cluster self-managed. This is necessary if there is only one cluster in your environment, or if this cluster should become the Management cluster in a multi-cluster environment.

> If you already have a self-managed or Management cluster in your environment, skip this page.

7.15.10.8.1 Make the new Kubernetes cluster manage itself

Follow these steps:

1. Deploy cluster lifecycle services on the workload cluster:

   ```
   dkp create capi-components --kubeconfig ${CLUSTER_NAME}.conf
   ```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).
   The output is similar to this:

   ```
   ✓ Initializing new CAPI components
   ```

2. Move the Cluster API objects from the bootstrap to the workload cluster:

   The cluster lifecycle services on the workload cluster are ready, but the workload cluster configuration is on the bootstrap cluster. The `move` command moves the configuration, which takes

the form of Cluster API Custom Resource objects, from the bootstrap to the workload cluster. This process is also called a Pivot[853].

```
dkp move capi-resources --to-kubeconfig ${CLUSTER_NAME}.conf
```

```
✓ Moving cluster resources
You can now view resources in the moved cluster by using the --kubeconfig flag
with kubectl. For example: kubectl --kubeconfig=vsphere-example.conf get nodes
```

NOTE: To ensure only one set of cluster lifecycle services manages the workload cluster, Konvoy first pauses reconciliation of the objects on the bootstrap cluster, then creates the objects on the workload cluster. As Konvoy copies the objects, the cluster lifecycle services on the workload cluster reconcile the objects. The workload cluster becomes self-managed after Konvoy creates all the objects. If it fails, the move command can be safely retried.

3. Wait for the cluster control-plane to be ready:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf wait --for=condition=ControlPlaneRead
y "clusters/${CLUSTER_NAME}" --timeout=20m
```

```
cluster.cluster.x-k8s.io/vsphere-example condition met
```

4. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

   NOTE: After moving the cluster lifecycle services to the workload cluster, remember to use Konvoy with the workload cluster kubeconfig.

```
dkp describe cluster --kubeconfig ${CLUSTER_NAME}.conf -c ${CLUSTER_NAME}
```

```
NAME                                                                READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/vsphere-example-1                                           True
13h
```

853 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

```
├──ClusterInfrastructure - VSphereCluster/vsphere-example-1           True
13h
├──ControlPlane - KubeadmControlPlane/vsphere-example-control-plane   True
13h
│  ├──Machine/vsphere-example-control-plane-7llgd                     True
13h
│  ├──Machine/vsphere-example-control-plane-vncbl                     True
13h
│  └──Machine/vsphere-example-control-plane-wbgrm                     True
13h
└──Workers
   └──MachineDeployment/vsphere-example-md-0                          True
13h
   ├──Machine/vsphere-example-md-0-74c849dc8c-67rv4                   True
13h
   ├──Machine/vsphere-example-md-0-74c849dc8c-n2skc                   True
13h
   ├──Machine/vsphere-example-md-0-74c849dc8c-nkftv                   True
13h
   └──Machine/vsphere-example-md-0-74c849dc8c-sqklv                   True
13h
```

5. Remove the bootstrap cluster, if desired, as the workload cluster is now self-managed:

```
dkp delete bootstrap
```

```
✓ Deleting bootstrap cluster
```

7.15.10.8.2 Known limitations

Be aware of these limitations in the current release of DKP Konvoy.

- Before making a workload cluster self-managed, be sure that its control plane nodes have sufficient permissions for running Cluster API controllers.
- DKP Konvoy supports moving only one set of cluster objects from the bootstrap cluster to the workload cluster, or vice-versa.
- DKP Konvoy only supports moving all namespaces in the cluster; DKP does not support migration of individual namespaces.

7.15.10.8.3 Next Step:

Explore vSphere Air-gapped Cluster (see page 1178)

7.15.10.9 Explore vSphere Air-gapped Cluster

7.15.10.9.1 Get the `kubeconfig` File for the New Kubernetes Cluster

1. Fetch the kubeconfig file with the command:

```
dkp get kubeconfig -c ${air-gapped_NAME} > ${air-gapped_NAME}.conf
```

7.15.10.9.2 Create a StorageClass with a vSphere Datastore

Follow these steps:

1. Access the Datastore tab in the vSphere client and select a datastore by name.
2. Copy the URL of that datastore from the information dialog that displays.
3. Return to the DKP CLI, and delete the existing `StorageClass` with the command:

```
kubectl delete storageclass vsphere-raw-block-sc
```

4. Run the following command to create a new StorageClass, supplying the correct values for your environment:

```
cat <<EOF > vsphere-raw-block-sc.yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
  name: vsphere-raw-block-sc
provisioner: csi.vsphere.vmware.com
parameters:
  datastoreurl: "<url>"
volumeBindingMode: WaitForFirstConsumer
EOF
```

7.15.10.9.3 Explore Nodes and Pods in the New Cluster

1. List the Nodes with this command:

```
kubectl --kubeconfig=${air-gapped_NAME}.conf get nodes
```

NOTE: It may take a few minutes for the Status to move to Ready while the Pod network is deployed. The Node's Status should change to Ready soon after the calico-node DaemonSet Pods are Ready.

The output resembles this example:

```
NAME                                          STATUS   ROLES                  AGE
VERSION
d2iq-e2e-air-gapped-1-control-plane-7llgd     Ready    control-plane,master   20h
v1.25.4
d2iq-e2e-air-gapped-1-control-plane-vncbl     Ready    control-plane,master   19h
v1.25.4
d2iq-e2e-air-gapped-1-control-plane-wbgrm     Ready    control-plane,master   19h
v1.25.4
d2iq-e2e-air-gapped-1-md-0-74c849dc8c-67rv4   Ready    <none>                 19h
v1.25.4
d2iq-e2e-air-gapped-1-md-0-74c849dc8c-n2skc   Ready    <none>                 19h
v1.25.4
d2iq-e2e-air-gapped-1-md-0-74c849dc8c-nkftv   Ready    <none>                 19h
v1.25.4
d2iq-e2e-air-gapped-1-md-0-74c849dc8c-sqklv   Ready    <none>                 19h
v1.25.4
```

2. List the pods with the command:

```
kubectl --kubeconfig=${air-gapped_NAME}.conf get pods -A
```

The output resembles the following example:

```
NAMESPACE              NAME
READY   STATUS     RESTARTS      AGE
calico-system          calico-kube-controllers-57fbd7bd59-qqd96
1/1     Running    0             20h
calico-system          calico-node-2m524
1/1     Running    3 (19h ago)   19h
```

```
calico-system            calico-node-bbhg5
1/1     Running    0             20h
calico-system            calico-node-cc5lf
1/1     Running    2 (19h ago)   19h
calico-system            calico-node-cwg7x
1/1     Running    1 (19h ago)   19h
calico-system            calico-node-d59hn
1/1     Running    1 (19h ago)   19h
calico-system            calico-node-qmmcz
1/1     Running    0             19h
calico-system            calico-node-wdqhx
1/1     Running    0             19h
calico-system            calico-typha-655489d8cc-b5jnt
1/1     Running    0             20h
calico-system            calico-typha-655489d8cc-q92x9
1/1     Running    0             19h
calico-system            calico-typha-655489d8cc-vjlkx
1/1     Running    0             19h
kube-system              cluster-autoscaler-68c759fbf6-7d2ck
0/1     Init:0/1   0             20h
kube-system              coredns-78fcd69978-qn4qt
1/1     Running    0             20h
kube-system              coredns-78fcd69978-wqpmg
1/1     Running    0             20h
kube-system              etcd-d2iq-e2e-air-gapped-1-control-plane-7llgd
1/1     Running    0             20h
kube-system              etcd-d2iq-e2e-air-gapped-1-control-plane-vncbl
1/1     Running    0             19h
kube-system              etcd-d2iq-e2e-air-gapped-1-control-plane-wbgrm
1/1     Running    0             19h
kube-system              kube-apiserver-d2iq-e2e-air-gapped-1-control-plane-7llgd
1/1     Running    0             20h
kube-system              kube-apiserver-d2iq-e2e-air-gapped-1-control-plane-vncbl
1/1     Running    0             19h
kube-system              kube-apiserver-d2iq-e2e-air-gapped-1-control-plane-wbgrm
1/1     Running    0             19h
kube-system              kube-controller-manager-d2iq-e2e-air-gapped-1-control-
plane-7llgd   1/1     Running    1 (19h ago)   20h
kube-system              kube-controller-manager-d2iq-e2e-air-gapped-1-control-plane-
vncbl   1/1     Running    0             19h
kube-system              kube-controller-manager-d2iq-e2e-air-gapped-1-control-plane-
wbgrm   1/1     Running    0             19h
kube-system              kube-proxy-cpscs
1/1     Running    0             19h
kube-system              kube-proxy-hhmxq
1/1     Running    0             19h
kube-system              kube-proxy-hxhnk
1/1     Running    0             19h
kube-system              kube-proxy-nsrbp
1/1     Running    0             19h
kube-system              kube-proxy-scxfg
1/1     Running    0             20h
```

```
kube-system              kube-proxy-tth4k
1/1     Running    0             19h
kube-system              kube-proxy-x2xfx
1/1     Running    0             19h
kube-system              kube-scheduler-d2iq-e2e-air-gapped-1-control-plane-7llgd
1/1     Running    1 (19h ago)   20h
kube-system              kube-scheduler-d2iq-e2e-air-gapped-1-control-plane-vncbl
1/1     Running    0             19h
kube-system              kube-scheduler-d2iq-e2e-air-gapped-1-control-plane-wbgrm
1/1     Running    0             19h
kube-system              kube-vip-d2iq-e2e-air-gapped-1-control-plane-7llgd
1/1     Running    1 (19h ago)   20h
kube-system              kube-vip-d2iq-e2e-air-gapped-1-control-plane-vncbl
1/1     Running    0             19h
kube-system              kube-vip-d2iq-e2e-air-gapped-1-control-plane-wbgrm
1/1     Running    0             19h
kube-system              vsphere-cloud-controller-manager-4zj7q
1/1     Running    0             19h
kube-system              vsphere-cloud-controller-manager-87tgm
1/1     Running    0             19h
kube-system              vsphere-cloud-controller-manager-xqmn4
1/1     Running    1 (19h ago)   20h
node-feature-discovery   node-feature-discovery-master-84c67dcbb6-txfw9
1/1     Running    0             20h
node-feature-discovery   node-feature-discovery-worker-8tg2l
1/1     Running    3 (19h ago)   19h
node-feature-discovery   node-feature-discovery-worker-c5f6q
1/1     Running    0             19h
node-feature-discovery   node-feature-discovery-worker-fjfkm
1/1     Running    0             19h
node-feature-discovery   node-feature-discovery-worker-x6tz8
1/1     Running    0             19h
tigera-operator          tigera-operator-d499f5c8f-r2srj
1/1     Running    1 (19h ago)   20h
vmware-system-csi        vsphere-csi-controller-7ffd6884cc-d7rql
7/7     Running    5 (19h ago)   20h
vmware-system-csi        vsphere-csi-controller-7ffd6884cc-k82cm
7/7     Running    2 (19h ago)   20h
vmware-system-csi        vsphere-csi-controller-7ffd6884cc-qttkp
7/7     Running    1 (19h ago)   20h
vmware-system-csi        vsphere-csi-node-678hw
3/3     Running    0             19h
vmware-system-csi        vsphere-csi-node-6tbsh
3/3     Running    0             19h
vmware-system-csi        vsphere-csi-node-9htwr
3/3     Running    5 (20h ago)   20h
vmware-system-csi        vsphere-csi-node-g8r6l
3/3     Running    0             19h
vmware-system-csi        vsphere-csi-node-ghmr6
3/3     Running    0             19h
vmware-system-csi        vsphere-csi-node-jhvgm
3/3     Running    0             19h
```

```
vmware-system-csi        vsphere-csi-node-rp77r
3/3     Running    0             19h
```

When you are ready, delete your cluster and clean up your environment (see page 1184).

7.15.11 vSphere Certificate Renewal

7.15.11.1 Configure Automated Renewal for Managed Kubernetes PKI Certficates

7.15.11.2 Certificate Renewal

During cluster creation, Kubernetes establishes a Public Key Infrastructure (PKI) for generating the TLS certificates needed for securing cluster communication for various components such as `etcd`, `kubernetes-apiserver` and `kube-proxy`. The certificates created by these components have a default expiration of one year and are renewed when an administrator updates the cluster.

Kubernetes provides a facility to renew all certificates automatically during control plane updates. For administrators who need long-running clusters or clusters that are not upgraded often, `dkp` provides automated certificate renewal, without a cluster upgrade.

7.15.11.2.1 Requirements

This feature requires that you install Python 3.5 or higher version on all control plane hosts.

7.15.11.3 Prerequisites

- Complete the bootstrap Cluster Lifecycle (see page 1142) topic.

7.15.11.3.1 Create a cluster with automated certificate renewal

To enable the automated certificate renewal, create a Konvoy cluster using the `certificate-renew-interval` flag:

```
dkp create cluster vsphere --certificate-renew-interval=60 --cluster-name=long-
running
```

The `certificate-renew-interval` is the number of days after which Kubernetes-managed PKI certificates will be renewed. For example, an `certificate-renew-interval` value of 30 means the certificates are renewed every 30 days.

7.15.11.3.2 Technical details

The following manifests are modified on the control plane hosts, and are located at `/etc/kubernetes/manifests`. Modifications to these files requires SUDO access.

```
kube-controller-manager.yaml
kube-apiserver.yaml
kube-scheduler.yaml
kube-proxy.yaml
```

The following annotation indicates the time each component was reset:

```
metadata:
  annotations:
    konvoy.d2iq.io/restartedAt: $(date +%s)
```

This only occurs when the PKI certificates are older than the interval given at cluster creation time. This is activated by a `systemd` `timer` called `renew-certs.timer` that triggers an associated `systemd` service called `renew-certs.service` that runs on all of the control plane hosts.

7.15.11.3.3 Debugging

To debug the automatic certificate renewal feature, a cluster administrator can look at several different components to see if the certificates were renewed. For example, an administrator might start with a look at the control plane pod definition to check the last reset time. To determine if a scheduler pod was properly reset, run the command:

```
kubectl get pod -n kube-system kube-scheduler-nodename -o yaml
```

The output of the command is similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    konvoy.d2iq.io/restartedAt: "1626124940.735733"
```

Administrators who want more details on the execution of the `systemd` service can use `ssh` to connect to the control plane hosts, and then use the `systemctl` and `journalctl` commands that follow to help diagnose potential issues.

7.15.11.3.3.1 Check timer status

To check the status of the timers, when they last ran, and when they are scheduled to run next, use the command:

```
systemctl list-timers
```

7.15.11.3.3.2 Check renew certs status

To check the status of the `renew-certs` service, use the command:

```
systemctl status renew-certs
```

7.15.11.3.3.3 Review logs

To get the logs of the last run of the service, use the command:

```
journalctl logs -u renew-certs
```

7.15.12 Delete vSphere Cluster

7.15.12.1 Prepare to delete a self-managed workload cluster

> A self-managed workload cluster cannot delete itself. If your workload cluster is self-managed, you must create a bootstrap cluster and move the cluster lifecycle services to the bootstrap cluster before deleting the workload cluster.

If you did not make your workload cluster self-managed, as described in Make New Cluster Self-Managed (see page 1156), see Delete the workload cluster (see page 1187).

1. Create a bootstrap cluster:

   The bootstrap cluster will host the Cluster API controllers that reconcile the cluster objects marked for deletion. To avoid using the wrong kubeconfig, the following steps use **explicit** kubeconfig paths and contexts.

   ```
   dkp create bootstrap --kubeconfig $HOME/.kube/config
   ```

   The output resembles this example:

   ```
   ✓ Creating a bootstrap cluster
   ✓ Initializing new CAPI components
   ```

2. Move the Cluster API objects from the workload to the bootstrap cluster: The cluster lifecycle services on the bootstrap cluster are ready, but the workload cluster configuration is on the workload cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the workload to the bootstrap cluster. This process is also called a Pivot[854].

   ```
   dkp move \
       --from-kubeconfig ${CLUSTER_NAME}.conf \
       --from-context konvoy-${CLUSTER_NAME}-admin@konvoy-${CLUSTER_NAME} \
       --to-kubeconfig $HOME/.kube/config \
       --to-context kind-konvoy-capi-bootstrapper
   ```

   ```
   INFO[2021-06-09T11:47:11-07:00] Running pivot command
   fromClusterKubeconfig=aws-example.conf fromClusterContext= src="move/
   move.go:83" toClusterKubeconfig=/home/clusteradmin/.kube/config
   toClusterContext=
   INFO[2021-06-09T11:47:36-07:00] Pivot operation complete.
   src="move/move.go:108"
   INFO[2021-06-09T11:47:36-07:00] You can now view resources in the moved cluster
   by using the --kubeconfig flag with kubectl. For example: kubectl --
   kubeconfig=/home/clusteradmin/.kube/config get nodes  src="move/move.go:155"
   ```

3. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

854 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

```
dkp describe cluster --kubeconfig $HOME/.kube/config -c ${CLUSTER_NAME}
```

```
NAME                                                                READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/d2iq-e2e-cluster_name-1                                     True
13h
├──ClusterInfrastructure - VSphereCluster/d2iq-e2e-cluster_name-1    True
13h
├──ControlPlane - KubeadmControlPlane/d2iq-control-plane            True
13h
│  ├──Machine/d2iq--control-plane-7llgd                             True
13h
│  ├──Machine/d2iq--control-plane-vncbl                             True
13h
│  └──Machine/d2iq--control-plane-wbgrm                             True
13h
└──Workers
   └──MachineDeployment/d2iq--md-0                                  True
13h
   ├──Machine/d2iq--md-0-74c849dc8c-67rv4                           True
13h
   ├──Machine/d2iq--md-0-74c849dc8c-n2skc                           True
13h
   ├──Machine/d2iq--md-0-74c849dc8c-nkftv                           True
13h
   └──Machine/d2iq--md-0-74c849dc8c-sqklv                           True
13h
```

After moving the cluster lifecycle services to the workload cluster, remember to use dkp with the workload cluster kubeconfig. Use DKP with the bootstrap cluster to delete the workload cluster.

4. Wait for the cluster control-plane to be ready:

```
kubectl --kubeconfig $HOME/.kube/config wait --for=condition=controlplaneready
"clusters/${CLUSTER_NAME}" --timeout=60m
```

The output should be similar to this example:

```
d2iq-e2e-cluster-1-control-plane/vsphere-example condition met
```

Persistent Volumes (PVs) are not deleted automatically by design in order to preserve your data. However, they take up storage space if not deleted. You must delete PVs manually. Information for backup of a cluster and PVs is on the page in documentation called Back up your Cluster's Applications and Persistent Volumes (see page 736) .

With Vsphere clusters, `dkp delete` doesn't delete the virtual disks backing the PVs for DKP add ons. Therefore internal VMware cluster runs out of storage eventually. Theses PVs are only visible if VSAN is installed which gives users a Container Native Storage tab.

7.15.12.2 Delete the workload cluster

1. Make sure your vSphere credentials are up-to-date. Refresh the credentials using this command:

```
dkp update bootstrap credentials vsphere --kubeconfig $HOME/.kube/config
```

2. To delete a cluster, you would use `dkp delete cluster` and pass in the name of the cluster you are trying to delete with `--cluster-name` flag. You would use `kubectl get clusters` to get those details (`--cluster-name` and `--namespace`) of the Kubernetes cluster to delete it.
NOTE: Do not use `dkp get clusters` since that gets you Kommander cluster details rather than Konvoy kubernetes cluster details.

```
kubectl get clusters
```

3. Delete the Kubernetes cluster and wait a few minutes:

   Before deleting the cluster, DKP deletes all Services of type LoadBalancer on the cluster.

   To skip this step, use the flag `--delete-kubernetes-resources=false` .

```
dkp delete cluster --cluster-name=${CLUSTER_NAME} --kubeconfig $HOME/.kube/
config
```

```
INFO[2022-03-30T11:53:42-07:00] Running cluster delete command
clusterName=d2iq-e2e-cluster-1 managementClusterKubeconfig= namespace=default
 src="cluster/delete.go:95"
```

```
INFO[2022-03-30T11:53:42-07:00] Waiting for cluster to be fully deleted
src="cluster/delete.go:123"
INFO[2022-03-30T12:14:03-07:00] Deleted default/d2iq-e2e-cluster-1 cluster
src="cluster/delete.go:129"
```

7.15.12.3 Delete the Bootstrap Cluster

After the workload cluster is deleted, you can delete the bootstrap cluster.

```
dkp delete bootstrap --kubeconfig $HOME/.kube/config
```

```
INFO[2021-06-09T12:15:20-07:00] Deleting bootstrap cluster
src="bootstrap/bootstrap.go:182"
```

7.15.12.4 Next Step:

Once your cluster is built in the Konvoy component of DKP for your infrastructure/environment, you will install the Kommander (see page 1227) component of DKP to see your dashboard and continue customization.

7.15.12.5 Known limitations

Be aware of these limitations in the current release of DKP Konvoy.

- The DKP Konvoy version used to create the workload cluster must match the DKP Konvoy version used to delete the workload cluster.

7.15.13 Manage vSphere Node Pools

Node pools are part of a cluster and managed as a group, and you can use a node pool to manage a group of machines using the same common properties. When Konvoy creates a new default cluster, there is one node pool for the worker nodes, and all nodes in that new node pool have the same configuration.

You can create additional node pools for more specialized hardware or configuration. For example, if you want to tune your memory usage on a cluster where you need maximum memory for some machines and minimal memory on other machines, you would create a new node pool with those specific resource needs.

Konvoy implements node pools using Cluster API MachineDeployments[855].

7.15.13.1 Create vSphere Node Pools

Creating a node pool is useful when you need to run workloads that require machines with specific resources, such as a GPU, additional memory, or specialized network or storage hardware.

7.15.13.1.1 Prepare the environment

Follow these steps:

1. Set the environment variable to the name you assigned this cluster with the command:

```
export CLUSTER_NAME=my-vsphere-cluster
```

2. If your workload cluster is self-managed, as described in Make the New Cluster Self-Managed (see page 1156), configure `kubectl` to use the kubeconfig for the cluster:

```
export KUBECONFIG=${CLUSTER_NAME}.conf
```

3. Define your node pool name:

```
export NODEPOOL_NAME=example
```

7.15.13.1.2 Create a vSphere node pool

Create a new vSphere node pool with 3 replicas using this command:

```
dkp create nodepool vsphere ${NODEPOOL_NAME} \
  --cluster-name=${CLUSTER_NAME} \
  --network=example_network \
  --data-center=example_datacenter \
  --data-store=example_datastore \
  --folder=example_folder \
  --server=example_vsphere_api_server_url\
  --resource-pool=example_resource_pool \
  --vm-template=example_vm_template \
```

855 https://cluster-api.sigs.k8s.io/developer/architecture/controllers/machine-deployment.html

```
  --replicas=3
```

The output resembles this example:

```
machinedeployment.cluster.x-k8s.io/example created
vspheremachinetemplate.infrastructure.cluster.x-k8s.io/example created
kubeadmconfigtemplate.bootstrap.cluster.x-k8s.io/example created
 ✓ Creating default/example nodepool resources
```

This example uses default values for brevity. Advanced users can use a combination of the `--dry-run` and `--output=yaml` or `--output-directory=<your-target-directory>/` flags to get a complete set of node pool objects to modify locally or store in version control.

7.15.13.2 List vSphere Node Pools

7.15.13.2.1 Listing node pools

Use this command to list the node pools of a given cluster. This returns specific properties of each node pool so that you can see the name of the MachineDeployments.

To list all node pools for a managed cluster, run the command:

```
dkp get nodepools --cluster-name=${CLUSTER_NAME} --kubeconfig=${CLUSTER_NAME}.conf
```

The expected output is similar to the following example, indicating the desired size of the node pool, the number of replicas ready in the node pool, and the Kubernetes version those nodes are running:

```
NODEPOOL                  DESIRED   READY   KUBERNETES
VERSION
demo-cluster-md-0         4         4       v1.25.4
example                   3         0       v1.25.4
```

7.15.13.3 Scale vSphere Node Pools

7.15.13.3.1 Scaling node pools

While you can run Cluster Autoscaler (see page 1193), you can also manually scale your node pools up or down when you need more finite control over your environment. For example, if you require 10 machines to run a process, you can manually set the scaling to run those 10 machines only. However, if also using the Cluster Autoscaler, you must stay within your minimum and maximum bounds.

7.15.13.3.1.1 Scaling up node pools

To scale up a node pool in a cluster, run the command that follows, replacing the value 5 with the actual number of replicas you need:

```
dkp scale nodepools ${NODEPOOL_NAME} --replicas=5 --cluster-name=${CLUSTER_NAME}
```

Your output should be similar to this example, indicating the scaling is in progress:

```
INFO[2021-07-26T08:54:35-07:00] Running scale nodepool command
clusterName=demo-cluster managementClusterKubeconfig= namespace=default src="nodepool
/scale.go:82"
INFO[2021-07-26T08:54:35-07:00] Nodepool example scaled to 5 replicas
clusterName=demo-cluster managementClusterKubeconfig= namespace=default src="nodepool
/scale.go:94"
```

After a few minutes, you can list the node pools with the command:

```
dkp get nodepools --cluster-name=${CLUSTER_NAME} --kubeconfig=${CLUSTER_NAME}.conf
```

Your output should be similar to this example, with the number of DESIRED and READY replicas increased to 5:

```
NODEPOOL                    DESIRED             READY               KUBERNETES
VERSION
example                     5                   5                   v1.25.4
demo-cluster-md-0           4                   4                   v1.25.4
```

7.15.13.3.1.2 Scaling down node pools

To scale down a node pool, run the command:

```
dkp scale nodepools ${NODEPOOL_NAME} --replicas=4 --cluster-name=${CLUSTER_NAME}
```

```
INFO[2021-07-26T08:54:35-07:00] Running scale nodepool command
clusterName=demo-cluster managementClusterKubeconfig= namespace=default src="nodepool
/scale.go:82"
INFO[2021-07-26T08:54:35-07:00] Nodepool example scaled to 4 replicas
clusterName=demo-cluster managementClusterKubeconfig= namespace=default src="nodepool
/scale.go:94"
```

In a default cluster, the nodes to delete are selected at random. This behavior is controller by CAPI's delete policy[856]. However, when using the Konvoy CLI to scale down a node pool, you can specify the Kubernetes Nodes you want to delete.

To do this, set the flag `--nodes-to-delete` with a list of nodes as shown in the next command. This adds an annotation `cluster.x-k8s.io/delete-machine=yes` to the matching Machine object that contains `status.NodeRef` with the node names from `--nodes-to-delete`.

```
dkp scale nodepools ${NODEPOOL_NAME} --replicas=3 --nodes-to-delete=<> --cluster-
name=${CLUSTER_NAME}
```

```
INFO[2021-07-26T08:54:35-07:00] Running scale nodepool command
clusterName=demo-cluster managementClusterKubeconfig= namespace=default src="nodepool
/scale.go:82"
INFO[2021-07-26T08:54:35-07:00] Nodepool example scaled to 3 replicas
clusterName=demo-cluster managementClusterKubeconfig= namespace=default src="nodepool
/scale.go:94"
```

7.15.13.3.1.3 Scaling node pools when using cluster autoscaler

If you configured the cluster autoscaler[857] for the `demo-cluster-md-0` node pool, the value of `--replicas` must be within the minimum and maximum bounds.

For example, assuming you have the these annotations:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment ${NODEPOOL_NAME}
cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size=2
kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment ${NODEPOOL_NAME}
cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size=6
```

856 https://github.com/kubernetes-sigs/cluster-api/blob/v0.4.0/api/v1alpha4/machineset_types.go#L85-L105
857 https://docs.d2iq.com/dkp/konvoy/2.2/choose-infrastructure/vsphere/nodepools/cluster-autoscaler

Try to scale the node pool to 7 replicas with the command:

```
dkp scale nodepools ${NODEPOOL_NAME} --replicas=7 -c demo-cluster
```

This action results in an error similar to:

```
INFO[2021-07-26T09:46:37-07:00] Running scale nodepool command
clusterName=demo-cluster managementClusterKubeconfig= namespace=default src="nodepool
/scale.go:82"
Error: failed to scale nodepool: scaling MachineDeployment is forbidden: desired
replicas 7 is greater than the configured max size annotation cluster.x-k8s.io/
cluster-api-autoscaler-node-group-max-size: 6
```

Similarly, scaling down to a number of replicas less than the configured `min-size` also returns an error.

7.15.13.4 vSphere Cluster Autoscaler

7.15.13.4.1 Cluster Autoscaler

Cluster Autoscaler[858] provides the ability to automatically scale-up or scale-down the number of worker nodes in a cluster, based on the number of pending pods to be scheduled. Running the Cluster Autoscaler is optional.

Unlike Horizontal-Pod Autoscaler[859], Cluster Autoscaler does not depend on any Metrics server and does not need Prometheus or any other metrics source.

The Cluster Autoscaler looks at the following annotations on a MachineDeployment to determine its scale-up and scale-down ranges:

```
cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size
cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size
```

The full list of command line arguments to the Cluster Autoscaler controller is found this topic[860].

For more information about how Cluster Autoscaler works, see these documents:

- What is Cluster Autoscaler[861]

858 https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/cloudprovider/clusterapi
859 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-fast-is-hpa-when-combined-with-ca
860 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#what-are-the-parameters-to-ca
861 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#what-is-cluster-autoscaler

- How does scale-up work[862]
- How does scale-down work[863]
- CAPI Provider for Cluster Autoscaler[864]

7.15.13.4.1.1 Cluster Autoscaler Prerequisites

Before you begin, you must have:

- A Bootstrap Cluster Lifecycle (see page 1142).
- Created a new Kubernetes Cluster (see page 1145).
- A Self-Managed Cluster (see page 1156).

7.15.13.4.1.2 Run Cluster Autoscaler

The Cluster Autoscaler controller runs on the workload cluster. Upon creation of the workload cluster, this controller does not have all the objects required to function correctly until after a `dkp move` is issued from the bootstrap cluster.

Run the following steps to enable Cluster Autoscaler:

1. Ensure the Cluster Autoscaler controller is up and running (no restarts and no errors in the logs)

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf logs deployments/cluster-autoscaler
cluster-autoscaler -n kube-system -f
```

2. Enable Cluster Autoscaler by setting the min & max ranges

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment $
{NODEPOOL_NAME} cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size=2
kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment $
{NODEPOOL_NAME} cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size=6
```

3. The Cluster Autoscaler logs will show that the worker nodes are associated with node-groups and that pending pods are being watched.
4. To demonstrate that it is working properly, create a large deployment which will trigger pending pods.

```
cat <<EOF | kubectl --kubeconfig=${CLUSTER_NAME}.conf apply -f -
apiVersion: apps/v1
```

862 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-does-scale-up-work
863 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-does-scale-down-work
864 https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/cloudprovider/clusterapi

```
kind: Deployment
metadata:
  name: busybox-deployment
  labels:
    app: busybox
spec:
  replicas: 600
  selector:
    matchLabels:
      app: busybox
  template:
    metadata:
      labels:
        app: busybox
    spec:
      containers:
      - name: busybox
        image: busybox:latest
        command:
          - sleep
          - "3600"
        imagePullPolicy: IfNotPresent
      restartPolicy: Always
EOF
```

Cluster Autoscaler scales up the number of Worker Nodes until there are no pending pods.

5. Scale down the number of replicas for `busybox-deployment` with the command:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf scale --replicas=30 deployment/
busybox-deployment
```

6. Cluster Autoscaler starts to scale down the number of Worker Nodes after the default timeout of 10 minutes.

7.15.13.5 Replace a vSphere Node

7.15.13.5.1 Prerequisites

Before you begin, you must:

- Create a workload cluster (see page 1145) or create an air-gapped workload cluster (see page 1171).
- Make the new cluster self-managed (see page 1156).

7.15.13.5.2 Replace a worker node

In certain situations, you may want to delete a worker node and have Cluster API[865] replace it with a newly-provisioned machine.

1. Identify the name of the node to delete.

   List the nodes:

   ```
   kubectl --kubeconfig ${CLUSTER_NAME}.conf get nodes
   ```

   The output from this command resembles the following:

   ```
   NAME                                      STATUS   ROLES                  AGE
   VERSION
   d2iq-e2e-cluster-1-control-plane-7llgd    Ready    control-plane,master   20h
   v1.25.4
   d2iq-e2e-cluster-1-control-plane-vncbl    Ready    control-plane,master   20h
   v1.25.4
   d2iq-e2e-cluster-1-control-plane-wbgrm    Ready    control-plane,master   19h
   v1.25.4
   d2iq-e2e-cluster-1-md-0-74c849dc8c-67rv4  Ready    <none>                 20h
   v1.25.4
   d2iq-e2e-cluster-1-md-0-74c849dc8c-n2skc  Ready    <none>                 20h
   v1.25.4
   d2iq-e2e-cluster-1-md-0-74c849dc8c-nkftv  Ready    <none>                 20h
   v1.25.4
   d2iq-e2e-cluster-1-md-0-74c849dc8c-sqklv  Ready    <none>                 20h
   v1.25.4
   ```

2. Export a variable with the node name to use in the next steps:

   This example uses the name `d2iq-e2e-cluster-1-md-0-74c849dc8c-67rv4`.

   ```
   export NAME_NODE_TO_DELETE="d2iq-e2e-cluster-1-md-0-74c849dc8c-67rv4"
   ```

3. Delete the Machine resource with the command:

865 https://cluster-api.sigs.k8s.io/

```
NAME_MACHINE_TO_DELETE=$(kubectl --kubeconfig ${CLUSTER_NAME}.conf get machine
-ojsonpath="{.items[?
(@.status.nodeRef.name==\"$NAME_NODE_TO_DELETE\")].metadata.name}")
kubectl --kubeconfig ${CLUSTER_NAME}.conf delete machine
"$NAME_MACHINE_TO_DELETE"
```

```
machine.cluster.x-k8s.io "d2iq-e2e-cluster-1-md-0-74c849dc8c-67rv4" deleted
```

The command does not return immediately, but it does return after the Machine resource is deleted.

A few minutes after the Machine resource is deleted, the corresponding Node resource is also deleted.

4. Observe the Machine resource replacement using this command:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf get machinedeployment
```

```
NAME                      CLUSTER              REPLICAS   READY   UPDATED
UNAVAILABLE   PHASE       AGE   VERSION
d2iq-e2e-cluster-1-md-0   d2iq-e2e-cluster-1   4          3       4         1
ScalingUp   20h   v1.25.4
```

In this example, there exist 4 replicas, but only 3 are ready. One replica is unavailable, and the `ScalingUp` phase means a new Machine is being created.

5. Identify the replacement Machine using this command:

```
export NAME_NEW_MACHINE=$(kubectl --kubeconfig ${CLUSTER_NAME}.conf get
machines \
    -l=cluster.x-k8s.io/deployment-name=${CLUSTER_NAME}-md-0 \
    -ojsonpath='{.items[?(@.status.phase=="Provisioning")].metadata.name}
{"\n"}')
echo "$NAME_NEW_MACHINE"
```

If the output is empty, the new Machine has probably exited the `Provisioning` phase and entered the `Running` phase.

6. Identify the replacement Node using this command:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf get nodes \
    -o=jsonpath="{.items[?(@.metadata.annotations.cluster\.x-k8s\.io/
machine==\"$NAME_NEW_MACHINE\")].metadata.name}"
```

The output should be similar to this example:

```
d2iq-e2e-cluster-1-md-0-74c849dc8c-rc528
```

If the output is empty, the Node resource is not yet available, or does not yet have the expected annotation. Wait a few minutes, then repeat the command.

7.15.13.6 Delete vSphere Node Pools

Deleting a node pool deletes the Kubernetes nodes and the underlying infrastructure. DKP drains all nodes prior to deletion and reschedules the pods running on those nodes.

To delete a node pool from a managed cluster, run the command:

```
dkp delete nodepool ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME}
```

Here, `example` is the node pool to be deleted.

The expected output is similar to the following example, indicating the node pool is being deleted:

```
INFO[2021-07-28T17:14:26-07:00] Running nodepool delete command
Nodepool=example clusterName=d2iq-e2e-cluster-1 managementClusterKubeconfig=
namespace=default src="nodepool/delete.go:80"
```

Deleting an invalid node pool results in output similar to this example command output:

```
dkp delete nodepool ${CLUSTER_NAME}-md-invalid --cluster-name=${CLUSTER_NAME}

INFO[2021-07-28T17:11:44-07:00] Running nodepool delete command
Nodepool=demo-cluster-md-invalid clusterName=d2iq-e2e-cluster-1
 managementClusterKubeconfig= namespace=default src="nodepool/delete.go:80"
Error: failed to get nodepool with name demo-cluster-md-invalid in namespace default
: failed to get nodepool with name demo-cluster-md-invalid in namespace default :
machinedeployments.cluster.x-k8s.io "demo-cluster-md-invalid" not found
```

7.16 GCP Infrastructure

The following procedure describes creating a DKP cluster on GCP.

7.16.1 GCP Prerequisites

7.16.1.1 Prerequisites

Before beginning a DKP installation, verify that you have:

- An x86_64-based Linux or macOS machine with a supported version of the operating system.
- Download (see page 71) the `dkp` binary for Linux, or macOS. To check which version of DKP you installed for compatibility reasons, run the `dkp version -h` command (dkp version (see page 1534)).
- A Container engine/runtime installed is required to install DKP:
  - Version Docker®[866] container engine version 18.09.2 or higher installed for Linux or MacOS - On macOS, Docker runs in a virtual machine which needs configured with at least 8 GB of memory.
  - Version 4.0 of Podman[867] or higher for Linux. Host requirements found here: Host Requirements[868].
- kubectl[869] for interacting with the running cluster.
- Install the GCP `gcloud` CLI by following the https://cloud.google.com/sdk/docs/install

7.16.1.2 Control plane nodes

You must have at least three control plane nodes. Each control plane node should have at least:

- 4 cores
- 16 GiB memory

866 https://docs.docker.com/get-docker/
867 https://podman.io/getting-started/installation
868 https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
869 https://kubernetes.io/docs/tasks/tools/#kubectl

- Approximately 80 GiB of free space for the volume used for `/var/lib/kubelet` and `/var/lib/containerd`.
- Disk usage must be below 85% on the root volume.

DKP on GCP defaults to deploying an `n2-standard-4` instance with an 80GiB root volume for control plane nodes, which meets the above requirements.

7.16.1.3 Worker nodes

You must have at least four worker nodes. The specific number of worker nodes required for your environment can vary depending on the cluster workload and size of the nodes. Each worker node should have at least:

- 8 cores
- 32 GiB memory
- Around 80 GiB of free space for the volume used for `/var/lib/kubelet` and `/var/lib/containerd`.
- Disk usage must be below 85% on the root volume.

DKP on GCP defaults to deploying a `n2-standard-8` instance with an 80GiB root volume for worker nodes, which meets the above requirements.

7.16.1.4 GCP Prerequisite Roles

- If you are creating the cluster on a non-GCP instance or one that does not have the required `Editor` role:
  - (option 1) Create a GCP Service Account using the following `gcloud` commands:

```
export GCP_PROJECT=<your GCP project ID>
export GCP_SERVICE_ACCOUNT_USER=<some new service account user>
export GOOGLE_APPLICATION_CREDENTIALS="$HOME/.gcloud/credentials.json"

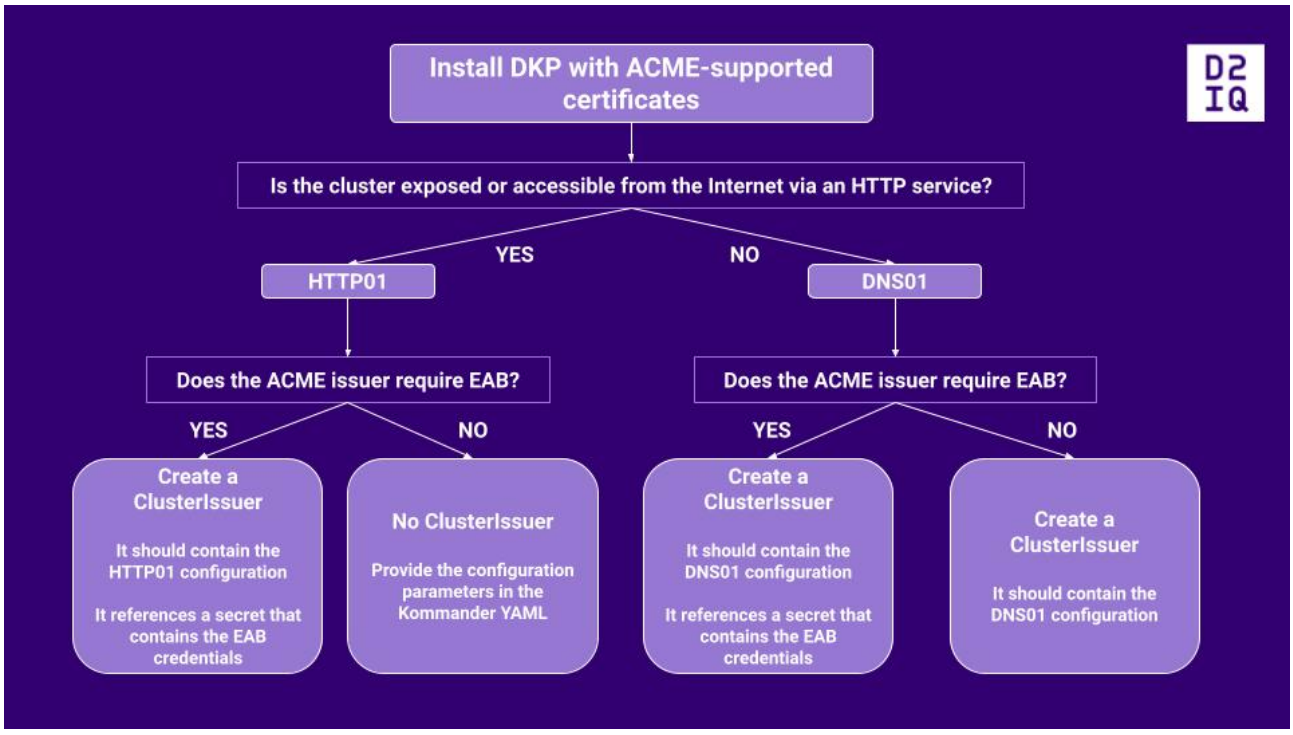
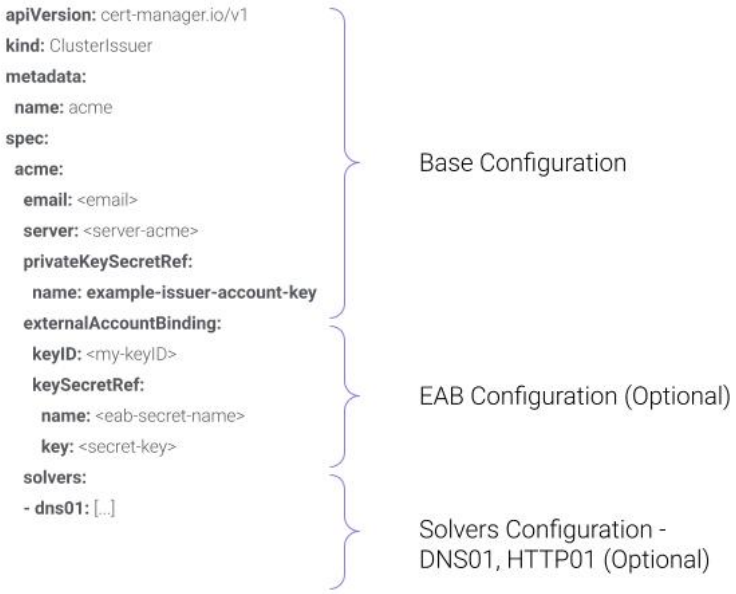
gcloud iam service-accounts create "$GCP_SERVICE_ACCOUNT_USER" --
project=$GCP_PROJECT
gcloud projects add-iam-policy-binding $GCP_PROJECT --
member="serviceAccount:
$GCP_SERVICE_ACCOUNT_USER@$GCP_PROJECT.iam.gserviceaccount.com" --
role=roles/editor
gcloud iam service-accounts keys create $GOOGLE_APPLICATION_CREDENTIALS
--iam-
account="$GCP_SERVICE_ACCOUNT_USER@$GCP_PROJECT.iam.gserviceaccount.com"
```

  - (option 2) Retrieve the credentials for an existing service account using the following `gcloud` commands:

```
export GCP_PROJECT=<your GCP project ID>
export GCP_SERVICE_ACCOUNT_USER=<existing service account user>
export GOOGLE_APPLICATION_CREDENTIALS="$HOME/.gcloud/credentials.json"

gcloud iam service-accounts keys create $GOOGLE_APPLICATION_CREDENTIALS
--iam-
account="$GCP_SERVICE_ACCOUNT_USER@$GCP_PROJECT.iam.gserviceaccount.com"
```

- Export the static credentials that will be used to create the cluster:

```
export GCP_B64ENCODED_CREDENTIALS=$(base64 < "$
{GOOGLE_APPLICATION_CREDENTIALS}" | tr -d '\n')
```

- To create a GCP Service Account with the `Editor` role, the user creating the GCP Service Account needs the `Editor`, `RoleAdministrator`, and `SecurityAdmin` roles. However, those pre-defined roles grant more permissions than the minimum set needed to create a DKP cluster.

NOTE: A minimal set of roles and permissions needed for the user creating the GCP Service Account is the `Editor` role plus the following additional permissions:

- `compute.disks.setIamPolicy`
- `compute.instances.setIamPolicy`
- `iam.roles.create`
- `iam.roles.delete`
- `iam.roles.update`
- `iam.serviceAccounts.setIamPolicy`
- `resourcemanager.projects.setIamPolicy`

For more information on GCP service accounts, see GCP's documentation: https://cloud.google.com/iam/docs/creating-managing-service-accounts

7.16.2 GCP Konvoy Image Builder

Konvoy Image Builder (KIB) is a complete solution for building Cluster API compliant images.

This procedure describes how to use the Konvoy Image Builder (see page 1282) (KIB) to create a Cluster API[870] compliant GCP image. GCP images contain configuration information and software to create a specific, pre-configured, operating environment. For example, you can create a GCP image of your current computer system settings and software. The GCP image can then be replicated and distributed, creating your computer system for other users. KIB uses variable overrides (see page 1330) to specify base image and container images to use in your new GCP image.

Google Cloud Platform does not publish images. You must first build the image using Konvoy Image Builder. For more information regarding images and clusters, refer to the GCP Infrastructure (see page 1199) section of the documentation.

7.16.2.1 Prerequisites

Before you begin, you must:

- Check the supported DKP version (see page 1543).
- Check the Supported Infrastructure Operating Systems (see page 62)
- Download the KIB (see page 1283) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
- Create a working `Docker` setup.
- On Debian-based Linux distributions, install a version of the cri-tools[871] package known to be compatible with both the Kubernetes and container runtime versions.
- Verify that your Google Cloud project does not have the Enable OS Login feature enabled. See below for more information:

The Enable OS Login feature is sometimes enabled by default in GCP projects. If the OS login feature is enabled, KIB will not be able to `ssh` to the VM instances it creates and will not be able to successfully create an image.
To check if it is enabled, use the commands on this page https://cloud.google.com/compute/docs/metadata/setting-custom-metadata#console_2 to inspect the metadata configured in in your project. If you find the the `enable-oslogin` flag set to TRUE, you must remove (or set it to FALSE) to successfully use KIB.

7.16.2.2 GCP Prerequisite Roles

- If you are creating your image on either a non-GCP instance or one that does not have the required roles (see page 1199):

870 https://cluster-api.sigs.k8s.io/
871 https://github.com/kubernetes-sigs/cri-tools

- (option 1) Create a service account using the following `gcloud` commands:

```
export GCP_PROJECT=<your GCP project ID>
export GCP_SERVICE_ACCOUNT_USER=<some new service account user>
export GOOGLE_APPLICATION_CREDENTIALS="$HOME/.gcloud/credentials.json"

gcloud iam service-accounts create "${GCP_SERVICE_ACCOUNT_USER}" --
project=${GCP_PROJECT}
gcloud projects add-iam-policy-binding ${GCP_PROJECT} --
member="serviceAccount:${GCP_SERVICE_ACCOUNT_USER}@$
{GCP_PROJECT}.iam.gserviceaccount.com" --role=roles/
compute.instanceAdmin.v1
gcloud projects add-iam-policy-binding ${GCP_PROJECT} --
member="serviceAccount:${GCP_SERVICE_ACCOUNT_USER}@$
{GCP_PROJECT}.iam.gserviceaccount.com" --role=roles/
iam.serviceAccountUser
gcloud iam service-accounts keys create $
{GOOGLE_APPLICATION_CREDENTIALS} --iam-account="$
{GCP_SERVICE_ACCOUNT_USER}@${GCP_PROJECT}.iam.gserviceaccount.com"
```

- (option 2) If you have already created a service account, retrieve the credentials for an existing service account using the following `gcloud` commands:

```
export GCP_PROJECT=<your GCP project ID>
export GCP_SERVICE_ACCOUNT_USER=<existing service account user>
export GOOGLE_APPLICATION_CREDENTIALS="$HOME/.gcloud/credentials.json"

gcloud projects add-iam-policy-binding ${GCP_PROJECT} --
member="serviceAccount:${GCP_SERVICE_ACCOUNT_USER}@$
{GCP_PROJECT}.iam.gserviceaccount.com" --role=roles/
compute.instanceAdmin.v1
gcloud projects add-iam-policy-binding ${GCP_PROJECT} --
member="serviceAccount:${GCP_SERVICE_ACCOUNT_USER}@$
{GCP_PROJECT}.iam.gserviceaccount.com" --role=roles/
iam.serviceAccountUser
gcloud iam service-accounts keys create $
{GOOGLE_APPLICATION_CREDENTIALS} --iam-account="$
{GCP_SERVICE_ACCOUNT_USER}@${GCP_PROJECT}.iam.gserviceaccount.com"
```

- Export the static credentials that will be used to create the cluster:

```
export GCP_B64ENCODED_CREDENTIALS=$(base64 < "$
{GOOGLE_APPLICATION_CREDENTIALS}" | tr -d '\n')
```

7.16.2.3 Build the GCP image

1. Run the `konvoy-image` command to build and validate the image:

```
./konvoy-image build gcp --project-id ${GCP_PROJECT} --network ${NETWORK_NAME}
images/gcp/ubuntu-2004.yaml
```

2. KIB will run and print out the name of the created image, you will use this name when creating a Kubernetes cluster. See sample output below:

```
...
==> ubuntu-2004-focal-v20220419: Deleting instance...
    ubuntu-2004-focal-v20220419: Instance has been deleted!
==> ubuntu-2004-focal-v20220419: Creating image...
==> ubuntu-2004-focal-v20220419: Deleting disk...
    ubuntu-2004-focal-v20220419: Disk has been deleted!
==> ubuntu-2004-focal-v20220419: Running post-processor: manifest
Build 'ubuntu-2004-focal-v20220419' finished after 7 minutes 46 seconds.

==> Wait completed after 7 minutes 46 seconds

==> Builds finished. The artifacts of successful builds are:
--> ubuntu-2004-focal-v20220419: A disk image was created: konvoy-ubuntu-2004-1-2
3-7-1658523168
--> ubuntu-2004-focal-v20220419: A disk image was created: konvoy-ubuntu-2004-1-2
3-7-1658523168
```

3. To find a list of images you have created in your account, run the following command:

```
gcloud compute images list --no-standard-images
```

Konvoy Image Builder (see page 1282) has a more detailed section in the documentation if you need to refer there for compatible versions with DKP and other specific information.

7.16.3 Bootstrap GCP

To create Kubernetes clusters, DKP uses Cluster API[872] (CAPI) controllers. These controllers run on a Kubernetes cluster. To get started, you need a *bootstrap* cluster. By default, DKP creates a bootstrap cluster for you in a Docker container using the Kubernetes-in-Docker (KIND[873]) tool.

7.16.3.1 Prerequisites

Before you begin, you must:

- Complete the steps in Prerequisites (see page 1199).
- Ensure the `dkp` binary can be found in your `$PATH`.

7.16.3.2 Bootstrap Cluster Lifecycle Services

1. If an HTTP proxy is required for the bootstrap cluster, set the local `http_proxy`, `https_proxy`, and `no_proxy` environment variables. They are copied into the bootstrap cluster.
2. Create a bootstrap cluster:

   ```
   dkp create bootstrap --with-gcp-bootstrap-credentials=true
   ```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).
3. The output looks similar to:

   ```
   ✓ Creating a bootstrap cluster
   ✓ Initializing new CAPI components
   ```

4. DKP creates a bootstrap cluster using KIND[874] as a library. DKP then deploys the following Cluster API[875] providers on the cluster:
   - Core Provider[876]
   - GCP Infrastructure Provider[877]
   - Kubeadm Bootstrap Provider[878]

872 https://cluster-api.sigs.k8s.io/
873 https://github.com/kubernetes-sigs/kind
874 https://github.com/kubernetes-sigs/kind
875 https://cluster-api.sigs.k8s.io/
876 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/
877 https://github.com/kubernetes-sigs/cluster-api-provider-gcp
878 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/bootstrap/kubeadm

- Kubeadm ControlPlane Provider[879]

5. DKP waits until the controller-manager and webhook deployments of these providers are ready. List these deployments using this command:

```
kubectl get --all-namespaces deployments -l=clusterctl.cluster.x-k8s.io
```

You will then receive the following output:

```
NAMESPACE                           NAME
READY   UP-TO-DATE   AVAILABLE   AGE
capa-system                         capa-controller-manager
1/1     1            1           1h
capg-system                         capg-controller-manager
1/1     1            1           1h
capi-kubeadm-bootstrap-system       capi-kubeadm-bootstrap-controller-manager
1/1     1            1           1h
capi-kubeadm-control-plane-system   capi-kubeadm-control-plane-controller-
manager   1/1     1            1           1h
capi-system                         capi-controller-manager
1/1     1            1           1h
cappp-system                        cappp-controller-manager
1/1     1            1           1h
capv-system                         capv-controller-manager
1/1     1            1           1h
capz-system                         capz-controller-manager
1/1     1            1           1h
cert-manager                        cert-manager
1/1     1            1           1h
cert-manager                        cert-manager-cainjector
1/1     1            1           1h
cert-manager                        cert-manager-webhook
1/1     1            1           1h
```

Once completed, move onto creating a new GCP cluster (see page 1206).

7.16.4 Create a New GCP Cluster

7.16.4.1 Name your Cluster

1. Give your cluster a unique name suitable for your environment.

   In GCP it is critical that the name is unique, as no two clusters in the same GCP account can have the same name.

2. Set the environment variable:

879 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/controlplane/kubeadm

```
export CLUSTER_NAME=<gcp-example>
```

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[880] for more naming information.

To increase Docker Hub's rate limit[881] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username=<username> --registry-mirror-password=<password>` on the `dkp create cluster` command.

7.16.4.2 Create a New GCP Cluster

Availability zones (AZs) are isolated locations within data center regions from which public cloud services originate and operate. Because all the nodes in a node pool are deployed in a single Availability Zone, you may wish to create additional node pools is to ensure your cluster has nodes deployed in multiple Availability Zones.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single zone. You may create additional node pools in other zones with the `dkp create nodepool` command. The default region for the availability zones is `us-west1`.

Google Cloud Platform does not publish images. You must first build the image using Konvoy Image Builder (see page 1282).

1. Create an image using Konvoy Image Builder (KIB) (see page 1282) and then export the image name:

```
export IMAGE_NAME=projects/${GCP_PROJECT}/global/images/<image_name_from_kib>
```

880 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/
881 https://docs.docker.com/docker-hub/download-rate-limit/

2. (Optional) You can modify Control Plane Audit logs settings using the information contained in the page Configuring the Control Plane (see page 1351).
3. (Optional) Determine what VPC Network to use. All GCP accounts come with a preconfigured VPC Network named `default`, which will be used if you do not specify a different network.
   To use a different VPC network for your cluster, create one by following these instructions for Create and Manage VPC Networks[882]. Then specify the `--network <new_vpc_network_name>` option on the create cluster command below. Follow the link for more information on GCP Cloud Nat[883] and network flag.

- Ensure your subnets[884] do not overlap with your host subnet because they cannot be changed after cluster creation. If you need to change the kubernetes subnets, you must do this at cluster creation. The default subnets used in DKP are:

```
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 192.168.0.0/16
    services:
      cidrBlocks:
      - 10.96.0.0/12
```

4. Create a Kubernetes cluster. The following example shows a common configuration. See dkp create cluster gcp (see page 1469) reference for the full list of cluster creation options:

```
dkp create cluster gcp \
--cluster-name=${CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--with-gcp-bootstrap-credentials=true
--project=${GCP_PROJECT} \
--image=${IMAGE_NAME}
--self-managed
```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

882 https://cloud.google.com/vpc/docs/create-modify-vpc-networks#create-auto-network
883 https://github.com/kubernetes-sigs/cluster-api-provider-gcp/blob/3406adaae0f8f65a615844e55d856d306eddacc1/docs/book/src/topics/prerequisites.md#cloud-nat
884 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/296452108

5. Wait for the cluster control-plane to be ready:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=20m
```

6. After the objects are created on the API server, the Cluster API controllers reconcile them. They create infrastructure and machines. As they progress, they update the Status of each object. Konvoy provides a command to describe the current status of the cluster:

```
dkp describe cluster -c ${CLUSTER_NAME}
```

```
NAME                                                                READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/gcp-example                                                 True
52s
├──ClusterInfrastructure - GCPCluster/gcp-example
├──ControlPlane - KubeadmControlPlane/gcp-example-control-plane
True                     52s
│  ├──Machine/gcp-example-control-plane-6fbzn
True                     2m32s
│  │  └──MachineInfrastructure - GCPMachine/gcp-example-control-plane-62g6s
│  ├──Machine/gcp-example-control-plane-jf6s2
True                     7m36s
│  │  └──MachineInfrastructure - GCPMachine/gcp-example-control-plane-bsr2z
│  └──Machine/gcp-example-control-plane-mnbfs
True                     54s
│     └──MachineInfrastructure - GCPMachine/gcp-example-control-plane-s8xsx
└──Workers
   └──MachineDeployment/gcp-example-md-0
True                     78s
     ├──Machine/gcp-example-md-0-68b86fddb8-8glsw
True                     2m49s
     │  └──MachineInfrastructure - GCPMachine/gcp-example-md-0-zls8d
     ├──Machine/gcp-example-md-0-68b86fddb8-bvbm7
True                     2m48s
     │  └──MachineInfrastructure - GCPMachine/gcp-example-md-0-5zcvc
     ├──Machine/gcp-example-md-0-68b86fddb8-k9499
True                     2m49s
     │  └──MachineInfrastructure - GCPMachine/gcp-example-md-0-k8h5p
     └──Machine/gcp-example-md-0-68b86fddb8-l6vfb
True                     2m49s
       └──MachineInfrastructure - GCPMachine/gcp-example-md-0-9h5vn
```

Once the cluster creation process has finished, move onto exploring your new cluster. (see page 1210)

7.16.5 Explore the GCP Cluster

This guide explains how to use the command line to interact with your newly deployed Kubernetes cluster.

Before you start, make sure you have created a workload cluster, as described in Create a New GCP Cluster (see page 1206).

7.16.5.1 Explore the new Kubernetes cluster

1. Get a kubeconfig file for the workload cluster:

   When the workload cluster is created, the cluster lifecycle services generate a kubeconfig file for the workload cluster, and write it to a *Secret*. The kubeconfig file is scoped to the cluster administrator.

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

2. Verify the API server is up :

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf get nodes
   ```

   NOTE: It may take a few minutes for the Status to move to Ready while the Pod network is deployed. The Nodes' Status should change to Ready soon after the calico-node DaemonSet Pods are Ready. You should receive the following output:

   ```
   NAME                              STATUS   ROLES                  AGE
   VERSION
   gcp-example-control-plane-9z77w   Ready    control-plane,master   4m44s
   v1.25.4
   gcp-example-control-plane-rtj9h   Ready    control-plane,master   104s
   v1.25.4
   gcp-example-control-plane-zbf9w   Ready    control-plane,master   3m23s
   v1.25.4
   gcp-example-md-0-88c46            Ready    <none>                 3m28s   v1.25
   .4
   gcp-example-md-0-fp8s7            Ready    <none>                 3m28s   v1.25
   .4
   gcp-example-md-0-qvnx7            Ready    <none>                 3m28s   v1.25
   .4
   gcp-example-md-0-wjdrg            Ready    <none>                 3m27s   v1.25
   .4
   ```

3. List the Pods with the command:

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf get pods -A
   ```

4. View the following output:

```
NAMESPACE                           NAME
READY   STATUS    RESTARTS       AGE
calico-system                       calico-kube-controllers-577c696df9-v2nzv
1/1     Running   0              5m23s
calico-system                       calico-node-4x5rk
1/1     Running   0              4m22s
calico-system                       calico-node-cxsgc
1/1     Running   0              4m23s
calico-system                       calico-node-dvlnm
1/1     Running   0              4m23s
calico-system                       calico-node-h6nlt
1/1     Running   0              4m23s
calico-system                       calico-node-jmkwq
1/1     Running   0              5m23s
calico-system                       calico-node-tnf54
1/1     Running   0              4m18s
calico-system                       calico-node-v6bwq
1/1     Running   0              2m39s
calico-system                       calico-typha-6d8c94bfdf-dkfvq
1/1     Running   0              5m23s
calico-system                       calico-typha-6d8c94bfdf-fdfn2
1/1     Running   0              3m43s
calico-system                       calico-typha-6d8c94bfdf-kjgzj
1/1     Running   0              3m43s
capa-system                         capa-controller-manager-6468bc488-w7nj9
1/1     Running   0              67s
capg-system                         capg-controller-manager-5fb47f869b-6jgms
1/1     Running   0              53s
capi-kubeadm-bootstrap-system       capi-kubeadm-bootstrap-controller-
manager-65ffc94457-7cjdn      1/1     Running   0              74s
capi-kubeadm-control-plane-system   capi-kubeadm-control-plane-controller-
manager-bc7b688d4-vv8wg   1/1     Running   0              72s
capi-system                         capi-controller-manager-dbfc7b49-dzvw8
1/1     Running   0              77s
cappp-system                        cappp-controller-manager-8444d67568-rmms2
1/1     Running   0              59s
capv-system                         capv-controller-manager-58b8ccf868-rbscn
1/1     Running   0              56s
capz-system                         capz-controller-manager-6467f986d8-dnvj4
1/1     Running   0              62s
cert-manager                        cert-manager-6888d6b69b-7b7m9
1/1     Running   0              91s
cert-manager                        cert-manager-cainjector-76f7798c9-gnp8f
1/1     Running   0              91s
cert-manager                        cert-manager-webhook-7d4b5d8484-gn5dr
1/1     Running   0              91s
gce-pd-csi-driver                   csi-gce-pd-controller-5bd587fbfb-lrx29
5/5     Running   0              5m40s
```

```
gce-pd-csi-driver                  csi-gce-pd-node-4cgd8
2/2     Running   0             4m22s
gce-pd-csi-driver                  csi-gce-pd-node-5qsfk
2/2     Running   0             4m23s
gce-pd-csi-driver                  csi-gce-pd-node-5w4bq
2/2     Running   0             4m18s
gce-pd-csi-driver                  csi-gce-pd-node-fbdbw
2/2     Running   0             4m23s
gce-pd-csi-driver                  csi-gce-pd-node-h82lx
2/2     Running   0             4m23s
gce-pd-csi-driver                  csi-gce-pd-node-jzq58
2/2     Running   0             5m39s
gce-pd-csi-driver                  csi-gce-pd-node-k6bz9
2/2     Running   0             2m39s
kube-system                        cluster-autoscaler-7f695dc48f-v5kvh
1/1     Running   0             5m40s
kube-system                        coredns-64897985d-hbkqd
1/1     Running   0             5m38s
kube-system                        coredns-64897985d-m8g5j
1/1     Running   0             5m38s
kube-system                        etcd-gcp-example-control-plane-9z77w
1/1     Running   0             5m32s
kube-system                        etcd-gcp-example-control-plane-rtj9h
1/1     Running   0             2m37s
kube-system                        etcd-gcp-example-control-plane-zbf9w
1/1     Running   0             4m17s
kube-system                        kube-apiserver-gcp-example-control-
plane-9z77w                 1/1     Running   0              5m32s
kube-system                        kube-apiserver-gcp-example-control-plane-
rtj9h                   1/1     Running   0              2m38s
kube-system                        kube-apiserver-gcp-example-control-plane-
zbf9w                   1/1     Running   0              4m17s
kube-system                        kube-controller-manager-gcp-example-
control-plane-9z77w         1/1     Running   0              5m33s
kube-system                        kube-controller-manager-gcp-example-
control-plane-rtj9h         1/1     Running   0              2m37s
kube-system                        kube-controller-manager-gcp-example-
control-plane-zbf9w         1/1     Running   0              4m17s
kube-system                        kube-proxy-bskz2
1/1     Running   0             4m18s
kube-system                        kube-proxy-gdkn5
1/1     Running   0             4m23s
kube-system                        kube-proxy-knvb9
1/1     Running   0             4m22s
kube-system                        kube-proxy-tcj7r
1/1     Running   0             4m23s
kube-system                        kube-proxy-thdpl
1/1     Running   0             5m38s
kube-system                        kube-proxy-txxmb
1/1     Running   0             4m23s
kube-system                        kube-proxy-vq6kv
1/1     Running   0             2m39s
```

```
kube-system                         kube-scheduler-gcp-example-control-
plane-9z77w                 1/1     Running   0             5m33s
kube-system                         kube-scheduler-gcp-example-control-plane-
rtj9h                   1/1     Running   0             2m37s
kube-system                         kube-scheduler-gcp-example-control-plane-
zbf9w                   1/1     Running   0             4m17s
node-feature-discovery              node-feature-discovery-master-7d5985467-
lh7dc                    1/1     Running   0             5m40s
node-feature-discovery              node-feature-discovery-worker-5qtvg
1/1     Running   0             3m40s
node-feature-discovery              node-feature-discovery-worker-66rwx
1/1     Running   0             3m40s
node-feature-discovery              node-feature-discovery-worker-7h92d
1/1     Running   0             3m35s
node-feature-discovery              node-feature-discovery-worker-b4666
1/1     Running   0             3m40s
tigera-operator                     tigera-operator-5f9bdc5c59-j9tnr
1/1     Running   0             5m38s
```

When you're done exploring the cluster, move onto making your cluster self-managed. (see page 1213)

7.16.6 Make the New GCP Cluster Self-Managed

DKP deploys all cluster lifecycle services to a bootstrap cluster, which then deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster, which makes the workload cluster self-managed. This section describes how to make a workload cluster self-managed.

Before starting, ensure you create a workload cluster as described in Create a New GCP Cluster (see page 498).

This page contains instructions on how to make your cluster self-managed. This is necessary if there is only one cluster in your environment, or if this cluster should become the Management cluster in a multi-cluster environment.

If you already have a self-managed or Management cluster in your environment, skip this page.

7.16.6.1 Make the new Kubernetes cluster manage itself

Follow these steps:

1. Deploy cluster lifecycle services on the workload cluster:

   ```
   dkp create capi-components --kubeconfig ${CLUSTER_NAME}.conf
   ```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful.

More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).
The output is similar to this:

```
✓ Initializing new CAPI components
```

2. Move the Cluster API objects from the bootstrap to the workload cluster:

   The cluster lifecycle services on the workload cluster are ready, but the workload cluster configuration is on the bootstrap cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the bootstrap to the workload cluster. This process is also called a Pivot[885].

   ```
   dkp move capi-resources --to-kubeconfig ${CLUSTER_NAME}.conf
   ```

   ```
   ✓ Moving cluster resources
   You can now view resources in the moved cluster by using the --kubeconfig flag
   with kubectl. For example: kubectl --kubeconfig=gcp-example.conf get nodes
   ```

   NOTE: To ensure only one set of cluster lifecycle services manages the workload cluster, DKP first pauses reconciliation of the objects on the bootstrap cluster, then creates the objects on the workload cluster. As DKP copies the objects, the cluster lifecycle services on the workload cluster reconcile the objects. The workload cluster becomes self-managed after DKP creates all the objects. If it fails, the `move` command can be safely retried.

3. Wait for the cluster control-plane to be ready:

   ```
   kubectl --kubeconfig ${CLUSTER_NAME}.conf wait --for=condition=ControlPlaneRead
   y "clusters/${CLUSTER_NAME}" --timeout=20m
   ```

   ```
   cluster.cluster.x-k8s.io/gcp-example condition met
   ```

4. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

   NOTE: After moving the cluster lifecycle services to the workload cluster, remember to use DKP with the workload cluster kubeconfig.

885 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

```
dkp describe cluster --kubeconfig ${CLUSTER_NAME}.conf -c ${CLUSTER_NAME}
```

```
NAME                                                                READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/gcp-example                                                 True
14s
├──ClusterInfrastructure - GCPCluster/gcp-example
├──ControlPlane - KubeadmControlPlane/gcp-example-control-plane
True                     14s
│ ├──Machine/gcp-example-control-plane-6fbzn
True                     17s
│ │ └──MachineInfrastructure - GCPMachine/gcp-example-control-plane-62g6s
│ ├──Machine/gcp-example-control-plane-jf6s2
True                     17s
│ │ └──MachineInfrastructure - GCPMachine/gcp-example-control-plane-bsr2z
│ └──Machine/gcp-example-control-plane-mnbfs
True                     17s
│   └──MachineInfrastructure - GCPMachine/gcp-example-control-plane-s8xsx
└──Workers
   └──MachineDeployment/gcp-example-md-0
True                     17s
     ├──Machine/gcp-example-md-0-68b86fddb8-8glsw
True                     17s
     │ └──MachineInfrastructure - GCPMachine/gcp-example-md-0-zls8d
     ├──Machine/gcp-example-md-0-68b86fddb8-bvbm7
True                     17s
     │ └──MachineInfrastructure - GCPMachine/gcp-example-md-0-5zcvc
     ├──Machine/gcp-example-md-0-68b86fddb8-k9499
True                     17s
     │ └──MachineInfrastructure - GCPMachine/gcp-example-md-0-k8h5p
     └──Machine/gcp-example-md-0-68b86fddb8-l6vfb
True                     17s
       └──MachineInfrastructure - GCPMachine/gcp-example-md-0-9h5vn
```

5. Remove the bootstrap cluster, as the workload cluster is now self-managed:

```
dkp delete bootstrap --kubeconfig $HOME/.kube/config
✓ Deleting bootstrap cluster
```

7.16.6.2 Known Limitations

DKP only supports moving all namespaces in the cluster; DKP does not support migration of individual namespaces.

7.16.7 Manage GCP Node Pools

Node pools are part of a cluster and managed as a group, and you can use a node pool to manage a group of machines using the same common properties. When Konvoy creates a new default cluster, there is one node pool for the worker nodes and all nodes in that new node pool have the same configuration. You can create additional node pools for more specialized hardware or configuration. For example, if you want to tune your memory usage on a cluster where you need maximum memory for some machines and minimal memory on other machines, you would create a new node pool with those specific resource needs.

Konvoy implements node pools using Cluster API MachineDeployments[886].

7.16.7.1 Create GCP Node Pools

Creating a node pool is useful when you need to run workloads that require machines with specific resources, such as additional memory, or specialized network or storage hardware.

7.16.7.1.1 Prerequisites

Before you begin, make sure you have created a GCP cluster. (see page 1206)

7.16.7.1.1.1 Prepare the environment

Follow these steps:

1. Set the environment variable to the name you assigned this cluster.

   ```
   export CLUSTER_NAME=gcp-example
   ```

2. If your workload cluster is self-managed, as described in Make the New Cluster Self-Managed (see page 1213), configure `kubectl` to use the kubeconfig for the cluster.

   ```
   export KUBECONFIG=${CLUSTER_NAME}.conf
   ```

3. Define your node pool name.

   ```
   export NODEPOOL_NAME=example
   ```

7.16.7.1.1.2 Create a GCP node pool

Availability zones (AZs) are isolated locations within data center regions from which public cloud services originate and operate. Because all the nodes in a node pool are deployed in a single Availability Zone, you

886 https://cluster-api.sigs.k8s.io/developer/architecture/controllers/machine-deployment.html

may wish to create additional node pools is to ensure your cluster has nodes deployed in multiple Availability Zones.

Create a new AWS node pool with 3 replicas using this command:

Set the `--zone` flag to a zone[887] in the same same region as your cluster.

```
dkp create nodepool gcp ${NODEPOOL_NAME} \
    --cluster-name=${CLUSTER_NAME} \
    --image $IMAGE_NAME \
    --zone us-west1-b \
    --replicas=3
```

```
machinedeployment.cluster.x-k8s.io/example created
⠦ Creating default/example nodepool resources
gcpmachinetemplate.infrastructure.cluster.x-k8s.io/example created
kubeadmconfigtemplate.bootstrap.cluster.x-k8s.io/example created
 ✓ Creating default/example nodepool resources
```

This example uses default values for brevity. Use flags to define custom instance types, and other properties.

Advanced users can use a combination of the `--dry-run` and `--output=yaml` or or `--output-directory=<your-target-directory>/` flags to get a complete set of node pool objects to modify locally or store in version control.

7.16.7.2 List GCP Node Pools

List node pools for a cluster

Use this command to list the node pools of a given cluster. This returns specific properties of each node pool so that you can see the name of the MachineDeployments.

To list all node pools for a managed cluster, run:

```
dkp get nodepool --cluster-name=${CLUSTER_NAME}
```

The expected output is similar to the following example, indicating the desired size of the node pool, the number of replicas ready in the node pool, and the Kubernetes version those nodes are running:

```
NODEPOOL                  DESIRED   READY   KUBERNETES
VERSION
example                   3         3       v1.25.4
gcp-example-2-md-0        4         4       v1.25.4
```

887 https://cloud.google.com/compute/docs/regions-zones

7.16.7.3 Scale GCP Node Pools

While you can run Cluster Autoscaler[888], you can also manually scale your node pools up or down when you need more finite control over your environment. For example, if you require 10 machines to run a process, you can manually set the scaling to run those 10 machines only. However, if also using the Cluster Autoscaler, you must stay within your minimum and maximum bounds.

7.16.7.3.1 Scaling Up Node Pools

To scale up a node pool in a cluster, run:

```
dkp scale nodepool ${NODEPOOL_NAME} --replicas=5 --cluster-name=${CLUSTER_NAME}
```

Your output should be similar to this example, indicating the scaling is in progress:

```
✓ Scaling node pool example to 5 replicas
```

After a few minutes, you can list the node pools to:

```
dkp get nodepool --cluster-name=${CLUSTER_NAME}
```

Your output should be similar to this example, with the number of DESIRED and READY replicas increased to 5:

```
NODEPOOL                        DESIRED             READY
KUBERNETES VERSION
example                         5                   5                   v1.25.4
gcp-example-md-0                4                   4                   v1.25.4
```

7.16.7.3.2 Scaling Down Node Pools

To scale down a node pool, run:

```
dkp scale nodepool ${NODEPOOL_NAME} --replicas=4 --cluster-name=${CLUSTER_NAME}
```

```
✓ Scaling node pool example to 4 replicas
```

After a few minutes, you can list the node pools using this command:

888 https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/cloudprovider/clusterapi

```
dkp get nodepool --cluster-name=${CLUSTER_NAME}
```

Your output should be similar to this example, with the number of DESIRED and READY replicas decreased to 4:

```
NODEPOOL                          DESIRED              READY
KUBERNETES VERSION
example                           4                    4                    v1.25.4
gcp-example-md-0                  4                    4                    v1.25.4
```

In a default cluster, the nodes to delete are selected at random. This behavior is controlled by CAPI's delete policy[889]. However, when using the DKP CLI to scale down a node pool, it is also possible to specify the Kubernetes Nodes you want to delete.

To do this, set the flag `--nodes-to-delete` with a list of nodes as below. This adds an annotation `cluster.x-k8s.io/delete-machine=yes` to the matching Machine object that contains `status.NodeRef` with the node names from `--nodes-to-delete`.

```
dkp scale nodepool ${NODEPOOL_NAME} --replicas=3 --cluster-name=${CLUSTER_NAME}
```

```
✓ Scaling node pool example to 3 replicas
```

7.16.7.3.3 Scaling Node Pools When Using Cluster Autoscaler

If you configured the cluster autoscaler[890] for the `demo-cluster-md-0` node pool, the value of `--replicas` must be within the minimum and maximum bounds.

For example, assuming you have these annotations:

```
kubectl annotate machinedeployment ${NODEPOOL_NAME} cluster.x-k8s.io/cluster-api-
autoscaler-node-group-min-size=2
kubectl annotate machinedeployment ${NODEPOOL_NAME} cluster.x-k8s.io/cluster-api-
autoscaler-node-group-max-size=6
```

Try to scale the node pool to 7 replicas with the command:

```
dkp scale nodepool ${NODEPOOL_NAME} --replicas=7 --cluster-name=${CLUSTER_NAME}
```

Which results in an error similar to:

889 https://github.com/kubernetes-sigs/cluster-api/blob/v0.4.0/api/v1alpha4/machineset_types.go#L85-L105
890 https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/cloudprovider/clusterapi

```
 ✗ Scaling node pool example to 7 replicas
failed to scale nodepool: scaling MachineDeployment is forbidden: desired replicas 7
 is greater than the configured max size annotation cluster.x-k8s.io/cluster-api-
autoscaler-node-group-max-size: 6
```

Similarly, scaling down to a number of replicas less than the configured `min-size` also returns an error.

7.16.7.4 Delete GCP Node Pools

Delete node pools in a cluster

Deleting a node pool deletes the Kubernetes nodes and the underlying infrastructure. All nodes will be drained prior to deletion and the pods running on those nodes will be rescheduled.

To delete a node pool from a managed cluster, run:

```
dkp delete nodepool ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME}
```

Here, `example` is the node pool to be deleted.

The expected output will be similar to the following example, indicating the node pool is being deleted:

```
✓ Deleting default/example nodepool resources
```

Deleting an invalid node pool results in output similar to this example:

```
dkp delete nodepool ${CLUSTER_NAME}-md-invalid --cluster-name=${CLUSTER_NAME}
```

```
MachineDeployments or MachinePools.infrastructure.cluster.x-k8s.io "no
MachineDeployments or MachinePools found for cluster gcp-example" not found
```

7.16.7.5 GCP Cluster Autoscaler

Configure autoscaler for node pools

Cluster Autoscaler[891] provides the ability to automatically scale-up or scale-down the number of worker nodes in a cluster, based on the number of pending pods to be scheduled. Running the Cluster Autoscaler is optional.

Unlike Horizontal-Pod Autoscaler[892], Cluster Autoscaler does not depend on any Metrics server and does not need Prometheus or any other metrics source.

891 https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/cloudprovider/clusterapi
892 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-fast-is-hpa-when-combined-with-ca

The Cluster Autoscaler looks at the following annotations on a MachineDeployment to determine its scale-up and scale-down ranges:

```
cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size
cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size
```

The full list of command line arguments to the Cluster Autoscaler controller is on the Kubernetes public GitHub repository[893].

For more information about how Cluster Autoscaler works, see these documents:

- What is Cluster Autoscaler[894]
- How does scale-up work[895]
- How does scale-down work[896]
- CAPI Provider for Cluster Autoscaler[897]

7.16.7.5.1 Cluster Autoscaler Prerequisites

Before you begin, you must have:

- A Bootstrap Cluster Lifecycle (see page 1205).
- A New Kubernetes Cluster (see page 1206).
- A Self-Managed Cluster (see page 1213).

7.16.7.5.2 Run Cluster Autoscaler

The Cluster Autoscaler controller runs on the workload cluster. Upon creation of the workload cluster, this controller does not have all the objects required to function correctly until after a `dkp move` is issued from the bootstrap cluster.

Run the following steps to enable Cluster Autoscaler:

1. Ensure the Cluster Autoscaler controller is up and running (no restarts and no errors in the logs)

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf logs deployments/cluster-autoscaler
   cluster-autoscaler -n kube-system -f
   ```

2. Enable Cluster Autoscaler by setting the min & max ranges

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment $
   {NODEPOOL_NAME} cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size=2
   ```

893 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#what-are-the-parameters-to-ca
894 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#what-is-cluster-autoscaler
895 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-does-scale-up-work
896 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-does-scale-down-work
897 https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/cloudprovider/clusterapi

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment $
{NODEPOOL_NAME} cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size=6
```

3. The Cluster Autoscaler logs will show that the worker nodes are associated with node-groups and that pending pods are being watched.
4. To demonstrate that it is working properly, create a large deployment which will trigger pending pods (For this example we used GCP n2-standard-8 worker nodes. If you have larger worker-nodes, you should scale up the number of replicas accordingly).

```
cat <<EOF | kubectl --kubeconfig=${CLUSTER_NAME}.conf apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: busybox-deployment
  labels:
    app: busybox
spec:
  replicas: 600
  selector:
    matchLabels:
      app: busybox
  template:
    metadata:
      labels:
        app: busybox
    spec:
      containers:
      - name: busybox
        image: busybox:latest
        command:
          - sleep
          - "3600"
        imagePullPolicy: IfNotPresent
      restartPolicy: Always
EOF
```

5. Cluster Autoscaler will scale up the number of Worker Nodes until there are no pending pods.
6. Scale down the number of replicas for `busybox-deployment`.

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf scale --replicas=30 deployment/
busybox-deployment
```

7. Cluster Autoscaler starts to scale down the number of Worker Nodes after the default timeout of 10 minutes.

7.16.8 Delete a GCP Cluster

7.16.8.1 Prepare to Delete a Workload Cluster

> A self-managed workload cluster cannot delete itself. If your workload cluster is self-managed, you must create a bootstrap cluster and move the cluster lifecycle services to the bootstrap cluster before deleting the workload cluster.

If you did not make your workload cluster self-managed, as described in Make the New GCP Cluster Self-Managed (see page 1213), proceed to the Delete the workload cluster (see page 0) section below.

1. Create a bootstrap cluster:

   The bootstrap cluster will host the Cluster API controllers that reconcile the cluster objects marked for deletion:

   NOTE: To avoid using the wrong `kubeconfig`, the following steps use explicit `kubeconfig` paths and contexts.

   ```
   dkp create bootstrap --with-gcp-bootstrap-credentials=true --kubeconfig
   $HOME/.kube/config
   ```

   ```
   ✓ Creating a bootstrap cluster
   ✓ Initializing new CAPI components
   ```

2. Move the Cluster API objects from the workload to the bootstrap cluster: The cluster lifecycle services on the bootstrap cluster are ready, but the workload cluster configuration is on the workload cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the workload to the bootstrap cluster. This process is also called a Pivot[898].

   ```
   dkp move capi-resources \
       --from-kubeconfig ${CLUSTER_NAME}.conf \
       --to-kubeconfig $HOME/.kube/config
   ```

3. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

898 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

```
dkp describe cluster --kubeconfig $HOME/.kube/config -c ${CLUSTER_NAME}
```

```
NAME                                                                  READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/gcp-example                                                   True
34s
├──ClusterInfrastructure - GCPCluster/gcp-example
├──ControlPlane - KubeadmControlPlane/gcp-example-control-plane
True                     34s
│ ├──Machine/gcp-example-control-plane-6fbzn
True                     37s
│ │ └──MachineInfrastructure - GCPMachine/gcp-example-control-plane-62g6s
│ ├──Machine/gcp-example-control-plane-jf6s2
True                     37s
│ │ └──MachineInfrastructure - GCPMachine/gcp-example-control-plane-bsr2z
│ └──Machine/gcp-example-control-plane-mnbfs
True                     37s
│   └──MachineInfrastructure - GCPMachine/gcp-example-control-plane-s8xsx
└──Workers
   └──MachineDeployment/gcp-example-md-0
True                     37s
     ├──Machine/gcp-example-md-0-68b86fddb8-8glsw
True                     37s
     │ └──MachineInfrastructure - GCPMachine/gcp-example-md-0-zls8d
     ├──Machine/gcp-example-md-0-68b86fddb8-bvbm7
True                     37s
     │ └──MachineInfrastructure - GCPMachine/gcp-example-md-0-5zcvc
     ├──Machine/gcp-example-md-0-68b86fddb8-k9499
True                     37s
     │ └──MachineInfrastructure - GCPMachine/gcp-example-md-0-k8h5p
     └──Machine/gcp-example-md-0-68b86fddb8-l6vfb
True                     37s
       └──MachineInfrastructure - GCPMachine/gcp-example-md-0-9h5vn
```

NOTE: After moving the cluster lifecycle services to the workload cluster, remember to use dkp with the workload cluster kubeconfig.

Persistent Volumes (PVs) are not deleted automatically by design in order to preserve your data. However, they take up storage space if not deleted. You must delete PVs manually. Information for backup of a cluster and PVs is on the page in documentation called Back up your Cluster's Applications and Persistent Volumes (see page 736) .

7.16.8.2 Delete the Workload Cluster

1. To delete a cluster, you would use `dkp delete cluster` and pass in the name of the cluster you are trying to delete with `--cluster-name` flag. You would use `kubectl get clusters` to get those details (`--cluster-name` and `--namespace`) of the Kubernetes cluster to delete it.
   NOTE: Do not use `dkp get clusters` since that gets you Kommander cluster details rather than Konvoy kubernetes cluster details.

   ```
   kubectl get clusters
   ```

2. Use `dkp` with the bootstrap cluster to delete the workload cluster. Delete the Kubernetes cluster and wait a few minutes:

   Before deleting the cluster, dkp deletes all Services of type LoadBalancer on the cluster. To skip this step, use the flag `--delete-kubernetes-resources=false` .

   ```
   dkp delete cluster --kubeconfig $HOME/.kube/config --cluster-name $
   {CLUSTER_NAME}
   ```

   ```
   ✓ Deleting Services with type LoadBalancer for Cluster default/gcp-example
   ✓ Deleting ClusterResourceSets for Cluster default/gcp-example
   ✓ Deleting cluster resources
   ✓ Waiting for cluster to be fully deleted
   Deleted default/gcp-example cluster
   ```

   After the workload cluster is deleted, delete the bootstrap cluster.

7.16.8.3 Delete the Bootstrap Cluster

1. Delete the bootstrap cluster using `dkp delete` :

```
dkp delete bootstrap --kubeconfig $HOME/.kube/config
```

```
✓ Deleting bootstrap cluster
```

7.16.8.4 Next Step:

Once your cluster is built in the Konvoy component of DKP for your infrastructure/environment, you will install the Kommander (see page 1227) component of DKP to see your dashboard and continue customization.

7.16.8.5 Known Limitations

The DKP version used to create the workload cluster must match the DKP version used to delete the workload cluster.

8 Additional Kommander Configuration

The Kommander component of DKP can be customized for different environments and installation types. See the following sections for more information on custom settings:

8.1 Kommander Additional Install Configurations

You can configure the Kommander component of DKP during the initial installation, and also post-installation using the DKP CLI.

> Review the Management cluster application requirements (see page 113) and Workspace platform application requirements (see page 114) to ensure that your cluster has sufficient resources.

8.1.1 Initialize a *Kommander Installer Configuration File*

To begin configuring Kommander, run the following command to initialize a default configuration file:

- For a non-air-gapped environment, run this command:

```
dkp install kommander --init > kommander.yaml
```

- For an air-gapped environment, run this command:

```
dkp install kommander --init --airgapped > kommander.yaml
```

> If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in these commands for them to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

8.1.2 Configure Applications

After you have a default configuration file, you can then configure each `app` either inline **or** by referencing another YAML file. The configuration values for each `app` correspond to the Helm Chart values for the application.

After the initial deployment of Kommander, you can find the application Helm Charts by checking the `spec.chart.spec.sourceRef` field of the associated `HelmRelease`:

```
kubectl get helmreleases <application> -o yaml -n kommander
```

8.1.2.1 Inline configuration (using values)

In this example, you configure the `centralized-grafana` application with resource limits by defining the Helm Chart values in the Kommander configuration file.

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  centralized-grafana:
    values: |
      grafana:
        resources:
          limits:
            cpu: 150m
            memory: 100Mi
          requests:
            cpu: 100m
            memory: 50Mi
...
```

8.1.2.2 Reference another YAML file (using valuesFrom)

Alternatively, you could create another YAML file containing the configuration for `centralized-grafana` and reference that using `valuesFrom`. Point to this file by using either a relative path (from the configuration file location) or by using an absolute path.

```
cat > centralized-grafana.yaml <<EOF
grafana:
  resources:
    limits:
      cpu: 150m
      memory: 100Mi
    requests:
      cpu: 100m
      memory: 50Mi
EOF
```

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  centralized-grafana:
    valuesFrom: centralized-grafana.yaml
...
```

8.1.3 Minimal Kommander Installation

You can install Kommander with a bare minimum of applications on a small environment with smaller memory, storage, and CPU requirements for testing and demo purposes. Refer to the Install DKP on a Small Environment (see page 1276) documentation for more information.

8.1.4 Install with Configuration File

In the following command, the `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you set the context to **install Kommander on the right cluster**. For alternatives and recommendations around setting your context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

Add the `--installer-config` flag to the `kommander install` command to use a custom configuration file. To reconfigure applications, you can also run this command after the initial installation.

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

TIP: Sometimes, applications require a longer period of time to deploy, which causes the installation to time out. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.

8.1.5 Verify Installation

After the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander (see page 1279) in that section of documentation.

Then you will be able to Log in to the Kommander UI. (see page 1280)

8.1.6 Custom Domains and Certificates

Configure a custom domain during installation

DKP supports configuring a custom domain name and certificate for accessing the UI and other platform services.

This section provides instructions and examples on how to configure the **DKP installation** to add a customized domain and certificate on your **Essential cluster** (see page 98) or your **Management cluster** (see page 97).

If you want to customize the domain and certificate after installing DKP or on an **Attached** (see page 97) or **Managed** (see page 97) cluster, this sections does not apply to your use case. For this, refer to Configure Custom Domains or Custom Certificates (see page 765).

Section contents:

8.1.6.1 Why to set up a Custom Domain or Certificate?

8.1.6.1.1 Reasons for Using a Custom DNS Domain

DKP supports the customization of domains to allow you to use your own domain or hostname for your services. For example, you can set up your DKP UI or any of your clusters to be accessible with your custom domain name instead of the domain provided by default.

- To set up a custom domain (without a custom certificate), refer to Configure a Custom Domain without a Custom Certificate (see page 1240).

8.1.6.1.2 Reasons for Using a Custom Certificate

DKP's default CA identity supports the encryption of data exchange and traffic (between your client and your environment's server). To configure an additional security layer that validates your environment's server authenticity, DKP supports configuring a custom certificate issued by a trusted Certificate Authority either directly in a Secret or managed automatically using the ACME protocol (for example, Let's Encrypt).

Changing the default certificate for any of your clusters can be helpful. For example, you can adapt it to classify your DKP UI or any other type of service as trusted (when accessing a service via a browser).

To set up a custom domain and certificate, refer to the following pages respectively:

- Configure a custom domain and certificate as part of the cluster's installation process (see page 1232). This is only possible for your Management/Essential cluster (see page 97).
- Update your cluster's current domain and certificate configuration as part of your cluster's Day 2 operations (see page 765). You can do this for any cluster type (see page 97) in your environment.

> Using Let's Encrypt or other public ACME certificate authorities **does not work in air-gapped scenarios**, as these services require connection to the Internet for their setup. For air-gapped environments, you can either use self-signed certificates issued by the cluster (the default configuration), or a certificate created manually using a trusted Certificate Authority.

8.1.6.1.3 Next Topic:

Certificate Authority (CA) Specifics (see page 1231)

8.1.6.2 Certificate Authority (CA) Specifics

The following table defines some values you require to set up a custom certificate according to your Certificate Authority (CA).

Certificate Authority	**Prerequisites**	***Kommander Installer* values**
Let's Encrypt	None	Generated automatically by Kommander when `acme` is enabled
ZeroSSL	An access and a secret key provided by ZeroSSL	`acme.server: https://acme.zerossl.com/v2/DV90`

Certificate Authority	Prerequisites	*Kommander Installer* values
SSL.com	An access and a secret key provided by SSL	`acme.server: https://acme.ssl.com/sslcom-dv-rsa`

Use these values to Configure your Custom Domain and Certificate (see page 1232).

8.1.6.2.1 Next Topic:

Configure your Custom Domain and Certificate (see page 1232)

8.1.6.3 Configure the Kommander Installation with a Custom Domain and Certificate

This page contains instructions on how to set up custom certificates for your cluster during the installation of DKP. This allows most browsers to validate the certificate for the cluster when users try to log into the operations portal.

> Refer to Certificate Authority (CA) Specifics (see page 1231) for more information on values that are specific to your Certificate Authority or CA.

There are three main options:

I want to use an automatically-generated certificate with ACME and require basic configuration*

8.1.6.3.1 I want to use an automatically-generated certificate with ACME and require basic configuration*

When you enable ACME, by default DKP generates an ACME-supported certificate with an HTTP01 solver. The `cert-manager` automatically issues a trusted certificate for the configured custom domain, and takes care of renewing the certificate before expiration.

1. Open the *Kommander Installer Configuration File* or `<kommander.yaml>` file:
   a. If you do not have the `<kommander.yaml>` file, initialize the configuration file (see page 1227), so you can edit it in the following steps. **WARNING**: Initialize this file only ONCE, otherwise you will overwrite previous customizations.
   b. If you have initialized the configuration file already, open the `<kommander.yaml>` with the editor of your choice.
2. In that file, configure the custom domain for your cluster:

```
[...]
clusterHostname: <mycluster.example.com>
[...]
```

3. Enable ACME by adding `acme` value, the issuer's server and your e-mail. If you don't provide a server, DKP sets up **Let's Encrypt** as your certificate provider:

```
acme:
  email: <your_email>
  server: <your_server>
[...]
```

4. Use the configuration file to install Kommander (see page 0).

*basic configuration: ACME server without EAB (External Account Bindings) and HTTP solver

I want to use an automatically-generated certificate with ACME and require advanced configuration (e.g. EAB, DNS solver, etc.)

8.1.6.3.2 I want to use an automatically-generated certificate with ACME and require advanced configuration

If you require additional configuration options like DNS solver, EAB, among others, create a `ClusterIssuer` with the required configurations before you run the installation of Kommander. The `cert-manager` automatically issues a trusted certificate for the configured custom domain, and takes care of renewing the certificate before expiration.

> To read more about the `ClusterIssuer`, other objects, and where to store them, refer to Advanced Configuration: ClusterIssuer (see page 1236) and Advanced Configuration: Important Concepts (see page 1239).

1. Create a `ClusterIssuer` and store it in the target cluster. It **must** be called `kommander-acme-issuer`:
   a. If you require an **HTTP** solver, adapt the following example with the properties required for your certificate and execute the command:

```
cat <<EOF | kubectl apply -f -
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: kommander-acme-issuer # This part is important
spec:
```

```
  acme:
    email: <your_email>
    server: <https://acme.server.example>
    skipTLSVerify: true
    privateKeySecretRef:
      name: kommander-acme-issuer-account # Set this to <name>-account
    solvers:
    - http01:
        ingress:
          ingressTemplate:
            metadata:
              annotations:
                kubernetes.io/ingress.class: kommander-traefik
                "traefik.ingress.kubernetes.io/router.priority":
"2147483647"
EOF
```

Note: The values `kommander-acme-issuer`, `kommander-acme-issuer-account` and `"traefik.ingress.kubernetes.io/router.priority": "2147483647"` are not placeholders and MUST be filled out exactly as in the example.

b. If you require a **DNS** solver, adapt the following example with the properties required for your certificate and execute the command:

```
cat <<EOF | kubectl apply -f -
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: kommander-acme-issuer # This part is important
spec:
  acme:
    email: <your_email>
    server: <https://acme.server.example>
    privateKeySecretRef:
      name: kommander-acme-issuer-account # Set this to <name>-account
    solvers:
      - dns01:
          route53:
            region: us-east-1
            role: arn:aws:iam::YYYYYYYYYYYY:role/dns-manager
EOF
```

Note: The values `kommander-acme-issuer`, `kommander-acme-issuer-account` and `"traefik.ingress.kubernetes.io/router.priority": "2147483647"` are not placeholders and MUST be filled out exactly as in the example.

2. Optional: If you require External Account Bindings to link your ACME account to an external database, refer to https://cert-manager.io/docs/configuration/acme/#external-account-bindings.
3. Optional: Create a DNS record, by setting up an external-dns service (see page 855). This way, the `external-dns` will take care of pointing the DNS record to the ingress of the cluster automatically.

Note: You can also create a DNS record manually, that maps your domain name or IP address to the cluster ingress. In this case, finish installing Kommander and then manually create the DNS record pointing to the load balancer address.

4. Open the **Kommander Installer Configuration File** or `kommander.yaml` file:
   a. If you do not have the `kommander.yaml` file, initialize the configuration file (see page 1227), so you can edit it in the following steps. **WARNING**: Initialize this file only ONCE, otherwise you will overwrite previous customizations.
   b. If you have initialized the configuration file already, open the `kommander.yaml` with the editor of your choice.
5. In that file, configure the cluster to use your custom domain:

```
[...]
clusterHostname: <mycluster.example.com>
[...]
```

6. Enable ACME by configuring the issuer's server and your e-mail:

```
[...]
acme:
  email: <your_email>
  server: <your_server>
[...]
```

7. Use the configuration file to install Kommander (see page 0).

I have a manually-generated certificate

8.1.6.3.3 I have a manually-generated certificate

D2iQ supports the use of a manually-created certificate. In this case, there is no certificate controller that handles the renewal and update of your certificate automatically, so you will have to take care of these tasks manually.

8.1.6.3.3.1 Prerequisites:

- Obtain the PEM files of your certificate and store them in the target cluster's namespace:
  - Certificate
  - certificate's private key
  - CA bundle (containing the root and intermediate certificates)

8.1.6.3.3.2 Configure the manually-generated certificate

1. Open the **Kommander Installer Configuration File** or `<kommander.yaml>` file:

a. If you do not have the `<kommander.yaml>` file, initialize the configuration file (see page 1227), so you can edit it in the following steps. **WARNING**: Initialize this file only ONCE, otherwise you will overwrite previous customizations.
b. If you have initialized the configuration file already, open the `<kommander.yaml>` with the editor of your choice.

2. In the *Kommander Installer Configuration file*, provide your custom domain and the paths to the PEM files of your certificate:

```
[...]
clusterHostname: <mycluster.example.com>
ingressCertificate:
  certificate: <certs/cert.pem>
  private_key: <certs/key.pem>
  ca: <certs/ca.pem>
[...]
```

3. Use the configuration file to install Kommander (see page 0).

Certificates issued by another Issuer

You can also configure a certificate issued by another Certificate Authority. In this case, the CA will determine which information to include in the configuration.

- Refer to https://cert-manager.io/docs/configuration/ for configuration examples.
- The `ClusterIssuer`'s name **MUST BE** `kommander-acme-issuer`.

8.1.6.3.4 Next Step:

Verification and Troubleshooting for Custom Certificates (see page 1241)

8.1.6.3.5 Related Topics:

- Advanced Configuration: ClusterIssuer (see page 1236)
- Advanced Configuration: Important Concepts (see page 1239)
- Configure a Custom Domain without a Custom Certificate (see page 1240)

8.1.6.3.6 Advanced Configuration: ClusterIssuer

When you enable ACME (see page 1232), by default DKP generates an ACME-supported certificate with an HTTP01 solver that is provided by Let's Encrypt.

You can also set up an advanced configuration for a Custom Domain and Certificate (see page 1233). In these cases, the custom configuration cannot be done completely via the `installer config` file, but must be specified further in a `ClusterIssuer`.

Whether it is sufficient to establish the configuration of your custom certificate in the `installer config` file only, or you require a `ClusterIssuer` to define further configuration options depends on the degree of customization.

> ⚠ If you require a `ClusterIssuer`, you MUST create it before you run the Kommander installation.

8.1.6.3.6.1 When do You Need a ClusterIssuer?

The configuration of the `ClusterIssuer` resource depends on your DKP landscape:



8.1.6.3.6.2 How do You Configure a ClusterIssuer?

The following image describes the configurable fields of a `ClusterIssuer`:



For more information on the available options, refer to the ACME section in the cert-manager documentation[899].

8.1.6.3.6.3 Examples:

Refer to Configure the Kommander Installation with a Custom Domain and Certificate (see page 1232) for configuration steps and examples.

> ⚠ If you need to make changes in the configuration of your domain or certificate *after* you have installed DKP, or if you want to set up a custom domain and certificate for Attached or Managed clusters (see page 97), modify the `ingress` in the `KommanderCluster` object as shown in the Custom domains and certificates configuration (see page 765) section.

8.1.6.3.6.4 Related topics:

Why to set up a Custom Domain or Certificate? (see page 1230)

Configure the Kommander Installation with a Custom Domain and Certificate (see page 1232)

Advanced Configuration: Important Concepts (see page 1239)

899 https://cert-manager.io/docs/configuration/acme/

8.1.6.3.7 Advanced Configuration: Important Concepts

If you set up an advanced configuration of your custom domain (see page 1233), ensure you understand the following concepts.

8.1.6.3.7.1 IssuerRef, ClusterIssuerRef or certificateSecretRef?

If you use a certificate issued and managed automatically by `cert-manager`, you need an *Issuer* or *Cluster Issuer* that you reference in your `KommanderCluster` resource. The referenced object must contain the information of your certificate provider.

If you want to use a manually-created certificate, you need a *secret* that you reference in your `KommanderCluster` resource.

8.1.6.3.7.2 Management, Managed or Attached cluster? Location of the KommanderCluster and Issuer objects

In the Management (see page 97) or Essential (see page 98) cluster, both the `KommanderCluster` and *issuer* objects are stored on the same cluster. The *issuer* can be referenced as an `Issuer`, `ClusterIssuer` or `certificateSecretRef`.

In Managed (see page 97) and Attached (see page 97) clusters, the `KommanderCluster` object is stored on the Management cluster. The `Issuer`, `ClusterIssuer` or `certificateSecretRef` is stored on the Managed or Attached cluster.

For more information on `ClusterIssuer` objects, refer to Advanced Configuration: ClusterIssuer (see page 1236).

8.1.6.3.7.3 HTTP or DNS solver?

When configuring a certificate for your DKP cluster, you can set up an *HTTP solver* or a *DNS solver*. The HTTP protocol exposes your cluster to the public Internet, whereas DNS keeps your traffic hidden. If you use HTTP, your cluster must be publically accessible (via the ingress or load balancer). If you use DNS, this is not a requirement.

8.1.6.3.7.4 Related topics:

Why to set up a Custom Domain or Certificate? (see page 1230)

Configure your Custom Domain and Certificate (see page 1232)

Advanced Configuration: ClusterIssuer (see page 1236)

8.1.6.3.8 Configure a Custom Domain without a Custom Certificate

To configure Kommander to use a custom domain, the domain name must be provided in an installation config file. If you want to set up a custom domain and certificate, refer to Configure your Custom Domain and Certificate (see page 1232).

1. Open the *Kommander Installer Configuration File* or `<kommander.yaml>` file:
   a. If you do not have the `<kommander.yaml>` file, initialize the configuration file (see page 1227), so you can edit it in the following steps. **WARNING**: Initialize this file only ONCE, otherwise you will overwrite previous customizations.
   b. If you have initialized the configuration file already, open the `<kommander.yaml>` with the editor of your choice.
2. In that file, configure the custom domain for your cluster by adding this line:

   ```
   [...]
   clusterHostname: <mycluster.example.com>
   [...]
   ```

3. This configuration can be used when installing or reconfiguring Kommander by passing it to the `dkp install kommander` command:

   ```
   dkp install kommander --installer-config <kommander.yaml> --kubeconfig=$
   {CLUSTER_NAME}.conf
   ```

   Note: To ensure Kommander is installed on the right cluster, use the `--kubeconfig=cluster_name.conf` flag as an alternative to KUBECONFIG.
4. After the command completes, obtain the cluster ingress IP address or hostname using the following command:

   ```
   kubectl -n kommander get svc kommander-traefik -o go-template='{{with
   index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}{{ "\n"}}'
   ```

If required, create a DNS record (for example, by using external-dns (see page 1240)) for your custom hostname that resolves to the cluster ingress load balancer hostname or IP address. If the previous command returns a hostname, you should create a CNAME DNS entry that resolves to that hostname. If the cluster ingress is an IP address, create a DNS A record.

The domain must be resolvable from the client (your browser) and from the cluster. If you set up an `external-dns` service, it will take care of pointing the DNS record to the ingress of the cluster automatically. If you are manually creating a DNS record, you have to install Kommander first to obtain the load balancer address required for the DNS record. The Configure a Custom Certificate (see page 1232) page contains more details and examples on how and when to set up the DNS record.

8.1.6.3.8.1 Related topics:

Why to set up a Custom Domain or Certificate? (see page 1230)

Configure your Custom Domain and Certificate (see page 1232)

Advanced Configuration: Important Concepts (see page 1239)

Advanced Configuration: ClusterIssuer (see page 1236)

8.1.6.4 Verification and Troubleshooting for Custom Certificates

If you want to ensure the customization for a domain and certificate is completed, or if you want to obtain more information on the status of the customization, display the status information for the `KommanderCluster`.

1. Inspect the modified `KommanderCluster` object:

   ```
   kubectl describe kommandercluster -n <workspace_name> <cluster_name>
   ```

2. If the ingress is still being provisioned, the output looks similar to this:

   ```
   [...]
   Conditions:
     Last Transition Time:  2022-06-24T07:48:31Z
     Message:               Ingress service object was not found in the cluster
     Reason:                IngressServiceNotFound
     Status:                False
     Type:                  IngressAddressReady
   [...]
   ```

   If the provisioning has been completed, the output looks similar to this:

   ```
   [...]
   Conditions:
     Last Transition Time:  2022-06-28T13:43:33Z
   ```

```
  Message:               Ingress service address has been provisioned
  Reason:                IngressServiceAddressFound
  Status:                True
  Type:                  IngressAddressReady
  Last Transition Time:  2022-06-28T13:42:24Z
  Message:               Certificate is up to date and has not expired
  Reason:                Ready
  Status:                True
  Type:                  IngressCertificateReady
[...]
```

The same command also prints the actual customized values for the `KommanderCluster.Status.Ingress`. Here is an example:

```
[...]
   ingress:
    address: 172.20.255.180
    caBundle: LS0tLS1CRUdJTiBD...<output has been shortened>...DQVRFLS0tLS0K
[...]
```

8.1.6.4.1 Related topics:

Why to set up a Custom Domain or Certificate? (see page 1230)

Configure your Custom Domain and Certificate (see page 1232)

Advanced Configuration: Important Concepts (see page 1239)

Advanced Configuration: ClusterIssuer (see page 1236)

8.1.7 Install Kommander with an External Load Balancer

8.1.7.1 Load Balancing for External Traffic in DKP

DKP includes a load balancing solution for the supported cloud infrastructure providers (see page 62) and for pre-provisioned environments. For more information, see Load Balancing for external traffic (see page 854) in DKP.

If you want to use a **non-DKP load balancer** (for example, as an alternative to MetalLB in pre-provisioned environments), DKP supports setting up an **external load balancer**.

When enabled, the external load balancer routes incoming traffic requests to a single point of entry in your cluster. Users and services can then access the **DKP UI** through an established IP or DNS address.

In DKP environments, the external load balancer must be configured without TLS termination.

8.1.7.2 Configure Kommander to use an External Load Balancer

To configure an external load balancer, configure a custom hostname (static IP or dynamic DNS address) and specify the target `nodePorts` for your cluster.

1. Open the **Kommander Installer Configuration File** or `kommander.yaml` file:
   a. If you do not have the `kommander.yaml` file, initialize the configuration file (see page 1227), so you can edit it in the following steps. **WARNING**: Initialize this file only ONCE, otherwise you will overwrite previous customizations.
   b. If you have initialized the configuration file already, open the `kommander.yaml` with the editor of your choice.
2. In that file, add the following line for the IP address or DNS name:
   ⚠ ACME does not support the automatic creation of a certificate if you select an IP address for your `clusterHostname`.

   ```
   [...]
   clusterHostname: <mycluster.example.com OR IP_address>
   [...]
   ```

3. Optional: If you require a custom certificate for your `clusterHostname`, see Configure the Kommander Installation with a Custom Domain and Certificate (see page 1232).
4. In the same **Kommander Installer Configuration File**, configure Kommander to use the `NodePort` service by adding a custom configuration under `traefik`:
   ⚠ You can specify the `nodePort` entry points for the load balancer. Ensure the port is within the Kubernetes default (30 000 - 32 768). If not specified, Kommander assigns a port dynamically.

   ```
   traefik:
     enabled: true
     values: |-
       ports:
         web:
           nodePort: 32080 #if not specified, will be assigned dynamically
         websecure:
           nodePort: 32443 #if not specified, will be assigned dynamically
       service:
         type: NodePort
   ```

5. Use the configuration file to install Kommander (see page 1229).

8.1.7.3 Configure the External Load Balancer to Target the Specified Ports

The `traefik` service of the Kommander component now actively listens on the pod IPs, and is accessible through the specified ports on every node.

Configure the load balancer targets to include every worker node address (DNS name or IP address) and node port combination by following this format:

```
<node1>:<nodePort_web> # for example, my.node1.internal:32080
<node2>:<nodePort_web>
<node3>:<nodePort_web>
[...]
<node1>:<nodePort_websecure> # for example, my.node1.internal:32443
<node2>:<nodePort_websecure>
<node3>:<nodePort_websecure>
[...]
```

The exact configuration depends on your load balancer provider.

8.1.8 Configure HTTP Proxy

Configure HTTP proxy for the Kommander cluster(s)

Kommander supports environments where access to the Internet is restricted, and must be made through an HTTP/HTTPS proxy.

In these environments, you must configure Kommander to use the HTTP/HTTPS proxy. In turn, Kommander configures all platform services to use the HTTP/HTTPS proxy.

Kommander follows a common convention for using an HTTP proxy server. The convention is based on three environment variables, and is supported by many, though not all, applications.

- `HTTP_PROXY` : the HTTP proxy server address
- `HTTPS_PROXY` : the HTTPS proxy server address
- `NO_PROXY` : a list of IPs and domain names that are not subject to proxy settings

8.1.8.1 Prerequisites

In the examples below:

1. The `curl` command-line tool is available on the host.
2. The proxy server address is `http://proxy.company.com:3128`.
3. The HTTP and HTTPS proxy server addresses use the `http` scheme.
4. The proxy server can reach `www.google.com` using HTTP or HTTPS.

Verify the cluster nodes can access the Internet through the proxy server. On each cluster node, run:

```
curl --proxy http://proxy.company.com:3128 --head http://www.google.com
curl --proxy http://proxy.company.com:3128 --head https://www.google.com
```

If the proxy is working for HTTP and HTTPS, respectively, the `curl` command returns a `200 OK` HTTP response.

8.1.8.2 Enable Gatekeeper

Gatekeeper acts as a Kubernetes mutating webhook[900]. You can use this to mutate the Pod resources with `HTTP_PROXY`, `HTTPS_PROXY` and `NO_PROXY` environment variables.

1. Create (if necessary) or update the Kommander installation configuration file. If one does not already exist, then create it using the following commands:

   ```
   dkp install kommander --init > kommander.yaml
   ```

2. Append this `apps` section to the `kommander.yaml` file with the following values to enable Gatekeeper and configure it to add HTTP proxy settings to the pods.

   NOTE: Only pods created after applying this setting will be mutated. Also, this will only affect pods in the namespace with the `"gatekeeper.d2iq.com/mutate=pod-proxy"` label.

   ```
   apps:
     gatekeeper:
       values: |
         disableMutation: false
         mutations:
           enablePodProxy: true
           podProxySettings:
             noProxy:
   "127.0.0.1,192.168.0.0/16,10.0.0.0/16,10.96.0.0/12,169.254.169.254,169.254.0.0/
   24,localhost,kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.de
   fault.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.cluster,.svc.c
   luster.local,.svc.cluster.local.,kubecost-prometheus-server.kommander,logging-
   operator-logging-fluentd.kommander.svc.cluster.local,elb.amazonaws.com"
   ```

900 https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/#mutatingadmissionwebhook

```
          httpProxy: "http://proxy.company.com:3128"
          httpsProxy: "http://proxy.company.com:3128"
        excludeNamespacesFromProxy: []
        namespaceSelectorForProxy:
          "gatekeeper.d2iq.com/mutate": "pod-proxy"
```

3. Create the `kommander` and `kommander-flux` namespaces, or the namespace where Kommander will be installed. Label the namespaces to activate the Gatekeeper mutation on them:

```
kubectl create namespace kommander
kubectl label namespace kommander gatekeeper.d2iq.com/mutate=pod-proxy

kubectl create namespace kommander-flux
kubectl label namespace kommander-flux gatekeeper.d2iq.com/mutate=pod-proxy
```

8.1.8.3 Create Gatekeeper ConfigMap in the Kommander Namespace

To configure Gatekeeper so that these environment variables are mutated in the pods, create the following `gatekeeper-overrides` ConfigMap in the `kommander` Workspace you created in a previous step:

```
export NAMESPACE=kommander
```

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: gatekeeper-overrides
  namespace: ${NAMESPACE}
data:
  values.yaml: |
    ---
    # enable mutations
    disableMutation: false
    mutations:
      enablePodProxy: true
      podProxySettings:
        noProxy:
"127.0.0.1,192.168.0.0/16,10.0.0.0/16,10.96.0.0/12,169.254.169.254,169.254.0.0/24,loc
alhost,kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cl
uster,kubernetes.default.svc.cluster.local,.svc,.svc.cluster,.svc.cluster.local,.svc.
cluster.local.,kubecost-prometheus-server.kommander,logging-operator-logging-
fluentd.kommander.svc.cluster.local,elb.amazonaws.com"
        httpProxy: "http://proxy.company.com:3128"
        httpsProxy: "http://proxy.company.com:3128"
      excludeNamespacesFromProxy: []
```

```
      namespaceSelectorForProxy:
        "gatekeeper.d2iq.com/mutate": "pod-proxy"
EOF
```

Set the `httpProxy` and `httpsProxy` environment variables to the address of the HTTP and HTTPS proxy servers, respectively. Set the `noProxy` environment variable to the addresses that should be accessed directly, not through the proxy.

Performing this step before installing Kommander allows the Flux components to respect the proxy configuration in this ConfigMap.

8.1.8.4 HTTP Proxy Configuration Considerations

To ensure that core components work correctly, always add these addresses to the `noProxy`:

- Loopback addresses (`127.0.0.1` and `localhost`)
- Kubernetes API Server addresses
- Kubernetes Pod IPs (for example, `192.168.0.0/16`). This comes from two places:
  - Calico pod CIDR - Defaults to `192.168.0.0/16`
  - The `podSubnet` is configured in CAPI objects and needs to match above Calico's - Defaults to `192.168.0.0/16` (same as above)
- Kubernetes Service addresses (for example, `10.96.0.0/12`, `kubernetes`, `kubernetes.default`, `kubernetes.default.svc`, `kubernetes.default.svc.cluster`, `kubernetes.default.svc.cluster.local`, `.svc`, `.svc.cluster`, `.svc.cluster.local`, `.svc.cluster.local.`)
- Auto-IP addresses `169.254.169.254,169.254.0.0/24`

In addition to the values above, the following settings are needed when installing on AWS:

- The default VPC CIDR range of `10.0.0.0/16`
- `kube-apiserver` internal/external ELB address

- The `NO_PROXY` variable contains the Kubernetes Services CIDR. This example uses the default CIDR, `10.96.0.0/12`. If your cluster's CIDR is different, update the value in the `NO_PROXY` field.
- Based on the order in which the Gatekeeper Deployment is Ready (in relation to other Deployments), not all the core services are guaranteed to be mutated with the proxy environment variables. Only the user deployed workloads are guaranteed to be mutated with the proxy environment variables. If you need a core service to be mutated with your proxy environment variables, you can restart the AppDeployment for that core service.

8.1.8.5 Install Kommander

Kommander installs with the DKP CLI. Install Kommander using the configuration files and ConfigMap from previous steps:

> **NOTE:** To ensure Kommander is installed on the workload cluster, use the `--kubeconfig=cluster_name.conf` flag:

```
dkp install kommander --installer-config kommander.yaml
```

8.1.8.6 Configure Workspace or Project

Configure the Workspace or Project in which you want to use the proxy. To have Gatekeeper mutate the manifests, create the `Workspace` (or `Project`) with the following label:

```
labels:
  gatekeeper.d2iq.com/mutate: "pod-proxy"
```

This can be done when creating the Workspace (or Project) from the UI OR by running the following command from the CLI once the namespace is created:

```
kubectl label namespace <NAMESPACE> "gatekeeper.d2iq.com/mutate=pod-proxy"
```

8.1.8.7 Configure HTTP Proxy in Attached Clusters

To ensure that Gatekeeper is deployed before everything else in the attached clusters that you want to configure with proxy configuration, you must manually create the exact Namespace of the Workspace in which the cluster is going to be attached, *before* attaching the cluster:

Execute the following command in the attached cluster before attaching it to the host cluster:

```
kubectl create namespace <NAMESPACE>
```

Then, to configure the pods in this namespace to use proxy configuration, you must label the Workspace with `gatekeeper.d2iq.com/mutate=pod-proxy` when creating it so that Gatekeeper deploys a `validatingwebhook` to mutate the pods with proxy configuration.

```
kubectl label namespace <NAMESPACE> "gatekeeper.d2iq.com/mutate=pod-proxy"
```

8.1.8.8 Create Gatekeeper ConfigMap in the Workspace Namespace

To configure Gatekeeper so that these environment variables are mutated in the pods, create the following `gatekeeper-overrides` ConfigMap in the Workspace Namespace:

```
export NAMESPACE=<NAMESPACE>
```

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: gatekeeper-overrides
  namespace: ${NAMESPACE}
data:
  values.yaml: |
    ---
    # enable mutations
    disableMutation: false
    mutations:
      enablePodProxy: true
      podProxySettings:
        noProxy:
"127.0.0.1,192.168.0.0/16,10.0.0.0/16,10.96.0.0/12,169.254.169.254,169.254.0.0/24,loc
alhost,kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cl
uster,kubernetes.default.svc.cluster.local,.svc,.svc.cluster,.svc.cluster.local,.svc.
cluster.local.,kubecost-prometheus-server.kommander,logging-operator-logging-
fluentd.kommander.svc.cluster.local,elb.amazonaws.com"
        httpProxy: "http://proxy.company.com:3128"
        httpsProxy: "http://proxy.company.com:3128"
      excludeNamespacesFromProxy: []
      namespaceSelectorForProxy:
        "gatekeeper.d2iq.com/mutate": "pod-proxy"
EOF
```

Set the `httpProxy` and `httpsProxy` environment variables to the address of the HTTP and HTTPS proxy servers, respectively. Set the `noProxy` environment variable to the addresses that should be accessed directly, not through the proxy. The list of the recommended settings is in the section *HTTP Proxy Configuration Considerations* above.

8.1.8.9 Configure Your Applications

In a default installation with `gatekeeper` enabled, you can have proxy environment variables applied to all your pods automatically by adding the following label to your namespace:

```
"gatekeeper.d2iq.com/mutate": "pod-proxy"
```

No further manual changes are required.

8.1.8.10 Manually Configure Your Application

> ⚠ If Gatekeeper is not installed, and you need to use an HTTP proxy, you must manually configure your applications.

Some applications follow the convention of `HTTP_PROXY`, `HTTPS_PROXY`, and `NO_PROXY` environment variables.

In this example, the environment variables are set for a container in a Pod:

See Define Environment Variables for a Container[901] for more details.

8.1.8.10.1 Next Steps:

Now select your environment, and finish your Kommander Installation in one of the following:

Install Kommander in an Air-gapped Environment (see page 1258)

Install Kommander in a Non-air-gapped Environment (see page 1269)

Install Kommander on a Small Environment (see page 1276)

8.1.9 DKP Kommander Configuration Reference

8.1.9.1 Configuration Parameters

This page contains the configuration parameters for the Kommander component of DKP.

For additional information about configuring the Kommander component of DKP during initial installation, see Kommander Install Configuration (see page 1227).

901 https://kubernetes.io/docs/tasks/inject-data-application/define-environment-variable-container/#define-an-environment-variable-for-a-container

Parameter	Description	Default Value

apps
(AppConfig list (see page 1253))
	List of platform applications that will be installed on the management cluster.	<pre>apps:
		  dex:
		    enabled: true
		  dex-k8s-authenticator:
		    enabled: true
		  dkp-insights-management:
		    enabled: true
		  gatekeeper:
		    enabled: true
		  gitea:
		    enabled: true
		  grafana-logging:
		    enabled: true
		  grafana-loki:
		    enabled: true
		  kommander:
		    enabled: true
		  kube-prometheus-stack:
		    enabled: true
		    values: <shortened for brevity>
		  kubefed:
		    enabled: true
		  kubernetes-dashboard:
		    enabled: true
		  kubetunnel:
		    enabled: true
		  logging-operator:
		    enabled: true
		  prometheus-adapter:
		    enabled: true
		  reloader:
		    enabled: true
		   rook-ceph:
		    enabled: true
		   rook-ceph-cluster:
		    enabled: true
		  traefik:
		    enabled: true
		  traefik-forward-auth-mgmt:
		    enabled: true
		  velero:
		    enabled: true</pre>

Parameter	Description	Default Value
ageEncryptionSecretName	Defines the name of the secret to store the Age encryption in.	`sops-age`
clusterHostName	Allows users to provide a hostname that is used for accessing the cluster's ingresses.	
ingressCertificate (see page 1254)	Allows users to provide a custom certificate that's used for TLS in the cluster's ingresses.	
acme	Enable automatic ingress certificate management via ACME.	
appManagementImageTag	Specifies image tag of the AppManagement container.	
appManagementImageRepository	Specifies the image repository of AppManagement container	
appManagementKubetoolsImageRepository	Specifies the image repository of AppManagement Kubetools container	
kommanderChartsVersion	Specifies DKP Kommander Helm chart version.	
airgapped (see page 1254)	Specifies parameters for an airgapped environment.	
catalog Enterprise Gov Advanced	Specifies parameters for installing default catalog repositories. For additional information, see Configure an Enterprise catalog (see page 1256).	

8.1.9.2 AppConfig

Parameter	Description	Default Value
enabled	Denotes whether the specific app should be deployed or not.	false
valuesFrom	File path containing the values that are passed onto the application's `HelmRelease`. This a Helm values files for all applications at the moment. The path in this field must either be a relative file location which is then interpreted to be relative to the location of the configuration file or an absolute path.  Only one of `valuesFrom` or `values` may be set; both cannot be set.	
values	Contains the values that are passed to the the application's `HelmRelease`.  Only one of `valuesFrom` or `values` may be set; both cannot be set.	

8.1.9.3 IngressCertificate

Parameter	Description	Default Value
certificate	The path to a certificate PEM file.	
private_key	The path to the certificate's private key (PEM).	
ca	The path to the certificate's CA bundle; a PEM file containing root and intermediate certificates.	

8.1.9.4 Airgapped

Parameter	Description	Default Value
enabled	Specifies if installation happens in an air-gapped environment.	
helmMirrorImageTag	Specifies an image tag of Helm-mirror container.	
helmMirrorImageRepository	Specifies image repository of Helm-mirror container.	

8.1.9.4.1 Next Step:

Configure HTTP Proxy (see page 1244)

8.1.10 Ensure you have a Default StorageClass

Konvoy handles the creation of a default `StorageClass`. This `StorageClass` is required to install Kommander.

1. Execute the following command to verify one is configured:

```
kubectl get sc
```

The output should look similar to this. Note the `(default)` after the name:

```
NAME              PROVISIONER       RECLAIMPOLICY   VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION   AGE
ebs-sc (default)   ebs.csi.aws.com   Delete          WaitForFirstConsumer
false                  41s
```

2. If the desired `StorageClass` is not set as default, add the following annotation to the `StorageClass` manifest:

```
annotations:
  storageclass.kubernetes.io/is-default-class: "true"
```

More information on setting a `StorageClass` as default can be found at Changing the default storage class[902] in the Kubernetes documentation.

8.2 Helm and Chart Bundle CLI Commands

8.2.1 Kommander Charts Bundle

The charts bundle is a gzipped tar archive containing Helm charts, which are required during Kommander installation. Create the charts bundle with the Kommander CLI or downloaded along with the DKP CLI. Execute this command to create the charts bundle:

```
dkp create chart-bundle
```

DKP creates `charts-bundle.tar.gz`. Optionally, specify the output using the `-o` parameter:

```
dkp create chart-bundle -o [name of the output file]
```

8.2.2 DKP Internal Helm Repository

The DKP charts bundle is pushed to DKP's internal Helm repository. To inspect the contents:

902 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class

```
dkp get charts
```

Individual charts can be removed using:

```
dkp delete chart [chartName] [chartVersion]
```

It is possible to push new charts as well:

```
dkp push chart [chartTarball]
```

Or push a new bundle:

```
dkp push chart-bundle [chartsTarball]
```

Check the built-in help text for each command for more information.

8.3 Configure an Enterprise Catalog

Enterprise Gov Advanced

Configure an Enterprise catalog for DKP

DKP supports configuring default catalogs for clusters with Enterprise license.

8.3.1 Configure a Default Enterprise Catalog

To configure DKP to use a default catalog repository, add these values to your existing and already configured `kommander.yaml` (or to your newly-created `kommander.yaml`, if this is your first installation):

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
```

```
    ref:
      tag: v2.5.0
```

NOTE: If you install Kommander with the previous file, you will install the default applications with default configurations, and default infrastructure settings (this omits configurations for air-gapped environments, configured custom domains, and any other customizations). If you only want to enable catalog applications to an existing configuration, add these values to an existing installer configuration file to maintain your Management cluster's settings.

Use this configuration when installing or reconfiguring DKP by passing it to the `dkp install kommander` command:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

In the previous command, the `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you set the context to **install Kommander on the right cluster**. For alternatives and recommendations around setting your context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

TIP: Sometimes, applications require a longer period of time to deploy, which causes the installation to time out. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.

The following section describes each label:

Label	Description
`kommander.d2iq.io/project-default-catalog-repository`	Indicates this acts as a Catalog Repository in all projects
`kommander.d2iq.io/workspace-default-catalog-repository`	Indicates this acts as a Catalog Repository in all workspaces
`kommander.d2iq.io/gitapps-gitrepository-type`	Indicates this Catalog Repository (and all its Applications) are certified to run on DKP

8.3.2 Configure an Enterprise Catalog after Installation/Upgrade

When configuring the catalog repository post-upgrade, run `dkp install kommander --init > kommander.yaml` and update it accordingly with any custom configuration. This ensures you are using the proper default configuration values for the new DKP version.

8.3.3 Next Step:

Custom Domains and Certificates (see page 1230)

8.4 Install Kommander in an Air-gapped Environment

How to install Kommander using an Air-gapped installation

Depending on your configuration, there are three different ways you can install DKP to an air-gapped environment.
Ensure you follow the correct procedure for your configuration type below:

DKP Essential:

- Install Kommander in an air-gapped environment (see page 1260)
- Install air-gapped Kommander with DKP Insights (see page 1262)

DKP Enterprise:

- Install air-gapped Kommander with DKP Catalog Applications (see page 1264)
- Install air-gapped Kommander with DKP Insights and DKP Catalog Applications (see page 1267)

8.4.1 Prerequisites

Before installing, ensure you have:

- A registry containing all the necessary installation images, including the Kommander images. See below for how to push the necessary images to this registry.
- Download (see page 71) the Complete DKP Air-gapped Bundle for this release (i.e. `dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz`)
- Connectivity with clusters attaching to the management cluster:
  - Both management and attached clusters must be able to connect to the registry.
  - The management cluster must be able to connect to all attached cluster's API servers.
  - The management cluster must be able to connect to any load balancers created for platform services on the management cluster.

- All the prerequisites covered in air-gapped DKP installation[903].
- Sufficient resources on your cluster to run Kommander. Review the Management cluster application requirements (see page 113) and Workspace platform application requirements (see page 114) for application requirements.
- A load balancer to route external traffic which is provided by your cloud provider. For on-premises deployments, you must configure MetalLB. See Load Balancing (see page 854) topic for more details.

8.4.1.1 Load the Images into Your Registry

Follow these steps:

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz`, extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz && cd dkp-v2.5.2
   ```

2. See the `NOTICES.txt` file for 3rd party software attributions in the `container-images/` directory.
3. Set an environment variable with your registry address:

   ```
   export REGISTRY_ADDRESS=<registry-address>:<registry-port>
   ```

4. Run the following command to load the air-gapped `kommander` image bundle into your private registry:

   ```
   dkp push image-bundle --image-bundle ./container-images/kommander-image-bundle-
   v2.5.2.tar --to-registry $REGISTRY_ADDRESS
   ```

5. If you are using DKP catalog applications, run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

   ```
   dkp push image-bundle --image-bundle ./container-images/dkp-catalog-
   applications-image-bundle-v2.5.2.tar --to-registry $REGISTRY_ADDRESS
   ```

6. If you are using DKP Insights, run the following command to load the `dkp-insights` image bundle into your private registry:

903 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29917765/AWS+Air-gapped+Prerequisites#Air-gapped-installation-prerequisites

```
dkp push image-bundle --image-bundle ./container-images/dkp-insights-image-
bundle-v2.5.2.tar --to-registry $REGISTRY_ADDRESS
```

It can take a while to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

8.4.1.2 Next Step

Choose the correct air-gapped environment from the pages in this section.

- Air-gapped Essential (see page 1260)
- Air-gapped Enterprise (see page 1264)

8.4.2 Air-gapped Essential

DKP Essential is a self-managed single cluster Kubernetes solution that gives you a feature-rich, easy-to-deploy, and easy-to-manage entry-level cloud container platform. The DKP Essential license gives the user access to the entire Konvoy cluster environment, and to the Kommander platform application manager.

Depending on your configuration, there are two different ways you can install DKP to an air-gapped environment in DKP Essential:

- Kommander in an Air-gapped Environment (see page 1260)
- Kommander in Air-gapped with DKP Insights (see page 1262)

For more information regarding Essential vs. Enterprise, refer to the Licenses (see page 72) section of the documentation.

8.4.2.1 Kommander in an Air-gapped Environment

Follow these steps to install the Kommander component of DKP Essential in an air-gapped environment with basic setup.

8.4.2.1.1 Installation

1. Create the configuration file (see page 1227) by running `dkp install kommander --init --airgapped > kommander.yaml` for the air-gapped deployment.
2. In the same file, if you are installing Kommander in an AWS VPC, set the Traefik annotation to create an internal facing ELB by setting the following:

```
apps:
  traefik:
    enabled: true
    values: |
      service:
```

```
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true"
```

3. To install Kommander in your air-gapped environment using the above configuration file, enter the following command:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.5.0.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.5.0.tar.gz
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

In the previous command, the `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you set the context to **install Kommander on the right cluster**. For alternatives and recommendations around setting your context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

TIP: Sometimes, applications require a longer period of time to deploy, which causes the installation to time out. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.

8.4.2.1.1.1 Next Steps:

- Verify Installation
- Log in to the Kommander UI

Verify Installation

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander (see page 1279) in that section of documentation.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

Log in to the UI

Then you will be able to log in to Kommander UI. (see page 1280)

8.4.2.2 Kommander in Air-gapped with DKP Insights

Follow these steps to install the Kommander component of DKP Essential in an air-gapped environment with DKP Insights.

8.4.2.2.1 Install Kommander

Follow these steps:

1. Create the configuration file (see page 1227) by running `dkp install kommander --init --airgapped > kommander.yaml` for the air-gapped deployment.
2. In `kommander.yaml`, if you are installing Kommander in an AWS VPC, set the Traefik annotation to create an internal facing ELB by setting the following:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  ...
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true"
...
```

3. In `kommander.yaml`, enable DKP Insights by setting the following:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  ...
  dkp-insights-management:
    enabled: true
```

```
...
catalog:
  repositories:
    - name: insights-catalog-applications
      labels:
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      path: ./application-repositories/dkp-insights-v2.5.0.tar.gz
...
```

Install DKP with Insights enabled by running:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.5.0.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.5.0.tar.gz
\
--charts-bundle ./application-charts/dkp-insights-charts-bundle-v2.5.0.tar.gz
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

In the previous command, the `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you set the context to **install Kommander on the right cluster**. For alternatives and recommendations around setting your context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

TIP: Sometimes, applications require a longer period of time to deploy, which causes the installation to time out. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.

8.4.2.2.2 Next Steps:

- Verify Installation
- Log in to the Kommander UI

8.4.2.2.2.1 Verify Installation

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander (see page 1279) in that section of documentation.

> **NOTE**: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

8.4.2.2.2.2 Log in to the UI

Then you will be able to log in to Kommander UI. (see page 1280)

See Deployment of Catalog Applications in Workspaces (see page 594) to deploy Insights Engine.

8.4.3 Air-gapped Enterprise

Enterprise Gov Advanced

DKP Enterprise is a multi-cluster solution centered around a management cluster that manages multiple attached or managed Kubernetes clusters via a centralized management dashboard.

Depending on your configuration, there are two different ways you can install DKP to an air-gapped environment for DKP Enterprise:

- Install Air-gapped Kommander with DKP Catalog Applications (see page 1264)
- Install Air-gapped Kommander with DKP Insights and DKP Catalog Applications (see page 1267)

For more information regarding Essential vs. Enterprise, refer to the Licenses (see page 72) section of the documentation.

8.4.3.1 Install Air-gapped Kommander with DKP Catalog Applications

Enterprise Gov Advanced

Follow these steps to install the Kommander component of DKP Enterprise in an air-gapped environment with Catalog Applications.

8.4.3.1.1 Install Kommander Air-gapped

Follow these steps:

1. Create the configuration file (see page 1227) by running `dkp install kommander --init --airgapped > kommander.yaml` for the air-gapped deployment.
2. In `kommander.yaml`, if you are installing Kommander in an AWS VPC, set the Traefik annotation to create an internal facing ELB by setting the following:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  ...
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true"
...
```

3. In `kommander.yaml`, enable DKP Catalog Applications by setting the following:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
...
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      path: ./dkp-catalog-applications-v2.5.2.tar.gz
```

4. To customize your Kommander installation (custom domains and certificates, HTTP proxy, Storage Class, etc.), see Kommander Additional Install Configuration (see page 1227) for more details.
5. To install the Kommander component of DKP in your air-gapped environment using the above configuration file, run the following command:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.5.2.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.5.2.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.5.2.tar.gz
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

In the previous command, the `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you set the context to **install Kommander on the right cluster**. For alternatives and recommendations around setting your context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

> **TIP:** Sometimes, applications require a longer period of time to deploy, which causes the installation to time out. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.

8.4.3.1.2 Next Steps:

- Verify Kommander Installation (see page 1279)
- Log in to the UI with Kommander (see page 1280)

8.4.3.1.2.1 Verify Installation

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander (see page 1279) in that section of documentation.

> **NOTE**: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

8.4.3.1.2.2 Log in to the UI

Then you will be able to log in to Kommander UI. (see page 1280)

See Deployment of Catalog Applications in Workspaces (see page 594) to deploy the Catalog Applications.

8.4.3.2 Install Air-gapped Kommander with DKP Insights and DKP Catalog Applications

Enterprise Gov Advanced

Follow these steps to install the Kommander component of DKP Enterprise in an air-gapped environment with DKP Insights and Catalog Applications.

8.4.3.2.1 Install Kommander

Follow these steps:

1. Create the configuration file (see page 1227) by running `dkp install kommander --init --airgapped > kommander.yaml` for the air-gapped deployment.
2. In `kommander.yaml`, if you are installing Kommander in an AWS VPC, set the Traefik annotation to create an internal facing ELB by setting the following:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  ...
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true"
...
```

3. In `kommander.yaml`, enable DKP Insights and DKP Catalog Applications by setting the following:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  ...
  dkp-insights-management:
    enabled: true
...
catalog:
  repositories:
    - name: insights-catalog-applications
      labels:
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      path: ./application-repositories/dkp-insights-v2.5.2.tar.gz
```

```
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      path: ./application-repositories/dkp-catalog-applications-v2.5.2.tar.gz
```

4. Follow the steps on the Configure an Enterprise catalog (see page 1256) page to configure the DKP catalog applications.
5. To install the Kommander component of DKP in your air-gapped environment using the above configuration file, run the following command:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.5.2.tar.gz \
--charts-bundle dkp-kommander-charts-bundle-v2.5.2.tar.gz \
--charts-bundle dkp-catalog-applications-charts-bundle-v2.5.2.tar.gz \
--charts-bundle dkp-insights-charts-bundle-v2.5.2.tar.gz
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

In the previous command, the `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you set the context to **install Kommander on the right cluster**. For alternatives and recommendations around setting your context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

TIP: Sometimes, applications require a longer period of time to deploy, which causes the installation to time out. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.

8.4.3.2.2 Next Steps:

- Verify Installation
- Log in to the Kommander UI

8.4.3.2.2.1 Verify Installation

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander (see page 1279) in that section of documentation.

> **NOTE**: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

8.4.3.2.2.2 Log in to the UI

Then you will be able to log in to Kommander UI. (see page 1280)

See Deployment of Catalog Applications in Workspaces (see page 594) to deploy Insights Engine.

8.4.3.2.2.3 See Also

For information on how to activate an Insights license key in DKP, see (2.5) DKP Insights Activating a License Key[904]

> Installing DKP Insights without a valid license results in higher resource consumption.

8.5 Install Kommander in a Non-air-gapped Environment

To install the Kommander component of DKP in a Non-air-gapped Environment, ensure you meet the following prerequisites:

8.5.1 Prerequisites

Before installing Kommander:

- Ensure you have the version of the CLI that matches the DKP version you want to install.
- You have configured a Konvoy cluster using the Basic Installations (see page 124) for infrastructure provider specific instructions on building a cluster-based on environment.
- Review the Management Cluster Application Requirements (see page 113) and Workspace Platform Application Defaults and Resource Requirements (see page 114) to ensure that your cluster has sufficient resources.

904 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/213322715/%282.5%29+DKP+Insights+Activating+a+License+Key

- Ensure you have a default StorageClass (see page 1254) configured.
- If you want to customize your cluster's domain or certificate, ensure you review the respective documentation sections:
  - Configure custom domains and certificates during the Kommander installation (see page 1230)
  - Configure custom domains and certificates after Kommander has been installed (see page 764)

8.5.2 Install Kommander

To customize your *Kommander Installer Configuration File* (custom domains and certificates, HTTP proxy, Storage Class, etc.), see Kommander Additional Install Configuration (see page 1227) for more details.

In the following command, the `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you set the context to **install Kommander on the right cluster**. For alternatives and recommendations around setting your context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

Install Kommander (without a custom configuration, GPU, or other pre-installation configurations):

```
dkp install kommander --kubeconfig=${CLUSTER_NAME}.conf
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

TIP: Sometimes, applications require a longer period of time to deploy, which causes the installation to time out. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.

8.5.3 Verify Installation

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander (see page 1279) in that section of documentation.

8.5.3.1 Next Step

Access the Kommander UI (see page 1280)

8.6 Install Kommander in a Pre-provisioned Environment

The Kommander installation for pre-provisioned environment consists on the following high-level steps:

- Pre-provisioned Prerequisites for Kommander (see page 1271)
- Non-air-gapped environment: Prepare the Kommander Installer Configuration File (see page 1271)
- Air-gapped Environment: Prepare the Kommander Installer Configuration File (see page 1273)

8.6.1 Pre-provisioned Prerequisites for Kommander

To install the Kommander component of DKP in a pre-provisioned environment, ensure you meet the following prerequisites:

8.6.1.1 Prerequisites

As part of the installation of DKP:

- Ensure you meet the general pre-provisioned prerequisites specified in Prerequisites for Install (see page 119), and have configured cluster using Pre-provisioned Install Options (see page 125) instructions.

After installing **Konvoy** and *before* installing **Kommander**, ensure you have:

- The version of the CLI that matches the DKP version you want to install.
- Sufficient resources on your cluster to run Kommander. Review the Management Cluster Application Requirements (see page 113) and Workspace Platform Application Defaults and Resource Requirements (see page 114) for application requirements.
- Ensure you meet the storage requirements (see page 896).
- A load balancer to route external traffic, which is provided by your cloud provider. For on-premises deployments, you must configure MetalLB. See Load Balancing (see page 854) topic for more details.
- Ensure you have configured a default StorageClass (see page 1254).
- Ensure you have added at least 40 GB of raw storage to each of your worker nodes (see page 897) in your cluster.

8.6.1.2 Next Step:

Prepare the Kommander Installer Configuration File (see page 1271)

8.6.2 Non-air-gapped environment: Prepare the Kommander Installer Configuration File

You **MUST** modify the Kommander installer configuration file (`<kommander.yaml>`) before installing the Kommander component of DKP in a pre-provisioned environment.

Prepare the installer file and install Kommander:

1. Initialize a *Kommander Installer Configuration File* (see page 1227).
2. Edit the installer file to include configuration overrides for the `rook-ceph-cluster`. DKP default configuration ships ceph with PVC based storage[905] which requires your CSI provider to support PVC with type `volumeMode: Block`. As this is not possible with the default local static provisioner (see page 103), you can install ceph in host storage[906] mode.

   You can choose whether Ceph's object storage daemon (osd) pods should consume all or just some of the devices on your nodes. Include **one** of the following Overrides:

   a. To automatically assign all raw storage devices on all nodes to the Ceph cluster:

   ```
   rook-ceph-cluster:
       enabled: true
       values: |
         cephClusterSpec:
           storage:
             storageClassDeviceSets: []
             useAllDevices: true
             useAllNodes: true
             deviceFilter: "<<value>>"
   ```

   b. To assign specific storage devices on all nodes to the Ceph cluster:

   ```
   rook-ceph-cluster:
       enabled: true
       values: |
         cephClusterSpec:
           storage:
             storageClassDeviceSets: []
             useAllNodes: true
             useAllDevices: false
             deviceFilter: "^sdb."
   ```

   Note: If you want to assign specific devices to specific nodes using the `deviceFilter` option, refer to Specific Nodes and Devices[907]. For general information on the `deviceFilter` value, refer to Storage Selection Settings[908].

3. **Optional:** You can add other configuration overrides to your *Kommander installer configuration file*, for example:
   - If you are using GPU nodes, append the values required for the usage of a GPU Toolkit (see page 872).

905 https://rook.io/docs/rook/v1.10/CRDs/Cluster/pvc-cluster/
906 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/
907 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/#specific-nodes-and-devices
908 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

- If you want to customize your cluster's domain or certificate, review Custom Domains and Certificates (see page 1230) during the Kommander installation.
- If you require an Enterprise catalog, review Configure an Enterprise Catalog (see page 1256).
- If you require an HTTPS proxy, review Configure HTTP Proxy (see page 1244).
- Other configurations available at: Kommander Additional Install Configurations (see page 1227)

4. Run the following command by replacing the placeholder `<kommander.yaml>` with the name of your *Kommander installer configuration file*:

```
dkp install kommander --installer-config <kommander.yaml> --kubeconfig=$
{CLUSTER_NAME}.conf
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

In the previous command, the `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you set the context to **install Kommander on the right cluster**. For alternatives and recommendations around setting your context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

TIP: Sometimes, applications require a longer period of time to deploy, which causes the installation to time out. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.

8.6.3 Air-gapped Environment: Prepare the Kommander Installer Configuration File

Follow these steps:

1. Initialize a *Kommander Installer Configuration File* (see page 1227).
2. Edit the installer file to include configuration overrides for the `rook-ceph-cluster`. DKP default configuration ships ceph with PVC based storage[909] which requires your CSI provider to support PVC with type `volumeMode: Block`. As this is not possible with the default local static provisioner, you can install ceph in host storage[910] mode.

909 https://rook.io/docs/rook/v1.10/CRDs/Cluster/pvc-cluster/
910 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/

You can choose whether Ceph's object storage daemon (osd) pods should consume all or just some of the devices on your nodes. Include **one** of the following Overrides:

a. To automatically assign all raw storage devices on all nodes to the Ceph cluster:

```
rook-ceph-cluster:
    enabled: true
    values: |
      cephClusterSpec:
        storage:
          storageClassDeviceSets: []
          useAllDevices: true
          useAllNodes: true
          deviceFilter: "<<value>>"
```

b. To assign specific storage devices on all nodes to the Ceph cluster:

```
rook-ceph-cluster:
    enabled: true
    values: |
      cephClusterSpec:
        storage:
          storageClassDeviceSets: []
          useAllNodes: true
          useAllDevices: false
          deviceFilter: "^sdb."
```

:note: **Note:** If you want to assign specific devices to specific nodes using the `deviceFilter` option, refer to Specific Nodes and Devices[911]. For general information on the `deviceFilter` value, refer to Storage Selection Settings[912].

3. In `kommander.yaml`, if you are installing Kommander in an AWS VPC, set the Traefik annotation to create an internal facing ELB by setting the following:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  ...
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true"
...
```

911 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/#specific-nodes-and-devices
912 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

4. In `kommander.yaml`, enable DKP Catalog Applications by setting the following:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
...
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      path: ./dkp-catalog-applications-v2.5.2.tar.gz
```

5. **Optional:** You can add other configuration overrides to your **Kommander installer configuration file**, for example:
   - If you want to **customize your cluster's domain or certificate**, review Custom Domains and Certificates (see page 1230) during the Kommander installation.
   - If you require an Enterprise catalog, review Configure an Enterprise Catalog (see page 1256).
   - If you require an HTTPS proxy, review Configure HTTP Proxy (see page 1244).
   - Other configurations available at: Kommander Additional Install Configurations (see page 1227)
6. To install DKP in your air-gapped environment using the above configuration file, run the following command:

```
dkp install kommander --installer-config ./kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.5.2.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.5.2.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.5.2.tar.gz
```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

   In the previous command, the `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you set the context to **install Kommander on the right cluster**. For alternatives and recommendations around setting your context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

TIP: Sometimes, applications require a longer period of time to deploy, which causes the installation to time out. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.

8.7 Install Kommander on a Small Environment

You can install the Kommander component of DKP on a small, single-cluster environment with smaller memory, storage, and CPU requirements for testing and demo purposes. This topic describes methods for installing Kommander in these environments.

Enterprise considerations: D2iQ recommends performing testing and demo tasks in a single-cluster environment. The Enterprise license is designed for multi-cluster environments and fleet management, which require a minimum amount of resources (see page 113). Applying an Enterprise license key to the previous installation adds modifications to your environment that can exhaust a small environment's resources.

8.7.1 Prerequisites

Ensure you have done the following:

- You have acquired a DKP license.
- You have installed the Konvoy component (see page 124).
- You have reviewed the prerequisite section pertaining to your air-gapped (see page 1258), or networked (see page 1269) environment.

8.7.2 Minimal Kommander installation

The YAML file that is used to install a minimal configuration of Kommander contains the bare minimum setup that allows you to deploy applications, and access the DKP UI. It does **NOT** include applications for cost monitoring, logging, alerting, object storage, etc.

In this YAML file you can find the lines that correspond to all platform applications which would be included in a normal Kommander setup. Applications that have `enabled` set to `false` are not taken into account during installation. If you want to test an additional application, you can enable it individually to be installed by setting `enabled` to `true` on the corresponding line in the YAML file.

For example, if you want to enable the logging stack, set `enabled` to `true` for `grafana-logging`, `grafana-loki` `logging-operator`, `rook-ceph` and `rook-ceph-cluster`. Note that depending on the size of your cluster, enabling several platform applications could exhaust your cluster's resources.

Some applications depend on other applications to work properly. Refer to the dependencies documentation (see page 568) to find out which other applications you need to enable to test the target application.

1. Initialize your Kommander installation and name it `kommander_minimal.yaml`:

```
dkp install kommander --init --kubeconfig=${CLUSTER_NAME}.conf -oyaml >
kommander_minimal.yaml
```

2. Edit your `kommander_minimal.yaml` file to match the following example:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  dex:
    enabled: true
  dex-k8s-authenticator:
    enabled: true
  dkp-insights-management:
    enabled: false
  gatekeeper:
    enabled: true
  gitea:
    enabled: true
  grafana-logging:
    enabled: false
  grafana-loki:
    enabled: false
  kommander:
    enabled: true
  kube-prometheus-stack:
    enabled: false
  kubernetes-dashboard:
    enabled: false
  kubefed:
    enabled: true
  kubetunnel:
    enabled: false
  logging-operator:
    enabled: false
  prometheus-adapter:
```

```
    enabled: false
  reloader:
    enabled: true
  rook-ceph:
    enabled: false
  rook-ceph-cluster:
    enabled: false
  traefik:
    enabled: true
  traefik-forward-auth-mgmt:
    enabled: true
  velero:
    enabled: false
ageEncryptionSecretName: sops-age
clusterHostname: ""
```

3. Install Kommander on your cluster with the following command:

```
dkp install kommander --installer-config ./kommander_minimal.yaml --
kubeconfig=${CLUSTER_NAME}.conf
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 922).

In the previous command, the `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you set the context to **install Kommander on the right cluster**. For alternatives and recommendations around setting your context, refer to Provide Context for Commands with a kubeconfig File (see page 99).

TIP: Sometimes, applications require a longer period of time to deploy, which causes the installation to time out. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.

8.7.3 Verify Installation

After the Konvoy cluster is built and you install Kommander, you need to verify your installation of Kommander (see page 1279) in that section of documentation. If the Kommander installation fails, or you wish to reconfigure applications, you can rerun the install command, and you can view the progress by increasing the log verbosity by adding the flag `-v 2`.

Then you will be able to log in to Kommander UI. (see page 1280)

8.8 Verify Kommander Installation

Once the Konvoy cluster is built and Kommander has been installed, you will want to verify your installation of Kommander. After the CLI successfully installs the components, it will wait for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Released helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Released` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
```

```
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

8.8.1 Failed HelmReleases

If any application fails to successfully deploy, you can check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

8.8.2 Next Step

Log in to the UI with Kommander (see page 1280).

8.9 Log in to the UI with Kommander

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 524). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 524):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

8.9.1 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 510) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

8.9.2 Next Step

Day 2 - Cluster Operations Management (see page 510)

9 Additional Configurations

When installing DKP for a project, line-of-business, or enterprise, the first step is to determine the infrastructure on which you want to deploy.

The infrastructure you select then determines the specific requirements for a successful installation. Day 1 - Basic Install (see page 124) section gave you a recommended install. For further customization or advanced questions, this Day 1 section of the documentation will provide answers.

If you have decided to uninstall DKP, refer to the same infrastructure documentation you have selected in this Day 1 section for specific steps to take down your cluster.

The different Infrastructures and components for further configuration are listed below:

9.1 Section Contents

9.2 Konvoy Image Builder

Konvoy Image Builder (KIB) is a complete solution for building Cluster API compliant images. The goal of Konvoy Image Builder is to produce a common operating surface to run Konvoy across heterogeneous infrastructure. KIB relies on ansible to install software, configure, and sanitize systems for running Konvoy. Packer is used to build images for cloud environments. Goss is used to validate systems are capable of running Konvoy.

This section describes how to use KIB to create a Cluster API[913] compliant machine images. Machine images contain configuration information[914] and software to create a specific, pre-configured, operating environment. For example, you can create an image of your current computer system settings and software. The machine image can then be replicated and distributed, creating your computer system for other users. KIB uses variable overrides (see page 1330) to specify base image and container images to use in your new machine image. The variable overrides files for Nvidia and FIPS can be ignored unless adding an overlay feature.

913 https://cluster-api.sigs.k8s.io/
914 https://github.com/mesosphere/konvoy-image-builder/tree/main/images

9.2.1 Prerequisites

Before you begin, you must ensure your versions of KIB and DKP are compatible:

- Check the supported DKP version (see page 1543).
- Check the Supported Infrastructure Operating Systems (see page 62).
- Download the Konvoy Image Builder bundle from the KIB Version column of the chart below for your version of DKP prefixed with `konvoy-image-bundle` for your Operating System.
- Create a working `Docker` setup.

9.2.2 Compatible DKP to KIB Versions

Along with the KIB Bundle, we publish a file containing checksums for the bundle files. The recommendation for using these checksums is to verify the integrity of the downloads.

- On the link, download the package prefixed with `konvoy-image-bundle` for your OS.

DKP Version	KIB Version (Click for bundle download)
v2.5.2	v2.2.12[915]
v2.5.1	v2.2.8[916]
v2.5.0	v2.2.6[917]
v2.4.0	v1.24.5[918]
v2.3.3	v1.19.14[919]
v2.3.2	v1.19.14[920]
v2.3.1	v1.19.12[921]
v2.3.0	v1.19.12[922]

915 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v2.2.12
916 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v2.2.8
917 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v2.2.6
918 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v1.24.5
919 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v1.19.14
920 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v1.19.14
921 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v1.19.12
922 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v1.19.12

DKP Version	KIB Version (Click for bundle download)
v2.2.3	v1.17.4[923]
v2.2.2	v1.17.4[924]
v2.2.1	v1.13.2[925]
v2.2.0	v1.11.0[926]
v2.1.5	v1.5.1[927]
v2.1.4	v1.5.1[928]
v2.1.3	v1.5.0[929]
v2.1.2	v1.5.0[930]
v2.1.1	v1.5.0[931]
v2.1.0	v1.5.0[932]

KIB will run and print out the name of the created image for your infrastructure provider as shown on specific provider KIB pages below. Use this name when creating a Kubernetes cluster.

923 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v1.17.4
924 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v1.17.4
925 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v1.13.2
926 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v1.11.0
927 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v1.5.1
928 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v1.5.1
929 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v1.5.0
930 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v1.5.0
931 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v1.5.0
932 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v1.5.0

9.2.3 Next Steps:

Either return to Basic Install (see page 124) or Custom Install (see page 919) instructions, or for more KIB specific provider information you can continue to the provider link below for additional information:

- KIB with AWS (see page 1285)
- KIB for Azure (see page 1295)
- KIB with GCP (see page 1299)
- KIB for GPU (see page 1302)
- KIB with vSphere (see page 1305)
- KIB with Pre-provisioned Environments (see page 1312)
- Konvoy Image Builder CLI (see page 1312)
- Use Override Files with Konvoy Image Builder (see page 1330)

9.2.4 KIB with AWS

The following section describes how to use Konvoy Image Builder (KIB) with Amazon Web Services (AWS). A minimal set of permissions from the AWS Image Builder Book[933] should be met before beginning.

Section Contents

- Minimal IAM Permissions for KIB (see page 1285)
- Create a Custom AMI (see page 1288)
- AWS Air-gapped AMI (see page 1292)
- Add Custom Tags to Image (see page 1293)
- KIB for EKS (see page 1294)

9.2.4.1 Minimal IAM Permissions for KIB

Configure IAM Prerequisites before building an AWS Image

This section guides you in creating and using a minimally-scoped policy to create an Image for an AWS account using Konvoy Image Builder.

9.2.4.1.1 Prerequisites

Before applying the IAM Policies, verify the following:

- You have a valid AWS account with credentials configured[934] that can manage CloudFormation Stacks, IAM Policies, IAM Roles, and IAM Instance Profiles.
- The AWS CLI utility[935] is installed.

933 https://image-builder.sigs.k8s.io/capi/providers/aws.html#required-permissions-to-build-the-aws-amis
934 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html
935 https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html

9.2.4.1.2 Minimal Permissions

The following is an AWSCloudformation stack that creates the minimal policy to run KIB in AWS.

1. Copy the following contents into a file:

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  AWSIAMInstanceKIBUser:
    Properties:
      InstanceProfileName: KIBUserInstnaceProfile
      Roles:
        - Ref: KIBUserRole
    Type: AWS::IAM::InstanceProfile
  AWSIAMManagedPolicyKIBPolicy:
    Properties:
      Description: Minimal policy to run KIB in AWS
      ManagedPolicyName: kib-policy
      PolicyDocument:
        Statement:
          - Action:
              - ec2:AssociateRouteTable
              - ec2:AssociateRouteTable
              - ec2:AttachInternetGateway
              - ec2:AttachVolume
              - ec2:AuthorizeSecurityGroupIngress
              - ec2:CreateImage
              - ec2:CreateInternetGateway
              - ec2:CreateKeyPair
              - ec2:CreateRoute
              - ec2:CreateRouteTable
              - ec2:CreateSecurityGroup
              - ec2:CreateSubnet
              - ec2:CreateTags
              - ec2:CreateVolume
              - ec2:CreateVpc
              - ec2:DeleteInternetGateway
              - ec2:DeleteKeyPair
              - ec2:DeleteRouteTable
              - ec2:DeleteSecurityGroup
              - ec2:DeleteSnapshot
              - ec2:DeleteSubnet
              - ec2:DeleteVolume
              - ec2:DeleteVpc
              - ec2:DeregisterImage
              - ec2:DescribeAccountAttributes
              - ec2:DescribeImages
              - ec2:DescribeInstances
              - ec2:DescribeInternetGateways
              - ec2:DescribeKeyPairs
```

```
              - ec2:DescribeNetworkAcls
              - ec2:DescribeNetworkInterfaces
              - ec2:DescribeRegions
              - ec2:DescribeRouteTables
              - ec2:DescribeSecurityGroups
              - ec2:DescribeSubnets
              - ec2:DescribeVolume
              - ec2:DescribeVpcAttribute
              - ec2:DescribeVpcClassicLink
              - ec2:DescribeVpcClassicLinkDnsSupport
              - ec2:DescribeVpcs
              - ec2:DetachInternetGateway
              - ec2:DetachVolume
              - ec2:DisassociateRouteTable
              - ec2:ModifyImageAttribute
              - ec2:ModifySnapshotAttribute
              - ec2:ModifySubnetAttribute
              - ec2:ModifyVpcAttribute
              - ec2:RegisterImage
              - ec2:RevokeSecurityGroupEgress
              - ec2:RunInstances
              - ec2:StopInstances
              - ec2:TerminateInstances
            Effect: Allow
            Resource:
              - '*'
        Version: 2012-10-17
      Roles:
        - Ref: KIBUserRole
    Type: AWS::IAM::ManagedPolicy
    Version: 2012-10-17
  KIBUserRole:
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action:
              - sts:AssumeRole
            Effect: Allow
            Principal:
              Service:
                - ec2.amazonaws.com
          - Action:
              - sts:AssumeRole
            Effect: Allow
            Principal:
              AWS: arn:aws:iam::MYAWSACCOUNTID:root
        Version: 2012-10-17
      RoleName: kib-user-role
    Type: AWS::IAM::Role
```

2. Replace the following with the correct values:

- `MYFILENAME.yaml` - give your file a meaningful name.

- `MYSTACKNAME` - give your cloudformation stack a meaningful name.

3. Run the following command to create the stack:

```
aws cloudformation create-stack --template-body=file://MYFILENAME.yaml --stack-
name=MYSTACKNAME --capabilities  CAPABILITY_NAMED_IAM
```

9.2.4.2 Create a Custom AMI

9.2.4.2.1 Learn how to build a custom AMI for use with DKP

This procedure describes how to use the Konvoy Image Builder (see page 1282) (KIB) to create a Cluster API[936] compliant Amazon Machine Image (AMI). KIB uses variable overrides (see page 1330) to specify base image and container images to use in your new AMI.

The default AWS image is not recommended for use in production. We suggest using Konvoy Image Builder to create a custom AMI to take advantage of enhanced cluster operations. Explore the KIB with AWS (see page 1285) topics for more options.

9.2.4.2.2 Prerequisites

Before you begin, you must:

- Check the DKP Supported Kubernetes Versions (see page 1543).
- Check the Supported Infrastructure Operating Systems (see page 62)
- Download the KIB (see page 0) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
- Create a working `Docker` setup.
- Ensure you have met the minimal set of permissions from the AWS Image Builder Book[937].
- A Minimal IAM Permissions for KIB (see page 1285) to create an Image for an AWS account using Konvoy Image Builder.

9.2.4.2.3 Extract KIB Bundle

Extract the bundle and `cd` into the extracted `konvoy-image-bundle-$VERSION` folder. The bundled version of `konvoy-image` contains an embedded `docker` image that contains all the requirements for building.

936 https://cluster-api.sigs.k8s.io/
937 https://image-builder.sigs.k8s.io/capi/providers/aws.html#required-permissions-to-build-the-aws-amis

The `konvoy-image` binary and all supporting folders are also extracted. When run, `konvoy-image` bind mounts the current working directory (`${PWD}`) into the container to be used.

- Set environment variables for AWS access[938]. The following variables must be set using your credentials including required IAM (see page 1285):

```
export AWS_ACCESS_KEY_ID
export AWS_SECRET_ACCESS_KEY
export AWS_DEFAULT_REGION
```

- If you have an override file (see page 1330) to configure specific attributes of your AMI file, add it.

AMI images contain configuration information and software to create a specific, pre-configured, operating environment. For example, you can create an AMI image of your current computer system settings and software. The AMI image can then be replicated and distributed, creating your computer system for other users.

9.2.4.2.4 Build the Image

Depending on which version of DKP (see page 1282) you are running, steps and flags will be different. To deploy in a region where CAPI images are not provided, you need to use KIB to create your own image for the region. For a list of supported AWS regions, refer to the Published AMI[939]information from AWS.

9.2.4.2.4.1 Execute the following to begin image creation:

Run the `konvoy-image` command to build and validate the image.

```
konvoy-image build aws images/ami/centos-79.yaml
```

By default it builds in the `us-west-2` region. to specify another region set the `--region` flag:

```
konvoy-image build aws --region us-east-1 images/ami/centos-79.yaml
```

Ensure you have named the correct YAML file for your OS in the `konvoy-image build` command.

938 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-envvars.html
939 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html

Once KIB provisions the image successfully, the `ami` id is printed and written to the `manifest.json` file. This file has an `artifact_id` field whose value provides the name of the AMI ID as shown in the example below:

```
{
      "name": "rhel-7.9-fips",
      "builder_type": "amazon-ebs",
      "build_time": 1659486130,
      "files": null,
      "artifact_id": "us-west-2:ami-0f2ef742482e1b829",
      "packer_run_uuid": "0ca500d9-a5f0-815c-6f12-aceb4d46645b",
      "custom_data": {
        "containerd_version": "",
        "distribution": "RHEL",
        "distribution_version": "7.9",
        "kubernetes_cni_version": "",
        "kubernetes_version": "1.24.5+fips.0"
      }
    }
```

9.2.4.2.5 Related Information

For information on related topics or procedures, refer to the following:

- Creating GPU enabled on-premises configurations (see page 988)

9.2.4.2.6 Use Custom AMI to Build Cluster

9.2.4.2.6.1 Launch a DKP Cluster with a Custom AMI

To use the built `ami` with DKP, specify it with the `--ami` flag when calling cluster create.

```
dkp create cluster aws --cluster-name=$(whoami)-aws-cluster --ami ami-0123456789
```

9.2.4.2.6.2 Launch a DKP Cluster with Custom AMI Lookup

By default `konvoy-image` will name the AMI in such a way that `dkp` can discover the latest AMI for a base OS and Kubernetes version. To create a cluster that will use the latest AMI, specify the `--ami-format`, `--ami-base-os` and `--ami-owner` flags:

```
dkp create cluster aws --cluster-name=$(whoami)-aws-cluster --ami-format "konvoy-ami-
{{.BaseOS}}-?{{.K8sVersion}}-*" --ami-base-os centos-7 --ami-owner 123456789012
```

9.2.4.2.6.3 Using Custom Source AMIs

When using KIB for building machine images to Amazon, the default source AMIs that we provide are modeled by looking up an AMI based on the owner. Then we apply a filter for that operating system and version.

You can view an example of that with the provided `centos-79.yaml` snippet below:

```
download_images: true

packer:
  ami_filter_name: "CentOS Linux 7"
  ami_filter_owners: "125523088429"
  distribution: "CentOS"
  distribution_version: "7.9"
  source_ami: ""
  ssh_username: "centos"
  root_device_name: "/dev/sda1"
...
```

At times, a particular upstream AMI may not be available in your region or something could be renamed. Other times you want to provide a custom AMI. If this is the case, you will want to edit or create your own YAML file that looks up based on the `source_ami` field. For example, you can select images that are otherwise deprecated.

Once you select the source AMI that you want, you can declare that when running your build command:

```
konvoy-image build aws path/to/ami/centos-79.yaml --source-ami ami-0123456789
```

Alternatively, if you want to add it to your YAML file, or make your own file, you can do that as well. You add that AMI ID into the `source_ami` in the YAML file:

```
download_images: true

packer:
  ami_filter_name: ""
  ami_filter_owners: ""
  distribution: "CentOS"
  distribution_version: "7.9"
  source_ami: "ami-123456789"
  ssh_username: "centos"
  root_device_name: "/dev/sda1"
...
```

When you're done selecting your `source_ami`, you can build your KIB image as you would normally:

```
konvoy-image build aws path/to/ami/centos-79.yaml
```

9.2.4.3 AWS Air-gapped AMI

9.2.4.3.1 Create an Image using Konvoy Image Builder (KIB) for use in an air-gapped cluster

9.2.4.3.2 Prerequisites

- Before you begin, you must:
  - Check the DKP Supported Kubernetes Versions (see page 1543).
  - Check the Supported Infrastructure Operating Systems (see page 62)
  - Download the KIB (see page 0) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
  - Create a working Docker or other registry setup.
  - Ensure you have met the minimal set of permissions from the AWS Image Builder Book[940].
  - A Minimal IAM Permissions for KIB (see page 1285) to create an Image for an AWS account using Konvoy Image Builder.

The default AWS image is not recommended for use in production. We suggest using Konvoy Image Builder to create an AWS Air-gapped AMI to take advantage of enhanced cluster operations. Explore the KIB with AWS (see page 1285) topics for more options.

Using KIB (see page 1282), you can build an AMI without requiring access to the internet by providing an additional `--override` flag.

1. Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz` , extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.5.2_linux_amd64.tar.gz && cd dkp-v2.5.2/kib
   ```

2. Follow the instructions below to build an AMI.

Depending on which version of DKP (see page 1282) you are running, steps and flags will be different. To deploy in a region where CAPI images are not provided, you need to use KIB to create your own image for the region. For a list of supported AWS regions, refer to the Published AMI[941]information from AWS.

940 https://image-builder.sigs.k8s.io/capi/providers/aws.html#required-permissions-to-build-the-aws-amis
941 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html

9.2.4.3.2.1 Execute the following to begin image creation:

Run the `konvoy-image` command to build and validate the image.

```
konvoy-image build aws images/ami/centos-79.yaml --overrides overrides/offline.yaml
```

By default it builds in the `us-west-2` region. to specify another region set the `--region` flag:

```
konvoy-image build aws --region us-east-1 images/ami/centos-79.yaml --overrides
overrides/offline.yaml
```

When the command is complete the `ami` id is printed and written to `./manifest.json`.

After you complete these steps, you can seed your docker registry.

9.2.4.4 Add Custom Tags to Image

As certain cloud environments enforce the tagging of resources, DKP platform users have inquired about whether is possible to add custom tags on an AMI that was built with Konvoy Image Builder (KIB). Currently, adding a custom tag as an attribute to the AMI is only possible by modifying the `packer.json` file.

In order to add a tag, perform the commands below:

1. Generate manifest files for KIB by executing the following command. In this example, we're building a CentOS 7.9 image:

   ```
   ./konvoy-image build images/ami/centos-79.yaml
   ```

2. Send `SIGINT` to the process to kill it after seeing the output `...writing new packer configuration to work/centos-7-xxxxxxx-yyyyy`. During the attempt to build the AMI a packer.json is generated under the path `work/centos-7-xxxxxxx-yyyyy/packer.json`.

3. Edit the packer.json file by adding the parameter `"run_tags"` to the `packer.json` file as seen in the example below:

   ```
    "builders": [
   {
   "name": "{{(user `distribution`) | lower}}-{{user `distribution_version`}}
   {{user `build_name_extra`}}",
   "type": "amazon-ebs",
   "run_tags": {"my_custom_tag": "tag"},
   ```

```
"instance_type": "{{user `aws_instance_type`}}",
"source_ami_filter": {
"filters": {
"virtualization-type": "hvm",
"name": "{{user `ami_filter_name`}}",
"root-device-type": "ebs",
"architecture": "x86_64"
},
```

4. Use the packer manifest option to build the AMI:

```
./konvoy-image build aws images/ami/centos-79.yaml --packer-manifest=work/
centos-7-1658174984-TycMM/packer.json
2022/09/07 18:23:33 writing new packer configuration to work/centos-7-166257501
3-zJUhP
2022/09/07 18:23:33 starting packer build
centos-7.9: output will be in this color.

==> centos-7.9: Prevalidating any provided VPC information
==> centos-7.9: Prevalidating AMI Name: konvoy-ami-centos-7-1.25.4-1662575013
centos-7.9: Found Image ID: ami-0686851c4e7b1a8e1
==> centos-7.9: Creating temporary keypair: packer_6318e1a6-9f45-c01b-c7ca-
f5404735f709
==> centos-7.9: Creating temporary security group for this instance:
packer_6318e1aa-7448-d74b-9b90-166f26d21619
==> centos-7.9: Authorizing access to port 22 from [0.0.0.0/0] in the temporary
security groups...
==> centos-7.9: Launching a source AWS instance...
centos-7.9: Adding tag: "my_custom_tag": "tag"
centos-7.9: Instance ID: i-04d457b20713dcf7d
==> centos-7.9: Waiting for instance (i-04d457b20713dcf7d) to become ready...
==> centos-7.9: Using SSH communicator to connect: 34.222.153.107
==> centos-7.9: Waiting for SSH to become available...
```

Now that your tags have been added to the AMI, you can continue to Advanced Configuration (see page 931) for AWS and finish building your clusters.

9.2.4.5 KIB for EKS

AWS EKS best practices discourage building custom images. The Amazon EKS Optimized AMI[942] is the preferred way to deploy containers for EKS. If the image is customized, it breaks some of the autoscaling and security capabilities of EKS.

942 https://docs.aws.amazon.com/eks/latest/userguide/eks-optimized-amis.html

9.2.5 KIB for Azure

9.2.5.1 Learn how to build a custom Azure Image for use with DKP

This procedure describes how to use the Konvoy Image Builder (see page 1282) (KIB) to create a Cluster API[943] compliant Azure Virtual Machine (VM) Image. The VM Image contains the base operating system you specify and all the necessary Kubernetes components. The Konvoy Image Builder uses variable `overrides` to specify the base image and container images to use in your new Azure VM image.

The default Azure image is not recommended for use in production. We suggest using KIB for Azure to build the image in order to take advantage of enhanced cluster operations. To explore more information on this topic refer to the Azure Infrastructure (see page 1033).

9.2.5.2 Prerequisites

Before you begin, you must:

- Check the DKP 2.5.0 Supported Kubernetes Versions (see page 1543)
- Check the Supported Infrastructure Operating Systems (see page 62)
- Download the Konvoy Image Builder (see page 1282) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
- Create a working `Docker` setup

9.2.5.3 Extract the KIB Bundle

Extract the bundle and `cd` into the extracted `konvoy-image-bundle-$VERSION_$OS` folder. The bundled version of `konvoy-image` contains an embedded `docker` image that contains all the requirements for building.

The `konvoy-image` binary and all supporting folders are also extracted. When extracted, `konvoy-image` bind mounts the current working directory (`${PWD}`) into the container to be used.

943 https://cluster-api.sigs.k8s.io/

9.2.5.4 Configure Azure Prerequisites

> If you have already followed the Azure Prerequisites (see page 1033) topic steps, then the environment variables needed by KIB (`[AZURE_CLIENT_SECRET`, `AZURE_CLIENT_ID`, `AZURE_TENANT_ID`, `AZURE_SUBSCRIPTION_ID]`) are set and do not need repeated if you are still working in the same window.

If you have not executed the Azure Prerequisite (see page 1033) steps, they are listed below.

1. Sign in to Azure:

```
az login
```

```
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "id": "b1234567-abcd-11a1-a0a0-1234a5678b90",
    "isDefault": true,
    "managedByTenants": [],
    "name": "Mesosphere Developer Subscription",
    "state": "Enabled",
    "tenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "user": {
      "name": "user@azuremesosphere.onmicrosoft.com",
      "type": "user"
    }
  }
]
```

2. Create an Azure Service Principal (SP) by running the following command:

   If an SP with the name exists, this command will rotate the password.

```
az ad sp create-for-rbac --role contributor --name "$(whoami)-konvoy" --
scopes=/subscriptions/$(az account show --query id -o tsv) --query
"{ client_id: appId, client_secret: password, tenant_id: tenant }"
```

```
{
  "client_id": "7654321a-1a23-567b-b789-0987b6543a21",
  "client_secret": "Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C",
  "tenant_id": "a1234567-b132-1234-1a11-1234a5678b90"
}
```

3. Set the `AZURE_CLIENT_SECRET` environment variable:

```
export AZURE_CLIENT_SECRET="<azure_client_secret>" #
Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C
export AZURE_CLIENT_ID="<client_id>"               # 7654321a-1a23-567b-
b789-0987b6543a21
export AZURE_TENANT_ID="<tenant_id>"               # a1234567-b132-1234-1a11-12
34a5678b90
export AZURE_SUBSCRIPTION_ID="<subscription_id>"   # b1234567-abcd-11a1-
a0a0-1234a5678b90
```

4. Ensure you have an override file (see page 1330) to configure specific attributes[944] of your Azure image.

9.2.5.5 Build the Image

Run the `konvoy-image` command to build and validate the image.

```
konvoy-image build azure --client-id ${AZURE_CLIENT_ID} --tenant-id $
{AZURE_TENANT_ID} --overrides override-source-image.yaml images/azure/ubuntu-2004.yam
l
```

By default, the image builder builds in the `westus2` location. To specify another location set the `--location` flag (shown in example below is how to change the location to `eastus`):

```
konvoy-image build azure --client-id ${AZURE_CLIENT_ID} --tenant-id $
{AZURE_TENANT_ID} --location eastus --overrides override-source-image.yaml images/
azure/centos-7.yaml
```

When the command is complete, the image id is printed and written to the `./manifest.json` file. This file has an `artifact_id` field whose value provides the name of the image. You should then specify this image id when creating the cluster.

944 https://github.com/mesosphere/konvoy-image-builder/tree/7447074a6d910e71ad2e61fc4a12820d073ae5ae/images/azure

9.2.5.6 Image Gallery

By default Konvoy Image Builder will create a Resource Group, Gallery, and Image Name to store the resulting image in. To specify a specific Resource Group, Gallery, or Image Name flags may be specified:

```
--gallery-image-locations string     a list of locations to publish the image
(default same as location)
--gallery-image-name string          the gallery image name to publish the image to
--gallery-image-offer string         the gallery image offer to set (default "dkp")
--gallery-image-publisher string     the gallery image publisher to set (default
"dkp")
--gallery-image-sku string           the gallery image sku to set
--gallery-name string                the gallery name to publish the image in
(default "dkp")
--resource-group string              the resource group to create the image in
(default "dkp")
```

When creating your cluster, you will then add this flag during the create process for your custom image: `--compute-gallery-id "<Managed Image Shared Image Gallery Id>"`. See Create a New Azure Cluster (see page 1041) for specific consumption of image commands.

The SKU and Image Name will default to the values found in the image YAML.

9.2.5.7 Marketplace Images for Rocky Linux

Similar to Image Gallery, additional flags allow DKP to create a cluster with Marketplace based images for Rocky Linux 9.0: `--plan-offer`, `--plan-publisher` and `--plan-sku`.

- If you use these fields in the override file (see page 1330) when you create a machine image with KIB, you must also set the corresponding flags when you create your cluster with DKP.
- Conversely, if you do not use these fields when you create a machine image with KIB, you do not need to set these flags when you create your cluster with DKP.

Rocky Linux YAML

```
---
download_images: true

packer:
  distribution: "rockylinux-9"                # Offer
  distribution_version: "rockylinux-9"  # SKU
  # Azure Rocky linux official image: https://portal.azure.com/#view/
Microsoft_Azure_Marketplace/GalleryItemDetailsBladeNopdl/id/
erockyenterprisesoftwarefoundationinc1653071250513.rockylinux-9
  image_publisher: "erockyenterprisesoftwarefoundationinc1653071250513"
  image_version: "latest"
  ssh_username: "azureuser"
```

```
  plan_image_sku: "rockylinux-9" # SKU
  plan_image_offer: "rockylinux-9" # offer
  plan_image_publisher: "erockyenterprisesoftwarefoundationinc1653071250513" #
publisher

build_name: "rocky-90-az"
packer_builder_type: "azure"
python_path: ""
```

9.2.5.8 KIB for AKS

Because custom virtual machine images are discouraged in AKS, Konvoy Image Builder (KIB) does not include any support for building custom machine images for AKS. If the image is customized, it breaks some of the autoscaling and security capabilities of AKS. The AKS managed image ensures everything is patched and tuned for container workload.

9.2.6 KIB with GCP

This procedure describes how to use the Konvoy Image Builder (see page 1282) (KIB) to create a Cluster API[945] compliant GCP image. GCP images contain configuration information and software to create a specific, pre-configured, operating environment. For example, you can create a GCP image of your current computer system settings and software. The GCP image can then be replicated and distributed, creating your computer system for other users. KIB uses variable overrides (see page 1330) to specify base image and container images to use in your new GCP image.

Google Cloud Platform does not publish images. You must first build the image using Konvoy Image Builder. For more information regarding images and clusters, refer to the GCP Infrastructure (see page 1199) section of the documentation.

9.2.6.1 Prerequisites

Before you begin, you must:

- Check the supported DKP version (see page 1543).
- Check the Supported Infrastructure Operating Systems (see page 62)
- Download the KIB (see page 1283) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
- Create a working `Docker` setup.
- On Debian-based Linux distributions, install a version of the cri-tools[946] package known to be compatible with both the Kubernetes and container runtime versions.

945 https://cluster-api.sigs.k8s.io/
946 https://github.com/kubernetes-sigs/cri-tools

- Verify that your Google Cloud project does not have the Enable OS Login feature enabled. See below for more information:

> ⚠ The Enable OS Login feature is sometimes enabled by default in GCP projects. If the OS login feature is enabled, KIB will not be able to `ssh` to the VM instances it creates and will not be able to successfully create an image.
> To check if it is enabled, use the commands on this page https://cloud.google.com/compute/docs/metadata/setting-custom-metadata#console_2 to inspect the metadata configured in in your project. If you find the the `enable-oslogin` flag set to TRUE, you must remove (or set it to FALSE) to successfully use KIB.

9.2.6.2 GCP Prerequisite Roles

- If you are creating your image on either a non-GCP instance or one that does not have the required roles (see page 1199):
  - (option 1) Create a service account using the following `gcloud` commands:

```
export GCP_PROJECT=<your GCP project ID>
export GCP_SERVICE_ACCOUNT_USER=<some new service account user>
export GOOGLE_APPLICATION_CREDENTIALS="$HOME/.gcloud/credentials.json"

gcloud iam service-accounts create "${GCP_SERVICE_ACCOUNT_USER}" --
project=${GCP_PROJECT}
gcloud projects add-iam-policy-binding ${GCP_PROJECT} --
member="serviceAccount:${GCP_SERVICE_ACCOUNT_USER}@$
{GCP_PROJECT}.iam.gserviceaccount.com" --role=roles/
compute.instanceAdmin.v1
gcloud projects add-iam-policy-binding ${GCP_PROJECT} --
member="serviceAccount:${GCP_SERVICE_ACCOUNT_USER}@$
{GCP_PROJECT}.iam.gserviceaccount.com" --role=roles/
iam.serviceAccountUser
gcloud iam service-accounts keys create $
{GOOGLE_APPLICATION_CREDENTIALS} --iam-account="$
{GCP_SERVICE_ACCOUNT_USER}@${GCP_PROJECT}.iam.gserviceaccount.com"
```

  - (option 2) If you have already created a service account, retrieve the credentials for an existing service account using the following `gcloud` commands:

```
export GCP_PROJECT=<your GCP project ID>
export GCP_SERVICE_ACCOUNT_USER=<existing service account user>
export GOOGLE_APPLICATION_CREDENTIALS="$HOME/.gcloud/credentials.json"
```

```
gcloud projects add-iam-policy-binding ${GCP_PROJECT} --
member="serviceAccount:${GCP_SERVICE_ACCOUNT_USER}@$
{GCP_PROJECT}.iam.gserviceaccount.com" --role=roles/
compute.instanceAdmin.v1
gcloud projects add-iam-policy-binding ${GCP_PROJECT} --
member="serviceAccount:${GCP_SERVICE_ACCOUNT_USER}@$
{GCP_PROJECT}.iam.gserviceaccount.com" --role=roles/
iam.serviceAccountUser
gcloud iam service-accounts keys create $
{GOOGLE_APPLICATION_CREDENTIALS} --iam-account="$
{GCP_SERVICE_ACCOUNT_USER}@${GCP_PROJECT}.iam.gserviceaccount.com"
```

- Export the static credentials that will be used to create the cluster:

```
export GCP_B64ENCODED_CREDENTIALS=$(base64 < "$
{GOOGLE_APPLICATION_CREDENTIALS}" | tr -d '\n')
```

9.2.6.3 Build the GCP image

1. Run the `konvoy-image` command to build and validate the image:

```
./konvoy-image build gcp --project-id ${GCP_PROJECT} --network ${NETWORK_NAME}
images/gcp/ubuntu-2004.yaml
```

2. KIB will run and print out the name of the created image, you will use this name when creating a Kubernetes cluster. See sample output below:

```
...
==> ubuntu-2004-focal-v20220419: Deleting instance...
    ubuntu-2004-focal-v20220419: Instance has been deleted!
==> ubuntu-2004-focal-v20220419: Creating image...
==> ubuntu-2004-focal-v20220419: Deleting disk...
    ubuntu-2004-focal-v20220419: Disk has been deleted!
==> ubuntu-2004-focal-v20220419: Running post-processor: manifest
Build 'ubuntu-2004-focal-v20220419' finished after 7 minutes 46 seconds.

==> Wait completed after 7 minutes 46 seconds

==> Builds finished. The artifacts of successful builds are:
--> ubuntu-2004-focal-v20220419: A disk image was created: konvoy-ubuntu-2004-1-2
3-7-1658523168
--> ubuntu-2004-focal-v20220419: A disk image was created: konvoy-ubuntu-2004-1-2
3-7-1658523168
```

3. To find a list of images you have created in your account, run the following command:

```
gcloud compute images list --no-standard-images
```

9.2.6.4 Create a Network (optional)

Building an image requires a Network[947] with firewall rules that allow SSH access to the VM instance.

1. Set your GCP Project ID for your `gcp` account unless already set previously:

   ```
   export GCP_PROJECT=<your GCP project ID>
   ```

2. Run the following to create a new network:

   ```
   export NETWORK_NAME=kib-ssh-network
   gcloud compute networks create "${NETWORK_NAME}" --project="${GCP_PROJECT}" --
   subnet-mode=auto --mtu=1460 --bgp-routing-mode=regional
   ```

3. Create the firewall rule to allow Ingress access on port 22:

   ```
   gcloud compute firewall-rules create "${NETWORK_NAME}-allow-ssh" --project="$
   {GCP_PROJECT}" --network="projects/${GCP_PROJECT}/global/networks/$
   {NETWORK_NAME}" --description="Allows TCP connections from any source to any
   instance on the network using port 22." --direction=INGRESS --priority=65534 --
   source-ranges=0.0.0.0/0 --action=ALLOW --rules=tcp:22
   ```

With your KIB image now created, you can now move on to create your cluster and set up your Cluster API[948] (CAPI) controllers..

9.2.7 KIB for GPU

Create a GPU supported OS image using Konvoy Image Builder

Using the Konvoy Image Builder (see page 1282), you can build an image that has support to use NVIDIA GPU hardware to support GPU workloads.

NOTE: The NVIDIA driver requires a specific Linux kernel version. Make sure that the base image for the OS version has the required kernel version.

947 https://cloud.google.com/vpc/docs/vpc
948 https://cluster-api.sigs.k8s.io/

See Supported Infrastructure Operating Systems (see page 62) for a list of OS versions and the corresponding kernel versions known to work with the NVIDIA driver.

If the NVIDIA runfile[949] installer has not been downloaded, then retrieve and install the download first by running the following command. The first line in the command below downloads and installs the runfile and the second line places it in the artifacts directory (you must create an `artifacts` directory if you don't already have one).

```
curl -O https://download.nvidia.com/XFree86/Linux-x86_64/470.82.01/NVIDIA-Linux-
x86_64-470.82.01.run
mv NVIDIA-Linux-x86_64-470.82.01.run artifacts
```

DKP supported NVIDIA driver[950] version is 470.x.

To build an image for use on GPU enabled hardware, perform the following steps.

1. In your `overrides/nvidia.yaml` file, add the following to enable GPU builds. You can also access and use the overrides repo[951] or in the documentation under Nvidia GPU Override File (see page 1334) or Offline Nvidia Override file (see page 1335).
   a. Non-air-gapped GPU override:

```
gpu:
  types:
    - nvidia
build_name_extra: "-nvidia"
```

   b. Air-gapped GPU override:

```
# Use this file when building a machine image, not as a override secret
for preprovisioned environments
nvidia_runfile_local_file: "{{ playbook_dir}}/../artifacts/
{{ nvidia_runfile_installer }}"
gpu:
  types:
    - nvidia

build_name_extra: "-nvidia"
```

949 https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html#runfile
950 https://www.nvidia.com/Download/Find.aspx
951 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

2. Build your image using the following Konvoy Image Builder commands, making sure to include the flag `--instance-type` that specifies an AWS instance that has an available GPU:
   AWS Example:

   ```
   konvoy-image build --region us-east-1 --instance-type=p2.xlarge --source-
   ami=ami-12345abcdef images/ami/centos-7.yaml --overrides overrides/nvidia.yaml
   ```

   In this example, we chose an instance type with an NVIDIA GPU using the `--instance-type` flag, and we provided the NVIDIA overrides using the `--overrides` flag. See KIB with AWS (see page 1285) for more information on creating an AMI.

Additional helpful information can be found in the NVIDIA Device Plug-in[952] for Kubernetes instructions and the Installation Guide of Supported Platforms[953].

See also: NVIDIA documentation[954]

When the Konvoy Image Builder image is complete, the custom `ami` id is printed and written to `./manifest.json`. To use the built `ami` with Konvoy, specify it with the `--ami` flag when calling cluster create.

Additional helpful information can be found in the NVIDIA Device Plug-in[955] for Kubernetes instructions and the Installation Guide of Supported Platforms[956].

9.2.7.1 Verification

To verify that the NVIDIA driver is working, connect to the node and execute this command:

```
nvidia-smi
```

When drivers are successfully installed, the display looks like the following:

```
Fri Jun 11 09:05:31 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.73.01    Driver Version: 460.73.01    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:1E.0 Off |                    0 |
```

952 https://github.com/NVIDIA/k8s-device-plugin/blob/master/README.md
953 https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html
954 https://nvidia.custhelp.com/app/answers/detail/a_id/131/kw/driver%20installation%20docs/related/1
955 https://github.com/NVIDIA/k8s-device-plugin/blob/master/README.md
956 https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html

```
| N/A   35C    P0    73W / 149W |      0MiB / 11441MiB |     99%      Default |
|                                |                      |                  N/A |
+--------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

9.2.8 KIB with vSphere

vSphere for DKP using Konvoy Image Builder

This section assumes you have access to a vSphere environment and have credentials with proper privileges and access. Refer to the vCenter[957] and vSphere Client[958] documentation for details.

- Create a vSphere Base OS Image (see page 1305)
- Create a vSphere Virtual Machine Template (see page 1307)

9.2.8.1 Create a vSphere Base OS Image

Creating a base OS image from DVD ISO files is a one-time process. Building a base OS image creates a base vSphere template in your vSphere environment. The base OS image is used by Konvoy Image Builder (see page 1282)(KIB) to create a VM template to configure Kubernetes nodes by the DKP vSphere provider.

9.2.8.1.1 Create the Base OS Image

For vSphere, a username and password is populated by `SSH_USERNAME` and the user can use authorization via `SSH_PASSWORD` or `SSH_PRIVATE_KEY_FILE` environment variables and required by default by packer[959]. This user should have administrator privileges. It is possible to configure a custom user and password when building the OS image, however, that requires the Konvoy Image Builder (KIB) configuration to be overridden (see page 1330). DKP advises not to use usernames and passwords for security reasons, but instead to use private and public keys. If that is not possible, passwords should be generated and a minimum of 20 characters long.

While creating the base OS image, it is important to take into consideration the following elements:

- Storage configuration: D2iQ recommends customizing disk partitions and not configuring a SWAP partition.
- Network configuration: as KIB must download and install packages, activating the network is required.

957 https://www.vmware.com/products/vcenter-server.html
958 https://docs.vmware.com/en/VMware-vSphere/index.html
959 https://github.com/mesosphere/konvoy-image-builder/blob/0523fd2c5e6e1ad1d4962f60a47039aa145a6e42/pkg/packer/manifests/vsphere/packer.pkr.hcl

- Connect to Red Hat: if using RHEL, registering with Red Hat is required to configure software repositories and install software packages.
- Software selection: D2iQ recommends choosing **Minimal Install**.
- DKP recommends to install with the packages provided by the operating system package managers. Use the version that corresponds to the major version of your operating system.
- DKP advises not to use usernames and passwords for security reasons, but instead to use private and public keys. If that is not possible, passwords should be generated and a minimum of 20 characters long.

9.2.8.1.1.1 Disk Size

For each cluster you create using this base OS image, ensure you establish the disk size of the **root file system** based on:

- The minimum DKP storage requirements (see page 110).
- The minimum storage requirements for your organization.

Defaults

Clusters are created with a default disk size of 80 GB.

> ⚠ **For clusters created with the default disk size, the base OS image root file system must be exactly 80 GB.** The root file system cannot be reduced automatically when a machine first boots.

Customization

You can also specify a custom disk size when you create a cluster (see page 0) (see the flags of the Create new vSphere Cluster (see page 1462) command). This allows you to use one base OS image to create multiple clusters that have different storage requirements.

> ⚠ Before specifying a disk size when you create a cluster, take into account:
>
> 1. **For some base OS images, the custom disk size option has no effect on the size of the root file system.** This is because some root file systems, for example, those contained in an LVM Logical Volume, cannot be resized automatically when a machine first boots.
> 2. **The specified custom disk size must be equal to, or larger than the size of the base OS image root file system.** This is because a root file system cannot be reduced automatically when a machine first boots.

For additional information and examples of KIB vSphere overrides, see this page: Base Image Override Files (see page 1337)

The next step is to create a vSphere VM template (see page 1307) that contains the CAPI and Kubernetes objects.

Refer to the vCenter[960] and vSphere Client[961] documentation for details.

9.2.8.1.2 Next Step:

Create a vSphere Virtual Machine Template (see page 1307)

9.2.8.2 Create a vSphere Virtual Machine Template

Create a vSphere template for your cluster from a base OS image

You must have at least one image before creating a new cluster. As long as you have an image, this step in your configuration is not required each time since that image can be used to spin up a new cluster. However, if you need different images for different environments or providers, you will need to create a new custom image.

The Konvoy Image Builder (see page 1305) (KIB) uses the values in `image.yaml` and the input base OS image to create a vSphere template directly on the vCenter server. Using KIB, you can build an image without requiring access to the internet by providing an additional `--override` flag.

9.2.8.2.1 Prerequisites

- Users need to create a base OS image (see page 1305) in their vSphere client before starting this procedure.
- Konvoy Image Builder (KIB) downloaded (see page 71) and extracted
- **Air-gapped Only**
  - Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz`, extract the tarball to a local directory:

    ```
    tar -xzvf dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz && cd dkp-v2.5.0/kib
    ```

  - Follow the instructions to build a vSphere template below and set the override `--overrides overrides/offline.yaml` flag.

9.2.8.2.2 Create a vSphere Template for Your Cluster from a Base OS Image

Using the base OS image created in a previous procedure, DKP creates the new vSphere template directly on the vCenter server.

960 https://www.vmware.com/products/vcenter-server.html
961 https://docs.vmware.com/en/VMware-vSphere/index.html

1. Set the following vSphere environment variables on the bastion VM host:

```
export VSPHERE_SERVER=your_vCenter_APIserver_URL
export VSPHERE_USERNAME=your_vCenter_user_name
export VSPHERE_PASSWORD=your_vCenter_password
```

2. Copy the base OS image file created in the vSphere Client to your desired location on the bastion VM host, and make a note of the path and file name.

3. Create an `image.yaml` file and add the following variables for vSphere. DKP uses this file and these variables as inputs in the next step.
NOTE: You will need to replace OS name below based on your OS - EX: “rhel-79” to “rhel-86”. See other YAML examples[962] for copy and paste below last step.

```
---
download_images: true
build_name: "rhel-79"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-
init-vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-rhel-7.9" # change default value with your base template name
  vsphere_guest_os_type: "rhel7_64Guest"
  guest_os_type: "rhel7-64"
  # goss params
  distribution: "RHEL"
  distribution_version: "7.9"
# Use following overrides to select the authentication method that can be used
with base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # if set to true, ssh_password and ssh_private_key
will be ignored
```

962 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

4. Create a vSphere VM template with your variation of the following command:

   ```
   konvoy-image build images/ova/<image.yaml>
   ```

   Any additional configurations can be added to this command using `--overrides` flags as shown below:

   a. for FIPS, add this flag - `--overrides overrides/fips.yaml`
   b. for air-gapped, add this flag `--overrides overrides/offline-fips.yaml`

5. The Konvoy Image Builder (see page 1305) (KIB) uses the values in `image.yaml` and the input base OS image to create a vSphere template directly on the vCenter server. This template contains the required artifacts needed to create a Kubernetes cluster.
   When KIB provisions the OS image[963] successfully, it creates a manifest file. The `artifact_id` field of this file contains the name of the AMI ID (AWS), template name (vSphere), or image name (GCP/Azure), for example:

   ```
   {
         "name": "vsphere-clone",
         "builder_type": "vsphere-clone",
         "build_time": 1644985039,
         "files": null,
         "artifact_id": "konvoy-ova-vsphere-rhel-84-1.21.6-1644983717",
         "packer_run_uuid": "260e8110-77f8-ca94-e29e-ac7a2ae779c8",
         "custom_data": {
           "build_date": "2022-02-16T03:55:17Z",
           "build_name": "vsphere-rhel-84",
           "build_timestamp": "1644983717",
           [...]
         }
       }
   ```

6. Next steps are to deploy a DKP cluster using your vSphere template.

9.2.8.2.2.1 Additional OS YAML files for copy/paste:

RHEL 8.6

```
---
download_images: true
build_name: "rhel-86"
```

963 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

```
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "os-qualification-templates/d2iq-base-RHEL-86" # change default value
with your base template name
  vsphere_guest_os_type: "rhel8_64Guest"
  guest_os_type: "rhel8-64"
  # goss params
  distribution: "RHEL"
  distribution_version: "8.6"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

Ubuntu 20.04

```
---
download_images: true
build_name: "ubuntu-2004"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "os-qualification-templates/d2iq-base-Ubuntu-20.04" # change default
 value with your base template name
```

```
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "ubuntu2004-64"
  # goss params
  distribution: "ubuntu"
  distribution_version: "20.04"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

Rocky Linux 9.1

```
---
download_images: true
build_name: "rocky-91"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "os-qualification-templates/d2iq-base-RockyLinux-9.1" # change default
 value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "rocky9-64"
  # goss params
  distribution: "rocky"
  distribution_version: "9.1"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

9.2.9 KIB with Pre-provisioned Environments

In Pre-provisioned environments, KIB runs inside the `bootstrap cluster`, against the list of nodes that you define. Whereas with other providers, you run KIB outside the cluster manually to build your images.

For Pre-provisioned, you define a set of nodes that already exist. During the cluster creation process, KIB is built into DKP and automatically runs the machine configuration process (which KIB uses to build images with other providers) against the set of nodes that you defined. This results in your pre-existing/'pre-provisioned' nodes being configured properly. The remainder of the cluster provisioning happens automatically after that.

- In a **Pre-provisioned environment** (see page 125), you have existing machines and DKP consumes them to form a cluster.
- When you have another provisioner (for example, cloud providers such AWS (see page 261), vSphere (see page 378) and others), you build images with KIB and DKP consumes the images to provision machines and form a cluster.

9.2.10 Konvoy Image Builder CLI

9.2.10.1 Other resources:

- Main Github KIB CLI[964]
- DKP CLI (see page 1436)

9.2.10.2 konvoy-image build

build and provision images

9.2.10.2.1 Synopsis

Build and Provision images. Specifying AWS arguments is deprecated and will be removed in a future version. Use the `aws` subcommand instead.

964 https://github.com/mesosphere/konvoy-image-builder/tree/main/docs/cli

```
konvoy-image build <image.yaml> [flags]
```

9.2.10.2.2 Options

```
      --ami-groups stringArray           a list of AWS groups which are allowed use
the image, using 'all' result in a public image
      --ami-regions stringArray          a list of regions to publish amis
      --ami-users stringArray            a list AWS user accounts which are allowed
use the image
      --containerd-version string        the version of containerd to install
      --dry-run                          do not create artifacts, or delete them
after creating. Recommended for tests.
      --extra-vars strings               flag passed Ansible's extra-vars
  -h, --help                             help for build
      --instance-type string             instance type used to build the AMI; the
type must be present in the region in which the AMI is built
      --kubernetes-version string        The version of kubernetes to install.
Example: 1.21.6
      --overrides strings                a comma separated list of override YAML
files
      --packer-manifest string           provide the path to a custom packer manifest
      --packer-on-error string           [advanced] set error strategy for packer.
strategies [cleanup, abort, run-cleanup-provisioner]
      --packer-path string               the location of the packer binary (default
"packer")
      --region string                    the region in which to build the AMI
      --source-ami string                the ID of the AMI to use as the source; must
be present in the region in which the AMI is built
      --source-ami-filter-name string    restricts the set of source AMIs to ones
whose Name matches filter
      --source-ami-filter-owner string   restricts the source AMI to ones with this
 owner ID
      --work-dir string                  path to custom work directory generated by
the generate command
```

9.2.10.2.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.2.10.2.4 SEE ALSO

- konvoy-image build aws (see page 1314) - build and provision aws images
- konvoy-image build azure (see page 1315) - build and provision azure images

- konvoy-image build gcp (see page 1316) - build and provision gcp images

9.2.10.2.5 konvoy-image build aws

build and provision aws images

```
konvoy-image build aws <image.yaml> [flags]
```

9.2.10.2.5.1 Examples

```
aws --region us-west-2 --source-ami=ami-12345abcdef images/ami/centos-79.yaml
```

9.2.10.2.5.2 Options

```
      --ami-groups stringArray           a list of AWS groups which are allowed use
the image, using 'all' result in a public image
      --ami-regions stringArray          a list of regions to publish amis
      --ami-users stringArray            a list AWS user accounts which are allowed
use the image
      --containerd-version string        the version of containerd to install
      --dry-run                          do not create artifacts, or delete them
after creating. Recommended for tests.
      --extra-vars strings               flag passed Ansible's extra-vars
  -h, --help                             help for aws
      --instance-type string             instance type used to build the AMI; the
type must be present in the region in which the AMI is built
      --kubernetes-version string        The version of kubernetes to install.
Example: 1.21.6
      --overrides strings                a comma separated list of override YAML
files
      --packer-manifest string           provide the path to a custom packer manifest
      --packer-on-error string           [advanced] set error strategy for packer.
strategies [cleanup, abort, run-cleanup-provisioner]
      --packer-path string               the location of the packer binary (default
"packer")
      --region string                    the region in which to build the AMI
      --source-ami string                the ID of the AMI to use as the source; must
be present in the region in which the AMI is built
      --source-ami-filter-name string    restricts the set of source AMIs to ones
whose Name matches filter
      --source-ami-filter-owner string   restricts the source AMI to ones with this
 owner ID
      --work-dir string                  path to custom work directory generated by
the generate command
```

9.2.10.2.5.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.2.10.2.5.4 SEE ALSO

- konvoy-image build (see page 1312) - build and provision images

9.2.10.2.6 konvoy-image build azure

build and provision azure images

```
konvoy-image build azure <image.yaml> [flags]
```

9.2.10.2.6.1 Examples

```
azure --location westus2 --subscription-id <sub_id> images/azure/centos-79.yaml
```

9.2.10.2.6.2 Options

```
      --client-id string                      the client id to use for the build
      --cloud-endpoint string                 Azure cloud endpoint. Which can be one
of [Public USGovernment China] (default "Public")
      --containerd-version string             the version of containerd to install
      --dry-run                               do not create artifacts, or delete them
after creating. Recommended for tests.
      --extra-vars strings                    flag passed Ansible's extra-vars
      --gallery-image-locations stringArray   a list of locations to publish the
image (default [westus])
      --gallery-image-name string             the gallery image name to publish the
image to
      --gallery-image-offer string            the gallery image offer to set (default
 "dkp")
      --gallery-image-publisher string        the gallery image publisher to set
(default "dkp")
      --gallery-image-sku string              the gallery image sku to set
      --gallery-name string                   the gallery name to publish the image
in (default "dkp")
  -h, --help                                  help for azure
```

```
      --instance-type string                  the Instance Type to use for the build
(default "Standard_D2s_v3")
      --kubernetes-version string             The version of kubernetes to install.
Example: 1.21.6
      --location string                       the location in which to build the
image (default "westus2")
      --overrides strings                     a comma separated list of override YAML
files
      --packer-manifest string                provide the path to a custom packer
manifest
      --packer-on-error string                [advanced] set error strategy for
 packer. strategies [cleanup, abort, run-cleanup-provisioner]
      --packer-path string                    the location of the packer binary
(default "packer")
      --resource-group string                 the resource group to create the image
in (default "dkp")
      --subscription-id string                the subscription id to use for the
build
      --tenant-id string                      the tenant id to use for the build
      --work-dir string                       path to custom work directory generated
by the generate command
```

9.2.10.2.6.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.2.10.2.6.4 SEE ALSO

- konvoy-image build (see page 1312) - build and provision images

9.2.10.2.7 konvoy-image build gcp

build and provision gcp images

```
konvoy-image build gcp <image.yaml> [flags]
```

9.2.10.2.7.1 Examples

```
gcp ... images/gcp/centos-79.yaml
```

9.2.10.2.7.2 Options

```
      --containerd-version string   the version of containerd to install
      --dry-run                     do not create artifacts, or delete them after
creating. Recommended for tests.
      --extra-vars strings          flag passed Ansible's extra-vars
  -h, --help                        help for gcp
      --kubernetes-version string   The version of kubernetes to install. Example:
1.21.6
      --network string              the network to use when creating an image
      --overrides strings           a comma separated list of override YAML files
      --packer-manifest string      provide the path to a custom packer manifest
      --packer-on-error string      [advanced] set error strategy for packer.
strategies [cleanup, abort, run-cleanup-provisioner]
      --packer-path string          the location of the packer binary (default
"packer")
      --project-id string           the project id to use when storing created image
      --region string               the region in which to launch the instance
(default "us-west1")
      --work-dir string             path to custom work directory generated by the
generate command
```

9.2.10.2.7.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.2.10.2.7.4 SEE ALSO

- konvoy-image build (see page 1312) - build and provision images

9.2.10.3 konvoy-image completion

Generate the autocompletion script for the specified shell

9.2.10.3.1 Synopsis

Generate the autocompletion script for konvoy-image for the specified shell. See each sub-command's help for details on how to use the generated script.

9.2.10.3.2 Options

```
  -h, --help   help for completion
```

9.2.10.3.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.2.10.3.4 SEE ALSO

- konvoy-image completion bash (see page 1318) - Generate the autocompletion script for bash
- konvoy-image completion fish (see page 1319) - Generate the autocompletion script for fish
- konvoy-image completion powershell (see page 1320) - Generate the autocompletion script for powershell
- konvoy-image completion zsh (see page 1321) - Generate the autocompletion script for zsh

9.2.10.3.5 konvoy-image completion bash

Generate the autocompletion script for bash

9.2.10.3.5.1 Synopsis

Generate the autocompletion script for the bash shell.

This script depends on the 'bash-completion' package. If it is not installed already, you can install it via your OS's package manager.

To load completions in your current shell session:

```
source <(konvoy-image completion bash)
```

To load completions for every new session, execute once:

Linux:

```
konvoy-image completion bash > /etc/bash_completion.d/konvoy-image
```

macOS:

```
konvoy-image completion bash > $(brew --prefix)/etc/bash_completion.d/konvoy-image
```

You will need to start a new shell for this setup to take effect.

```
konvoy-image completion bash
```

9.2.10.3.5.2 Options

```
  -h, --help              help for bash
      --no-descriptions   disable completion descriptions
```

9.2.10.3.5.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.2.10.3.5.4 SEE ALSO

- konvoy-image completion (see page 1317) - Generate the autocompletion script for the specified shell

9.2.10.3.6 konvoy-image completion fish

Generate the autocompletion script for fish

9.2.10.3.6.1 Synopsis

Generate the autocompletion script for the fish shell.

To load completions in your current shell session:

```
konvoy-image completion fish | source
```

To load completions for every new session, execute once:

```
konvoy-image completion fish > ~/.config/fish/completions/konvoy-image.fish
```

You will need to start a new shell for this setup to take effect.

```
konvoy-image completion fish [flags]
```

9.2.10.3.6.2 Options

```
  -h, --help              help for fish
      --no-descriptions   disable completion descriptions
```

9.2.10.3.6.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.2.10.3.6.4 SEE ALSO

- konvoy-image completion (see page 1317) - Generate the autocompletion script for the specified shell

9.2.10.3.7 konvoy-image completion powershell

Generate the autocompletion script for powershell

9.2.10.3.7.1 Synopsis

Generate the autocompletion script for powershell.

To load completions in your current shell session:

```
konvoy-image completion powershell | Out-String | Invoke-Expression
```

To load completions for every new session, add the output of the above command to your powershell profile.

```
konvoy-image completion powershell [flags]
```

9.2.10.3.7.2 Options

```
  -h, --help              help for powershell
      --no-descriptions   disable completion descriptions
```

9.2.10.3.7.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.2.10.3.7.4 SEE ALSO

- konvoy-image completion (see page 1317) - Generate the autocompletion script for the specified shell

9.2.10.3.8 konvoy-image completion zsh

Generate the autocompletion script for zsh

9.2.10.3.8.1 Synopsis

Generate the autocompletion script for the zsh shell.

If shell completion is not already enabled in your environment you will need to enable it. You can execute the following once:

```
echo "autoload -U compinit; compinit" >> ~/.zshrc
```

To load completions in your current shell session:

```
source <(konvoy-image completion zsh)
```

To load completions for every new session, execute once:

Linux:

```
konvoy-image completion zsh > "${fpath[1]}/_konvoy-image"
```

macOS:

```
konvoy-image completion zsh > $(brew --prefix)/share/zsh/site-functions/_konvoy-image
```

You will need to start a new shell for this setup to take effect.

```
konvoy-image completion zsh [flags]
```

9.2.10.3.8.2 Options

```
  -h, --help              help for zsh
      --no-descriptions   disable completion descriptions
```

9.2.10.3.8.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.2.10.3.8.4 SEE ALSO

- konvoy-image completion (see page 1317) - Generate the autocompletion script for the specified shell

9.2.10.4 konvoy-image generate-docs

generate docs in path

```
konvoy-image generate-docs <PATH> [flags]
```

9.2.10.4.1 Examples

```
generate-docs /tmp/docs
```

9.2.10.4.2 Options

```
  -h, --help   help for generate-docs
```

9.2.10.4.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.2.10.5 konvoy-image generate

generate files relating to building images

9.2.10.5.1 Synopsis

Generate files relating to building images. Specifying AWS arguments is deprecated and will be removed in a future version. Use the `aws` subcommand instead.

```
konvoy-image generate <image.yaml> [flags]
```

9.2.10.5.2 Options

```
      --ami-groups stringArray           a list of AWS groups which are allowed use
the image, using 'all' result in a public image
      --ami-regions stringArray          a list of regions to publish amis
      --ami-users stringArray            a list AWS user accounts which are allowed
use the image
      --containerd-version string        the version of containerd to install
      --extra-vars strings               flag passed Ansible's extra-vars
  -h, --help                             help for generate
      --instance-type string             instance type used to build the AMI; the
type must be present in the region in which the AMI is built
      --kubernetes-version string        The version of kubernetes to install.
Example: 1.21.6
      --overrides strings                a comma separated list of override YAML
files
      --region string                    the region in which to build the AMI
```

```
      --source-ami string                the ID of the AMI to use as the source; must
be present in the region in which the AMI is built
      --source-ami-filter-name string    restricts the set of source AMIs to ones
whose Name matches filter
      --source-ami-filter-owner string   restricts the source AMI to ones with this
 owner ID
```

9.2.10.5.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.2.10.5.4 SEE ALSO

- konvoy-image generate aws (see page 1324) - generate files relating to building aws images
- konvoy-image generate azure (see page 1325) - generate files relating to building azure images

9.2.10.5.5 konvoy-image generate aws

generate files relating to building aws images

```
konvoy-image generate aws <image.yaml> [flags]
```

9.2.10.5.5.1 Examples

```
aws --region us-west-2 --source-ami=ami-12345abcdef images/ami/centos-79.yaml
```

9.2.10.5.5.2 Options

```
      --ami-groups stringArray           a list of AWS groups which are allowed use
the image, using 'all' result in a public image
      --ami-regions stringArray          a list of regions to publish amis
      --ami-users stringArray            a list AWS user accounts which are allowed
use the image
      --containerd-version string        the version of containerd to install
      --extra-vars strings               flag passed Ansible's extra-vars
  -h, --help                             help for aws
      --instance-type string             instance type used to build the AMI; the
type must be present in the region in which the AMI is built
```

```
      --kubernetes-version string        The version of kubernetes to install.
Example: 1.21.6
      --overrides strings                a comma separated list of override YAML
files
      --region string                    the region in which to build the AMI
      --source-ami string                the ID of the AMI to use as the source; must
be present in the region in which the AMI is built
      --source-ami-filter-name string    restricts the set of source AMIs to ones
whose Name matches filter
      --source-ami-filter-owner string   restricts the source AMI to ones with this
 owner ID
```

9.2.10.5.5.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.2.10.5.5.4 SEE ALSO

- konvoy-image generate (see page 1323) - generate files relating to building images

9.2.10.5.6 konvoy-image generate azure

generate files relating to building azure images

```
konvoy-image generate azure <image.yaml> [flags]
```

9.2.10.5.6.1 Examples

```
azure --location westus2 --subscription-id <sub_id> images/azure/centos-79.yaml
```

9.2.10.5.6.2 Options

```
      --client-id string                     the client id to use for the build
      --cloud-endpoint string                Azure cloud endpoint. Which can be one
of [Public USGovernment China] (default "Public")
      --containerd-version string            the version of containerd to install
      --extra-vars strings                   flag passed Ansible's extra-vars
```

```
      --gallery-image-locations stringArray   a list of locations to publish the
image (default [westus])
      --gallery-image-name string             the gallery image name to publish the
image to
      --gallery-image-offer string            the gallery image offer to set (default
 "dkp")
      --gallery-image-publisher string        the gallery image publisher to set
(default "dkp")
      --gallery-image-sku string              the gallery image sku to set
      --gallery-name string                   the gallery name to publish the image
in (default "dkp")
  -h, --help                                  help for azure
      --instance-type string                  the Instance Type to use for the build
(default "Standard_D2s_v3")
      --kubernetes-version string             The version of kubernetes to install.
Example: 1.21.6
      --location string                       the location in which to build the
image (default "westus2")
      --overrides strings                     a comma separated list of override YAML
files
      --resource-group string                 the resource group to create the image
in (default "dkp")
      --subscription-id string                the subscription id to use for the
build
      --tenant-id string                      the tenant id to use for the build
```

9.2.10.5.6.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.2.10.5.6.4 SEE ALSO

- konvoy-image generate (see page 1323) - generate files relating to building images

9.2.10.6 konvoy-image provision

provision to an inventory.yaml or hostname, note the comma at the end of the hostname

```
konvoy-image provision <inventory.yaml|hostname,> [flags]
```

9.2.10.6.1 Examples

```
provision --inventory-file inventory.yaml
```

9.2.10.6.2 Options

```
      --containerd-version string   the version of containerd to install
      --extra-vars strings          flag passed Ansible's extra-vars
  -h, --help                        help for provision
      --inventory-file string       an ansible inventory defining your infrastructure
      --kubernetes-version string   The version of kubernetes to install. Example:
1.21.6
      --overrides strings           a comma separated list of override YAML files
      --provider string             specify a provider if you wish to install
provider specific utilities
      --work-dir string             path to custom work directory generated by the
generate command
```

9.2.10.6.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.2.10.7 konvoy-image upload

Upload one of [artifacts]

9.2.10.7.1 Options

```
  -h, --help   help for upload
```

9.2.10.7.2 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
```

```
      --verbose   enable debug level logging (same as --v 5)
```

9.2.10.7.3 SEE ALSO

- konvoy-image upload artifacts (see page 1328) - upload offline artifacts to hosts defined in inventory-file

9.2.10.7.4 konvoy-image upload artifacts

upload offline artifacts to hosts defined in inventory-file

```
konvoy-image upload artifacts [flags]
```

9.2.10.7.4.1 Options

```
      --container-images-dir string   path to container images for install on remote
hosts.
      --containerd-bundle string      path to Containerd tar file for install on
remote hosts.
      --extra-vars strings            flag passed Ansible's extra-vars
  -h, --help                          help for artifacts
      --inventory-file string         an ansible inventory defining your
infrastructure (default "inventory.yaml")
      --nvidia-runfile string         path to nvidia runfile to place on remote
hosts.
      --os-packages-bundle string     path to os-packages tar file for install on
remote hosts.
      --overrides strings             a comma separated list of override YAML files
      --pip-packages-bundle string    path to pip-packages tar file for install on
remote hosts.
      --work-dir string               path to custom work directory generated by the
command
```

9.2.10.7.4.2 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.2.10.7.4.3 SEE ALSO

- konvoy-image upload (see page 1327) - Upload one of [artifacts]

9.2.10.8 konvoy-image validate

validate existing infrastructure

```
konvoy-image validate [flags]
```

9.2.10.8.1 Options

```
      --apiserver-port int      apiserver port (default 6443)
  -h, --help                    help for validate
      --inventory-file string   an ansible inventory defining your infrastructure
(default "inventory.yaml")
      --pod-subnet string       ip addresses used for the pod subnet (default
"192.168.0.0/16")
      --service-subnet string   ip addresses used for the service subnet (default
"10.96.0.0/12")
```

9.2.10.8.2 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.2.10.9 konvoy-image version

Version for konvoy-image

```
konvoy-image version [flags]
```

9.2.10.9.1 Options

```
  -h, --help   help for version
```

9.2.10.9.2 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.2.11 Use Override Files with Konvoy Image Builder

9.2.11.1 Learn how to use override files with Konvoy Imgage builder

Konvoy Image Builder (see page 1282) uses YAML[965] `override` files to configure specific attributes. These files provide information to override default values for certain parts of your image file. These `override` files can modify the version and parameters of the image description and the Ansible playbook.

There are 2 types of override files:

- Default override (see page 1330) files provided by Konvoy Image Builder for you to use for specific purposes.
- Custom override files (see page 1336) you create to use with Konvoy Image Builder.

9.2.11.2 Default Override Files

9.2.11.2.1 Learn how to use override files with DKP

DKP comes with default override files:

- FIPS Override Files (see page 1331)
- Offline Override File (see page 1334)
- Nvidia GPU Override File (see page 1334)
- Offline Nvidia GPU Override file (see page 1335)
- Oracle Redhat Linux Kernel Override File (see page 1336)

You can find these override files in the Konvoy Image Builder repo[966].

965 https://github.com/mesosphere/konvoy-image-builder/tree/7447074a6d910e71ad2e61fc4a12820d073ae5ae/images

966 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

9.2.11.2.2 FIPS Override Files

9.2.11.2.2.1 Cloud provisioners

Online FIPS Override File (Non-air-gapped)

Add the following FIPS Overrides file to your environment:

`--overrides overrides/fips.yaml`

```
---
k8s_image_registry: docker.io/mesosphere

fips:
  enabled: true

build_name_extra: -fips
kubernetes_build_metadata: fips.0
default_image_repo: hub.docker.io/mesosphere
kubernetes_rpm_repository_url: "https://packages.d2iq.com/konvoy/stable/linux/repos/
el/kubernetes-v{{ kubernetes_version }}-fips/x86_64"
docker_rpm_repository_url: "\
  https://containerd-fips.s3.us-east-2.amazonaws.com\
  /{{ ansible_distribution_major_version|int }}\
  /x86_64"
```

You can find all available Overrides files in the Konvoy Image Builder repo[967].

Offline FIPS Override File (Air-gapped)

Add the following FIPS Overrides file to your environment:

`--overrides overrides/offline-fips.yaml`

```
# fips os-packages
```

967 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

```
os_packages_local_bundle_file: "{{ playbook_dir }}/../artifacts/
{{ kubernetes_version }}_{{ ansible_distribution|lower }}
_{{ ansible_distribution_major_version }}_x86_64_fips.tar.gz"
containerd_local_bundle_file: "{{ playbook_dir }}/../artifacts/
{{ containerd_tar_file }}"
pip_packages_local_bundle_file: "{{ playbook_dir }}/../artifacts/pip-packages.tar.gz"
images_local_bundle_dir: "{{ playbook_dir}}/../artifacts/images"
```

You can find all available Overrides files in the Konvoy Image Builder repo[968].

9.2.11.2.2.2 Pre-provisioned environments

Online FIPS Override File (Pre-provisioned)

Add the following FIPS Overrides file to your environment:

1. If your pre-provisioned machines need to have a default Override file like FIPS, create a secret that includes the overrides in a file:

```
cat > fips.yaml << EOF
---
k8s_image_registry: docker.io/mesosphere

fips:
  enabled: true

build_name_extra: -fips
kubernetes_build_metadata: fips.0
default_image_repo: hub.docker.io/mesosphere
kubernetes_rpm_repository_url: "https://packages.d2iq.com/konvoy/stable/linux/
repos/el/kubernetes-v{{ kubernetes_version }}-fips/x86_64"
docker_rpm_repository_url: "\
  https://containerd-fips.s3.us-east-2.amazonaws.com\
  /{{ ansible_distribution_major_version|int }}\
  /x86_64"
EOF
```

2. Create the related secret by running the following command:

968 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

```
kubectl create secret generic $CLUSTER_NAME-user-overrides --from-
file=fips.yaml=fips.yaml
kubectl label secret $CLUSTER_NAME-user-overrides clusterctl.cluster.x-k8s.io/
move=
```

You can find all available Overrides files in the Konvoy Image Builder repo[969].

Offline FIPS Override File (Pre-provisioned Air-gapped)

Add the following FIPS Overrides file to your environment:

1. If your pre-provisioned machines need to have a default Override file like FIPS, create a secret that includes the overrides in a file:

```
cat > fips.yaml << EOF
# fips os-packages
os_packages_local_bundle_file: "{{ playbook_dir }}/../artifacts/
{{ kubernetes_version }}_{{ ansible_distribution|lower }}
_{{ ansible_distribution_major_version }}_x86_64_fips.tar.gz"
containerd_local_bundle_file: "{{ playbook_dir }}/../artifacts/
{{ containerd_tar_file }}"
pip_packages_local_bundle_file: "{{ playbook_dir }}/../artifacts/pip-
packages.tar.gz"
images_local_bundle_dir: "{{ playbook_dir}}/../artifacts/images"
EOF
```

2. Create the related secret by running the following command:

```
kubectl create secret generic $CLUSTER_NAME-user-overrides --from-
file=fips.yaml=fips.yaml
kubectl label secret $CLUSTER_NAME-user-overrides clusterctl.cluster.x-k8s.io/
move=
```

You can find all available Overrides files in the Konvoy Image Builder repo[970].

969 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides
970 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

9.2.11.2.3 Offline Override File

You can find these override files in the Konvoy Image Builder repo[971].

Add the following FIPS Overrides file to your environment:

`--overrides overrides/offline.yaml`

```
os_packages_local_bundle_file: "{{ playbook_dir }}/../artifacts/
{{ kubernetes_version }}_{{ ansible_distribution|lower }}
_{{ ansible_distribution_major_version }}_x86_64.tar.gz"
containerd_local_bundle_file: "{{ playbook_dir }}/../artifacts/
{{ containerd_tar_file }}"
pip_packages_local_bundle_file: "{{ playbook_dir }}/../artifacts/pip-packages.tar.gz"
images_local_bundle_dir: "{{ playbook_dir}}/../artifacts/images"
```

For Ubuntu 20.04, when Konvoy Image Builder runs, it will temporarily disable all defined Debian repositories by appending a `.disabled` suffix. At the end of installation, each repository will revert to its original name. In case of failures, the files will not be renamed back.

For GPU Offline Override File (see page 1335), you can use `nvidia-runfile` flag for GPU support if you have downloaded the runfile installer[972].

9.2.11.2.4 Nvidia GPU Override File

9.2.11.2.4.1 GPU Override File

This override file should be used to create images for use with DKP that can use GPU hardware. These override files can also be located in the Konvoy Image Builder repo[973].

`--overrides overrides/nvidia.yaml`

```
---
gpu:
  types:
    - nvidia
build_name_extra: "-nvidia"
```

971 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides
972 https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html#runfile
973 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

9.2.11.2.4.2 GPU Override File for Pre-provisioned environments

If you require your environment to consume GPU resources, add the following FIPS Overrides file to your environment:

1. If your pre-provisioned machines need to have a default Override file like FIPS, create a secret that includes the overrides in a file:

```
cat > nvidia.yaml << EOF
---
gpu:
  types:
    - nvidia
build_name_extra: "-nvidia"
EOF
```

2. Create the related secret by running the following command:

```
kubectl create secret generic $CLUSTER_NAME-user-overrides --from-
file=nvidia.yaml=nvidia.yaml
kubectl label secret $CLUSTER_NAME-user-overrides clusterctl.cluster.x-k8s.io/
move=
```

9.2.11.2.5 Offline Nvidia GPU Override file

This override file should be used to create images for use with DKP that can use GPU hardware. These override files can also be located in the Konvoy Image Builder repo[974].
`--overrides overrides/offline-nvidia.yaml`

```
# Use this file when building a machine image, not as a override secret for
 preprovisioned environments
nvidia_runfile_local_file: "{{ playbook_dir}}/../artifacts/
{{ nvidia_runfile_installer }}"
gpu:
  types:
    - nvidia

build_name_extra: "-nvidia"
```

974 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

9.2.11.2.5.1 GPU Override File for Air-gapped Pre-provisioned Environments

If you require your environment to consume GPU resources, add the following GPU Overrides file to your environment:

1. If your pre-provisioned machines need to have a default Override file like GPU, create a secret that includes the overrides in a file:

```
cat > nvidia.yaml << EOF
---
nvidia_runfile_local_file: "{{ playbook_dir}}/../artifacts/
{{ nvidia_runfile_installer }}"
gpu:
  types:
    - nvidia
build_name_extra: "-nvidia"
EOF
```

2. Create the related secret by running the following command:

```
kubectl create secret generic $CLUSTER_NAME-user-overrides --from-
file=nvidia.yaml=nvidia.yaml
kubectl label secret $CLUSTER_NAME-user-overrides clusterctl.cluster.x-k8s.io/
move=
```

9.2.11.2.6 Oracle Redhat Linux Kernel Override File

You can find these override files in the Konvoy Image Builder repo[975].

Add the following FIPS Overrides file to your environment:

`--overrides overrides/rhck.yaml`

```
---
oracle_kernel: RHCK
```

9.2.11.3 Custom Override Files

You can specify customization of your KIB images through the use of override files, which are used to specify alternate package libraries, Docker image repos, and other customizations. Some example custom override files are described in the following topics:

- Base Image Override Files (see page 1337)
- HTTP Proxy Override Files (see page 1341)

975 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

- Pre-Provisioned Override Files (see page 1341)
- Private Registry in Air-gapped Override (see page 1341)

9.2.11.3.1 Base Image Override Files

When using KIB to create an OS image that is compliant with DKP, the instructions on how to build and configure the image are included in the file located in `images/<builder-type>/<os-version.yaml`, as seen below:

```
./konvoy-image build images/<builder-type>/{os}-{version}.yaml
```

> In previous versions of KIB, only Azure and AWS providers were available, but now vSphere is an option as well.

Although there are several parameters specified by default in the Packer templates for each provider, it is possible to override the default values.

1. One option is to execute KIB with specific flags to override the values of the source AMI (`--source-ami`) , AMI region (`--ami-regions`), AWS EC2 instance type (`--aws-instance-type`), and so on. For a comprehensive list of these flags, please run:

```
./konvoy-image build --help
```

2. Another option is by creating a file with the parameters to be overridden and specify the `--overrides` flag as shown below:

```
./konvoy-image build images/<builder-type>/<os>-<version>.yaml --overrides
overrides.yaml
```

> While CLI flags can be used in combination with override files, CLI flags take priority over any override files.

9.2.11.3.1.1 Using Overrides flag:

AWS Example:

For example, when using the AWS Packer builder to override the above base image with another base image, create an override file and set the `source_ami` under the packer key. This overrides the image search and forces the use of the specified `source_ami`.

```
---
packer:
  source_ami: "ami-0123456789"
```

After creating the override file for our `source_ami`, we can pass our override file by using the `--overrides` flag when building our image:

```
./konvoy-image build aws images/ami/centos-7.yaml --overrides override-source-
ami.yaml
```

vSphere Example:

Create your vsphere overrides file `overrides/vcenter.yaml` and fill in the relevant details for your vSphere environment:

```
---
packer:
  vcenter_server: <FQDN of your vcenter>
  vsphere_username: <username for vcenter e.g. administrator@vsphere.local>
  vsphere_password: <password for vcenter>
  ssh_username: builder
  ssh_password: <password for your VMs builder user>
  linked_clone: false
  cluster: <vsphere cluster to use>
  datacenter: <vsphere datacenter to use>
  datastore: <vsphere datastore to use for templates>
  folder: <vsphere folder to store the template in>
  insecure_connection: "true"
  network: <a DHCP-enabled network in vcenter>
  resource_pool: <vsphere resource pool to use>
```

After creating the override file for our `source_ova`, we can pass our override file by using the `--overrides` flag when building our image:

```
konvoy-image build images/ova/ubuntu-2004.yaml \
  --overrides overrides/kube-version.yaml \
  --overrides overrides/vcenter.yaml
```

9.2.11.3.1.2 Override Credentials in Packer:

To abide to security practices, a user could set their own username and password while creating the base OS image and override the default credentials in KIB. To do this, create a file with the following content:

```
---
packer:
  ssh_username: "<USERNAME>"
  ssh_password: "<PASSWORD>"
```

For a complete list of the variables that can be modified for each Packer builder, users can refer to:

- AWS Packer template[976]
- Azure Packer template[977]
- GCP Packer template[978]
- vSphere Packer template[979]

AWS Example:

An AWS example would be the current base image description at `images/ami/centos-7.yaml` which is similar to the following:

```
---
# Example images/ami/centos-7.yaml
download_images: true
packer:
  ami_filter_name: "CentOS 7.9.2009 x86_64"
  ami_filter_owners: "125523088429"
  distribution: "CentOS"
  distribution_version: "7"
  source_ami: ""
  ssh_username: "centos"
  root_device_name: "/dev/sda1"
build_name: "centos-7"
packer_builder_type: "amazon"
python_path: ""
```

or

Azure Example:

An Azure example would be the current base image description at `images/azure/centos-79.yaml` which is similar to the following:

976 https://github.com/mesosphere/konvoy-image-builder/blob/main/pkg/packer/manifests/aws/packer.pkr.hcl
977 https://github.com/mesosphere/konvoy-image-builder/blob/main/pkg/packer/manifests/azure/packer.pkr.hcl
978 https://github.com/mesosphere/konvoy-image-builder/blob/main/pkg/packer/manifests/gcp/packer.pkr.hcl
979 https://github.com/mesosphere/konvoy-image-builder/blob/main/pkg/packer/manifests/vsphere/packer.pkr.hcl

```
---
# Example images/azure/centos-79.yaml
download_images: true
packer:
  distribution: "centos"  #offer
  distribution_version: "7_9-gen2"  #SKU
  image_publisher: "openlogic"
  image_version: "latest"
  ssh_username: "centos"

build_name: "centos-7"
packer_builder_type: "azure"
python_path: ""
```

or

vSphere Example:

A vSphere example would be the current base image description at `images/vsphere/rhel-79.yaml` which is similar to the following:

```
---
download_images: true
build_name: "rhel-79"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
cluster: "zone1"
  datacenter: "dc1"
  datastore: "esxi-06-disk1"
  folder: "cluster-api"
  insecure_connection: "false"
  network: "Airgapped"
  resource_pool: "Users"
  template: "base-rhel-7"
  vsphere_guest_os_type: "rhel7_64Guest"
  guest_os_type: "rhel7-64"
  #goss params
  distribution: "RHEL"
  distribution_version: "7.9"
```

See Supported Operating Systems (see page 62) for details.

9.2.11.3.2 HTTP Proxy Override Files

An HTTP proxy configuration can be used when creating your image. The Ansible playbooks will create `systemd` drop-in files for `containerd` and `kubelet` to configure the `http_proxy`, `http_proxy`, and `no_proxy` environment variables for the service from the file `/etc/konvoy_http_proxy.conf`.

To configure a proxy for use during image creation, create a new override file and specify the following:

```
# Example override-proxy.yaml
---
http_proxy: http://example.org:8080
https_proxy: http://example.org:8081
no_proxy: example.org,example.com,example.net
```

These values are only used for the image creation. After the image is created, the Ansible playbooks remove the `/etc/konvoy_http_proxy.conf` file.

The `dkp` command can be used to configure the `KubeadmConfigTemplate` object to create this file on bootup of the image with values supplied during the `dkp` invocation. This enables using different proxy settings for image creation and runtime.

9.2.11.3.3 Pre-Provisioned Override Files

Pre-provisioned environments require certain override files to work properly. Those override files must also have a secret secret that includes all of the overrides you wish to provide in one file. For example, if you wish to provide an override with Docker credentials and a different source for EPEL on a CentOS7 machine, you can create a file like this:

```
image_registries_with_auth:
- host: "registry-1.docker.io"
  username: "my-user"
  password: "my-password"
  auth: ""
  identityToken: ""

epel_centos_7_rpm: https://my-rpm-repostory.org/epel/epel-release-latest-7.noarch.rpm
```

9.2.11.3.4 Private Registry in Air-gapped Override

To create an Air-gapped registry override for Docker hub, run the following command:

```
[plugins."io.containerd.grpc.v1.cri".registry.mirrors."docker.io"]
 endpoint = ["https://my-local-registry.local/v2/harbor-registry","https://
registry-1.docker.io"]
```

Tell the override how to create a wildcard mirror with this command:

```
[plugins."io.containerd.grpc.v1.cri".registry.mirrors."*"]
 endpoint = ["https://my-local-registry.local/v2/harbor-registry"]
```

9.3 FIPS 140-2 Compliance

Understand FIPS-140 Operating Mode and Requirements

Developed by a working group of government, industry operators, and vendors, the Federal Information Processing Standard (FIPS), FIPS-140 defines security requirements for cryptographic modules. FIPS defines what cryptographic cyphers can be used. Kubernetes uses encryption by default between various components and FIPS support ensures that the ciphers used for those communications meet those standards. The standard provides for a wide spectrum of data sensitivity, transaction values, and a diversity of application environment security situations. The standard specifies four security levels for each of eleven requirement areas. Each successive level offers increased security.

NIST introduced FIPS 140-2 validation, by accredited third party laboratories, as a formal, rigorous process to protect sensitive digitally-stored information not under Federal security classifications.

9.3.1 FIPS Support in DKP

DKP supports provisioning a FIPS-enabled Kubernetes control plane. Core Kubernetes components are compiled using a version of Go, called goboring, which uses a FIPS-certified cryptographic module[980] for all cryptographic functions.

Before provisioning DKP, you will need to follow your OS vendor's instructions to ensure that your OS, or OS images, are prepared for operating in FIPS mode[981].

> You cannot apply FIPS-mode to an existing cluster, you must create a new cluster with FIPS enabled. Similarly, a FIPS-mode cluster must remain a FIPS-mode cluster; you cannot change the cluster's FIPS status after you create it.

9.3.2 Infrastructure Requirements for FIPS-140-2 Mode

To ensure proper operations in FIPS mode, be sure that your environment meets these requirements:

FIPS 140 Mode Performance Impact (see page 1348)

980 https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp3702.pdf

981 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/chap-federal_standards_and_regulations

9.3.2.1 Supported Operating Systems

Supported Operating Systems for FIPS mode are Red Hat Enterprise Linux and CentOS. See the Supported Operating Systems (see page 62) for details on the tested and supported versions.

9.3.3 Deploying a Cluster in FIPS mode

In order to create a cluster in FIPS mode, we must inform the bootstrap controllers of the appropriate image repository and version tags of the official D2iQ FIPS builds of Kubernetes.

9.3.3.1 Supported FIPS Builds

Component	Repository	Version
Kubernetes	docker.io/mesosphere[982]	v1.25.4+fips.0
etcd	docker.io/mesosphere[983]	3.5.5+fips.0

AWS Example:

When creating a cluster, use the following command line options:

- `--ami <fips enabled AMI created in the previous step>` (AWS only)
- `--kubernetes-version <version>+fips.<build>`
- `--etcd-version <version>+fips.<build>`
- `--kubernetes-image-repository docker.io/mesosphere`
- `--etcd-image-repository docker.io/mesosphere`

For example:

```
dkp create cluster aws --cluster-name myFipsCluster \
--ami=ami-03dcaa75d45aca36f \
--kubernetes-version=v1.25.4+fips.0 \
--kubernetes-image-repository=docker.io/mesosphere \
--etcd-image-repository=docker.io/mesosphere \
--etcd-version=3.5.5+fips.0
```

vSphere Example:

982 https://hub.docker.com/layers/mesosphere/kube-apiserver/v1.25.4_fips.0/images/sha256-cacb721f96ec8764d187ad3c21557630f5f4d96fea4a03ca35d0388b509d970d?context=explore

983 https://hub.docker.com/layers/mesosphere/etcd/v3.5.5_fips.0/images/sha256-ba07521fa1875efa0cc623533eb5557e40a6f8fdc8a71a86751a649ce53de1d0?context=explore

```
dkp create cluster vsphere \
  --cluster-name ${CLUSTER_NAME} \
  --network <NETWORK_NAME> \
  --control-plane-endpoint-host <xxx.yyy.zzz.000> \
  --data-center <DATACENTER_NAME> \
  --data-store <DATASTORE_NAME> \
  --folder <FOLDER_NAME> \
  --server <VCENTER_API_SERVER_URL> \
  --ssh-public-key-file <SSH_PUBLIC_KEY_FILE> \
  --resource-pool <RESOURE_POOL_NAME> \
  --vm-template <TEMPLATE_NAME> \
  --self-managed \
  --kubernetes-version=v1.25.4+fips.0 \
  --kubernetes-image-repository=docker.io/mesosphere \
  --etcd-image-repository=docker.io/mesosphere --etcd-version=3.5.5+fips.0
```

9.3.4 Create FIPS 140 Images: Non-air-gapped Environment

9.3.4.1 Use Konvoy Image Builder to create images with FIPS-compliant binaries

9.3.4.2 Non-air-gapped Environment Create FIPS-140 images

KIB can produce images containing FIPS-140 compliant binaries. Use the `fips.yaml` override file (see page 1331) provided with the image bundles.

> You can also find these override files in the Konvoy Image Builder repo[984].

9.3.4.2.1 Examples:

> The below snippets will create images with FIPS-compliant Kubernetes components. If you need the underlying OS to be FIPS-compliant, then you will need to provide the specific FIPS-compliant OS image, using the `--source-ami` flag for AWS.

984 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

- A non-air-gapped environment example of override file use is the command below, which produces a FIPS-compliant image on RHEL 8.4 for AWS:
  Replace `ami` with your infrastructure provisioner

```
konvoy-image build --overrides overrides/fips.yaml images/ami/rhel-84.yaml
```

- vSphere FIPS-complaint using `image.yaml` created during VM Template (see page 1307) configuration:

```
konvoy-image build --overrides overrides/fips.yaml images/ova/<image.yaml>
```

Here is a list of FIPS Override Files (see page 1331).

9.3.5 Create FIPS 140 Images: Air-gapped Environment

KIB can produce images containing FIPS-140 compliant binaries. Use the `fips.yaml` override file[985] provided with the image bundles.

> You can also find these override files in the Konvoy Image Builder repo[986].

9.3.5.1 Examples:

> The below snippets will create images with FIPS-compliant Kubernetes components. If you need the underlying OS to be FIPS-compliant, then you will need to provide the specific FIPS-compliant OS image, using the `--source-ami` flag for AWS.

- An air-gapped environment example of override file use is the command below which produces an AWS FIPS-compliant image on RHEL 8.4:

```
konvoy-image build --overrides offline-fips.yaml --overrides overrides/fips.yaml
images/ami/rhel-84.yaml
```

- vSphere FIPS-compliant air-gapped environment example:

985 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/46039044
986 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

```
konvoy-image build --overrides offline-fips.yaml --overrides overrides/fips.yaml
images/ova/<image.yaml>
```

9.3.5.2 Pre-provisioned FIPS Infrastructure

If you are targeting a Pre-provisioned Installs (see page 125), you can create a FIPS-compliant cluster by doing the following:

1. Create a Pre-provisioned: Bootstrap Cluster (see page 1097)
2. Create a secret on the bootstrap cluster with the contents from `fips.yaml` override file[987] and any other user overrides you wish to provide

```
kubectl create secret generic $CLUSTER_NAME-fips-overrides --from-
file=overrides.yaml=overrides.yaml
kubectl label secret $CLUSTER_NAME-fips-overrides clusterctl.cluster.x-k8s.io/move=
```

Here is a list of FIPS Override Files (see page 1331).

9.3.6 Validate FIPS in Cluster

You can use the FIPS validation tool to verify that specific components and services are FIPS-compliant. The tool checks the components by comparing their file signatures against ones stored in a signed signature file, and by checking that services are using the certified algorithms.

9.3.6.1 Run FIPS validation

To verify the cluster is FIPS compliant, run `dkp check cluster fips`. This command reads from the signature files embedded in the `dkp` executable in order to validate that specific components and services are FIPS-compliant. Run the command:

```
dkp check cluster fips
```

Upon successful completion, the command's output displays details about the deployment in JSON format. If validation fails, the output will say which components fail and a list of the nodes that failed validation will return.

The full command usage and flags include:

```
dkp check cluster fips [flags]
```

987 https://github.com/mesosphere/konvoy-image-builder/blob/main/overrides/fips.yaml

Flags:

```
  -h, --help                          Help for fips
      --kubeconfig string             Path to the kubeconfig file for the fips
cluster. If unspecified, default discovery rules apply.
  -n, --namespace string              If present, the namespace scope for this CLI
request. (default "default")
      --output-configmap string       ConfigMap to store result of the fips check.
(default "check-cluster-fips-output") (DEPRECATED: This flag will be removed in a
future release.)
      --signature-configmap string    ConfigMap with fips signature data to verify.
      --signature-file string         File containing fips signature data.
      --timeout duration              The length of time to wait before giving up.
Zero means wait forever (e.g. 1s, 2m, 3h). (default 10m0s)
```

9.3.6.1.1 Run FIPS validation with custom signature file

To validate FIPS-mode operation with the a custom signature file, you can use the `signature-file` flag, as in the following command. You also need to use the `signature-configmap` flag to set the name of the ConfigMap used to store your custom signature file.

```
dkp check cluster fips \
 --signature-file custom.json.asc \
 --signature-configmap custom-signature-file
```

9.3.6.1.2 Run FIPS validation with existing ConfigMap

If you already have a signature file stored in a ConfigMap, you can omit the `signature-file` flag, as in the following command:

```
dkp check cluster fips \
 --signature-configmap prod-rhel8-fips-signatures
```

9.3.6.2 Download Signature Files

The following signature files are already embedded in the `dkp` executable. They are provided for reference. You do not need to download them to run the FIPS check.

9.3.6.2.1 DKP Version 2.5.x

Operating System version	Kubernetes version	containerd version	Signature File URL
CentOS 7.9	v1.25.4	1.6.17	CentOS 7.9[988]
Oracle 7.9	v1.25.4	1.6.17	Oracle 7.9[989]
RHEL 7.9	v1.25.4	1.6.17	RHEL 7.9[990]
RHEL 8.4	v1.25.4	1.6.17	RHEL 8.4[991]
RHEL 8.6	v1.25.4	1.6.17	RHEL 8.6[992]

9.3.7 FIPS 140 Mode Performance Impact

9.3.7.1 Understand the performance impact from operating your cluster in FIPS 140 mode

The Go language cryptographic module, Goboring, relies on CGO's foreign function interface to call C-language functions exposed by the cryptographic module. Each call into the C library starts with a base overhead of 200ns.

One benchmark[993] finds that the time to encrypt a single AES-128 block increased from 13ns to 209ns over the internal golang implementation. The preferred mode of D2iQ's FIPS module is `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`.

The aggregate impact on a stable control plane seems to be an increase of around 10% CPU utilization over default operation. Workloads that do not directly interact with the control plane are not affected.

988 https://downloads.d2iq.com/dkp/fips/v2.5.2/manifest-centos-79.json.asc
989 https://downloads.d2iq.com/dkp/fips/v2.5.2/manifest-oracle-79.json.asc
990 https://downloads.d2iq.com/dkp/fips/v2.5.2/manifest-rhel-79.json.asc
991 https://downloads.d2iq.com/dkp/fips/v2.5.2/manifest-rhel-84.json.asc
992 https://downloads.d2iq.com/dkp/fips/v2.5.2/manifest-rhel-86.json.asc
993 https://github.com/golang/go/issues/21525

9.4 Local Registry Tools

Kubernetes does not natively provide a registry for hosting container images which contain the applications you want to deploy on Kubernetes. Instead, Kubernetes expects you to use an external solution for storing and sharing container images.

There are a variety of Kubernetes registry options out there that are compatible with DKP. The list below refers to the local registry tools required if running an air-gapped environment.

9.4.1 Air-Gapped Registry Prerequisites

DKP in an air-gapped environment requires a local container registry of trusted images to enable production level Kubernetes cluster management. In an environment with access to the internet, you retrieve artifacts from specialized repositories dedicated to them such as Docker images contained in DockerHub and Helm Charts that come from a dedicated Helm Chart repository. However, in an air-gapped environment, you need local repositories to store Helm charts, Docker images and other artifacts. Tools such as jFrog, Harbor and Nexus handle multiple types of artifacts in one local repository.

If you want to use images from this local registry to deploy applications inside your Kubernetes cluster, you'll need to set up a secret for a private registry. The secret contains your login data, which Kubernetes needs to connect to your private repository.

9.4.1.1 JFrog Artifactory

JFrog can function as a container registry, as well as an automated management tool for binaries and artifacts of all types. If you use **JFrog Artifactory**[994] or **JFrog Container Registry**, you must update to a new version of the software. Any build newer than version **7.11** will work, as we have confirmed that older versions are not compatible.

9.4.1.2 Nexus Registry

Nexus Repository[995] is a package registry for all of your Docker images and Helm Chart repositories and supports Proxy, Hosted, and Group repositories. It can be used a single registry for all your Kubernetes deployments.

9.4.1.3 Harbor Registry

Install Harbor[996] and configure any https access required as well as the system level parameters in the `harbor.yml` file. Then run the installer script. If you are upgrading from a previous version of Harbor, you update the configuration file and migrate your data to fit the database schema of the later version. For information about upgrading, see Upgrading Harbor[997]. Any newer version than **Harbor Registry v2.1.1**-5f52168e will support OCI images.

994 https://jfrog.com/artifactory/
995 https://www.nexusregistry.com/info/
996 https://goharbor.io/docs/2.0.0/install-config/download-installer/
997 https://goharbor.io/docs/2.0.0/administration/upgrade/

9.4.1.4 Bastion Host

If you have not set up a Bastion Host (see page 1162) yet, refer to that section of the Documentation.

9.4.2 Next Topic:

(2.5) Air-gapped Seed Docker Registry (see page 1350)

9.4.3 Use an Alternative Mirror

9.4.3.1 Use an Alternative Mirror

To apply private registry configurations during the create operation, add the appropriate flags to the `create cluster` command:

Registry configuration	Flag
CA certificate chain to use while communicating with the registry mirror using TLS	`--registry-mirror-cacert file`
URL of a container registry to use as a mirror in the cluster	`--registry-mirror-url string`

This is useful when using an internal registry and when Internet access is not available (air-gapped installations). Only use one image registry per cluster.

When the cluster is up and running, you can deploy and test workloads.

9.4.3.2 Alternative Mirror Example:

```
dkp create cluster preprovisioned \
    --cluster-name ${CLUSTER_NAME} \
    --registry-mirror-cacert /tmp/registry.pem \
    --registry-mirror-url https://registry.example.com
```

9.5 Air-gapped Seed the Registry

Before creating a Kubernetes cluster, you need the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1162) and

either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1349) page for more information.

1. Assuming you have downloaded (see page 71) `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz` , extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz && cd dkp-v2.5.0
   ```

2. Set an environment variable with your registry address with this command:

   ```
   export REGISTRY_ADDRESS=<registry-address>:<registry-port>
   export REGISTRY_USERNAME=<username>
   export REGISTRY_PASSWORD=<password>
   ```

3. Execute the following command to load the air-gapped image bundle into your private registry:

   ```
   dkp push image-bundle --image-bundle ./container-images/konvoy-image-bundle-v2.5.0.tar --to-registry $REGISTRY_ADDRESS --to-registry-username $REGISTRY_USERNAME --to-registry-password $REGISTRY_PASSWORD
   ```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

You are now ready to create an air-gapped bootstrap cluster for a custom cluster for your infrastructure provider (see page 919), or create an air-gapped cluster from the Day 1 - Basic Installs (see page 124) section for your provider.

9.6 Configure the Control Plane

9.6.1 Prerequisites

Before you begin, make sure you have created your cluster using a bootstrap cluster from the respective Infrastructure Providers (see page 919) section.

Users can make modifications to the `KubeadmControlplane` cluster-api object to configure different kubelet options. Please see the following guide if you wish to configure your control plane beyond the existing options that are available from flags.

9.6.1.1 Modifying Audit Logs

In order to modify control plane option, get the appropriate `cluster-api` objects that describe the cluster by running the following command:

> The following example uses AWS, but can be used for `gcp`, `azure`, `preprovisioned`, and `vsphere` clusters.

```
dkp create cluster aws -c {MY_CLUSTER_NAME} -o yaml --dry-run >>
{MY_CLUSTER_NAME}.yaml
```

When you open `{MY_CLUSTER_NAME}.yaml` with your favorite text editor, look for the `KubeadmControlPlane` object for your cluster. Below is an example object:

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: my-cluster-control-plane
  namespace: default
spec:
  kubeadmConfigSpec:
    clusterConfiguration:
      apiServer:
        extraArgs:
          audit-log-maxage: "30"
          audit-log-maxbackup: "10"
          audit-log-maxsize: "100"
          audit-log-path: /var/log/audit/kube-apiserver-audit.log
          audit-policy-file: /etc/kubernetes/audit-policy/apiserver-audit-policy.yaml
          cloud-provider: aws
          encryption-provider-config: /etc/kubernetes/pki/encryption-config.yaml
        extraVolumes:
        - hostPath: /etc/kubernetes/audit-policy/
          mountPath: /etc/kubernetes/audit-policy/
          name: audit-policy
        - hostPath: /var/log/kubernetes/audit
          mountPath: /var/log/audit/
          name: audit-logs
```

```
      controllerManager:
        extraArgs:
          cloud-provider: aws
          configure-cloud-routes: "false"
      dns: {}
      etcd:
        local:
          imageTag: 3.5.5
      networking: {}
      scheduler: {}
    files:
    - content: |
        # Taken from https://github.com/kubernetes/kubernetes/blob/master/cluster/
gce/gci/configure-helper.sh
        # Recommended in Kubernetes docs
        apiVersion: audit.k8s.io/v1
        kind: Policy
        rules:
          # The following requests were manually identified as high-volume and low-
risk,
          # so drop them.
          - level: None
            users: ["system:kube-proxy"]
            verbs: ["watch"]
            resources:
              - group: "" # core
                resources: ["endpoints", "services", "services/status"]
          - level: None
            # Ingress controller reads 'configmaps/ingress-uid' through the unsecured
port.
            # TODO(#46983): Change this to the ingress controller service account.
            users: ["system:unsecured"]
            namespaces: ["kube-system"]
            verbs: ["get"]
            resources:
              - group: "" # core
                resources: ["configmaps"]
          - level: None
            users: ["kubelet"] # legacy kubelet identity
            verbs: ["get"]
            resources:
              - group: "" # core
                resources: ["nodes", "nodes/status"]
          - level: None
            userGroups: ["system:nodes"]
            verbs: ["get"]
            resources:
              - group: "" # core
                resources: ["nodes", "nodes/status"]
          - level: None
            users:
              - system:kube-controller-manager
```

```
            - system:kube-scheduler
            - system:serviceaccount:kube-system:endpoint-controller
          verbs: ["get", "update"]
          namespaces: ["kube-system"]
          resources:
            - group: "" # core
              resources: ["endpoints"]
        - level: None
          users: ["system:apiserver"]
          verbs: ["get"]
          resources:
            - group: "" # core
              resources: ["namespaces", "namespaces/status", "namespaces/finalize"]
        - level: None
          users: ["cluster-autoscaler"]
          verbs: ["get", "update"]
          namespaces: ["kube-system"]
          resources:
            - group: "" # core
              resources: ["configmaps", "endpoints"]
        # Don't log HPA fetching metrics.
        - level: None
          users:
            - system:kube-controller-manager
          verbs: ["get", "list"]
          resources:
            - group: "metrics.k8s.io"
        # Don't log these read-only URLs.
        - level: None
          nonResourceURLs:
            - /healthz*
            - /version
            - /swagger*
        # Don't log events requests.
        - level: None
          resources:
            - group: "" # core
              resources: ["events"]
        # node and pod status calls from nodes are high-volume and can be large,
don't log responses for expected updates from nodes
        - level: Request
          users: ["kubelet", "system:node-problem-detector",
"system:serviceaccount:kube-system:node-problem-detector"]
          verbs: ["update","patch"]
          resources:
            - group: "" # core
              resources: ["nodes/status", "pods/status"]
          omitStages:
            - "RequestReceived"
        - level: Request
          userGroups: ["system:nodes"]
          verbs: ["update","patch"]
```

```
            resources:
              - group: "" # core
                resources: ["nodes/status", "pods/status"]
            omitStages:
              - "RequestReceived"
          # deletecollection calls can be large, don't log responses for expected
namespace deletions
          - level: Request
            users: ["system:serviceaccount:kube-system:namespace-controller"]
            verbs: ["deletecollection"]
            omitStages:
              - "RequestReceived"
          # Secrets, ConfigMaps, and TokenReviews can contain sensitive & binary
data,
          # so only log at the Metadata level.
          - level: Metadata
            resources:
              - group: "" # core
                resources: ["secrets", "configmaps"]
              - group: authentication.k8s.io
                resources: ["tokenreviews"]
            omitStages:
              - "RequestReceived"
          # Get responses can be large; skip them.
          - level: Request
            verbs: ["get", "list", "watch"]
            resources:
              - group: "" # core
              - group: "admissionregistration.k8s.io"
              - group: "apiextensions.k8s.io"
              - group: "apiregistration.k8s.io"
              - group: "apps"
              - group: "authentication.k8s.io"
              - group: "authorization.k8s.io"
              - group: "autoscaling"
              - group: "batch"
              - group: "certificates.k8s.io"
              - group: "extensions"
              - group: "metrics.k8s.io"
              - group: "networking.k8s.io"
              - group: "node.k8s.io"
              - group: "policy"
              - group: "rbac.authorization.k8s.io"
              - group: "scheduling.k8s.io"
              - group: "settings.k8s.io"
              - group: "storage.k8s.io"
            omitStages:
              - "RequestReceived"
          # Default level for known APIs
          - level: RequestResponse
            resources:
              - group: "" # core
```

```
              - group: "admissionregistration.k8s.io"
              - group: "apiextensions.k8s.io"
              - group: "apiregistration.k8s.io"
              - group: "apps"
              - group: "authentication.k8s.io"
              - group: "authorization.k8s.io"
              - group: "autoscaling"
              - group: "batch"
              - group: "certificates.k8s.io"
              - group: "extensions"
              - group: "metrics.k8s.io"
              - group: "networking.k8s.io"
              - group: "node.k8s.io"
              - group: "policy"
              - group: "rbac.authorization.k8s.io"
              - group: "scheduling.k8s.io"
              - group: "settings.k8s.io"
              - group: "storage.k8s.io"
            omitStages:
              - "RequestReceived"
          # Default level for all other requests.
          - level: Metadata
            omitStages:
              - "RequestReceived"
      path: /etc/kubernetes/audit-policy/apiserver-audit-policy.yaml
      permissions: "0600"
    - content: |
        #!/bin/bash
        # CAPI does not expose an API to modify KubeProxyConfiguration
        # this is a workaround to use a script with preKubeadmCommand to modify the
kubeadm config files
        # https://github.com/kubernetes-sigs/cluster-api/issues/4512
        for i in $(ls /run/kubeadm/ | grep 'kubeadm.yaml\|kubeadm-join-config.yaml');
do
          cat <<EOF>> "/run/kubeadm//$i"
        ---
        kind: KubeProxyConfiguration
        apiVersion: kubeproxy.config.k8s.io/v1alpha1
        metricsBindAddress: "0.0.0.0:10249"
        EOF
        done
      path: /run/kubeadm/konvoy-set-kube-proxy-configuration.sh
      permissions: "0700"
    - content: |
        [metrics]
          address = "0.0.0.0:1338"
          grpc_histogram = false
      path: /etc/containerd/conf.d/konvoy-metrics.toml
      permissions: "0644"
    - content: |
        #!/bin/bash
        systemctl restart containerd
```

```
      SECONDS=0
      until crictl info
      do
        if (( SECONDS > 60 ))
        then
           echo "Containerd is not running. Giving up..."
           exit 1
        fi
        echo "Containerd is not running yet. Waiting..."
        sleep 5
      done
    path: /run/konvoy/restart-containerd-and-wait.sh
    permissions: "0700"
  - contentFrom:
      secret:
        key: value
        name: my-cluster-etcd-encryption-config
    owner: root:root
    path: /etc/kubernetes/pki/encryption-config.yaml
    permissions: "0640"
  format: cloud-config
  initConfiguration:
    localAPIEndpoint: {}
    nodeRegistration:
      kubeletExtraArgs:
        cloud-provider: aws
      name: '{{ ds.meta_data.local_hostname }}'
  joinConfiguration:
    discovery: {}
    nodeRegistration:
      kubeletExtraArgs:
        cloud-provider: aws
      name: '{{ ds.meta_data.local_hostname }}'
  preKubeadmCommands:
  - systemctl daemon-reload
  - /run/konvoy/restart-containerd-and-wait.sh
  - /run/kubeadm/konvoy-set-kube-proxy-configuration.sh
machineTemplate:
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: AWSMachineTemplate
    name: my-cluster-control-plane
    namespace: default
  metadata: {}
replicas: 3
rolloutStrategy:
  rollingUpdate:
    maxSurge: 1
  type: RollingUpdate
version: v1.25.4
```

If you use the above example as-is, ensure you update your Kubeneters version number on the final line by replacing the x.

Now a user can configure the fields below for the log backend. The log backend will write audit events to a file in JSON[998] format. You can configure the log audit backend using the `kube-apiserver` flags shown below:

```
audit-log-maxage
audit-log-maxbackup
audit-log-maxsize
audit-log-path
```

See upstream documentation[999]for more information.

After modifying the values appropriately, you can create the cluster by running the command below:

```
kubectl create -f {MY_CLUSTER_NAME}.yaml
```

Once the cluster is created, users can get the corresponding kubeconfig for the cluster by running the following command:

```
dkp get kubeconfig -c {MY_CLUSTER_NAME} >> {MY_CLUSTER_NAME}.conf
```

9.6.1.2 Viewing the Audit Logs

Fluent Bit is disabled on the management cluster by default, to view the audit logs run the command below:

```
dkp diagnose --kubeconfig={MY_CLUSTER_NAME}.conf
```

A file similar to `support-bundle-2022-08-15T02_28_48.tar.gz` will be created. Untar the file using a command similar to the example below:

998 https://jsonlines.org/
999 https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/#log-backend

```
tar -xzf support-bundle-2022-08-15T02_28_48.tar.gz
```

Navigate to the node-diagnostics sub directory from the extracted file with a command like the one shown below:

```
cd support-bundle-2022-08-15T02_28_48/node-diagnostics
```

Finally, to find the audit logs run the following command:

```
$ find . -type f | grep audit.log
./ip-10-0-142-117.us-west-2.compute.internal/data/kube_apiserver_audit.log
./ip-10-0-148-139.us-west-2.compute.internal/data/kube_apiserver_audit.log
./ip-10-0-128-181.us-west-2.compute.internal/data/kube_apiserver_audit.log
```

See other related pages below:

Fluent bit (see page 818)

9.7 Update Cluster Nodepools

Upgrading a node pool involves draining the existing nodes in the node pool and replacing them with new nodes. In order to ensure minimum downtime and maintain high availability of the critical application workloads during the upgrade process, we recommend deploying Pod Disruption Budget (Disruptions[1000]) for your critical applications.

The Pod Disruption Budget will prevent any impact on critical applications as a result of misconfiguration or failures during the upgrade process.

9.7.1 Prerequisites:

- Deploy Pod Disruption Budget[1001] (PDB)
- Konvoy Image Builder (see page 1282) (KIB)

9.7.2 Steps:

1. Deploy Pod Disruption Budget for your critical applications. If your application can tolerate only one replica to be unavailable at a time, then you can set Pod disruption budget as shown in the following example. The example below is for NVIDIA GPU node pools, but the process is the same for all node pools.

1000 https://kubernetes.io/docs/concepts/workloads/pods/disruptions/
1001 https://kubernetes.io/docs/concepts/workloads/pods/disruptions/

Repeat this step for each additional node pool.

Create the file: `pod-disruption-budget-nvidia.yaml`

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: nvidia-critical-app
spec:
  maxUnavailable: 1
  selector:
    matchLabels:
      app: nvidia-critical-app
```

Apply the YAML file above using the following command:

```
kubectl create -f pod-disruption-budget-nvidia.yaml
```

2. Prepare OS image for your node pool using Konvoy Image Builder (see page 1282).

9.7.3 Related Topics:

Upgrade Konvoy (see page 1373)

9.8 Verify your Cluster and DKP Installation

Check DKP components to verify the status of your cluster

This section contains information on how to verify a DKP installation.

9.8.1 Check the Cluster Infrastructure and Nodes

DKP ships with some default diagnosis tools to check your cluster, such as the `describe` command. You can use those tools to validate your installation.

If you have not done so already, set the environment variable for your cluster name, substituting `dkp-example` with the name of your cluster:

```
export CLUSTER_NAME=dkp-example
```

Then, run this command to check the health of the cluster infrastructure:

```
dkp describe cluster --cluster-name=${CLUSTER_NAME}
```

For more details on the `dkp describe cluster` command, refer to dkp describe cluster (see page 1488).

A healthy cluster returns an output similar to:

```
NAME                                                          READY  SEVERITY
REASON  SINCE  MESSAGE
Cluster/dkp-example                                           True
121m
├──ClusterInfrastructure - AWSCluster/dkp-example              True
121m
├──ControlPlane - KubeadmControlPlane/dkp-example-control-plane  True
121m
│  ├──Machine/dkp-example-control-plane-h52t6                  True
121m
│  ├──Machine/dkp-example-control-plane-knrrh                  True
121m
│  └──Machine/dkp-example-control-plane-zmjjx                  True
121m
└──Workers
   └──MachineDeployment/dkp-example-md-0                       True
121m
     ├──Machine/dkp-example-md-0-88488cb74-2vxjq               True
121m
     ├──Machine/dkp-example-md-0-88488cb74-84xsd               True
121m
     ├──Machine/dkp-example-md-0-88488cb74-9xmc6               True
121m
     └──Machine/dkp-example-md-0-88488cb74-mjf6s               True
121m
```

Use this `kubectl` command to check to see if all cluster nodes are ready:

```
kubectl get nodes
```

The output should look similar to this, with all statuses set to `Ready`

```
NAME                                          STATUS   ROLES                  AGE
VERSION
ip-10-0-112-116.us-west-2.compute.internal    Ready    <none>                 135m
v1.21.6
ip-10-0-122-142.us-west-2.compute.internal    Ready    <none>                 135m
v1.21.6
ip-10-0-186-214.us-west-2.compute.internal    Ready    control-plane,master   133m
v1.21.6
ip-10-0-231-82.us-west-2.compute.internal     Ready    control-plane,master   135m
v1.21.6
ip-10-0-71-114.us-west-2.compute.internal     Ready    <none>                 135m
v1.21.6
ip-10-0-71-207.us-west-2.compute.internal     Ready    <none>                 135m
v1.21.6
ip-10-0-85-253.us-west-2.compute.internal     Ready    control-plane,master   137m
v1.21.6
```

To verify a successful installation, all of the previous commands should return as Ready or True.

9.8.2 Monitor the CAPI resources

Obtain a list of all resources that comprise a cluster and their statuses:

```
kubectl get cluster-api
```

9.8.3 Verify all Pods

All pods installed by DKP should be in `Running` or `Completed` status if the install is successful.

```
kubectl get pods --all-namespaces
```

9.8.4 Troubleshooting

If any pod is not in `Running` or `Completed` status, you need to investigate further. If it appears that something has not deployed properly or fully, run a diagnostic bundle (see page 910):

```
dkp diagnose
```

This collects information from pods and infrastructure. For more information, see more about generating a support bundle and the different configurations that it supports (see page 910).

9.9 GPU for Konvoy

In this version of DKP, the nodes with NVIDIA GPUs are configured with nvidia-gpu-operator (Overview — NVIDIA Cloud Native Technologies documentation[1002]) and NVIDIA drivers to support the container runtime.

This page will link you to all the relevant GPU pages necessary such as Updating Cluster Nodepools (see page 1359) or KIB for GPU (see page 1302). The remainder of GPU information (see page 860) is found in Day 2 Operations under Kommander component section of documentation.

9.10 Delete a DKP Cluster with One Command

You can use a single command line entry to delete a Kubernetes cluster on any of the platforms supported by DKP. Deleting a cluster means removing the cluster, all of its nodes, and all of the platform applications that were deployed on it as part of its creation. Use this command with extreme care, as it is not reversible!

You need to delete the attachment for any clusters attached in Kommander before running the `delete` command.

Persistent Volumes (PVs) are not deleted automatically by design in order to preserve your data. However, they take up storage space if not deleted. You must delete PVs manually. Information for backup of a cluster and PVs is on the page in documentation called Back up your Cluster's Applications and Persistent Volumes (see page 736) .

With Vsphere clusters, `dkp delete` doesn't delete the virtual disks backing the PVs for DKP add ons. Therefore internal VMware cluster runs out of storage eventually. Theses PVs are only visible if VSAN is installed which gives users a Container Native Storage tab.

Set the environment variable to be used throughout this documentation:

```
export CLUSTER_NAME=cluster-example
```

The basic DKP `delete` command structure is:

```
dkp delete cluster --cluster-name=${CLUSTER_NAME} --self-managed --kubeconfig=$
{CLUSTER_NAME}.conf
```

1002 https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/overview.html

When you use the `--self-managed` flag, the prerequisite components and resources are moved from the self-managed cluster before deleting. When you omit this flag (the default value is false) the resources are assumed to be installed in a management cluster. The default value is false, or no flag.

This command performs the following actions:

- Creates a local bootstrap cluster
- Moves controllers to it
- Deletes the management cluster
- Deletes the local bootstrap cluster

10 Upgrade DKP

The DKP upgrade represents an important step of your environment's lifecycle, as it ensures that you are up-to-date with the latest features and can benefit from the most recent improvements, enhanced cluster management, and better performance. This section describes how to upgrade your air-gapped and non-air-gapped environment to the latest version of DKP compatible with the latest Kubernetes version.

10.1 Prerequisite

Check what version of DKP you have downloaded currently using cli command dkp version (see page 1534).

10.2 Supported Upgrade Paths

Use this table to determine your correct upgrade path:

Upgrade DKP

	Upgrading from Release…					
…to Release		**2.4.0**	**2.4.1**	**2.5.0**	**2.5.1**	**2.5.2**
	2.4.0	NA	NA	NA	NA	NA
	2.4.1	Yes	NA	NA	NA	NA
	2.5.0	Yes	Yes	NA	NA	NA
	2.5.1	Yes	Yes	Yes	NA	NA
	2.5.2	Yes	Yes	Yes	Yes	NA
	2.6.0	No	No	Yes	Yes	Yes

10.2.1 Understand the Upgrade Process

For this release, you perform the upgrade sequentially, beginning with the DKP UI and then moving to upgrading clusters and CAPI components.

When upgrading DKP, the process is different depending on whether you run a stand-alone Management Cluster, or a multi-cluster environment that includes a combination of a Management cluster and managed or attached workspace clusters.

Start with your Management Cluster in the UI, and then, if more than one exists, proceed workspace by workspace until complete. You can then move to upgrading Konvoy, cluster by cluster.

The overall process for upgrading to the latest version of DKP is done on each Workspace or cluster, with the following processes:

- Before upgrading, we **strongly recommend** reading the release notes (see page 1537) and verifying your current setup against any possible breaking changes.
- Review the list of major Kubernetes changes (see page 1544) that may affect your system.

For **Kommander**, on your DKP Essential cluster (see page 98) or DKP Enterprise Management Cluster (see page 97):

1. **Upgrade the DKP UI** (see page 1369)and all Platform Applications.

   If you do not have any managed or attached clusters, skip to upgrading Konvoy on your DKP Essential or DKP Management Cluster.

On your Workspaces (which include DKP Essential, DKP Enterprise Management Cluster and any Managed (see page 97) or Attached (see page 97) clusters):

1. **Upgrade your Workspaces** (see page 1526), which upgrades all Platform Applications on your workspace.
2. **Upgrade all DKP Catalog applications** (see page 597) deployed to Workspaces.
3. **Upgrade all DKP Catalog applications** (see page 617) deployed to Projects.
4. **Verify any Custom Catalog applications** (see page 1373) and ensure they are compatible with the Kubernetes version included in the new release (see page 1537).

For **Konvoy**, on your DKP Essential cluster (see page 98) or DKP Enterprise Management Cluster (see page 97):

1. **Upgrade CAPI components** (see page 1373) (Essential or Enterprise section). This upgrades the CAPI controllers, which only run on the Management Cluster.
2. **Upgrade the Core Addons** (see page 1373) (Essential or Enterprise section). This upgrades multiple addons such as CSI, CNI, Cluster Autoscaler, and Node Feature Discovery.
3. **Upgrade the Kubernetes version** (see page 1373) (Essential or Enterprise section). This upgrades your cluster's control plane and node pools.

If you do not have any managed or attached clusters, you have finished the upgrade process and can start testing your environment. If you have managed or attached clusters, continue with the next section.

For **Konvoy**, on your Managed Clusters:

1. **Upgrade the Core Addons** (see page 0). This upgrades multiple addons such as CSI, CNI, Cluster Autoscaler, and Node Feature Discovery.
2. **Upgrade the Kubernetes version** (see page 0). This upgrades your cluster's control plane and node pools. We recommend you upgrade your Kubernetes version on any attached clusters.

10.3 Upgrade: For Air-gapped Environments Only

If you are operating in an air-gapped environment, a local container registry (see page 1349) containing all the necessary installation images, including the Kommander images is required. See below for how to push the necessary images to this registry.

- Download (see page 71) the Complete DKP Air-gapped Bundle for this release (i.e. `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz`) to load registry images as explained below.
- Connectivity with clusters attaching to the management cluster:
  - Both management and attached clusters must be able to connect to the local registry.
  - The management cluster must be able to connect to all attached cluster's API servers.
  - The management cluster must be able to connect to any load balancers created for platform services on the management cluster.
- For **Pre-provisioned air-gapped environments only**, you must run `konvoy-image upload artifacts` to copy the artifacts (see page 1082) onto the cluster hosts **before** you begin the Upgrade the CAPI Components section below.

```
konvoy-image upload artifacts \
    --container-images-dir=./artifacts/images/ \
    --os-packages-bundle=./artifacts/$OS_PACKAGES_BUNDLE \
    --containerd-bundle=artifacts/$CONTAINERD_BUNDLE \
    --pip-packages-bundle=./artifacts/pip-packages.tar.gz
```

Follow these steps to load the air-gapped image bundles into your private registry:

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz`, extract the tarball to a local directory:

```
tar -xzvf dkp-air-gapped-bundle_v2.5.0_linux_amd64.tar.gz && cd dkp-v2.5.0
```

2. See the `NOTICES.txt` file for 3rd party software attributions in the `container-images/` directory.
3. Set an environment variable with your registry address:

```
export REGISTRY_ADDRESS=<registry-address>:<registry-port>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

For the basic air-gapped `kommander` image bundle, run the command below:

1. Run the following command to load the image bundle:

```
dkp push image-bundle --image-bundle ./container-images/kommander-image-bundle-v2.5.0.tar --to-registry $REGISTRY_ADDRESS
```

For **DKP Catalog Applications**, also perform this image load:

1. Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push image-bundle --image-bundle ./container-images/dkp-catalog-applications-image-bundle-v2.5.0.tar --to-registry $REGISTRY_ADDRESS
```

If you are running **DKP Insights**, also perform this image load command:

1. Run the following command to load the `dkp-insights` image bundle into your private registry:

```
dkp push image-bundle --image-bundle ./container-images/dkp-insights-image-bundle-v2.5.0.tar --to-registry $REGISTRY_ADDRESS
```

Before creating a Kubernetes cluster, you need to load the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1162) and either the AWS EC2 instances or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1349) page for more information.

1. Execute the following command to load the air-gapped image bundle into your private registry:

```
dkp push image-bundle --image-bundle ./container-images/konvoy-image-bundle-
v2.5.0.tar --to-registry $REGISTRY_ADDRESS --to-registry-username
$REGISTRY_USERNAME --to-registry-password $REGISTRY_PASSWORD
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

10.4 Upgrade Kommander

This section describes how to upgrade your Kommander Management cluster and all Platform Applications to their supported versions in air-gapped, non-air-gapped and on-prem environments. To prevent compatibility issues, you must first upgrade Kommander on your Management Cluster before upgrading to DKP.

It is important you upgrade Kommander BEFORE upgrading the Kubernetes version (or Konvoy version for Managed Konvoy clusters) in attached clusters. This ensures that any changes required for new or changed Kubernetes API's are already present.

Page Contents

10.4.1 Prerequisites

- **REQUIRED** Before upgrading, create an on-demand backup (see page 772) of your current configuration with Velero.
- Download (see page 71) and install the supported DKP CLI binary of this release (see page 1537) on your computer.
- Ensure you are attempting to run a supported DKP upgrade (see page 1365).
- Ensure you are on Kubernetes version 1.24.x.
- If you have attached clusters, ensure they are on Kubernetes versions 1.24 or later.
- Review the DKP 2.5.0 Components and Applications (see page 1544) versions that are part of this upgrade.
- For air-gapped environments **with** DKP Catalog Applications: Load the **Kommander** and **DKP Catalog Applications** Docker images into your Docker registry (see page 1259)
- For air-gapped environments **with** DKP Insights: Load the **Kommander**, **DKP Insights** Docker images into your Docker registry (see page 1259)

- For air-gapped environments **without** DKP Catalog Applications: Load the **Kommander** Docker images into your Docker registry (see page 0)

DKP Upgrade with Insights Installed
If your environment has Insights installed, you must uninstall Insights BEFORE upgrading DKP.

10.4.2 Upgrade Kommander

Before running the following command, ensure that your `dkp` configuration **references the Kommander Management cluster**, otherwise it attempts to run the upgrade on the bootstrap cluster. You can do this by setting the `KUBECONFIG` environment variable to the appropriate kubeconfig file's location[1003].

If you have configured a **custom domain**, running the `upgrade command` could result in an inaccessibility of your services via your custom domain for a few minutes.

As stated earlier, an alternative to initializing the KUBECONFIG environment variable is to use the `-kubeconfig=cluster_name.conf` flag. This ensures that Kommander upgrades on the workload cluster.

1. Use the DKP CLI to upgrade Kommander and all the Platform Applications in the Management Cluster:
   a. For **air-gapped** environments:

```
dkp upgrade kommander --charts-bundle ./application-charts/dkp-
kommander-charts-bundle-v2.5.2.tar.gz --kommander-applications-
repository ./application-repositories/kommander-applications-v2.5.2.tar.
gz
```

   b. For **air-gapped** with **DKP Catalog Applications** in a multi-cluster environment:

```
dkp upgrade kommander --charts-bundle ./application-charts/dkp-
kommander-charts-bundle-v2.5.2.tar.gz --charts-bundle ./application-
charts/dkp-catalog-applications-charts-bundle-v2.5.2.tar.gz --kommander-
```

1003 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

```
applications-repository ./application-repositories/kommander-
applications-v2.5.2.tar.gz
```

After the upgrade, if you have DKP Catalog Applications deployed, update the GitRepository for `dkp-catalog-applications`.

⚠️⚠️⚠️ For the following section, ensure you modify the **most recent** `kommander.yaml` configuration file. It must be the file that reflects the current state of your environment. Reinstalling Kommander with an outdated `kommander.yaml` overwrites the list of platform applications that are currently running in your cluster. ⚠️⚠️⚠️

i. In the `kommander.yaml` you are currently using for your environment, update the DKP Catalog Applications by setting the correct DKP version:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
...
# The list of enabled/disabled apps here should reflect the
current state of the environment, including configuration
overrides!
...
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository:
"true"
        kommander.d2iq.io/workspace-default-catalog-repository:
"true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      path: ./dkp-catalog-applications-v2.5.2.tar.gz # modify this
 version to match the DKP upgrade version
```

ii. Refresh the `kommander.yaml` to apply the updated tarball:

⚠️ Before running this command, ensure the `kommander.yaml` is the configuration file you are currently using for your environment. Otherwise, your previous configuration will be lost. ⚠️

```
dkp install kommander --installer-config kommander.yaml
```

c. For **non-air-gapped** environments:

```
dkp upgrade kommander
```

After the upgrade, if you have DKP Catalog Applications deployed, update the GitRepository for `dkp-catalog-applications`.

i. Update the GitRepository with the tag of your updated DKP version on the `kommander` workspace:

```
kubectl patch gitrepository -n kommander dkp-catalog-applications
--type merge --patch '{"spec":{"ref":{"tag":"v2.5.2"}}}'
```

This command updates the catalog application repositories for all workspaces.

ii. For any additional workspaces created outside the `kommander.yaml` configuration:
Set the `WORKSPACE_NAMESPACE` environment variable to the namespace of the workspace:

```
export WORKSPACE_NAMESPACE=<workspace namespace>
```

iii. Update the GitRepository for the additional workspace:

```
kubectl patch gitrepository -n ${WORKSPACE_NAMESPACE} dkp-catalog-
applications --type merge --patch '{"spec":{"ref":
{"tag":"v2.5.2"}}}'
```

The output looks similar to this:

```
✓ Ensuring upgrading conditions are met
✓ Ensuring application definitions are updated
✓ Ensuring helm-mirror implementation is migrated to chartmuseum
...
```

2. For **Enterprise customers** (see page 75) (multi-cluster environment): Upgrade your additional Workspaces (see page 566) on a per-Workspace basis to upgrade the Platform Applications on other clusters than the Management Cluster. After upgrading your Workspaces, proceed with the Konvoy Upgrade (see page 1373).
For **Essential customers** (see page 78) (single-cluster environment): Proceed with the Konvoy Upgrade (see page 1373).

10.4.3 Troubleshooting

If the upgrade fails, run the following command to get more information on the upgrade process:

```
dkp upgrade kommander -v 6
```

If you find any `HelmReleases` in a “broken” release state such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
```

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

You can always go back to the DKP Upgrade overview (see page 0), to review the next steps depending on your environment and license type.

10.5 Upgrade Custom Applications

10.5.1 Verify the compatibility of Custom Applications with the current Kubernetes version

We recommend upgrading your Custom Applications to the latest compatible version as soon as possible. Since Custom Applications are not created, maintained or supported by D2iQ, you must upgrade them manually.

Ensure you validate any Custom Applications you run for compatibility issues against the Kubernetes version (see page 1537) in the new release. If the Custom Application's version is not compatible with the Kubernetes version, do not continue with the Konvoy upgrade. Otherwise, your custom Applications may stop running.

Once you have ensured your Custom Applications are compatible with the current Kubernetes version, return to the DKP Upgrade overview (see page 0) documentation, to review the next steps depending on your environment and license type.

10.6 Upgrade Konvoy

10.6.1 Steps to upgrade Konvoy via CLI

This section describes how to upgrade your DKP clusters to the latest Konvoy version. To prevent compatibility issues, review and ensure you are following the correct upgrade process for customers with a single-cluster or multi-cluster experience.

- DKP Enterprise Upgrade (see page 1373)
- DKP Essential Upgrade (see page 1384)
- Upgrade Cluster Node Pools (see page 1392)

10.6.2 DKP Enterprise Upgrade

Enterprise Gov Advanced

Upgrade your Konvoy environment within the DKP Enterprise license.

10.6.2.1 Prerequisites

- Create an on-demand backup (see page 772) of your current configuration with Velero.
- Follow the steps listed in the DKP upgrade overview (see page 1366).
- Check what version of DKP you have downloaded currently using cli command dkp version (see page 1534).
- Ensure that all platform applications in the management cluster have been upgraded to avoid compatibility issues with the Kubernetes version (see page 1543) included in this release. This is done automatically when upgrading Kommander (see page 1369), so ensure that you upgrade Kommander prior to upgrading Konvoy.
- Set the appropriate environment variables:
  - For Air-gapped, download the required bundles at our support site[1004].
  - For Azure, set the required environment variables (see page 1033).
  - For AWS and EKS (see page 940), set the required environment variables (see page 940).
  - For vSphere, set the required environment variables (see page 1133).
  - For GCP, set the required environment variables (see page 1199).
- **vSphere only:** If you want to resize your disk, ensure you have reviewed Create a vSphere Base OS Image (see page 1305).

10.6.2.2 Overview

To upgrade Konvoy for DKP Enterprise:

1. Upgrade the Cluster API (CAPI) components.
2. Upgrade the core addons.
3. Upgrade the Kubernetes version.
4. Upgrade the Managed clusters.
5. Upgrade the Kubernetes version of Managed clusters.

Perform all three steps on the management cluster first. Then, execute the second and third steps on additional managed clusters one cluster at a time. For the managed clusters, you use the KUBECONFIG for the management cluster and specify the name of the managed cluster(s) to upgrade. You must maintain your attached clusters manually. Review the documentation from your cloud provider for more information.

For a full list of DKP Enterprise features, see DKP Enterprise (see page 75).

1004 https://support.d2iq.com/hc/en-us

- For air-gapped environments, seed the docker registry as explained here: Air-gapped Seed the Registry[1005]
- For Pre-provisioned air-gapped environments only, you must run `konvoy-image upload artifacts` to copy the artifacts[1006] onto the cluster hosts before you begin the Upgrade the CAPI Components section below.

```
konvoy-image upload artifacts \
    --container-images-dir=./artifacts/images/ \
    --os-packages-bundle=./artifacts/$OS_PACKAGES_BUNDLE \
    --containerd-bundle=artifacts/$CONTAINERD_BUNDLE \
    --pip-packages-bundle=./artifacts/pip-packages.tar.gz
```

10.6.2.3 Upgrade the CAPI Components

New versions of DKP come pre-bundled with newer versions of CAPI, newer versions of infrastructure providers, or new infrastructure providers. When using a new version of the DKP CLI, upgrade all of these components first.

If you are running on more than one management cluster, you must upgrade the CAPI components on each of these clusters.

Ensure your `dkp` configuration references the management cluster where you want to run the upgrade by setting the `KUBECONFIG` environment variable, or using the `--kubeconfig` flag, in accordance with Kubernetes conventions[1007].

Execute the upgrade command for the CAPI components.

If you created CAPI components using flags to specify values, use those same flags during **Upgrade** to preserve existing values while setting additional values.

- Refer to dkp create cluster (see page 1454) for flag descriptions for `--with-aws-bootstrap-credentials` and `--aws-service-endpoints`

1005 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/133890151/Air-gapped+Seed+the+Registry
1006 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29918702
1007 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

- Refer to the HTTP section for details: Universal Configurations for all Infrastructures (see page 921)

```
dkp upgrade capi-components
```

The output resembles the following:

```
✓ Upgrading CAPI components
✓ Waiting for CAPI components to be upgraded
✓ Initializing new CAPI components
✓ Deleting Outdated Global ClusterResourceSets
```

If the upgrade fails, review the prerequisites section and ensure that you've followed the steps in the DKP Upgrade (see page 1365) overview. Furthermore, ensure you have adhered to the Prerequisites at the top of this page.

10.6.2.4 Upgrade the Core Addons

To install the core addons, DKP relies on the `ClusterResourceSet` Cluster API feature[1008]. In the CAPI component upgrade, we deleted the previous set of outdated global `ClusterResourceSets` because in past releases, some addons were installed using a global configuration. In order to support individual cluster upgrades, DKP now installs all addons with a unique set of `ClusterResourceSets` and corresponding referenced resources, all named using the cluster's name as a suffix. For example: `calico-cni-installation-my-aws-cluster`.

If you have modified any of the `ClusterResourceSet` definitions, these changes will **not** be preserved when running the command `dkp upgrade addons <provider>`. You must define the cloud provider before you use the `--dry-run -o yaml` options to save the new configuration to a file and remake the same changes upon each upgrade.

Your cluster comes preconfigured with a few different core addons that provide functionality to your cluster upon creation. These include: CSI, CNI, Cluster Autoscaler, and Node Feature Discovery. New versions of DKP may come pre-bundled with newer versions of these addons.

Perform the following steps to update these addons:

1. If you have any additional managed clusters, you will need to upgrade the core addons and Kubernetes version for each one.

1008 https://cluster-api.sigs.k8s.io/

2. Ensure your `dkp` configuration references the management cluster where you want to run the upgrade by setting the `KUBECONFIG` environment variable, or using the `--kubeconfig` flag, in accordance with Kubernetes conventions[1009].
3. Upgrade the core addons in a cluster using the `dkp upgrade addons` command specifying the cluster infrastructure (choose `aws`, `azure`, `vsphere`, `eks`, `gcp`, `preprovisioned`) and the name of the cluster.

If you need to verify or discover your cluster name to use with this example, first run the `kubectl get clusters` command.

Examples for upgrade core addons commands:

```
export CLUSTER_NAME=my-azure-cluster
dkp upgrade addons azure --cluster-name=${CLUSTER_NAME}
```

OR

```
export CLUSTER_NAME=my-aws-cluster
dkp upgrade addons aws --cluster-name=${CLUSTER_NAME}
```

The output for the AWS example should be similar to:

```
Generating addon resources
clusterresourceset.addons.cluster.x-k8s.io/calico-cni-installation-my-aws-cluster
upgraded
configmap/calico-cni-installation-my-aws-cluster upgraded
clusterresourceset.addons.cluster.x-k8s.io/tigera-operator-my-aws-cluster upgraded
configmap/tigera-operator-my-aws-cluster upgraded
clusterresourceset.addons.cluster.x-k8s.io/aws-ebs-csi-my-aws-cluster upgraded
configmap/aws-ebs-csi-my-aws-cluster upgraded
clusterresourceset.addons.cluster.x-k8s.io/cluster-autoscaler-my-aws-cluster upgraded
configmap/cluster-autoscaler-my-aws-cluster upgraded
clusterresourceset.addons.cluster.x-k8s.io/node-feature-discovery-my-aws-cluster
upgraded
configmap/node-feature-discovery-my-aws-cluster upgraded
clusterresourceset.addons.cluster.x-k8s.io/nvidia-feature-discovery-my-aws-cluster
upgraded
configmap/nvidia-feature-discovery-my-aws-cluster upgraded
```

1009 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

10.6.2.4.1 See Also

DKP upgrade addons (see page 1527) for more CLI command help.

10.6.2.5 Upgrade the Kubernetes Version

When upgrading the Kubernetes version of a cluster, first upgrade the control plane and then the node pools. If you have any additional managed or attached clusters, you need to upgrade the core addons and Kubernetes version for each one.

1. Build a new image if applicable.
    - If an AMI was specified when initially creating a cluster for AWS, you must build a new one with Konvoy Image Builder (see page 1288).
    - If an Azure Machine Image was specified for Azure, you must build a new one with Konvoy Image Builder (see page 1295).
    - If a vSphere template Image was specified for vSphere, you must build a new one with Konvoy Image Builder (see page 1305).
    - You must build a new GCP image with Konvoy Image Builder (see page 1299).
2. Upgrade the Kubernetes version of the control plane. Each cloud provider has distinctive commands. Below is the AWS command example. Select the drop-down menu next to your provider for compliant CLI.
   NOTE: The first example below is for AWS. If you created your initial cluster with a custom AMI using the `--ami` flag, it is required to set the `--ami` flag during the Kubernetes upgrade.

   ```
   dkp update controlplane aws --cluster-name=${CLUSTER_NAME} --kubernetes-
   version=v1.25.4
   ```

Azure

```
dkp update controlplane azure --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.25.4 --compute-gallery-id <Azure Compute Gallery built by KIB for
 Kubernetes v1.25.4>
```

- If these fields were specified in the override file (see page 1330) during image creation (see page 1295), the flags must be used in upgrade:
    - `--plan-offer`, `--plan-publisher` and `--plan-sku`
    - ```
      --plan-offer rockylinux-9
      --plan-publisher erockyenterprisesoftwarefoundationinc1653071250513
      --plan-sku rockylinux-9
      ```

vSphere
      ```

```
dkp update controlplane vsphere --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.25.4 --vm-template <vSphere template built by KIB for Kubernetes v1.25.4>
```

GCP

```
dkp update controlplane gcp --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.25.4 --image=projects/${GCP_PROJECT}/global/images/<GCP image built by KIB
for Kubernetes v1.25.4>
```

Pre-provisioned

```
dkp update controlplane preprovisioned --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.25.4
```

EKS

```
dkp update controlplane eks --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.24.7
```

The `nodes.cluster-api-provider-aws.sigs.k8s.io` role is **required** to be present in your AWS account when using EKS clusters with DKP 2.5.1. Refer to the DKP documentation regarding IAM Permissions (see page 1010) for more information.
See also: IAM Permissions Used-Kubernetes Cluster API Provider AWS[1010]

The output should be similar to the below example, with the provider name corresponding to the CLI you executed from the choices above:

```
Updating control plane resource controlplane.cluster.x-k8s.io/v1beta1,
Kind=KubeadmControlPlane default/my-aws-cluster-control-plane
Waiting for control plane update to finish.
 ✓ Updating the control plane
```

1010 https://cluster-api-aws.sigs.k8s.io/topics/iam-permissions.html?highlight=nodes.cluster-api-provider-aws.sigs.k8s.io#required-by-all-nodes

Some advanced options are available for various providers. To see all the options for your particular provider, run this command `dkp update controlplane aws|vsphere|preprovisioned|azure|gcp|eks --help` for more advance options like the example below:

This example for AWS AMI instance type: `aws: --ami, --instance-type` would be some of the options mentioned in the note above.

NOTE: The command `dkp update controlplane {provider}` has a 30 minute default timeout for the update process to finish. If you see the error "`timed out waiting for the condition`", you can check the control plane nodes version using the command `kubectl get machines -o wide $KUBECONFIG` before trying again.

Additional Considerations for upgrading a FIPS cluster:

If upgrading a FIPS cluster, to correctly upgrade the Kubernetes version, instead run the command shown below:

```
dkp update controlplane aws --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.25.4+fips.0 --ami=<ami-with-fips-id>
```

3. Upgrade the Kubernetes version of your node pools. Upgrading a nodepool involves draining the existing nodes in the nodepool and replacing them with new nodes. In order to ensure minimum downtime and maintain high availability of the critical application workloads during the upgrade process, we recommend deploying Pod Disruption Budget (Disruptions[1011]) for your critical applications. For more information, refer to Update Cluster Nodepools (see page 1359) documentation.

a. First, get a list of all node pools available in your cluster by running the following command:

```
dkp get nodepool --cluster-name ${CLUSTER_NAME}
```

b. Select the nodepool you want to upgrade with the command below:

```
export NODEPOOL_NAME=my-nodepool
```

c. Then update the selected nodepool using the command below. The first example command shows AWS language, so select the drop-down menu for your provider for the correct command. Execute the `update` command for each of the node pools listed in the previous command:

NOTE: The first example below is for AWS. If you created your initial cluster with a custom AMI using the `--ami` flag, it is required to set the `--ami` flag during the Kubernetes upgrade.

1011 https://kubernetes.io/docs/concepts/workloads/pods/disruptions/

```
dkp update nodepool aws ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.25.4
```

Azure

```
dkp update nodepool azure ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --
kubernetes-version=v1.25.4 --compute-gallery-id <Azure Compute Gallery built by KIB
for Kubernetes v1.25.4>
```

- If these fields were specified in the override file (see page 1330) during image creation (see page 1295), the flags must be used in upgrade:
  - `--plan-offer`, `--plan-publisher` and `--plan-sku`
  - ```
    --plan-offer rockylinux-9
    --plan-publisher erockyenterprisesoftwarefoundationinc1653071250513
    --plan-sku rockylinux-9
    ```

vSphere

```
dkp update nodepool vsphere ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --
kubernetes-version=v1.25.4 --vm-template <vSphere template built by KIB for
 Kubernetes v1.25.4>
```

GCP

```
dkp update nodepool gcp ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.25.4 --image=projects/${GCP_PROJECT}/global/images/<GCP image built by KIB
for Kubernetes v1.25.4>
```

Pre-provisioned

```
dkp update nodepool preprovisioned ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --
kubernetes-version=v1.25.4
```

EKS

```
dkp update nodepool eks ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.24.7
```
```

The `nodes.cluster-api-provider-aws.sigs.k8s.io` role is **required** to be present in your AWS account when using EKS clusters with DKP 2.5.1. Refer to the DKP documentation regarding IAM Permissions (see page 1010) for more information.
See also: IAM Permissions Used-Kubernetes Cluster API Provider AWS[1012]

The output should be similar to the following, with the name of the infrastructure provider shown accordingly:

```
Updating node pool resource cluster.x-k8s.io/v1beta1, Kind=MachineDeployment default/
my-aws-cluster-my-nodepool
Waiting for node pool update to finish.
 ✓ Updating the my-aws-cluster-my-nodepool node pool
```

d. Repeat this step for each additional node pool.

Additional Considerations for upgrading a FIPS cluster:

If upgrading a FIPS cluster, to correctly upgrade the Kubernetes version, instead run the command shown below:

```
dkp update nodepool aws ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.25.4+fips.0 --ami=<ami-with-fips-id>
```

When all nodepools have been updated, your upgrade is complete. For the overall process for upgrading to the latest version of DKP, refer back to DKP Upgrade (see page 1366) for more details.

10.6.2.6 Upgrade Managed Clusters

If you have managed clusters, follow these steps to upgrade each cluster:

1. Using the kubeconfig of your management cluster, find your cluster name and be sure to copy the information for all of your clusters:

   ```
   kubectl get clusters -A
   ```

2. Set your cluster variable:

   ```
   export CLUSTER_NAME=<your-managed-cluster-name>
   ```

3. Set your cluster's workspace variable:

1012 https://cluster-api-aws.sigs.k8s.io/topics/iam-permissions.html?highlight=nodes.cluster-api-provider-aws.sigs.k8s.io#required-by-all-nodes

```
export CLUSTER_WORKSPACE=<your-workspace-namespace>
```

4. Then, upgrade the core addons (replacing `aws` with whatever infrastructure provider you would be using):

```
dkp upgrade addons aws --cluster-name=${CLUSTER_NAME} -n ${CLUSTER_WORKSPACE}
```

5. Check to see if you have any cluster resource sets that need to be cleaned up:

```
kubectl get clusterresourcesets -n ${CLUSTER_WORKSPACE}
```

6. Delete the ClusterResourceSet for nvidia-feature-discovery by running:

```
kubectl delete clusterresourceset nvidia-feature-discovery-${CLUSTER_NAME} -n $
{CLUSTER_WORKSPACE}
```

7. Delete ConfigMap ClusterResourceSet referred to by running the following command, ensure you use using `nvidia-feature-discover-${CLUSTER_NAME}`. If there is no related ConfigMap, then you can move on to the next step.

```
kubectl delete configmap nvidia-feature-discovery-${CLUSTER_NAME} -n $
{CLUSTER_WORKSPACE}
```

8. Get the kubeconfig for the managed cluster by running:

```
dkp get kubeconfig -c ${CLUSTER_NAME} -n ${CLUSTER_WORKSPACE} >> $
{CLUSTER_NAME}.conf
```

9. Delete the corresponding daemonset on the remote cluster by running the following command. If there is no related daemonset, then you can move on to the next step.

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf delete daemonset nvidia-feature-
discovery-gpu-feature-discovery -n node-feature-discovery
```

10.6.2.6.1 Upgrade Kubernetes Version on a Managed Cluster

After you complete the previous steps for all managed clusters and you update your core addons, begin upgrading the Kubernetes version.

> You should first complete the upgrade of your Kommander Management Cluster before upgrading any managed clusters.

1. Use this command to start upgrading the Kubernetes version:

```
dkp update controlplane aws --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.25.4 -n ${CLUSTER_WORKSPACE}
```

2. Get a list of all node pools available in your cluster by running the following command:

```
dkp get nodepools -c ${CLUSTER_NAME} -n ${CLUSTER_WORKSPACE}

export NODEPOOL_NAME=<my-nodepool>
```

3. Use this command to upgrade the node pools:

```
dkp update nodepool aws ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --
kubernetes-version=v1.25.4 -n ${CLUSTER_WORKSPACE}
```

10.6.3 DKP Essential Upgrade

Upgrade your Konvoy environment within the DKP Essential License.

10.6.3.1 Prerequisites

- Create an on-demand backup (see page 772) of your current configuration with Velero.
- Follow the steps listed in the DKP upgrade overview (see page 1366).
- Check what version of DKP you have downloaded currently using cli command dkp version (see page 1534).
- Ensure that all platform applications in the management cluster are upgraded to avoid compatibility issues with the Kubernetes version (see page 1543) included in this release. This is done automatically when upgrading Kommander (see page 1369), so ensure that you upgrade Kommander prior to upgrading Konvoy.
- Set the appropriate environment variables:

    - For air-gapped: Download the required bundles either at our support site[1013] or by using the CLI.
    - For Azure, set the required environment variables (see page 1033).
    - For AWS, set the required environment variables (see page 940).
    - For vSphere, set the required environment variables (see page 1133).
    - For GCP, set the required environment variables (see page 494).
- **vSphere only:** If you want to resize your disk, ensure you have reviewed Create a vSphere Base OS Image (see page 1305).

10.6.3.2 Overview

To upgrade Konvoy for DKP Essential:

1. Upgrade the Cluster API (CAPI) components.
2. Upgrade the core addons.
3. Upgrade the Kubernetes version.

If you have more than one Essential cluster, repeat all of these steps for each Essential cluster. For a full list of DKP Essential features, see DKP Essential (see page 78).

- For air-gapped environments, you must run `konvoy-image upload artifacts` to copy the artifacts (see page 1082) onto the cluster hosts before you begin the Upgrade the CAPI Components section below.
    - ```
      konvoy-image upload artifacts \
          --container-images-dir=./artifacts/images/ \
          --os-packages-bundle=./artifacts/$OS_PACKAGES_BUNDLE \
          --containerd-bundle=artifacts/$CONTAINERD_BUNDLE \
          --pip-packages-bundle=./artifacts/pip-packages.tar.gz
      ```
- For air-gapped environments, seed the docker registry as explained here: Air-gapped Seed the Registry (see page 1350)

10.6.3.2.1 Upgrade the CAPI components

New versions of DKP come pre-bundled with newer versions of CAPI, newer versions of infrastructure providers, or new infrastructure providers. When using a new version of the DKP CLI, upgrade all of these components first.

1013 https://support.d2iq.com/hc/en-us
      ```

Ensure your `dkp` configuration references the management cluster where you want to run the upgrade by setting the `KUBECONFIG` environment variable, or using the `--kubeconfig` flag, in accordance with Kubernetes conventions[1014].

Run the following upgrade command for the CAPI components:

If you created CAPI components using flags to specify values, use those same flags during **Upgrade** to preserve existing values while setting additional values.

- Refer to dkp create cluster (see page 1454) for flag descriptions for `--with-aws-bootstrap-credentials` and `--aws-service-endpoints`
- Refer to the HTTP section for details: Universal Configurations for all Infrastructures (see page 921)

```
dkp upgrade capi-components
```

The command should output something similar to the following:

```
✓ Upgrading CAPI components
✓ Waiting for CAPI components to be upgraded
✓ Initializing new CAPI components
✓ Deleting Outdated Global ClusterResourceSets
```

If the upgrade fails, review the prerequisites section and ensure that you've followed the steps in the DKP upgrade overview (see page 1366).

10.6.3.3 Upgrade the Core Addons

To install the core addons, DKP relies on the `ClusterResourceSet` Cluster API feature[1015]. In the CAPI component upgrade, we deleted the previous set of outdated global `ClusterResourceSets` because in past releases, some addons were installed using a global configuration. In order to support individual cluster upgrades, DKP now installs all addons with a unique set of `ClusterResourceSets` and corresponding referenced resources, all named using the cluster's name as a suffix. For example: `calico-cni-installation-my-aws-cluster`.

1014 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/
1015 https://cluster-api.sigs.k8s.io/

If you modify any of the `ClusterResourceSet` definitions, these changes are **not** be preserved when running the command `dkp upgrade addons`. You must use the `--dry-run -o yaml` options to save the new configuration to a file and continue the same changes upon each upgrade.

Your cluster comes preconfigured with a few different core addons that provide functionality to your cluster upon creation. These include: CSI, CNI, Cluster Autoscaler, and Node Feature Discovery. New versions of DKP may come pre-bundled with newer versions of these addons.

If you have more than one essential cluster, ensure your `dkp` configuration references the cluster where you want to run the upgrade by setting the `KUBECONFIG` environment variable, or using the `--kubeconfig` flag, in accordance with Kubernetes conventions[1016].

Perform the following steps to update your addons.

1. Ensure your `dkp` configuration references the cluster where you want to run the upgrade by setting the `KUBECONFIG` environment variable, or using the `--kubeconfig` flag, in accordance with Kubernetes conventions[1017].
2. Upgrade the core addons in a cluster using the `dkp upgrade addons` command specifying the cluster infrastructure (choose `aws`, `azure`, `vsphere`, `eks`, `gcp`, `preprovisioned`) and the name of the cluster.

Examples for upgrade core addons commands:

```
export CLUSTER_NAME=my-azure-cluster
dkp upgrade addons azure --cluster-name=${CLUSTER_NAME}
```

OR

```
export CLUSTER_NAME=my-aws-cluster
dkp upgrade addons aws --cluster-name=${CLUSTER_NAME}
```

The output for the AWS example should be similar to:

```
Generating addon resources
clusterresourceset.addons.cluster.x-k8s.io/calico-cni-installation-my-aws-cluster
upgraded
configmap/calico-cni-installation-my-aws-cluster upgraded
clusterresourceset.addons.cluster.x-k8s.io/tigera-operator-my-aws-cluster upgraded
configmap/tigera-operator-my-aws-cluster upgraded
```

1016 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/
1017 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

```
clusterresourceset.addons.cluster.x-k8s.io/aws-ebs-csi-my-aws-cluster upgraded
configmap/aws-ebs-csi-my-aws-cluster upgraded
clusterresourceset.addons.cluster.x-k8s.io/cluster-autoscaler-my-aws-cluster upgraded
configmap/cluster-autoscaler-my-aws-cluster upgraded
clusterresourceset.addons.cluster.x-k8s.io/node-feature-discovery-my-aws-cluster
upgraded
configmap/node-feature-discovery-my-aws-cluster upgraded
clusterresourceset.addons.cluster.x-k8s.io/nvidia-feature-discovery-my-aws-cluster
upgraded
configmap/nvidia-feature-discovery-my-aws-cluster upgraded
```

10.6.3.3.1 See Also

DKP upgrade addons (see page 1527) for more CLI command help.

10.6.3.4 Upgrade the Kubernetes version

When upgrading the Kubernetes version of a cluster, first upgrade the control plane and then the node pools.

1. Build a new image if applicable.
    - If an AMI was specified when initially creating a cluster for AWS, you must build a new one with Konvoy Image Builder (see page 1288).
    - If an Azure Machine Image was specified for Azure, you must build a new one with Konvoy Image Builder (see page 1295).
    - If a vSphere template Image was specified for vSphere, you must build a new one with Konvoy Image Builder (see page 1140).
    - You must build a new GCP image with Konvoy Image Builder (see page 1299).
2. Upgrade the Kubernetes version of the control plane. Each cloud provider has distinctive commands. Below is the AWS command example. Select the drop-down menu next to your provider for compliant CLI.
   NOTE: The first example below is for AWS. If you created your initial cluster with a custom AMI using the `--ami` flag, it is required to set the `--ami` flag during the Kubernetes upgrade.

   ```
   dkp update controlplane aws --cluster-name=${CLUSTER_NAME} --kubernetes-
   version=v1.25.4
   ```

If you need to verify or discover your cluster name to use with this example, first run the `kubectl get clusters` command.

Azure

```
dkp update controlplane azure --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.25.4 --compute-gallery-id <Azure Compute Gallery built by KIB for
 Kubernetes v1.25.4>
```

- If these fields were specified in the override file (see page 1330) during image creation (see page 1295), the flags must be used in upgrade:
  - `--plan-offer`, `--plan-publisher` and `--plan-sku`
  - ```
    --plan-offer rockylinux-9
    --plan-publisher erockyenterprisesoftwarefoundationinc1653071250513
    --plan-sku rockylinux-9
    ```

vSphere

```
dkp update controlplane vsphere --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.25.4 --vm-template <vSphere template built by KIB for Kubernetes v1.25.4>
```

GCP

```
dkp update controlplane gcp --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.25.4 --image=projects/${GCP_PROJECT}/global/images/<GCP image built by KIB
for Kubernetes v1.25.4>
```

Pre-provisioned

```
dkp update controlplane preprovisioned --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.25.4
```

EKS

```
dkp update controlplane eks --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.24.7
```

The output should be similar to the below example, with the provider name corresponding to the CLI you executed from the choices above:

Some advanced options are available for various providers. An example would be regarding AMI instance type: `aws: --ami, --instance-type`. To see all the options for your particular provider, run this command `dkp update controlplane aws|vsphere|preprovisioned|gcp|azure --help` for more advance options.
```

NOTE: The command `dkp update controlplane {provider}` has a 30 minute default timeout for the update process to finish. If you see the error "`timed out waiting for the condition`", you can check the control plane nodes version using the command `kubectl get machines -o wide $KUBECONFIG` before trying again.

The output should be similar to:

```
Updating control plane resource controlplane.cluster.x-k8s.io/v1beta1,
Kind=KubeadmControlPlane default/my-aws-cluster-control-plane
Waiting for control plane update to finish.
 ✓ Updating the control plane
```

```
kubectl set image -n kube-system daemonset.v1.apps/kube-proxy kube-proxy=docker.io/
mesosphere/kube-proxy:v1.25.4_fips.0
```

3. Upgrade the Kubernetes version of your node pools. Upgrading a node pool involves draining the existing nodes in the node pool and replacing them with new nodes. In order to ensure minimum downtime and maintain high availability of the critical application workloads during the upgrade process, we recommend deploying Pod Disruption Budget (Disruptions[1018]) for your critical applications. For more information, refer to Update Cluster Nodepools (see page 1359) documentation.

a. First, get a list of all node pools available in your cluster by running the following command:

```
dkp get nodepool --cluster-name ${CLUSTER_NAME}
```

b. Select the nodepool you want to upgrade with the command below:

```
export NODEPOOL_NAME=my-nodepool
```

c. Then update the selected nodepool using the command below. The first example command shows AWS language, so select the drop-down menu for your provider for the correct command. Execute the `update` command for each of the node pools listed in the previous command.

NOTE: The first example below is for AWS. If you created your initial cluster with a custom AMI using the `--ami` flag, it is required to set the `--ami` flag during the Kubernetes upgrade.

```
dkp update nodepool aws ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.25.4
```

Azure

1018 https://kubernetes.io/docs/concepts/workloads/pods/disruptions/

```
dkp update nodepool azure ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --
kubernetes-version=v1.25.4 --compute-gallery-id <Azure Compute Gallery built by KIB
for Kubernetes v1.25.4>
```

- If these fields were specified in the override file (see page 1330) during image creation (see page 1295), the flags must be used in upgrade:
  - `--plan-offer`, `--plan-publisher` and `--plan-sku`
  - ```
    --plan-offer rockylinux-9
    --plan-publisher erockyenterprisesoftwarefoundationinc1653071250513
    --plan-sku rockylinux-9
    ```

vSphere

```
dkp update nodepool vsphere ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --
kubernetes-version=v1.25.4 --vm-template <vSphere template built by KIB for
 Kubernetes v1.25.4>
```

GCP

```
dkp update nodepool gcp ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.25.4 --image=projects/${GCP_PROJECT}/global/images/<GCP image built by KIB
for Kubernetes v1.25.4>
```

Pre-provisioned

```
dkp update nodepool preprovisioned ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --
kubernetes-version=v1.25.4
```

EKS

```
dkp update nodepool eks ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.24.7
```

The output should be similar to the following, with the name of the infrastructure provider shown accordingly:

```
Updating node pool resource cluster.x-k8s.io/v1beta1, Kind=MachineDeployment default/
my-aws-cluster-my-nodepool
Waiting for node pool update to finish.
 ✓ Updating the my-aws-cluster-my-nodepool node pool
```

4. Repeat this step for each additional node pool.
```

Additional Considerations for upgrading a FIPS cluster:

If upgrading a FIPS cluster, to correctly upgrade the Kubernetes version, instead run the command shown below:

```
dkp update nodepool aws ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-version=v1.25.4+fips.0 --ami=<ami-with-fips-id>
```

When all nodepools have been updated, your upgrade is complete. For the overall process for upgrading to the latest version of DKP, refer back to DKP Upgrade (see page 1366) for more details.

For the overall process for upgrading to the latest version of DKP, refer back to DKP Upgrade (see page 1366).

10.6.4 Upgrade Cluster Node Pools

Upgrading a node pool involves draining the existing nodes in the node pool and replacing them with new nodes. In order to ensure minimum downtime and maintain high availability of the critical application workloads during the upgrade process, we recommend deploying Pod Disruption Budget (Disruptions[1019]) for your critical applications.

The Pod Disruption Budget will prevent any impact on critical applications as a result of misconfiguration or failures during the upgrade process.

10.6.4.1 Prerequisites:

- Deploy Pod Disruption Budget[1020] (PDB)
- Konvoy Image Builder (see page 1282) (KIB)

10.6.4.2 Steps:

1. Deploy Pod Disruption Budget for your critical applications. If your application can tolerate only one replica to be unavailable at a time, then you can set Pod disruption budget as shown in the following example. The example below is for NVIDIA GPU node pools, but the process is the same for all node pools.

1019 https://kubernetes.io/docs/concepts/workloads/pods/disruptions/
1020 https://kubernetes.io/docs/concepts/workloads/pods/disruptions/

Repeat this step for each additional node pool.

Create the file: `pod-disruption-budget-nvidia.yaml`

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: nvidia-critical-app
spec:
  maxUnavailable: 1
  selector:
    matchLabels:
      app: nvidia-critical-app
```

Apply the YAML file above using the following command:

```
kubectl create -f pod-disruption-budget-nvidia.yaml
```

2. Prepare OS image for your node pool using Konvoy Image Builder (see page 1282).

10.6.4.3 Related Topics:

Upgrade Konvoy (see page 1373)

11 CLI and API Tools

CLI commands and API tools.

11.1 API Documentation

This document is automatically generated from the API definition in the code.

11.1.1 apps.kommander.mesosphere.io/v1alpha2

11.1.1.1 App

App is the Schema for a Service or Application in Kommander.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1021]	false
spec		GenericAppSpec (see page 1397)	false
status		object	false

11.1.1.2 AppDeployment

AppDeployment is the Schema for a concrete installation of a Service or Application in Kommander.

1021 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1022]	false
spec		AppDeploymentSpec (see page 1395)	false
status		object	false

11.1.1.3 AppDeploymentList

AppDeploymentList contains a list of AppDeployments.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1023]	false
items		[...]AppDeployment (see page 1394)	true

11.1.1.4 AppDeploymentSpec

AppDeploymentSpec defines an instance of an Application.

Field	Description	Scheme	Required
appRef	AppRef provides reference to a ClusterApp or App object.	TypedLocalObjectReference (see page 1398)	true
configOverrides	ConfigOverrides allows a user to define a ConfigMap that contains configuration overrides	corev1.LocalObjectReference[1024]	false

1022 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1023 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1024 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

11.1.1.5 AppList

AppList contains a list of Apps.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1025]	false
items		[...]App (see page 1394)	true

11.1.1.6 ClusterApp

ClusterApp is the Schema for a Service or Application in Kommander.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1026]	false
spec		GenericAppSpec (see page 1397)	false
status		object	false

11.1.1.7 ClusterAppList

ClusterAppList contains a list of ClusterApps.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1027]	false
items		[...]ClusterApp (see page 1396)	true

1025 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1026 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1027 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

11.1.1.8 CrossNamespaceGitRepositoryReference

CrossNamespaceGitRepositoryReference contains enough information to let you locate the typed referenced object at cluster level.

Field	Description	Scheme	Required
apiVersion	API version of the referent	string	false
kind	Kind of the referent	string	true
name	Name of the referent	string	true
namespace	Namespace of the referent, defaults to the Kustomization namespace	string	false

11.1.1.9 GenericAppSpec

GenericAppSpec defines the actual Application spec.

Field	Description	Scheme	Required
appId	AppID specifies the name of the application/workload	string	true
gitRepositoryRef	GitRepository is reference to the Flux's GitRepository, which in turn describes Git repository where the service resides	CrossNamespaceGitRepositoryReference (see page 1397)	true
version	Version depicts the version of the service in the semantic versioning format.	string	true

11.1.1.10 TypedLocalObjectReference

TypedLocalObjectReference contains enough information to let you locate the typed referenced object at cluster or local level.

Field	Description	Scheme	Required
apiVersion	API version of the referent	string	false
kind	Kind of the referent	string	true
name	Name of the referent	string	true

11.1.2 apps.kommander.mesosphere.io/v1alpha3

11.1.2.1 App

App is the Schema for a Service or Application in Kommander.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1028]	false
spec		GenericAppSpec (see page 1404)	false
status		object	false

11.1.2.2 AppDeployment

AppDeployment is the Schema for a concrete installation of a Service or Application in Kommander.

1028 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1029]	false
spec		AppDeploymentSpec (see page 1400)	false
status		object	false

11.1.2.3 AppDeploymentClusterCondition

Field	Description	Scheme	Required
lastTransitionTime	LastTransitionTime is the last time the condition transitioned from one status to another.	metav1.Time[1030]	false
status	Status of the condition, one of True, False, Unknown	string	true
type	Type indicates type of condition in CamelCase or in foo.example.com/CamelCase.	string	true

11.1.2.4 AppDeploymentList

AppDeploymentList contains a list of AppDeployments.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1031]	false

1029 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1030 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#time-v1-meta
1031 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

Field	Description	Scheme	Required
items		[...]AppDeployment (see page 1398)	true

11.1.2.5 AppDeploymentSpec

AppDeploymentSpec defines an instance of an Application.

Field	Description	Scheme	Required
appRef	AppRef provides reference to a ClusterApp or App object.	TypedLocalObjectReference (see page 1404)	true
clusterConfigOverrides	ClusterConfigOverrides defines a list of config overrides configmaps and label selectors to select clusters on which to apply the config overrides.	[...]ClusterConfigOverrides (see page 1402)	false
clusterSelector	ClusterSelector defines the label selectors to select clusters on which to enable the AppDeployment.	metav1.LabelSelector[1032]	false
configOverrides	ConfigOverrides allows a user to define a ConfigMap that contains configuration overrides at the workspace level.	corev1.LocalObjectReference[1033]	false

11.1.2.6 AppDeploymentStatus

AppDeploymentStatus defines the current state of the AppDeployment.

1032 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#labelselector-v1-meta
1033 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

Field	Description	Scheme	Required
clusters	Clusters tracks clusters that have the AppDeployment enabled along with a ref to the cluster config overrides configmap in the management cluster. Existence of cluster means that the cluster's default AppDeployment state has been processed. Nonexistence of cluster means that the cluster was newly attached and needs to be selected by default to enable the AppDeployment on it.	[...]ClusterDeploymentStatus (see page 1403)	false
observedGeneration	ObservedGeneration is the most recent generation observed for this AppDeployment. It corresponds to the AppDeployment's generation, which is updated on mutation by the API Server.	int64	false

11.1.2.7 AppList

AppList contains a list of Apps.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1034]	false
items		[...]App (see page 1398)	true

1034 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

11.1.2.8 ClusterApp

ClusterApp is the Schema for a Service or Application in Kommander.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1035]	false
spec		GenericAppSpec (see page 1404)	false
status		object	false

11.1.2.9 ClusterAppList

ClusterAppList contains a list of ClusterApps.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1036]	false
items		[...]ClusterApp (see page 1402)	true

11.1.2.10 ClusterConfigOverrides

ClusterConfigOverrides defines the cluster config overrides configmap and label selector to select clusters on which to apply the overrides configmap.

Field	Description	Scheme	Required
clusterSelector	ClusterSelector defines the label selectors to select clusters on which to deploy the configmap.	metav1.LabelSelector[1037]	false

1035 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1036 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1037 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#labelselector-v1-meta

Field	Description	Scheme	Required
configMapName	ConfigMapName is the name of the cluster config overrides configmap.	string	true

11.1.2.11 ClusterDeploymentStatus

Field	Description	Scheme	Required
clusterConfigOverrides Ref	ClusterConfigOverrides Ref is set to reference the overrides ConfigMap in the management cluster	corev1.LocalObjectReference[1038]	false
conditions		[...]AppDeploymentClusterCondition (see page 1399)	false
name		string	true

11.1.2.12 CrossNamespaceGitRepositoryReference

CrossNamespaceGitRepositoryReference contains enough information to let you locate the typed referenced object at cluster level.

Field	Description	Scheme	Required
apiVersion	API version of the referent	string	false
kind	Kind of the referent	string	true
name	Name of the referent	string	true

1038 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

Field	Description	Scheme	Required
namespace	Namespace of the referent, defaults to the Kustomization namespace	string	false

11.1.2.13 GenericAppSpec

GenericAppSpec defines the actual Application spec.

Field	Description	Scheme	Required
appId	AppID specifies the name of the application/workload	string	true
gitRepositoryRef	GitRepository is reference to the Flux's GitRepository, which in turn describes Git repository where the service resides	CrossNamespaceGitRepositoryReference (see page 1403)	true
version	Version depicts the version of the service in the semantic versioning format.	string	true

11.1.2.14 TypedLocalObjectReference

TypedLocalObjectReference contains enough information to let you locate the typed referenced object at cluster or local level.

Field	Description	Scheme	Required
apiVersion	API version of the referent	string	false
kind	Kind of the referent	string	true

Field	Description	Scheme	Required
name	Name of the referent	string	true

11.1.3 dispatch.d2iq.io/v1alpha2

11.1.3.1 GitopsRepository

GitopsRepository represents an SCM repository that backs a gitops resource. A gitops resource can be a FluxCD HelmRelease or Kustomization resource.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1039]	false
spec		GitopsRepositorySpec (see page 1405)	true

11.1.3.2 GitopsRepositoryList

GitopsRepositoryList contains a list of Repository.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1040]	false
items		[...]GitopsRepository (see page 1405)	true

11.1.3.3 GitopsRepositorySpec

GitopsRepositorySpec defines the desired state of GitopsRepository.

1039 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1040 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

Field	Description	Scheme	Required
cloneUrl	URL to clone the repository from.	string	false
secret	Reference to the secret to use when interacting with the Git provider API. The secret should contain the following fields: username: the username (if password is not a token). password: the password or token (required).	string	false
template	Template of the gitops resource backing the repository.	GitopsRolloutTemplate (see page 1406)	false

11.1.3.4 GitopsRolloutTemplate

Field	Description	Scheme	Required
path	Root path to fetch manifests from in the Git repository.	string	false
ref	The Git reference to checkout and monitor for changes, defaults to master branch. This field supersedes the `revision` field in v1alpha1 API and is a breaking change.	sourcectrlv1beta1.GitRepositoryRef[1041]	false
suspend	Whether to suspend periodic or webhook-notified sync.	bool	false

1041 https://fluxcd.io/flux/components/source/api/#source.toolkit.fluxcd.io/v1beta1.GitRepositoryRef

11.1.4 kommander.mesosphere.io/v1beta1

11.1.4.1 CAPIClusterReference

Field	Description	Scheme	Required
name		string	true
namespace		string	false

11.1.4.2 ClusterReference

ClusterReference holds a single reference to clusters provisioned via Kommander. Only one field is allowed to be set. Currently, only CAPI clusters are creatable, but this is left extensible for other provider types in the future.

Field	Description	Scheme	Required
capiCluster		CAPIClusterReference (see page 1407)	false

11.1.4.3 GenericClusterReference

GenericClusterReference sets the name of the cluster.

Field	Description	Scheme	Required
name		string	true

11.1.4.4 IngressSpec

IngressSpec holds settings for the cluster's Kommander managed Ingress.

Field	Description	Scheme	Required
certificateSecretRef	CertificateSecretRef is a reference to a secret of type TLS that holds a TLS certificate, private key and CA certificate. The certificate must be valid for the Ingress hostname. The secret must be located in the workspace's namespace on the target cluster.	corev1.LocalObjectReference[1042]	false
hostname	Hostname can be set to configure the cluster's Ingress with a custom domain name.	string	false
issuerRef	IssuerRef is a reference to an `Issuer` or `ClusterIssuer` on the target cluster to be used to create a `Certificate` for the cluster's Ingress. If the type is an Issuer, it must be located in the workspace's namespace on the cluster.	IssuerReference (see page 1409)	false

11.1.4.5 IngressStatus

IngressSpec holds information about the cluster's Kommander managed Ingress.

1042 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

Field	Description	Scheme	Required
address	Address is the main DNS name or IP address of the cluster's Ingress that should be used to access services on the cluster. If the field is empty, it means that the load balancer hasn't been populated yet.	string	false
caBundle	CABundle is a PEM encoded CA bundle which should be used to verify the TLS connection for the Ingress-served end-entity certificate.	[...]byte	false
useSystemCA	UseSystemCA is set to true if the Ingress end-entity certificate was issued by a public CA and it is expected that the root CA cert is included in a OS CA bundle (which should be used for TLS verification in that case). If this field is `true` the `CABundle` field is empty. default: false	bool	false

11.1.4.6 IssuerReference

IssuerReference is a reference to an issuer with a given name, kind and group.

Field	Description	Scheme	Required
group	Group of the issuer being referred to.	string	false

Field	Description	Scheme	Required
kind	Kind of the issuer being referred to.	string	false
name	Name of the issuer being referred to.	string	true

11.1.4.7 KommanderCluster

KommanderCluster is the Schema for the kommander clusters API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1043]	false
spec		KommanderClusterSpec (see page 1410)	false
status		KommanderClusterStatus (see page 1411)	false

11.1.4.8 KommanderClusterList

KommanderClusterList contains a list of Cluster.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1044]	false
items		[...]KommanderCluster (see page 1410)	true

11.1.4.9 KommanderClusterSpec

KommanderClusterSpec defines the desired state of Cluster.

1043 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1044 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

Field	Description	Scheme	Required
clusterRef		ClusterReference (see page 1407)	false
clusterTunnelConnectorRef	ClusterTunnelConnectorRef is a reference to TunnelConnector that should be used for connecting to cluster.	corev1.LocalObjectReference[1045]	false
ingress	Ingress configures the Kommander managed Ingress. If no value is provided, the default ingress will be provisioned.	IngressSpec (see page 1407)	false
kubeconfigRef		corev1.LocalObjectReference[1046]	false
namespaceRef	NamespaceRef will override the namespace on target cluster for the given Workspace. This is used in special circumstances such as when an essential cluster is attached and thus implicitly demoted from being a management cluster to a managed cluster thus forcing the kommander namespace to be recycled as the underlying workspace namespace.	corev1.LocalObjectReference[1047]	false

11.1.4.10 KommanderClusterStatus

KommanderClusterStatus defines the observed state of Cluster.

1045 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1046 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1047 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

Field	Description	Scheme	Required
type	ClusterType represents the type of the cluster. E.g. EKS, GKE, etc.	string	false
conditions		[...] metav1.Condition[1048]	false
dextfaclientRef	DexTFAClientRef holds a reference to a DexClient provisioned for Traefik Forward Auth running on managed cluster.	corev1.ObjectReference[1049]	false
ingress	Ingress holds information about the cluster's Ingress	IngressStatus (see page 1408)	false
kubefedclusterRef	KubefedClusterRef holds a reference to a kubefedcluster in the kubefed system namespace.	corev1.LocalObjectReference[1050]	false
clusterId	KubernetesClusterID is the stable cluster ID of the Kubernetes cluster that this Kommander cluster represents.	string	false
kubernetesVersion	KubernetesVersion is the Kubernetes version of the cluster.	string	false
phase	Phase represents the current phase of cluster actuation. E.g. Pending, Provisioning, Provisioned, Deleting, Failed, etc.	string	false

1048 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#condition-v1-meta
1049 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectreference-v1-core
1050 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

Field	Description	Scheme	Required
serviceEndpoints	ServiceEndpoints will be the addresses assigned to the Kubernetes exposed services	object	false

11.1.4.11 License

License is the Schema for the licenses API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1051]	false
spec		LicenseSpec (see page 1414)	false
status		LicenseStatus (see page 1415)	false

11.1.4.12 LicenseCondition

Field	Description	Scheme	Required
lastTransitionTime		metav1.Time[1052]	false
message		string	false
reason		string	false
status		string	true
type		string	true

1051 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1052 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#time-v1-meta

11.1.4.13 LicenseExternalAWS

Field	Description	Scheme	Required
licenseArn	LicenseArn is a fully qualified AWS arn to a license for Kommander.	string	false

11.1.4.14 LicenseExternalReference

Field	Description	Scheme	Required
awsLicense	AWSLicense holds a single reference to a license in AWS's License Manager	LicenseExternalAWS (see page 1414)	false
type	Type is the source of the external license referenced, i.e. AWS, GCP, Azure	string	true

11.1.4.15 LicenseList

LicenseList contains a list of License.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1053]	false
items		[...]License (see page 1413)	true

11.1.4.16 LicenseSpec

LicenseSpec defines the desired state of License.

1053 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

Field	Description	Scheme	Required
externalLicenseRef	ExternalLicenseRef holds a reference to an external license	LicenseExternalReference (see page 1414)	false
licenseRef	LicenseReference holds a single reference to the secret holding the license JWT	corev1.LocalObjectReference[1054]	false

11.1.4.17 LicenseStatus

LicenseStatus defines the observed state of License.

Field	Description	Scheme	Required
clusterCapacity	Maximum number of clusters that the license allows.	int32	true
conditions	Conditions relevant to the license (currently used to track term breaches)	[...]LicenseCondition (see page 1413)	false
coreCapacity	Maximum number of cores that the license allows.	int32	true
customerId	The customer's ID. This is the customer name provided from Salesforce.	string	true
dkpLevel	The DKP license level.	string	true
endDate	End date of the licensing period.	metav1.Time[1055]	false

1054 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1055 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#time-v1-meta

Field	Description	Scheme	Required
licenseId	The license's ID as provided from Salesforce.	string	true
productName	The product name the license is for.	string	true
productSKU	The product SKU the license is for.	string	true
startDate	Start date of the licensing period.	metav1.Time[1056]	false
valid	Indicates whether the license is valid, i.e. the secret containing the JWT exists and the JWT carries a valid D2iQ signature. This does NOT indicate whether the license has expired or terms have been breached.	bool	true
version	The license's version ID as provided when the license was created.	string	true

11.1.4.18 PlacementSelector

PlacementSelector defines the fields to select clusters where to apply the project.

Field	Description	Scheme	Required
clusterSelector		metav1.LabelSelector[1057]	false
clusters		[...]GenericClusterReference (see page 1407)	false

1056 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#time-v1-meta
1057 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#labelselector-v1-meta

11.1.4.19 VirtualGroup

VirtualGroup is the Schema for the virtualgroups API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1058]	false
spec		VirtualGroupSpec (see page 1418)	false

11.1.4.20 VirtualGroupClusterRoleBinding

VirtualGroupClusterRoleBinding is the Schema for the virtualgroupclusterrolebindings API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1059]	false
spec		VirtualGroupClusterRoleBindingSpec (see page 1418)	true

11.1.4.21 VirtualGroupClusterRoleBindingList

VirtualGroupClusterRoleBindingList contains a list of VirtualGroupClusterRoleBinding.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1060]	false
items		[...]VirtualGroupClusterRoleBinding (see page 1417)	true

1058 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1059 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1060 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

11.1.4.22 VirtualGroupClusterRoleBindingSpec

VirtualGroupClusterRoleBindingSpec defines the desired state of VirtualGroupClusterRoleBinding.

Field	Description	Scheme	Required
clusterRoleRef		corev1.LocalObjectReference[1061]	true
placement		PlacementSelector (see page 1416)	false
virtualGroupRef		corev1.LocalObjectReference[1062]	true

11.1.4.23 VirtualGroupList

VirtualGroupList contains a list of VirtualGroup.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1063]	false
items		[...]VirtualGroup (see page 1417)	true

11.1.4.24 VirtualGroupSpec

VirtualGroupSpec defines the desired state of VirtualGroup.

Field	Description	Scheme	Required
subjects		[...]rbacv1.Subject[1064]	false

1061 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1062 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1063 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1064 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#subject-v1-rbac-authorization-k8s-io

11.1.5 workspaces.kommander.mesosphere.io/v1alpha1

11.1.5.1 KommanderProjectRole

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1065]	false
spec		KommanderProjectRoleSpec (see page 1419)	false
status		KommanderProjectRoleStatus (see page 1420)	false

11.1.5.2 KommanderProjectRoleList

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1066]	false
items		[...]KommanderProjectRole (see page 1419)	true

11.1.5.3 KommanderProjectRoleSpec

Field	Description	Scheme	Required
projectObjectVerbs		[...]string	false
rules		[...] rbacv1.PolicyRule[1067]	false

1065 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1066 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1067 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#policyrule-v1-rbac-authorization-k8s-io

11.1.5.4 KommanderProjectRoleStatus

Field	Description	Scheme	Required
roleInProjectRef		corev1.LocalObjectReference[1068]	false
roleInWorkspaceRef		corev1.LocalObjectReference[1069]	false

11.1.5.5 KommanderWorkspaceRole

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1070]	false
spec		KommanderWorkspaceRoleSpec (see page 1421)	false
status		KommanderWorkspaceRoleStatus (see page 1421)	false

11.1.5.6 KommanderWorkspaceRoleList

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1071]	false
items		[...]KommanderWorkspaceRole (see page 1420)	true

1068 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1069 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1070 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1071 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

11.1.5.7 KommanderWorkspaceRoleSpec

Field	Description	Scheme	Required
rules		[...] rbacv1.PolicyRule[1072]	false
workspaceObjectVerbs		[...]string	false

11.1.5.8 KommanderWorkspaceRoleStatus

Field	Description	Scheme	Required
clusterRoleRef		corev1.LocalObjectReference[1073]	false
roleInWorkspaceRef		corev1.LocalObjectReference[1074]	false

11.1.5.9 Project

Project is a logical top-level container for a set of Kommander resources.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1075]	false
spec		ProjectSpec (see page 1424)	false
status		ProjectStatus (see page 1424)	false

1072 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#policyrule-v1-rbac-authorization-k8s-io
1073 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1074 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1075 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta

11.1.5.10 ProjectCondition

Field	Description	Scheme	Required
lastTransitionTime	Last time the condition transitioned from one status to another.	metav1.Time[1076]	false
message	A human readable message indicating details about the transition.	string	false
reason	The reason for the condition's last transition.	string	false
status	Status of the condition, one of True, False, Unknown.	string	true
type	Type of project condition.	ProjectConditionType	true

11.1.5.11 ProjectList

ProjectList is a list of Project objects.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1077]	false
items		[...]Project (see page 1421)	true

1076 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#time-v1-meta
1077 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

11.1.5.12 ProjectRole

ProjectRole is the Schema for the workspaces ProjectRole API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1078]	false
spec		ProjectRoleSpec (see page 1423)	false
status		ProjectRoleStatus (see page 1424)	false

11.1.5.13 ProjectRoleList

ProjectRoleList contains a list of ProjectRole.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1079]	false
items		[...]ProjectRole (see page 1423)	true

11.1.5.14 ProjectRoleSpec

Field	Description	Scheme	Required
rules		[...] rbacv1.PolicyRule[1080]	false

1078 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1079 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1080 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#policyrule-v1-rbac-authorization-k8s-io

11.1.5.15 ProjectRoleStatus

Field	Description	Scheme	Required
federatedRoleRef		corev1.LocalObjectReference[1081]	false

11.1.5.16 ProjectSpec

ProjectSpec describes the attributes on a Project.

Field	Description	Scheme	Required
namespaceName	NamespaceName specifies the optional namespace name to use for the project. This field is immutable, only settable on create.	string	false
placement		v1beta1.PlacementSelector[1082]	false
workspaceRef		corev1.LocalObjectReference[1083]	false

11.1.5.17 ProjectStatus

ProjectStatus is information about the current status of a Project.

1081 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1082 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/110854693/kommander.mesosphere.io+v1beta1#PlacementSelector
1083 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

Field	Description	Scheme	Required
conditions	Represents the latest available observations of a project's current state.	[...]ProjectCondition (see page 1422)	false
namespaceRef		corev1.LocalObjectReference[1084]	false

11.1.5.18 VirtualGroupKommanderClusterRoleBinding

VirtualGroupKommanderClusterRoleBinding is the Schema for the VirtualGroupWorkspaceRoleBinding API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1085]	false
spec		VirtualGroupKommanderClusterRoleBindingSpec (see page 1426)	true
status		VirtualGroupKommanderClusterRoleBindingStatus (see page 1426)	false

11.1.5.19 VirtualGroupKommanderClusterRoleBindingList

VirtualGroupKommanderClusterRoleBindingList contains a list of VirtualGroupKommanderClusterRoleBinding.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1086]	false

1084 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1085 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1086 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

Field	Description	Scheme	Required
items		[...]VirtualGroupKommanderClusterRoleBinding (see page 1425)	true

11.1.5.20 VirtualGroupKommanderClusterRoleBindingSpec

Field	Description	Scheme	Required
clusterRoleRef		corev1.LocalObjectReference[1087]	true
virtualGroupRef		corev1.LocalObjectReference[1088]	true

11.1.5.21 VirtualGroupKommanderClusterRoleBindingStatus

Field	Description	Scheme	Required
clusterRoleBindingRef		corev1.LocalObjectReference[1089]	false

11.1.5.22 VirtualGroupKommanderProjectRoleBinding

VirtualGroupKommanderProjectRoleBinding is the Schema to be used in the API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1090]	false

1087 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1088 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1089 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1090 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta

Field	Description	Scheme	Required
spec		VirtualGroupKommanderProjectRoleBindingSpec (see page 1427)	true
status		VirtualGroupKommanderProjectRoleBindingStatus (see page 1428)	false

11.1.5.23 VirtualGroupKommanderProjectRoleBindingList

VirtualGroupKommanderProjectRoleBindingList contains a list of VirtualGroupKommanderProjectRoleBinding.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1091]	false
items		[...]VirtualGroupKommanderProjectRoleBinding (see page 1426)	true

11.1.5.24 VirtualGroupKommanderProjectRoleBindingSpec

Field	Description	Scheme	Required
kommanderProjectRoleRef		corev1.LocalObjectReference[1092]	true
virtualGroupRef		corev1.LocalObjectReference[1093]	true

1091 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1092 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1093 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

11.1.5.25 VirtualGroupKommanderProjectRoleBindingStatus

Field	Description	Scheme	Required
roleBindingInProjectRef		corev1.LocalObjectReference[1094]	false
roleBindingInWorkspaceRef		corev1.LocalObjectReference[1095]	false

11.1.5.26 VirtualGroupKommanderWorkspaceRoleBinding

VirtualGroupKommanderWorkspaceRoleBinding is the Schema to be used in the API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1096]	false
spec		VirtualGroupKommanderWorkspaceRoleBindingSpec (see page 1429)	true
status		VirtualGroupKommanderWorkspaceRoleBindingStatus (see page 1429)	false

11.1.5.27 VirtualGroupKommanderWorkspaceRoleBindingList

VirtualGroupKommanderWorkspaceRoleBindingList contains a list of VirtualGroupKommanderWorkspaceRoleBinding.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1097]	false

1094 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1095 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1096 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1097 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

Field	Description	Scheme	Required
items		[...]VirtualGroupKommanderWorkspaceRoleBinding (see page 1428)	true

11.1.5.28 VirtualGroupKommanderWorkspaceRoleBindingSpec

Field	Description	Scheme	Required
kommanderWorkspaceRoleRef		corev1.LocalObjectReference[1098]	true
virtualGroupRef		corev1.LocalObjectReference[1099]	true

11.1.5.29 VirtualGroupKommanderWorkspaceRoleBindingStatus

Field	Description	Scheme	Required
clusterRoleBindingRef		corev1.LocalObjectReference[1100]	false
roleBindingInWorkspaceRef		corev1.LocalObjectReference[1101]	false

11.1.5.30 VirtualGroupProjectRoleBinding

VirtualGroupProjectRoleBinding is the Schema for the VirtualGroupProjectRoleBinding API.

1098 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1099 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1100 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1101 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1102]	false
spec		VirtualGroupProjectRoleBindingSpec (see page 1430)	true
status		VirtualGroupProjectRoleBindingStatus (see page 1431)	false

11.1.5.31 VirtualGroupProjectRoleBindingList

VirtualGroupProjectRoleBindingList contains a list of VirtualGroupProjectRoleBinding.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1103]	false
items		[...]VirtualGroupProjectRoleBinding (see page 1429)	true

11.1.5.32 VirtualGroupProjectRoleBindingSpec

Field	Description	Scheme	Required
projectRoleRef		corev1.LocalObjectReference[1104]	false
virtualGroupRef		corev1.LocalObjectReference[1105]	true

1102 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1103 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1104 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1105 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

Field	Description	Scheme	Required
workspaceRoleRef	WorkspaceRoleRef maybe a LocalObjectReference but the WorkspaceRole is not created in project namespace but in Workspace namespace. `Local` in LocalObjectReference means `Local to project's workspace` since there can only be one workspace the project is in.	corev1.LocalObjectReference[1106]	false

11.1.5.33 VirtualGroupProjectRoleBindingStatus

Field	Description	Scheme	Required
federatedRoleBindingRef		corev1.LocalObjectReference[1107]	false

11.1.5.34 VirtualGroupWorkspaceRoleBinding

VirtualGroupWorkspaceRoleBinding is the Schema for the VirtualGroupWorkspaceRoleBinding API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1108]	false
spec		VirtualGroupWorkspaceRoleBindingSpec (see page 1432)	true

1106 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1107 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1108 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta

Field	Description	Scheme	Required
status		VirtualGroupWorkspaceRoleBindingStatus (see page 1433)	false

11.1.5.35 VirtualGroupWorkspaceRoleBindingList

VirtualGroupWorkspaceRoleBindingList contains a list of VirtualGroupWorkspaceRoleBinding.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1109]	false
items		[...]VirtualGroupWorkspaceRoleBinding (see page 1431)	true

11.1.5.36 VirtualGroupWorkspaceRoleBindingSpec

Field	Description	Scheme	Required
placement		v1beta1.PlacementSelector[1110]	false
virtualGroupRef		corev1.LocalObjectReference[1111]	true
workspaceRoleRef		corev1.LocalObjectReference[1112]	true

1109 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1110 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/110854693/kommander.mesosphere.io+v1beta1#PlacementSelector
1111 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1112 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

11.1.5.37 VirtualGroupWorkspaceRoleBindingStatus

Field	Description	Scheme	Required
federatedClusterRoleBindingRef		corev1.LocalObjectReference[1113]	false

11.1.5.38 Workspace

Workspace is the Schema for the workspaces API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1114]	false
spec		WorkspaceSpec (see page 1436)	false
status		WorkspaceStatus (see page 1436)	false

11.1.5.39 WorkspaceCondition

Field	Description	Scheme	Required
lastTransitionTime	Last time the condition transitioned from one status to another.	metav1.Time[1115]	false
message	A human readable message indicating details about the transition.	string	false

1113 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1114 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1115 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#time-v1-meta

Field	Description	Scheme	Required
reason	The reason for the condition's last transition.	string	false
status	Status of the condition, one of True, False, Unknown.	string	true
type	Type of workspace condition.	string	true

11.1.5.40 WorkspaceList

WorkspaceList contains a list of Workspace.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1116]	false
items		[...]Workspace (see page 1433)	true

11.1.5.41 WorkspaceRole

WorkspaceRole is the Schema for the workspaces API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1117]	false
spec		WorkspaceRoleSpec (see page 1435)	false
status		WorkspaceRoleStatus (see page 1435)	false

1116 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1117 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta

11.1.5.42 WorkspaceRoleList

WorkspaceRoleList contains a list of WorkspaceRole.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1118]	false
items		[...]WorkspaceRole (see page 1434)	true

11.1.5.43 WorkspaceRoleSpec

Field	Description	Scheme	Required
aggregationRule		rbacv1.AggregationRule[1119]	false
rules		[...] rbacv1.PolicyRule[1120]	false

11.1.5.44 WorkspaceRoleStatus

Field	Description	Scheme	Required
federatedClusterRoleRef		corev1.LocalObjectReference[1121]	false

1118 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1119 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#aggregationrule-v1-rbac-authorization-k8s-io
1120 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#policyrule-v1-rbac-authorization-k8s-io
1121 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

11.1.5.45 WorkspaceSpec

Field	Description	Scheme	Required
clusterLabels		object	false
namespaceName	NamespaceName specifies the optional namespace name to use for the workspace. This field is immutable, only settable on create.	string	false

11.1.5.46 WorkspaceStatus

Field	Description	Scheme	Required
conditions	Represents the latest available observations of a workspace's current state.	[...]WorkspaceCondition (see page 1433)	false
namespaceRef		corev1.LocalObjectReference[1122]	false

11.2 CLI Commands

11.2.1 CLI Commands for DKP

The table of contents on the left lists the command groupings. Inside each section, you will find the individual commands for most actions. For KIB CLI, refer to the documentation section for Konvoy Image Builder CLI (see page 1312)commands.

- dkp attach (see page 1437)
- dkp check (see page 1438)
- dkp completion (see page 1440)
- dkp config (see page 1445)

1122 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

- dkp create (see page 1448)
- dkp delete (see page 1484)
- dkp describe (see page 1488)
- dkp detach (see page 1489)
- dkp diagnose (see page 1490)
- dkp get (see page 1493)
- dkp import (see page 1497)
- dkp install (see page 1498)
- dkp move (see page 1500)
- dkp open (see page 1501)
- dkp push (see page 1502)
- dkp scale (see page 1505)
- dkp serve (see page 1506)
- dkp update (see page 1507)
- dkp upgrade (see page 1522)
- dkp edit (see page 1533)
- dkp version (see page 1534)
- dkp cluster (see page 1535)

11.2.2 dkp attach

Attach one of [cluster]

11.2.2.1 Options

```
      --config string            Config file to use (default "/root/.kommander/
config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for attach
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

11.2.2.2 SEE ALSO

- dkp attach cluster (see page 1437) - Attach a cluster

11.2.2.3 dkp attach cluster

Attach a cluster

```
dkp attach cluster -n NAME --attached-kubeconfig FILENAME [flags]
```

11.2.2.3.1 Options

```
      --attached-kubeconfig string   Path of the kubeconfig file of the cluster to be
attached
  -h, --help                         help for cluster
  -n, --name string                  Desired name of the attached cluster
  -w, --workspace string             Name of the workspace of the attached cluster
```

11.2.2.3.2 Options inherited from parent commands

```
      --config string            Config file to use (default "/root/.kommander/
config")
      --context string           The name of the kubeconfig context to use
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

11.2.2.3.3 SEE ALSO

- dkp attach (see page 1437) - Attach one of [cluster]

11.2.3 dkp check

Check, one of [cluster]

11.2.3.1 Options

```
  -h, --help          help for check
  -v, --verbose int   Output verbosity
```

11.2.3.2 SEE ALSO

- dkp check cluster (see page 1439) - Check a cluster, one of [fips]

11.2.3.3 dkp check cluster

Check a cluster, one of [fips]

11.2.3.3.1 Options

```
  -h, --help   help for cluster
```

11.2.3.3.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.3.3.3 SEE ALSO

- dkp check (see page 1438) - Check, one of [cluster]
- dkp check cluster fips (see page 1439) - Validate the components in your cluster are FIPS compliant

11.2.3.3.4 dkp check cluster fips

Validate the components in your cluster are FIPS compliant

11.2.3.3.4.1 Synopsis

The check cluster fips command is used to validate that specific components and services are FIPS compliant by checking the signatures of the files against a signed signature file, and checking that services are using the certified algorithms.

Examples:

```
To use the built-in signature files for supported operating systems:

dkp check cluster fips

To use a custom signature file, named "manifest-rhel-84.json.asc":

dkp check cluster fips \
    --signature-file manifest-rhel-84.json.asc \
    --signature-configmap myconfigmap

The file will be copied to the ConfigMap. To use an existing ConfigMap:
```

```
dkp check cluster fips \
    --signature-configmap myconfigmap

The validation will be re-checked against the existing signature data.
```

```
dkp check cluster fips [flags]
```

11.2.3.3.4.2 Options

```
  -h, --help                          help for fips
      --kubeconfig string             Path to the kubeconfig file for the fips
cluster. If unspecified, default discovery rules apply.
  -n, --namespace string              If present, the namespace scope for this CLI
request. (default "default")
      --output-configmap string       ConfigMap to store result of the fips check.
(default "check-cluster-fips-output") (DEPRECATED: This flag will be removed in a
future release.)
      --signature-configmap string    ConfigMap with fips signature data to verify.
      --signature-file string         File containing fips signature data.
      --timeout duration              The length of time to wait before giving up.
Zero means wait forever (e.g. 300s, 30m, 3h). (default 10m0s)
```

11.2.3.3.4.3 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.3.3.4.4 SEE ALSO

- dkp check cluster (see page 1439) - Check a cluster, one of [fips]

11.2.4 dkp completion

Generate the autocompletion script for the specified shell

11.2.4.1 Synopsis

Generate the autocompletion script for dkp for the specified shell. See each sub-command's help for details on how to use the generated script.

11.2.4.2 Options

```
  -h, --help   help for completion
```

11.2.4.3 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.4.4 SEE ALSO

- dkp completion bash (see page 1442) - Generate the autocompletion script for bash
- dkp completion fish (see page 1441) - Generate the autocompletion script for fish
- dkp completion powershell (see page 1444) - Generate the autocompletion script for powershell
- dkp completion zsh (see page 1443) - Generate the autocompletion script for zsh

11.2.4.5 dkp completion fish

Generate the autocompletion script for fish

11.2.4.5.1 Synopsis

Generate the autocompletion script for the fish shell.

To load completions in your current shell session:

```
dkp completion fish | source
```

To load completions for every new session, execute once:

```
dkp completion fish > ~/.config/fish/completions/dkp.fish
```

You will need to start a new shell for this setup to take effect.

```
dkp completion fish [flags]
```

11.2.4.5.2 Options

```
  -h, --help              help for fish
      --no-descriptions   disable completion descriptions
```

11.2.4.5.3 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.4.5.4 SEE ALSO

- dkp completion (see page 1440) - Generate the autocompletion script for the specified shell

11.2.4.6 dkp completion bash

Generate the autocompletion script for bash

11.2.4.6.1 Synopsis

Generate the autocompletion script for the bash shell.

This script depends on the 'bash-completion' package. If it is not installed already, you can install it via your OS's package manager.

To load completions in your current shell session:

```
source <(dkp completion bash)
```

To load completions for every new session, execute once:

11.2.4.6.1.1 Linux:

```
dkp completion bash > /etc/bash_completion.d/dkp
```

11.2.4.6.1.2 macOS:

```
dkp completion bash > $(brew --prefix)/etc/bash_completion.d/dkp
```

You will need to start a new shell for this setup to take effect.

```
dkp completion bash
```

11.2.4.6.2 Options

```
  -h, --help              help for bash
      --no-descriptions   disable completion descriptions
```

11.2.4.6.3 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.4.6.4 SEE ALSO

- dkp completion (see page 1440) - Generate the autocompletion script for the specified shell

11.2.4.7 dkp completion zsh

Generate the autocompletion script for zsh

11.2.4.7.1 Synopsis

Generate the autocompletion script for the zsh shell.

If shell completion is not already enabled in your environment you will need to enable it. You can execute the following once:

```
echo "autoload -U compinit; compinit" >> ~/.zshrc
```

To load completions in your current shell session:

```
source <(dkp completion zsh); compdef _dkp dkp
```

To load completions for every new session, execute once:

11.2.4.7.1.1 Linux:

```
dkp completion zsh > "${fpath[1]}/_dkp"
```

11.2.4.7.1.2 macOS:

```
dkp completion zsh > $(brew --prefix)/share/zsh/site-functions/_dkp
```

You will need to start a new shell for this setup to take effect.

```
dkp completion zsh [flags]
```

11.2.4.7.2 Options

```
  -h, --help              help for zsh
      --no-descriptions   disable completion descriptions
```

11.2.4.7.3 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.4.7.4 SEE ALSO

- dkp completion (see page 1440) - Generate the autocompletion script for the specified shell

11.2.4.8 dkp completion powershell

Generate the autocompletion script for powershell

11.2.4.8.1 Synopsis

Generate the autocompletion script for powershell.

To load completions in your current shell session:

```
dkp completion powershell | Out-String | Invoke-Expression
```

To load completions for every new session, add the output of the above command to your powershell profile.

```
dkp completion powershell [flags]
```

11.2.4.8.2 Options

```
  -h, --help              help for powershell
      --no-descriptions   disable completion descriptions
```

11.2.4.8.3 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.4.8.4 SEE ALSO

- dkp completion (see page 1440) - Generate the autocompletion script for the specified shell

11.2.5 dkp config

Manage DKP Kommander's configuration

11.2.5.1 Options

```
      --config string            Config file to use (default "/root/.kommander/
config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for config
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

11.2.5.2 SEE ALSO

- dkp config get (see page 1446) - Retrieve DKP Kommander's configuration

- dkp config set (see page 1447) - Modify DKP Kommander's configuration

11.2.5.3 dkp config get

Retrieve DKP Kommander's configuration

11.2.5.3.1 Options

```
  -h, --help   help for get
```

11.2.5.3.2 Options inherited from parent commands

```
      --config string            Config file to use (default "/root/.kommander/
config")
      --context string           The name of the kubeconfig context to use
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

11.2.5.3.3 SEE ALSO

- dkp config (see page 1445) - Manage DKP Kommander's configuration
- dkp config get default-workspace (see page 1446) - Displays the name of the configured default Workspace

11.2.5.3.4 dkp config get default-workspace

Displays the name of the configured default Workspace

```
dkp config get default-workspace [flags]
```

11.2.5.3.4.1 Options

```
  -h, --help   help for default-workspace
```

11.2.5.3.4.2 Options inherited from parent commands

```
      --config string            Config file to use (default "/root/.kommander/
config")
      --context string           The name of the kubeconfig context to use
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

11.2.5.3.4.3 SEE ALSO

- dkp config get (see page 1446) - Retrieve DKP Kommander's configuration

11.2.5.4 dkp config set

Modify DKP Kommander's configuration

11.2.5.4.1 Options

```
  -h, --help   help for set
```

11.2.5.4.2 Options inherited from parent commands

```
      --config string            Config file to use (default "/root/.kommander/
config")
      --context string           The name of the kubeconfig context to use
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

11.2.5.4.3 SEE ALSO

- dkp config (see page 1445) - Manage DKP Kommander's configuration
- dkp config set default-workspace (see page 1448) - Set the name of the default Workspace to use in all commands when none is provided

11.2.5.4.4 dkp config set default-workspace

Set the name of the default Workspace to use in all commands when none is provided

```
dkp config set default-workspace [WORKSPACE] [flags]
```

11.2.5.4.4.1 Options

```
  -h, --help   help for default-workspace
```

11.2.5.4.4.2 Options inherited from parent commands

```
      --config string            Config file to use (default "/root/.kommander/
config")
      --context string           The name of the kubeconfig context to use
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

11.2.5.4.4.3 SEE ALSO

- dkp config set (see page 1447) - Modify DKP Kommander's configuration

11.2.6 dkp create

Create one of [appdeployment, bootstrap, capi-components, chart-bundle, cluster, image-bundle, nodepool, workspace]

11.2.6.1 Options

```
  -h, --help   help for create
```

11.2.6.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.6.3 SEE ALSO

- dkp create appdeployment (see page 1450) - Create an AppDeployment
- dkp create bootstrap (see page 1452) - Create bootstrap cluster
- dkp create capi-components (see page 1451) - Create the CAPI components in the cluster
- dkp create chart-bundle (see page 1453) - Create charts bundle based on a catalog applications git repository
- dkp create cluster (see page 1454) - Create a Kubernetes cluster, one of [aks, aws, azure, eks, gcp, preprovisioned, vsphere]
- dkp create image-bundle (see page 1452) - Create an image bundle
- dkp create nodepool (see page 1472) - Create a nodepool, one of [aks, aws, azure, eks, gcp, preprovisioned, vsphere]
- dkp create workspace (see page 1449) - Create a Workspace

11.2.6.4 dkp create workspace

Create a Workspace

```
dkp create workspace WORKSPACE_NAME [flags]
```

11.2.6.4.1 Options

```
      --config string            Config file to use (default "/root/.kommander/
config")
      --context string           The name of the kubeconfig context to use
      --dry-run                  Export in YAML format to stdout
  -h, --help                     help for workspace
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
  -n, --namespace string         Name of the Namespace to create for the workspace
  -o, --output string            Output format. One of: yaml|json (default "yaml")
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

11.2.6.4.2 SEE ALSO

- dkp create (see page 1448) - Create one of [appdeployment, bootstrap, capi-components, chart-bundle, cluster, image-bundle, nodepool, workspace]

11.2.6.5 dkp create appdeployment

Create an AppDeployment

11.2.6.5.1 Synopsis

Creates an AppDeployment in a workspace or project.

When [--clusters C1,C2..] is not specified, it targets all existing clusters in the specified workspace or project.

```
dkp create appdeployment APPDEPLOYMENT_NAME --app NAME [flags]
```

11.2.6.5.2 Options

```
      --add-cluster-config-overrides strings   Comma-separated list of mappings of
kommanderCluster name to cluster configuration override ConfigMap name to apply to
the targeting clusters. (e.g., cluster-1:override-1-cm,cluster-2:override-2-cm)(only
valid for dkp clusters version 2.3.x and up) (default [])
  -a, --app string                             Name of the App to deploy
      --clusters strings                       List of names of kommanderClusters to
select for the command. (e.g., cluster-1,cluster-2)(only valid for dkp clusters
version 2.3.x and up) (default [])
      --config string                          Config file to use (default "/
root/.kommander/config")
  -c, --config-overrides string                Name of the ConfigMap used to override
default configuration of the App
      --context string                         The name of the kubeconfig context to
use
      --dry-run                                Export in YAML format to stdout
  -h, --help                                   help for appdeployment
      --kubeconfig string                      Path to the kubeconfig file to use for
 CLI requests.
  -o, --output string                          Output format. One of: yaml|json
(default "yaml")
  -p, --project string                         Name of the project to create the
AppDeployment in. Requires workspace flag (workspace that the project belongs to).
      --request-timeout string                 The length of time to wait before
giving up on a single server request. Non-zero values should contain a corresponding
time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int                            Output verbosity
```

```
  -w, --workspace string                    Name of the workspace to create the
AppDeployment in
```

11.2.6.5.3 SEE ALSO

- dkp create (see page 1448) - Create one of [appdeployment, bootstrap, capi-components, chart-bundle, cluster, image-bundle, nodepool, workspace]

11.2.6.6 dkp create capi-components

Create the CAPI components in the cluster

```
dkp create capi-components [flags]
```

11.2.6.6.1 Options

```
      --aws-service-endpoints string     Custom AWS service endpoints in a semi-colon
separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=${URL};$
{SigningRegion2}...
  -h, --help                             help for capi-components
      --http-proxy string                HTTP proxy for CAPI controllers
      --https-proxy string               HTTPS proxy for CAPI controllers
      --kubeconfig string                Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --no-proxy strings                 No Proxy list for CAPI controllers (default
 [])
      --timeout duration                 The length of time to wait before giving up.
Zero means wait forever (e.g. 300s, 30m, 3h). (default 10m0s)
  -v, --verbose int                      Output verbosity
      --wait                             If true, wait for operations to complete
before returning. (default true)
      --with-aws-bootstrap-credentials   Set true to use AWS bootstrap credentials
from your environment. When false, the instance profile of the EC2 instance where the
CAPA controller is scheduled on will be used instead.
      --with-gcp-bootstrap-credentials   Set true to use GCP bootstrap credentials
from your environment. When false, the service account of the VM instance where the
CAPG controller is scheduled on will be used instead.
```

11.2.6.6.2 SEE ALSO

- dkp create (see page 1448) - Create one of [appdeployment, bootstrap, capi-components, chart-bundle, cluster, image-bundle, nodepool, workspace]

11.2.6.7 dkp create image-bundle

Create an image bundle

```
dkp create image-bundle [flags]
```

11.2.6.7.1 Options

```
  -h, --help                     help for image-bundle
      --images-file string       File containing list of images to create bundle
from, either as YAML configuration or a simple list of images
      --output-file string       Output file to write image bundle to (default
"images.tar")
      --overwrite                Overwrite image bundle file if it already exists
      --platform platformSlice   platforms to download images (required format: <os>/
<arch>[/<variant>]) (default [linux/amd64])
  -v, --verbose int              Output verbosity
```

11.2.6.7.2 SEE ALSO

- dkp create (see page 1448) - Create one of [appdeployment, bootstrap, capi-components, chart-bundle, cluster, image-bundle, nodepool, workspace]

11.2.6.8 dkp create bootstrap

Create bootstrap cluster

```
dkp create bootstrap [flags]
```

11.2.6.8.1 Options

```
      --aws-service-endpoints string     Custom AWS service endpoints in a semi-colon
separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=${URL};$
{SigningRegion2}...
  -h, --help                             help for bootstrap
      --http-proxy string                HTTP proxy for CAPI controllers
      --https-proxy string               HTTPS proxy for CAPI controllers
      --kind-cluster-image string        Kind node image for the bootstrap cluster (d
efault "mesosphere/konvoy-bootstrap:v2.5.2")
```

```
      --kubeconfig string                Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --no-proxy strings                 No Proxy list for CAPI controllers (default
 [])
      --timeout duration                 The length of time to wait before giving up.
Zero means wait forever (e.g. 300s, 30m, 3h). (default 10m0s)
  -v, --verbose int                      Output verbosity
      --wait                             If true, wait for operations to complete
before returning. (default true)
      --with-aws-bootstrap-credentials   Set true to use AWS bootstrap credentials
from your environment. When false, the instance profile of the EC2 instance where the
CAPA controller is scheduled on will be used instead.
      --with-gcp-bootstrap-credentials   Set true to use GCP bootstrap credentials
from your environment. When false, the service account of the VM instance where the
CAPG controller is scheduled on will be used instead.
```

11.2.6.8.2 SEE ALSO

- dkp create (see page 1448) - Create one of [appdeployment, bootstrap, capi-components, chart-bundle, cluster, image-bundle, nodepool, workspace]

11.2.6.9 dkp create chart-bundle

Create charts bundle based on a catalog applications git repository

```
dkp create chart-bundle [flags]
```

11.2.6.9.1 Options

```
      --catalog-repository string         Git repository containing catalog
application definitions
      --config string                     Config file to use (default "/
root/.kommander/config")
      --context string                    The name of the kubeconfig context to use
      --dry-run                           Export in YAML format to stdout
      --extra-charts strings              Extra charts to include in the bundle, in
the format <repo-url1>|<chart-name1>,<repo-url2>|<chart-name2>:<chart-version2>,...
(default [])
  -h, --help                              help for chart-bundle
      --kommander-charts-version string   Kommander helm charts version to download.
Default: download all available versions
      --kubeconfig string                 Path to the kubeconfig file to use for CLI
requests.
  -o, --output string                     File path to write charts bundle to
(default "charts-bundle.tar.gz")
```

```
      --request-timeout string            The length of time to wait before giving up
on a single server request. Non-zero values should contain a corresponding time unit
(e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
      --skip-charts strings               Charts to not to include in the bundle, in
the format <chart-name1>,<chart-name2>:<chart-version2>,... (default [])
  -v, --verbose int                       Output verbosity
```

11.2.6.9.2 SEE ALSO

- dkp create (see page 1448) - Create one of [appdeployment, bootstrap, capi-components, chart-bundle, cluster, image-bundle, nodepool, workspace]

11.2.6.10 dkp create cluster

Create a Kubernetes cluster, one of [aks, aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.6.10.1 Options

```
  -h, --help          help for cluster
  -v, --verbose int   Output verbosity
```

11.2.6.10.2 SEE ALSO

- dkp create (see page 1448) - Create one of [appdeployment, bootstrap, capi-components, chart-bundle, cluster, image-bundle, nodepool, workspace]
- dkp create cluster aks (see page 1465) - Create a Konvoy cluster in AKS
- dkp create cluster aws (see page 1456) - Create a Konvoy cluster in AWS
- dkp create cluster azure (see page 1459) - Create a Konvoy cluster in Azure
- dkp create cluster eks (see page 1454) - Create a Konvoy cluster in EKS
- dkp create cluster gcp (see page 1469) - Create a Konvoy cluster in GCP
- dkp create cluster preprovisioned (see page 1467) - Create a Konvoy cluster on pre-provisioned infrastructure
- dkp create cluster vsphere (see page 1462) - Create a Konvoy cluster in vSphere

11.2.6.10.3 dkp create cluster eks

Create a Konvoy cluster in EKS

```
dkp create cluster eks [flags]
```

11.2.6.10.3.1 Options

```
      --additional-security-group-ids strings   A comma separated list of existing
security group IDs to use for machines in addition to those created automatically (de
fault [])
      --additional-tags stringToString          Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys             If true, ignore any errors in
templates when a field or map key is missing in the template. Only applies to golang
and jsonpath output formats. (default true)
      --aws-service-endpoints string            Custom AWS service endpoints in a
semi-colon separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=$
{URL};${SigningRegion2}...
  -c, --cluster-name name                       Name used to prefix the cluster and
all the created resources.
      --dry-run                                 Only print the objects that would be
created, without creating them.
  -h, --help                                    help for eks
      --http-proxy string                       HTTP proxy for CAPI controllers
      --https-proxy string                      HTTPS proxy for CAPI controllers
      --kind-cluster-image string               Kind node image for the bootstrap
cluster (default "mesosphere/konvoy-bootstrap:v2.5.2")
      --kubeconfig string                       Path to the kubeconfig for the
management cluster. If unspecified, default discovery rules apply.
      --kubernetes-pod-network-cidr cidr        The Kubernetes Pod network CIDR to
use in the cluster (default 192.168.0.0/16)
      --kubernetes-service-cidr cidr            The Kubernetes Service CIDR to use in
the cluster (default 10.96.0.0/12)
      --kubernetes-version string               Kubernetes version (default "1.24.7")
  -n, --namespace string                        If present, the namespace scope for
this CLI request. (default "default")
      --no-proxy strings                        No Proxy list for CAPI controllers (d
efault [])
  -o, --output string                           Output format. One of: (json, yaml,
name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-
json, jsonpath-file).
      --output-directory string                 Used with --output=json|yaml. The
directory where to output resources to files. The directory must already exist.
      --region string                           AWS region to deploy cluster to
(default "us-west-2")
      --show-managed-fields                     If true, keep the managedFields when
printing objects in JSON or YAML format.
      --subnet-ids strings                      A comma separated list of existing
subnet IDs to use for the kube-apiserver ELB and all control-plane and worker nodes (d
efault [])
      --template string                         Template string or path to template
file to use when -o=go-template, -o=go-template-file. The template format is golang
templates [http://golang.org/pkg/text/template/#pkg-overview].
```

```
      --timeout duration                          The length of time to wait before
giving up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --vpc-id string                             Existing VPC ID to use for the
cluster
      --wait                                      If true, wait for operations to
complete before returning. (default true)
      --with-aws-bootstrap-credentials            Set true to use AWS bootstrap
credentials from your environment. When false, the instance profile of the EC2
instance where the CAPA controller is scheduled on will be used instead.
      --with-gcp-bootstrap-credentials            Set true to use GCP bootstrap
credentials from your environment. When false, the service account of the VM instance
where the CAPG controller is scheduled on will be used instead.
      --worker-availability-zone string           The AvailabilityZone in the region to
deploy the worker nodes to, if not set a random one will be selected (ex. us-west-2a)
      --worker-iam-instance-profile string        Name of the IAM instance profile to
assign to worker machines. (default "nodes.cluster-api-provider-aws.sigs.k8s.io")
      --worker-instance-type string               Worker machine instance type (default
 "m5.2xlarge")
      --worker-replicas int                       Number of workers (default 4)
```

11.2.6.10.3.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.6.10.3.3 SEE ALSO

- dkp create cluster (see page 1454) - Create a Kubernetes cluster, one of [aks, aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.6.10.4 dkp create cluster aws

Create a Konvoy cluster in AWS

```
dkp create cluster aws [flags]
```

11.2.6.10.4.1 Options

```
      --additional-security-group-ids strings       A comma separated list of
existing security group IDs to use for machines in addition to those created
automatically (default [])
      --additional-tags stringToString              Tags to apply to the provisioned
infrastructure (default [])
```

```
      --allow-missing-template-keys                 If true, ignore any errors in
templates when a field or map key is missing in the template. Only applies to golang
and jsonpath output formats. (default true)
      --ami string                                  AMI ID to use for machines
      --ami-base-os string                          Base OS for Lookup search (ex.
'centos-7', 'ubuntu-18.04', 'ubuntu-20.04') (default "ubuntu-20.04")
      --ami-format string                           Lookup Format string to generate
AMI search name from (default "capa-ami-{{.BaseOS}}-?{{.K8sVersion}}-*")
      --ami-owner string                            Owner ID for AMI Lookup search (d
efault "258751437250")
      --aws-service-endpoints string                Custom AWS service endpoints in a
semi-colon separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=$
{URL};${SigningRegion2}...
      --certificate-renew-interval int              The interval number of days
Kubernetes managed PKI certificates are renewed. For example, an Interval value of 30
 means the certificates will be refreshed every 30 days. A value of 0 disables the
feature. (default 0)
  -c, --cluster-name name                           Name used to prefix the cluster
and all the created resources.
      --control-plane-http-proxy string             HTTP proxy for control plane
machines
      --control-plane-https-proxy string            HTTPS proxy for control plane
machines
      --control-plane-iam-instance-profile string   Name of the IAM instance profile
to assign to control plane machines. (default "control-plane.cluster-api-provider-
aws.sigs.k8s.io")
      --control-plane-instance-type string          Control Plane machine instance
type (default "m5.xlarge")
      --control-plane-no-proxy strings              No Proxy list for control plane
machines (default [])
      --control-plane-replicas int                  Number of control plane nodes (de
fault 3)
      --dry-run                                     Only print the objects that would
be created, without creating them.
      --etcd-image-repository string                The image repository to use for
 pulling the etcd image
      --etcd-version string                         The version of etcd to use.
      --extra-sans strings                          A comma separated list of
additional Subject Alternative Names for the API Server signing cert (default [])
  -h, --help                                        help for aws
      --http-proxy string                           HTTP proxy for CAPI controllers
      --https-proxy string                          HTTPS proxy for CAPI controllers
      --internal-load-balancer                      Make the control plane load
balancer internal, i.e., reachable only within the VPC.
      --kind-cluster-image string                   Kind node image for the bootstrap
cluster (default "mesosphere/konvoy-bootstrap:v2.5.2")
      --kubeconfig string                           Path to the kubeconfig for the
management cluster. If unspecified, default discovery rules apply. This flag is
ignored if used with the --self-managed flag.
      --kubernetes-image-repository string          The image repository to use for
 pulling kubernetes images
```

```
      --kubernetes-pod-network-cidr cidr            The Kubernetes Pod network CIDR
to use in the cluster (default 192.168.0.0/16)
      --kubernetes-service-cidr cidr                The Kubernetes Service CIDR to
use in the cluster (default 10.96.0.0/12)
      --kubernetes-version string                   Kubernetes version (default
"1.25.4")
  -n, --namespace string                            If present, the namespace scope
for this CLI request. (default "default")
      --no-proxy strings                            No Proxy list for CAPI
controllers (default [])
  -o, --output string                               Output format. One of: (json,
yaml, name, go-template, go-template-file, template, templatefile, jsonpath,
jsonpath-as-json, jsonpath-file).
      --output-directory string                     Used with --output=json|yaml. The
directory where to output resources to files. The directory must already exist.
      --region string                               AWS region to deploy cluster to
(default "us-west-2")
      --registry-mirror-cacert file                 CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string             Password to authenticate to the
registry mirror with
      --registry-mirror-url string                  URL of a container registry to
use as a mirror in the cluster
      --registry-mirror-username string             Username to authenticate to the
registry mirror with
      --self-managed                                When set to true, the required
prerequisites are created before creating the cluster and the resulting cluster has
all necessary components deployed onto itself, so it can manage its own cluster
lifecycle. When set to false, a management cluster is used.(default false)
      --show-managed-fields                         If true, keep the managedFields
when printing objects in JSON or YAML format.
      --ssh-public-key-file string                  Path to the authorized SSH key
for the user
      --ssh-username string                         Name of the user to create on the
instance (default "konvoy")
      --subnet-ids strings                          A comma separated list of
existing subnet IDs to use for the kube-apiserver ELB and all control-plane and
worker nodes (default [])
      --template string                             Template string or path to
template file to use when -o=go-template, -o=go-template-file. The template format is
golang templates [http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                            The length of time to wait before
giving up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --vpc-id string                               Existing VPC ID to use for the
cluster
      --wait                                        If true, wait for operations to
complete before returning. This flag is ignored and will always be 'true' if used
with the --self-managed flag. (default true)
      --with-aws-bootstrap-credentials              Set true to use AWS bootstrap
credentials from your environment. When false, the instance profile of the EC2
instance where the CAPA controller is scheduled on will be used instead.
```

```
      --with-gcp-bootstrap-credentials              Set true to use GCP bootstrap
credentials from your environment. When false, the service account of the VM instance
where the CAPG controller is scheduled on will be used instead.
      --worker-availability-zone string             The AvailabilityZone in the
region to deploy the worker nodes to, if not set a random one will be selected (ex.
us-west-2a)
      --worker-http-proxy string                    HTTP proxy for nodes
      --worker-https-proxy string                   HTTPS proxy for nodes
      --worker-iam-instance-profile string          Name of the IAM instance profile
to assign to worker machines. (default "nodes.cluster-api-provider-aws.sigs.k8s.io")
      --worker-instance-type string                 Worker machine instance type
(default "m5.2xlarge")
      --worker-no-proxy strings                     No Proxy list for nodes (default
 [])
      --worker-replicas int                         Number of workers (default 4)
```

11.2.6.10.4.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.6.10.4.3 SEE ALSO

- dkp create cluster (see page 1454) - Create a Kubernetes cluster, one of [aks, aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.6.10.5 dkp create cluster azure

Create a Konvoy cluster in Azure

```
dkp create cluster azure [flags]
```

11.2.6.10.5.1 Options

```
      --additional-tags stringToString       Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys          If true, ignore any errors in templates
when a field or map key is missing in the template. Only applies to golang and
jsonpath output formats. (default true)
      --aws-service-endpoints string         Custom AWS service endpoints in a semi-
colon separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=${URL};$
{SigningRegion2}...
      --certificate-renew-interval int       The interval number of days Kubernetes
managed PKI certificates are renewed. For example, an Interval value of 30 means the
```

```
certificates will be refreshed every 30 days. A value of 0 disables the feature. (def
ault 0)
  -c, --cluster-name name                    Name used to prefix the cluster and all
the created resources.
      --compute-gallery-id string            Compute Gallery ID of a custom image,
e.g., '/subscriptions/<subscription id>/resourceGroups/<resource group name>/
providers/Microsoft.Compute/galleries/<gallery name>/images/<image definition name>/
versions/<version id>' (replacing placeholders with the values used when creating the
image)
      --control-plane-http-proxy string      HTTP proxy for control plane machines
      --control-plane-https-proxy string     HTTPS proxy for control plane machines
      --control-plane-machine-size string    Control Plane machine size (default
"Standard_D4s_v3")
      --control-plane-no-proxy strings       No Proxy list for control plane machines
(default [])
      --control-plane-replicas int           Number of control plane nodes (default 3)
      --dry-run                              Only print the objects that would be
created, without creating them.
      --etcd-image-repository string         The image repository to use for pulling
the etcd image
      --etcd-version string                  The version of etcd to use.
      --extra-sans strings                   A comma separated list of additional
Subject Alternative Names for the API Server signing cert (default [])
  -h, --help                                 help for azure
      --http-proxy string                    HTTP proxy for CAPI controllers
      --https-proxy string                   HTTPS proxy for CAPI controllers
      --kind-cluster-image string            Kind node image for the bootstrap
cluster (default "mesosphere/konvoy-bootstrap:v2.5.2")
      --kubeconfig string                    Path to the kubeconfig for the
management cluster. If unspecified, default discovery rules apply. This flag is
ignored if used with the --self-managed flag.
      --kubernetes-image-repository string   The image repository to use for pulling
kubernetes images
      --kubernetes-pod-network-cidr cidr     The Kubernetes Pod network CIDR to use
in the cluster (default 192.168.0.0/16)
      --kubernetes-service-cidr cidr         The Kubernetes Service CIDR to use in
the cluster (default 10.96.0.0/12)
      --kubernetes-version string            Kubernetes version (default "1.25.4")
      --location string                      Azure location to deploy cluster to
(default "westus")
  -n, --namespace string                     If present, the namespace scope for this
 CLI request. (default "default")
      --no-proxy strings                     No Proxy list for CAPI controllers (defa
ult [])
  -o, --output string                        Output format. One of: (json, yaml,
name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-
json, jsonpath-file).
      --output-directory string              Used with --output=json|yaml. The
directory where to output resources to files. The directory must already exist.
      --plan-offer string                    The offer for a Marketplace image or a
custom image sourced from a Marketplace image requiring Plan information.
```

```
      --plan-publisher string                The publisher for a Marketplace image or
a custom image sourced from a Marketplace image requiring Plan information.
      --plan-sku string                      The SKU for a Marketplace image or a
custom image sourced from a Marketplace image requiring Plan information.
      --registry-mirror-cacert file          CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string      Password to authenticate to the registry
mirror with
      --registry-mirror-url string           URL of a container registry to use as a
mirror in the cluster
      --registry-mirror-username string      Username to authenticate to the registry
mirror with
      --self-managed                         When set to true, the required
prerequisites are created before creating the cluster and the resulting cluster has
all necessary components deployed onto itself, so it can manage its own cluster
lifecycle. When set to false, a management cluster is used.(default false)
      --show-managed-fields                  If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-public-key-file string           Path to the authorized SSH key for the
user
      --ssh-username string                  Name of the user to create on the
instance (default "konvoy")
      --template string                      Template string or path to template file
to use when -o=go-template, -o=go-template-file. The template format is golang
templates [http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                     The length of time to wait before giving
up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --wait                                 If true, wait for operations to complete
before returning. This flag is ignored and will always be 'true' if used with the --
self-managed flag. (default true)
      --with-aws-bootstrap-credentials       Set true to use AWS bootstrap
credentials from your environment. When false, the instance profile of the EC2
instance where the CAPA controller is scheduled on will be used instead.
      --with-gcp-bootstrap-credentials       Set true to use GCP bootstrap
credentials from your environment. When false, the service account of the VM instance
where the CAPG controller is scheduled on will be used instead.
      --worker-availability-zone string      The availability zone in the region to
deploy the worker nodes to, if not set a random one will be selected (ex. 1). Not all
locations, including the default 'westus', support setting this flag, see https://
docs.microsoft.com/en-us/azure/availability-zones/az-overview.
      --worker-http-proxy string             HTTP proxy for nodes
      --worker-https-proxy string            HTTPS proxy for nodes
      --worker-machine-size string           Worker machine size (default
"Standard_D8s_v3")
      --worker-no-proxy strings              No Proxy list for nodes (default [])
      --worker-replicas int                  Number of workers (default 4)
```

11.2.6.10.5.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.6.10.5.3 SEE ALSO

- dkp create cluster (see page 1454) - Create a Kubernetes cluster, one of [aks, aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.6.10.6 dkp create cluster vsphere

Create a Konvoy cluster in vSphere

```
dkp create cluster vsphere [flags]
```

11.2.6.10.6.1 Options

```
      --allow-missing-template-keys          If true, ignore any errors in templates
when a field or map key is missing in the template. Only applies to golang and
jsonpath output formats. (default true)
      --aws-service-endpoints string         Custom AWS service endpoints in a semi-
colon separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=${URL};$
{SigningRegion2}...
      --certificate-renew-interval int       The interval number of days Kubernetes
managed PKI certificates are renewed. For example, an Interval value of 30 means the
certificates will be refreshed every 30 days. A value of 0 disables the feature. (def
ault 0)
  -c, --cluster-name name                    Name used to prefix the cluster and all
the created resources.
      --control-plane-cpus int               The number of virtual processors in a
control plane machine (default 4)
      --control-plane-disk-size int          The size of a control plane machine's
disk, in GB (default 80)
      --control-plane-endpoint-host string   The control plane endpoint address. To
use an external load balancer, set to its IP or hostname. To use the built-in virtual
IP, set to a static IPv4 address in the Layer 2 network of the control plane
machines. [Not for production use: To use a single-machine control plane, set to the
IP or hostname of the machine.]
      --control-plane-endpoint-port int      The control plane endpoint port. To use
an external load balancer, set to its listening port. (default 6443)
      --control-plane-http-proxy string      HTTP proxy for control plane machines
      --control-plane-https-proxy string     HTTPS proxy for control plane machines
```

```
      --control-plane-memory int              The size of a control plane machine's
memory, in GB (default 16)
      --control-plane-no-proxy strings        No Proxy list for control plane machines
(default [])
      --control-plane-replicas int            Number of control plane nodes (default 3)
      --data-center string                    The vSphere datacenter to deploy the
workload cluster on.
      --data-store string                     The vSphere datastore to deploy the
workload cluster on.
      --dry-run                               Only print the objects that would be
created, without creating them.
      --etcd-image-repository string          The image repository to use for pulling
the etcd image
      --etcd-version string                   The version of etcd to use.
      --extra-sans strings                    A comma separated list of additional
Subject Alternative Names for the API Server signing cert (default [])
      --folder string                         The vSphere folder for your VMs. Set to
"" to use the root vSphere folder.
  -h, --help                                  help for vsphere
      --http-proxy string                     HTTP proxy for CAPI controllers
      --https-proxy string                    HTTPS proxy for CAPI controllers
      --kind-cluster-image string             Kind node image for the bootstrap
cluster (default "mesosphere/konvoy-bootstrap:v2.5.2")
      --kubeconfig string                     Path to the kubeconfig for the
management cluster. If unspecified, default discovery rules apply. This flag is
ignored if used with the --self-managed flag.
      --kubernetes-image-repository string    The image repository to use for pulling
kubernetes images
      --kubernetes-pod-network-cidr cidr      The Kubernetes Pod network CIDR to use
in the cluster (default 192.168.0.0/16)
      --kubernetes-service-cidr cidr          The Kubernetes Service CIDR to use in
the cluster (default 10.96.0.0/12)
      --kubernetes-version string             Kubernetes version (default "1.25.4")
  -n, --namespace string                      If present, the namespace scope for this
 CLI request. (default "default")
      --network string                        The vSphere network to deploy the
workload cluster on.
      --no-proxy strings                      No Proxy list for CAPI controllers (defa
ult [])
  -o, --output string                         Output format. One of: (json, yaml,
name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-
json, jsonpath-file).
      --output-directory string               Used with --output=json|yaml. The
directory where to output resources to files. The directory must already exist.
      --registry-mirror-cacert file           CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string       Password to authenticate to the registry
mirror with
      --registry-mirror-url string            URL of a container registry to use as a
mirror in the cluster
      --registry-mirror-username string       Username to authenticate to the registry
mirror with
```

```
      --resource-pool string                The vSphere resource pool for the
workload cluster's virtual machines.
      --self-managed                        When set to true, the required
prerequisites are created before creating the cluster and the resulting cluster has
all necessary components deployed onto itself, so it can manage its own cluster
lifecycle. When set to false, a management cluster is used.(default false)
      --server string                       The vCenter server IP or FQDN.
      --show-managed-fields                 If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-public-key-file string          Path to the authorized SSH key for the
user
      --ssh-username string                 Name of the user to create on the
instance (default "konvoy")
      --storage-policy string               This is the vSphere storage policy. Set
it to "" if you don't want to use a storage policy.
      --template string                     Template string or path to template file
to use when -o=go-template, -o=go-template-file. The template format is golang
templates [http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                    The length of time to wait before giving
up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --tls-thumb-print string              sha1 thumbprint of the vcenter
certificate: openssl x509 -sha1 -fingerprint -in ca.crt -noout
      --virtual-ip-interface string         The network interface, e.g, 'eth0' or
'ens5', to use for the built-in virtual IP control plane endpoint. This interface
 must be available on every control plane machine. If the value is empty, the flag
does nothing. If the value is not empty, the built-in virtual IP control plane
endpoint is created, using values from --control-plane-endpoint-host and --control-
plane-endpoint-port.
      --vm-template string                  The virtual machine template to use for
 the workload cluster's virtual machines.
      --wait                                If true, wait for operations to complete
before returning. This flag is ignored and will always be 'true' if used with the --
self-managed flag. (default true)
      --with-aws-bootstrap-credentials      Set true to use AWS bootstrap
credentials from your environment. When false, the instance profile of the EC2
instance where the CAPA controller is scheduled on will be used instead.
      --with-gcp-bootstrap-credentials      Set true to use GCP bootstrap
credentials from your environment. When false, the service account of the VM instance
where the CAPG controller is scheduled on will be used instead.
      --worker-cpus int                     The number of virtual processors in a
worker machine (default 8)
      --worker-disk-size int                The size of a worker machine's disk, in
GB (default 80)
      --worker-http-proxy string            HTTP proxy for nodes
      --worker-https-proxy string           HTTPS proxy for nodes
      --worker-memory int                   The size of a worker machine's memory,
in GB (default 32)
      --worker-no-proxy strings             No Proxy list for nodes (default [])
      --worker-replicas int                 Number of workers (default 4)
```

11.2.6.10.6.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.6.10.6.3 SEE ALSO

- dkp create cluster (see page 1454) - Create a Kubernetes cluster, one of [aks, aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.6.10.7 dkp create cluster aks

Create a Konvoy cluster in AKS

```
dkp create cluster aks [flags]
```

11.2.6.10.7.1 Options

```
      --additional-tags stringToString     Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys        If true, ignore any errors in templates
when a field or map key is missing in the template. Only applies to golang and
jsonpath output formats. (default true)
      --aws-service-endpoints string       Custom AWS service endpoints in a semi-
colon separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=${URL};$
{SigningRegion2}...
      --certificate-renew-interval int     The interval number of days Kubernetes
managed PKI certificates are renewed. For example, an Interval value of 30 means the
certificates will be refreshed every 30 days. A value of 0 disables the feature. (def
ault 0)
  -c, --cluster-name name                  Name used to prefix the cluster and all
the created resources.
      --dry-run                            Only print the objects that would be
created, without creating them.
  -h, --help                               help for aks
      --http-proxy string                  HTTP proxy for CAPI controllers
      --https-proxy string                 HTTPS proxy for CAPI controllers
      --kind-cluster-image string          Kind node image for the bootstrap cluster
(default "mesosphere/konvoy-bootstrap:v2.5.2")
      --kubeconfig string                  Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-pod-network-cidr cidr   The Kubernetes Pod network CIDR to use in
the cluster (default 192.168.0.0/16)
```

```
      --kubernetes-service-cidr cidr       The Kubernetes Service CIDR to use in the
cluster (default 10.96.0.0/12)
      --kubernetes-version string          Kubernetes version. Run 'az aks get-
versions -o table --location <location>' to see available versions. See https://
docs.microsoft.com/en-us/azure/aks/supported-kubernetes-versions for more details.
Must be a patch version for v1.25.x.
      --location string                    Azure location to deploy cluster to
(default "westus")
  -n, --namespace string                   If present, the namespace scope for this
 CLI request. (default "default")
      --no-proxy strings                   No Proxy list for CAPI controllers (defaul
t [])
  -o, --output string                      Output format. One of: (json, yaml, name,
go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string            Used with --output=json|yaml. The
directory where to output resources to files. The directory must already exist.
      --show-managed-fields                If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-public-key-file string         Path to the authorized SSH key for the
user
      --ssh-username string                Name of the user to create on the instance
(default "konvoy")
      --system-machine-size string         System node pool machine size (default
"Standard_D4s_v3")
      --system-replicas int                Number of system nodes (default 3)
      --template string                    Template string or path to template file
to use when -o=go-template, -o=go-template-file. The template format is golang
templates [http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                   The length of time to wait before giving
up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --wait                               If true, wait for operations to complete
before returning. (default true)
      --with-aws-bootstrap-credentials     Set true to use AWS bootstrap credentials
from your environment. When false, the instance profile of the EC2 instance where the
CAPA controller is scheduled on will be used instead.
      --with-gcp-bootstrap-credentials     Set true to use GCP bootstrap credentials
from your environment. When false, the service account of the VM instance where the
CAPG controller is scheduled on will be used instead.
      --worker-http-proxy string           HTTP proxy for nodes
      --worker-https-proxy string          HTTPS proxy for nodes
      --worker-machine-size string         Worker machine size (default
"Standard_D8s_v3")
      --worker-no-proxy strings            No Proxy list for nodes (default [])
      --worker-replicas int                Number of workers (default 4)
```

11.2.6.10.7.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.6.10.7.3 SEE ALSO

- dkp create cluster (see page 1454) - Create a Kubernetes cluster, one of [aks, aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.6.10.8 dkp create cluster preprovisioned

Create a Konvoy cluster on pre-provisioned infrastructure

```
dkp create cluster preprovisioned [flags]
```

11.2.6.10.8.1 Options

```
      --allow-missing-template-keys             If true, ignore any errors in
templates when a field or map key is missing in the template. Only applies to golang
and jsonpath output formats. (default true)
      --aws-service-endpoints string            Custom AWS service endpoints in a
semi-colon separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=$
{URL};${SigningRegion2}...
      --certificate-renew-interval int          The interval number of days
Kubernetes managed PKI certificates are renewed. For example, an Interval value of 30
 means the certificates will be refreshed every 30 days. A value of 0 disables the
feature. (default 0)
  -c, --cluster-name name                       Name used to prefix the cluster and
all the created resources.
      --control-plane-endpoint-host string      The control plane endpoint address.
To use an external load balancer, set to its IP or hostname. To use the built-in
virtual IP, set to a static IPv4 address in the Layer 2 network of the control plane
machines. [Not for production use: To use a single-machine control plane, set to the
IP or hostname of the machine.]
      --control-plane-endpoint-port int         The control plane endpoint port. To
use an external load balancer, set to its listening port. (default 6443)
      --control-plane-http-proxy string         HTTP proxy for control plane machines
      --control-plane-https-proxy string        HTTPS proxy for control plane
machines
      --control-plane-no-proxy strings          No Proxy list for control plane
machines (default [])
      --control-plane-replicas int              Number of control plane nodes (defaul
t 3)
      --dry-run                                 Only print the objects that would be
created, without creating them.
      --etcd-image-repository string            The image repository to use for
 pulling the etcd image
      --etcd-version string                     The version of etcd to use.
      --extra-sans strings                      A comma separated list of additional
Subject Alternative Names for the API Server signing cert (default [])
```

```
  -h, --help                                    help for preprovisioned
      --http-proxy string                       HTTP proxy for CAPI controllers
      --https-proxy string                      HTTPS proxy for CAPI controllers
      --kind-cluster-image string               Kind node image for the bootstrap
cluster (default "mesosphere/konvoy-bootstrap:v2.5.2")
      --kubeconfig string                       Path to the kubeconfig for the
management cluster. If unspecified, default discovery rules apply. This flag is
ignored if used with the --self-managed flag.
      --kubernetes-image-repository string      The image repository to use for
 pulling kubernetes images
      --kubernetes-pod-network-cidr cidr        The Kubernetes Pod network CIDR to
use in the cluster (default 192.168.0.0/16)
      --kubernetes-service-cidr cidr            The Kubernetes Service CIDR to use in
the cluster (default 10.96.0.0/12)
      --kubernetes-version string               Kubernetes version (default "1.25.4")
  -n, --namespace string                        If present, the namespace scope for
this CLI request. (default "default")
      --no-proxy strings                        No Proxy list for CAPI controllers (d
efault [])
      --os-hint flatcar                         A hint which will allow the installer
to generate appropriate configurations for a target OS. Presently, only the hint for
 flatcar is supported.
  -o, --output string                           Output format. One of: (json, yaml,
name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-
json, jsonpath-file).
      --output-directory string                 Used with --output=json|yaml. The
directory where to output resources to files. The directory must already exist.
      --override-secret-name string             Name of the secret for any provided
overrides on a preprovisioned cluster.All overrides defined at provisioning should be
present in this secret.
      --pre-provisioned-inventory-file string   Path to PreprovisionedInventory
inventory file
      --registry-mirror-cacert file             CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string         Password to authenticate to the
registry mirror with
      --registry-mirror-url string              URL of a container registry to use as
a mirror in the cluster
      --registry-mirror-username string         Username to authenticate to the
registry mirror with
      --self-managed                            When set to true, the required
prerequisites are created before creating the cluster and the resulting cluster has
all necessary components deployed onto itself, so it can manage its own cluster
lifecycle. When set to false, a management cluster is used.(default false)
      --show-managed-fields                     If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-private-key-file string             Path to the private SSH key file used
by Konvoy to access the pre-provisioned hosts
      --ssh-public-key-file string              Path to the authorized SSH key for
 the user
      --ssh-username string                     Name of the user to create on the
instance (default "konvoy")
```

```
      --template string                          Template string or path to template
file to use when -o=go-template, -o=go-template-file. The template format is golang
templates [http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                         The length of time to wait before
giving up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --virtual-ip-interface string              The network interface, e.g, 'eth0' or
'ens5', to use for the built-in virtual IP control plane endpoint. This interface
 must be available on every control plane machine. If the value is empty, the flag
does nothing. If the value is not empty, the built-in virtual IP control plane
endpoint is created, using values from --control-plane-endpoint-host and --control-
plane-endpoint-port.
      --wait                                     If true, wait for operations to
complete before returning. This flag is ignored and will always be 'true' if used
with the --self-managed flag. (default true)
      --with-aws-bootstrap-credentials           Set true to use AWS bootstrap
credentials from your environment. When false, the instance profile of the EC2
instance where the CAPA controller is scheduled on will be used instead.
      --with-gcp-bootstrap-credentials           Set true to use GCP bootstrap
credentials from your environment. When false, the service account of the VM instance
where the CAPG controller is scheduled on will be used instead.
      --worker-http-proxy string                 HTTP proxy for nodes
      --worker-https-proxy string                HTTPS proxy for nodes
      --worker-no-proxy strings                  No Proxy list for nodes (default [])
      --worker-replicas int                      Number of workers (default 4)
```

11.2.6.10.8.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.6.10.8.3 SEE ALSO

- dkp create cluster (see page 1454) - Create a Kubernetes cluster, one of [aks, aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.6.10.9 dkp create cluster gcp

Create a Konvoy cluster in GCP

```
dkp create cluster gcp [flags]
```

11.2.6.10.9.1 Options

```
      --additional-tags stringToString               Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys                  If true, ignore any errors in
templates when a field or map key is missing in the template. Only applies to golang
and jsonpath output formats. (default true)
      --associate-public-ip-address                  Associate a public IP for all
machines. When set to false the specified network must have Cloud NAT configured to
provide internet access. (default true)
      --aws-service-endpoints string                 Custom AWS service endpoints in
a semi-colon separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=$
{URL};${SigningRegion2}...
      --certificate-renew-interval int               The interval number of days
Kubernetes managed PKI certificates are renewed. For example, an Interval value of 30
 means the certificates will be refreshed every 30 days. A value of 0 disables the
feature. (default 0)
  -c, --cluster-name name                            Name used to prefix the cluster
and all the created resources.
      --control-plane-http-proxy string              HTTP proxy for control plane
machines
      --control-plane-https-proxy string             HTTPS proxy for control plane
machines
      --control-plane-instance-type string           Control Plane machine instance
type (default "n2-standard-4")
      --control-plane-no-proxy strings               No Proxy list for control plane
machines (default [])
      --control-plane-replicas int                   Number of control plane nodes (d
efault 3)
      --control-plane-service-account-email string   Control Plane Service Account
email address (default "default")
      --dry-run                                      Only print the objects that
would be created, without creating them.
      --etcd-image-repository string                 The image repository to use for
 pulling the etcd image
      --etcd-version string                          The version of etcd to use.
      --extra-sans strings                           A comma separated list of
additional Subject Alternative Names for the API Server signing cert (default [])
  -h, --help                                         help for gcp
      --http-proxy string                            HTTP proxy for CAPI controllers
      --https-proxy string                           HTTPS proxy for CAPI controllers
      --image string                                 Full reference to an image to
use for all nodes (set either this or --image-family) (ex. 'projects/my-project/
global/images/konvoy-ubuntu-2004-1-99-99-1234567890')
      --image-family string                          Full reference to an image
family to use for all nodes (set either this or --image) (ex. 'projects/my-project/
global/images/family/konvoy-ubuntu-2004-{{.K8sVersion}}')
      --kind-cluster-image string                    Kind node image for the
bootstrap cluster (default "mesosphere/konvoy-bootstrap:v2.5.2")
```

```
      --kubeconfig string                              Path to the kubeconfig for the
management cluster. If unspecified, default discovery rules apply. This flag is
ignored if used with the --self-managed flag.
      --kubernetes-image-repository string             The image repository to use for
 pulling kubernetes images
      --kubernetes-pod-network-cidr cidr               The Kubernetes Pod network CIDR
to use in the cluster (default 192.168.0.0/16)
      --kubernetes-service-cidr cidr                   The Kubernetes Service CIDR to
use in the cluster (default 10.96.0.0/12)
      --kubernetes-version string                      Kubernetes version (default
"1.25.4")
  -n, --namespace string                               If present, the namespace scope
for this CLI request. (default "default")
      --network string                                 The GCP network name to deploy
the cluster to (default "default")
      --no-proxy strings                               No Proxy list for CAPI
controllers (default [])
  -o, --output string                                  Output format. One of: (json,
yaml, name, go-template, go-template-file, template, templatefile, jsonpath,
jsonpath-as-json, jsonpath-file).
      --output-directory string                        Used with --output=json|yaml.
The directory where to output resources to files. The directory must already exist.
      --project string                                 The GCP project name to deploy
the cluster to
      --region string                                  GCP region to deploy cluster to
(default "us-west1")
      --registry-mirror-cacert file                    CA certificate chain to use
while communicating with the registry mirror using TLS
      --registry-mirror-password string                Password to authenticate to the
registry mirror with
      --registry-mirror-url string                     URL of a container registry to
use as a mirror in the cluster
      --registry-mirror-username string                Username to authenticate to the
registry mirror with
      --self-managed                                   When set to true, the required
prerequisites are created before creating the cluster and the resulting cluster has
all necessary components deployed onto itself, so it can manage its own cluster
lifecycle. When set to false, a management cluster is used.(default false)
      --show-managed-fields                            If true, keep the managedFields
when printing objects in JSON or YAML format.
      --ssh-public-key-file string                     Path to the authorized SSH key
for the user
      --ssh-username string                            Name of the user to create on
the instance (default "konvoy")
      --template string                                Template string or path to
template file to use when -o=go-template, -o=go-template-file. The template format is
golang templates [http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                               The length of time to wait
before giving up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --wait                                           If true, wait for operations to
complete before returning. This flag is ignored and will always be 'true' if used
with the --self-managed flag. (default true)
```

```
      --with-aws-bootstrap-credentials               Set true to use AWS bootstrap
credentials from your environment. When false, the instance profile of the EC2
instance where the CAPA controller is scheduled on will be used instead.
      --with-gcp-bootstrap-credentials               Set true to use GCP bootstrap
credentials from your environment. When false, the service account of the VM instance
where the CAPG controller is scheduled on will be used instead.
      --worker-http-proxy string                     HTTP proxy for nodes
      --worker-https-proxy string                    HTTPS proxy for nodes
      --worker-instance-type string                  Worker machine instance type
(default "n2-standard-8")
      --worker-no-proxy strings                      No Proxy list for nodes (default
 [])
      --worker-replicas int                          Number of workers (default 4)
      --worker-service-account-email string          Worker machine Service Account
email address (default "default")
      --worker-zone string                           Zone in the region to deploy the
worker nodes to, if not set a random one will be selected (ex. us-west1-a)
```

11.2.6.10.9.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.6.10.9.3 SEE ALSO

- dkp create cluster (see page 1454) - Create a Kubernetes cluster, one of [aks, aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.6.11 dkp create nodepool

Create a nodepool, one of [aks, aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.6.11.1 Options

```
  -h, --help          help for nodepool
  -v, --verbose int   Output verbosity
```

11.2.6.11.2 SEE ALSO

- dkp create (see page 1448) - Create one of [appdeployment, bootstrap, capi-components, chart-bundle, cluster, image-bundle, nodepool, workspace]
- dkp create nodepool aks (see page 1476) - Create a nodepool in AKS
- dkp create nodepool aws (see page 1477) - Create a nodepool in AWS
- dkp create nodepool azure (see page 1473) - Create a nodepool in Azure

- dkp create nodepool eks (see page 1479) - Create a nodepool for EKS
- dkp create nodepool gcp (see page 1481) - Create a nodepool in GCP
- dkp create nodepool preprovisioned (see page 1482) - Create a nodepool for a Pre Provisioned Cluster.
- dkp create nodepool vsphere (see page 1474) - Create a nodepool in vSphere

11.2.6.11.3 dkp create nodepool azure

Create a nodepool in Azure

```
dkp create nodepool azure name [flags]
```

11.2.6.11.3.1 Options

```
      --additional-tags stringToString    Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys       If true, ignore any errors in templates
when a field or map key is missing in the template. Only applies to golang and
jsonpath output formats. (default true)
      --availability-zone string          The availability zone in the region to
deploy the worker nodes to, if not set a random one will be selected (ex. 1). Not all
locations, including the default 'westus', support setting this flag, see https://
docs.microsoft.com/en-us/azure/availability-zones/az-overview.
  -c, --cluster-name name                 Name used to prefix the cluster and all the
created resources.
      --compute-gallery-id string         Compute Gallery ID of a custom image, e.g.,
'/subscriptions/<subscription id>/resourceGroups/<resource group name>/providers/
Microsoft.Compute/galleries/<gallery name>/images/<image definition name>/versions/
<version id>' (replacing placeholders with the values used when creating the image)
      --dry-run                           Only print the objects that would be
created, without creating them.
  -h, --help                              help for azure
      --http-proxy string                 HTTP proxy for nodes
      --https-proxy string                HTTPS proxy for nodes
      --kubeconfig string                 Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string         Kubernetes version (default "1.25.4")
      --machine-size string               Worker machine size (default
"Standard_D8s_v3")
  -n, --namespace string                  If present, the namespace scope for this
 CLI request. (default "default")
      --no-proxy strings                  No Proxy list for nodes (default [])
  -o, --output string                     Output format. One of: (json, yaml, name,
go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string           Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
```

```
      --plan-offer string                The offer for a Marketplace image or a
custom image sourced from a Marketplace image requiring Plan information.
      --plan-publisher string            The publisher for a Marketplace image or a
custom image sourced from a Marketplace image requiring Plan information.
      --plan-sku string                  The SKU for a Marketplace image or a custom
image sourced from a Marketplace image requiring Plan information.
      --registry-mirror-cacert file      CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string  Password to authenticate to the registry
mirror with
      --registry-mirror-url string       URL of a container registry to use as a
mirror in the cluster
      --registry-mirror-username string  Username to authenticate to the registry
mirror with
      --replicas int                     Number of replicas (default 1)
      --show-managed-fields              If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-public-key-file string       Path to the authorized SSH key for the user
      --ssh-username string              Name of the user to create on the instance
(default "konvoy")
      --template string                  Template string or path to template file to
use when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                 The length of time to wait before giving
up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --wait                             If true, wait for operations to complete
before returning.
```

11.2.6.11.3.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.6.11.3.3 SEE ALSO

- dkp create nodepool (see page 1472) - Create a nodepool, one of [aks, aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.6.11.4 dkp create nodepool vsphere

Create a nodepool in vSphere

```
dkp create nodepool vsphere name [flags]
```

11.2.6.11.4.1 Options

```
      --additional-tags stringToString    Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys       If true, ignore any errors in templates
when a field or map key is missing in the template. Only applies to golang and
jsonpath output formats. (default true)
  -c, --cluster-name name                 Name used to prefix the cluster and all the
created resources.
      --cpus int                          The number of virtual processors in a
worker machine (default 8)
      --data-center string                The vSphere datacenter to deploy the
workload cluster on.
      --data-store string                 The vSphere datastore to deploy the
workload cluster on.
      --disk-size int                     The size of a worker machine's disk, in GB
(default 80)
      --dry-run                           Only print the objects that would be
created, without creating them.
      --folder string                     The vSphere folder for your VMs. Set to ""
 to use the root vSphere folder.
  -h, --help                              help for vsphere
      --http-proxy string                 HTTP proxy for nodes
      --https-proxy string                HTTPS proxy for nodes
      --kubeconfig string                 Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string         Kubernetes version (default "1.25.4")
      --memory int                        The size of a worker machine's memory, in
GB (default 32)
  -n, --namespace string                  If present, the namespace scope for this
 CLI request. (default "default")
      --network string                    The vSphere network to deploy the workload
cluster on.
      --no-proxy strings                  No Proxy list for nodes (default [])
  -o, --output string                     Output format. One of: (json, yaml, name,
go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string           Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --registry-mirror-cacert file       CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string   Password to authenticate to the registry
mirror with
      --registry-mirror-url string        URL of a container registry to use as a
mirror in the cluster
      --registry-mirror-username string   Username to authenticate to the registry
mirror with
      --replicas int                      Number of replicas (default 1)
      --resource-pool string              The vSphere resource pool for the workload
cluster's virtual machines.
```

```
      --server string                     The vCenter server IP or FQDN.
      --show-managed-fields               If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-public-key-file string        Path to the authorized SSH key for the user
      --ssh-username string               Name of the user to create on the instance
(default "konvoy")
      --storage-policy string             This is the vSphere storage policy. Set it
to "" if you don't want to use a storage policy.
      --template string                   Template string or path to template file to
use when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                  The length of time to wait before giving
up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --tls-thumb-print string            sha1 thumbprint of the vcenter certificate:
openssl x509 -sha1 -fingerprint -in ca.crt -noout
      --vm-template string                The virtual machine template to use for the
workload cluster's virtual machines.
      --wait                              If true, wait for operations to complete
before returning.
```

11.2.6.11.4.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.6.11.4.3 SEE ALSO

- dkp create nodepool (see page 1472) - Create a nodepool, one of [aks, aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.6.11.5 dkp create nodepool aks

Create a nodepool in AKS

```
dkp create nodepool aks name [flags]
```

11.2.6.11.5.1 Options

```
      --additional-tags stringToString   Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys      If true, ignore any errors in templates when
a field or map key is missing in the template. Only applies to golang and jsonpath
output formats. (default true)
  -c, --cluster-name name                Name used to prefix the cluster and all the
created resources.
```

```
      --dry-run                          Only print the objects that would be
created, without creating them.
  -h, --help                             help for aks
      --http-proxy string                HTTP proxy for nodes
      --https-proxy string               HTTPS proxy for nodes
      --kubeconfig string                Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string        Kubernetes version. Run 'az aks get-versions
-o table --location <location>' to see available versions. See https://
docs.microsoft.com/en-us/azure/aks/supported-kubernetes-versions for more details.
Must be a patch version for v1.25.x.
      --machine-size string              Worker machine size (default
"Standard_D8s_v3")
  -n, --namespace string                 If present, the namespace scope for this CLI
request. (default "default")
      --no-proxy strings                 No Proxy list for nodes (default [])
  -o, --output string                    Output format. One of: (json, yaml, name,
go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string          Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --replicas int                     Number of replicas (default 1)
      --show-managed-fields              If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-public-key-file string       Path to the authorized SSH key for the user
      --ssh-username string              Name of the user to create on the instance
(default "konvoy")
      --template string                  Template string or path to template file to
use when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                 The length of time to wait before giving up.
Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --wait                             If true, wait for operations to complete
before returning.
```

11.2.6.11.5.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.6.11.5.3 SEE ALSO

- dkp create nodepool (see page 1472) - Create a nodepool, one of [aks, aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.6.11.6 dkp create nodepool aws

Create a nodepool in AWS

```
dkp create nodepool aws [flags]
```

11.2.6.11.6.1 Options

```
      --additional-security-group-ids strings   A comma separated list of existing
security group IDs to use for machines in addition to those created automatically (de
fault [])
      --additional-tags stringToString          Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys             If true, ignore any errors in
templates when a field or map key is missing in the template. Only applies to golang
and jsonpath output formats. (default true)
      --ami string                              AMI ID to use for machines
      --ami-base-os string                      Base OS for Lookup search (ex.
'centos-7', 'ubuntu-18.04', 'ubuntu-20.04') (default "ubuntu-20.04")
      --ami-format string                       Lookup Format string to generate AMI
search name from (default "capa-ami-{{.BaseOS}}-?{{.K8sVersion}}-*")
      --ami-owner string                        Owner ID for AMI Lookup search (defau
lt "258751437250")
      --availability-zone string                The AvailabilityZone in the region to
deploy the worker nodes to, if not set a random one will be selected (ex. us-west-2a)
  -c, --cluster-name name                       Name used to prefix the cluster and
all the created resources.
      --dry-run                                 Only print the objects that would be
created, without creating them.
  -h, --help                                    help for aws
      --http-proxy string                       HTTP proxy for nodes
      --https-proxy string                      HTTPS proxy for nodes
      --iam-instance-profile string             Name of the IAM instance profile to
assign to worker machines. (default "nodes.cluster-api-provider-aws.sigs.k8s.io")
      --instance-type string                    Worker machine instance type (default
 "m5.2xlarge")
      --kubeconfig string                       Path to the kubeconfig for the
management cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string               Kubernetes version (default "1.25.4")
  -n, --namespace string                        If present, the namespace scope for
this CLI request. (default "default")
      --no-proxy strings                        No Proxy list for nodes (default [])
  -o, --output string                           Output format. One of: (json, yaml,
name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-
json, jsonpath-file).
      --output-directory string                 Used with --output=json|yaml. The
directory where to output resources to files. The directory must already exist.
      --registry-mirror-cacert file             CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string         Password to authenticate to the
registry mirror with
```

```
      --registry-mirror-url string              URL of a container registry to use as
a mirror in the cluster
      --registry-mirror-username string         Username to authenticate to the
registry mirror with
      --replicas int                            Number of replicas (default 1)
      --show-managed-fields                     If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-public-key-file string              Path to the authorized SSH key for
 the user
      --ssh-username string                     Name of the user to create on the
instance (default "konvoy")
      --template string                         Template string or path to template
file to use when -o=go-template, -o=go-template-file. The template format is golang
templates [http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                        The length of time to wait before
giving up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --wait                                    If true, wait for operations to
complete before returning.
```

11.2.6.11.6.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.6.11.6.3 SEE ALSO

- dkp create nodepool (see page 1472) - Create a nodepool, one of [aks, aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.6.11.7 dkp create nodepool eks

Create a nodepool for EKS

```
dkp create nodepool eks [flags]
```

11.2.6.11.7.1 Options

```
      --additional-security-group-ids strings   A comma separated list of existing
security group IDs to use for machines in addition to those created automatically (de
fault [])
      --additional-tags stringToString          Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys             If true, ignore any errors in
templates when a field or map key is missing in the template. Only applies to golang
and jsonpath output formats. (default true)
```

```
      --availability-zone string                The AvailabilityZone in the region to
deploy the worker nodes to, if not set a random one will be selected (ex. us-west-2a)
  -c, --cluster-name name                       Name used to prefix the cluster and
all the created resources.
      --dry-run                                 Only print the objects that would be
created, without creating them.
  -h, --help                                    help for eks
      --http-proxy string                       HTTP proxy for nodes
      --https-proxy string                      HTTPS proxy for nodes
      --iam-instance-profile string             Name of the IAM instance profile to
assign to worker machines. (default "nodes.cluster-api-provider-aws.sigs.k8s.io")
      --instance-type string                    Worker machine instance type (default
 "m5.2xlarge")
      --kubeconfig string                       Path to the kubeconfig for the
management cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string               Kubernetes version (default "1.24.7")
  -n, --namespace string                        If present, the namespace scope for
this CLI request. (default "default")
      --no-proxy strings                        No Proxy list for nodes (default [])
  -o, --output string                           Output format. One of: (json, yaml,
name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-
json, jsonpath-file).
      --output-directory string                 Used with --output=json|yaml. The
directory where to output resources to files. The directory must already exist.
      --replicas int                            Number of replicas (default 1)
      --show-managed-fields                     If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-public-key-file string              Path to the authorized SSH key for
 the user
      --ssh-username string                     Name of the user to create on the
instance (default "konvoy")
      --template string                         Template string or path to template
file to use when -o=go-template, -o=go-template-file. The template format is golang
templates [http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                        The length of time to wait before
giving up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --wait                                    If true, wait for operations to
complete before returning.
```

11.2.6.11.7.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.6.11.7.3 SEE ALSO

- dkp create nodepool (see page 1472) - Create a nodepool, one of [aks, aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.6.11.8 dkp create nodepool gcp

Create a nodepool in GCP

```
dkp create nodepool gcp name [flags]
```

11.2.6.11.8.1 Options

```
      --additional-tags stringToString    Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys       If true, ignore any errors in templates
when a field or map key is missing in the template. Only applies to golang and
jsonpath output formats. (default true)
      --associate-public-ip-address       Associate a public IP for all machines.
When set to false the specified network must have Cloud NAT configured to provide
internet access. (default true)
  -c, --cluster-name name                 Name used to prefix the cluster and all the
created resources.
      --dry-run                           Only print the objects that would be
created, without creating them.
  -h, --help                              help for gcp
      --http-proxy string                 HTTP proxy for nodes
      --https-proxy string                HTTPS proxy for nodes
      --image string                      Full reference to an image to use for all
nodes (set either this or --image-family) (ex. 'projects/my-project/global/images/
konvoy-ubuntu-2004-1-99-99-1234567890')
      --image-family string               Full reference to an image family to use
for all nodes (set either this or --image) (ex. 'projects/my-project/global/images/
family/konvoy-ubuntu-2004-{{.K8sVersion}}')
      --instance-type string              Worker machine instance type (default "n2-
standard-8")
      --kubeconfig string                 Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string         Kubernetes version (default "1.25.4")
  -n, --namespace string                  If present, the namespace scope for this
 CLI request. (default "default")
      --no-proxy strings                  No Proxy list for nodes (default [])
  -o, --output string                     Output format. One of: (json, yaml, name,
go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string           Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --registry-mirror-cacert file       CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string   Password to authenticate to the registry
mirror with
      --registry-mirror-url string        URL of a container registry to use as a
mirror in the cluster
```

```
      --registry-mirror-username string   Username to authenticate to the registry
mirror with
      --replicas int                      Number of replicas (default 1)
      --service-account-email string      Worker machine Service Account email
address (default "default")
      --show-managed-fields               If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-public-key-file string        Path to the authorized SSH key for the user
      --ssh-username string               Name of the user to create on the instance
(default "konvoy")
      --template string                   Template string or path to template file to
use when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                  The length of time to wait before giving
up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --wait                              If true, wait for operations to complete
before returning.
      --zone string                       Zone in the region to deploy the worker
nodes to, if not set a random one will be selected (ex. us-west1-a)
```

11.2.6.11.8.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.6.11.8.3 SEE ALSO

- dkp create nodepool (see page 1472) - Create a nodepool, one of [aks, aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.6.11.9 dkp create nodepool preprovisioned

Create a nodepool for a Pre Provisioned Cluster.

```
dkp create nodepool preprovisioned name [flags]
```

11.2.6.11.9.1 Options

```
      --additional-tags stringToString    Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys       If true, ignore any errors in templates
when a field or map key is missing in the template. Only applies to golang and
jsonpath output formats. (default true)
  -c, --cluster-name name                 Name used to prefix the cluster and all the
created resources.
```

```
      --dry-run                           Only print the objects that would be
created, without creating them.
  -h, --help                              help for preprovisioned
      --http-proxy string                 HTTP proxy for nodes
      --https-proxy string                HTTPS proxy for nodes
      --kubeconfig string                 Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string         Kubernetes version (default "1.25.4")
  -n, --namespace string                  If present, the namespace scope for this
 CLI request. (default "default")
      --no-proxy strings                  No Proxy list for nodes (default [])
      --os-hint flatcar                   A hint which will allow the installer to
generate appropriate configurations for a target OS. Presently, only the hint for
 flatcar is supported.
  -o, --output string                     Output format. One of: (json, yaml, name,
go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string           Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --override-secret-name string       Name of the secret for any provided
overrides on a preprovisioned cluster.All overrides defined at provisioning should be
present in this secret.
      --registry-mirror-cacert file       CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string   Password to authenticate to the registry
mirror with
      --registry-mirror-url string        URL of a container registry to use as a
mirror in the cluster
      --registry-mirror-username string   Username to authenticate to the registry
mirror with
      --replicas int                      Number of replicas (default 1)
      --show-managed-fields               If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-public-key-file string        Path to the authorized SSH key for the user
      --ssh-username string               Name of the user to create on the instance
(default "konvoy")
      --template string                   Template string or path to template file to
use when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                  The length of time to wait before giving
up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --wait                              If true, wait for operations to complete
before returning.
```

11.2.6.11.9.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.6.11.9.3 SEE ALSO

- dkp create nodepool (see page 1472) - Create a nodepool, one of [aks, aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.7 dkp delete

Delete one of [bootstrap, capi-components, chart, cluster, nodepool]

11.2.7.1 Options

```
  -h, --help   help for delete
```

11.2.7.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.7.3 SEE ALSO

- dkp delete bootstrap (see page 1484) - Delete bootstrap cluster
- dkp delete capi-components (see page 1485) - Delete the CAPI components from the cluster
- dkp delete chart (see page 1486) - Delete a chart from the repository
- dkp delete cluster (see page 1485) - Delete a Kubernetes cluster
- dkp delete nodepool (see page 1487) - Delete a nodepool for a given cluster

11.2.7.4 dkp delete bootstrap

Delete bootstrap cluster

```
dkp delete bootstrap [flags]
```

11.2.7.4.1 Options

```
  -h, --help                       help for bootstrap
      --kind-cluster-name string   Kind cluster name for the bootstrap cluster (defau
lt "konvoy-capi-bootstrapper")
```

```
      --kubeconfig string          Path to the kubeconfig for the management cluster.
If unspecified, default discovery rules apply.
      --timeout duration           The length of time to wait before giving up. Zero
means wait forever (e.g. 300s, 30m, 3h). (default 5m0s)
  -v, --verbose int                Output verbosity
      --wait                       If true, wait for operations to complete before
returning. (default true)
```

11.2.7.4.2 SEE ALSO

- dkp delete (see page 1484) - Delete one of [bootstrap, capi-components, chart, cluster, nodepool]

11.2.7.5 dkp delete capi-components

Delete the CAPI components from the cluster

```
dkp delete capi-components [flags]
```

11.2.7.5.1 Options

```
  -h, --help                 help for capi-components
      --kubeconfig string    Path to the kubeconfig for the management cluster. If
unspecified, default discovery rules apply.
      --timeout duration     The length of time to wait before giving up. Zero means
wait forever (e.g. 300s, 30m, 3h). (default 5m0s)
  -v, --verbose int          Output verbosity
      --wait                 If true, wait for operations to complete before
returning. (default true)
```

11.2.7.5.2 SEE ALSO

- dkp delete (see page 1484) - Delete one of [bootstrap, capi-components, chart, cluster, nodepool]

11.2.7.6 dkp delete cluster

Delete a Kubernetes cluster

```
dkp delete cluster [flags]
```

11.2.7.6.1 Options

```
      --aws-service-endpoints string     Custom AWS service endpoints in a semi-colon
separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=${URL};$
{SigningRegion2}...
  -c, --cluster-name name                Name used to prefix the cluster and all the
created resources.
      --delete-kubernetes-resources      Delete Kubernetes resources on the cluster
before deleting that cluster (Services with type LoadBalancer) (default true)
  -h, --help                             help for cluster
      --http-proxy string                HTTP proxy for CAPI controllers
      --https-proxy string               HTTPS proxy for CAPI controllers
      --kind-cluster-image string        Kind node image for the bootstrap cluster (d
efault "mesosphere/konvoy-bootstrap:v2.5.2")
      --kubeconfig string                Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
  -n, --namespace string                 If present, the namespace scope for this CLI
request. (default "default")
      --no-proxy strings                 No Proxy list for CAPI controllers (default
 [])
      --self-managed                     When set to true, the required prerequisites
and resources are moved from the self managed cluster before deleting. When set to
false, the resources are assumed installed in a management cluster. (default false)
      --timeout duration                 The length of time to wait before giving up.
Zero means wait forever (e.g. 300s, 30m, 3h). (default 15m0s)
  -v, --verbose int                      Output verbosity
      --wait                             If true, wait for operations to complete
before returning. This flag is ignored and will always be 'true' if used with the --
self-managed flag. (default true)
      --with-aws-bootstrap-credentials   Set true to use AWS bootstrap credentials
from your environment. When false, the instance profile of the EC2 instance where the
CAPA controller is scheduled on will be used instead.
      --with-gcp-bootstrap-credentials   Set true to use GCP bootstrap credentials
from your environment. When false, the service account of the VM instance where the
CAPG controller is scheduled on will be used instead.
```

11.2.7.6.2 SEE ALSO

- dkp delete (see page 1484) - Delete one of [bootstrap, capi-components, chart, cluster, nodepool]

11.2.7.7 dkp delete chart

Delete a chart from the repository

```
dkp delete chart [chartName] [chartVersion] [flags]
```

11.2.7.7.1 Options

```
      --config string            Config file to use (default "/root/.kommander/
config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for chart
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

11.2.7.7.2 SEE ALSO

- dkp delete (see page 1484) - Delete one of [bootstrap, capi-components, chart, cluster, nodepool]

11.2.7.8 dkp delete nodepool

Delete a nodepool for a given cluster

```
dkp delete nodepool name [flags]
```

11.2.7.8.1 Options

```
  -c, --cluster-name name   Name used to prefix the cluster and all the created
resources.
      --dry-run             Only print the objects that would be created, without
creating them.
  -h, --help                help for nodepool
      --kubeconfig string   Path to the kubeconfig for the management cluster. If
unspecified, default discovery rules apply.
  -n, --namespace string    If present, the namespace scope for this CLI request. (de
fault "default")
  -v, --verbose int         Output verbosity
```

11.2.7.8.2 SEE ALSO

- dkp delete (see page 1484) - Delete one of [bootstrap, capi-components, chart, cluster, nodepool]

11.2.8 dkp describe

Describe one of [cluster]

11.2.8.1 Options

```
  -h, --help          help for describe
  -v, --verbose int   Output verbosity
```

11.2.8.2 SEE ALSO

- dkp describe cluster (see page 1488) - Describe a Kubernetes cluster status

11.2.8.3 dkp describe cluster

Describe a Kubernetes cluster status

```
dkp describe cluster [flags]
```

11.2.8.3.1 Options

```
  -c, --cluster-name name   Name used to prefix the cluster and all the created
resources.
  -h, --help                help for cluster
      --kubeconfig string   Path to the kubeconfig for the management cluster. If
unspecified, default discovery rules apply.
  -n, --namespace string    If present, the namespace scope for this CLI request. (de
fault "default")
```

11.2.8.3.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.8.3.3 SEE ALSO

- dkp describe (see page 1488) - Describe one of [cluster]

11.2.9 dkp detach

Detach one of [cluster]

11.2.9.1 Options

```
      --config string            Config file to use (default "/root/.kommander/
config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for detach
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

11.2.9.2 SEE ALSO

- dkp detach cluster (see page 1489) - Detach a cluster

11.2.9.3 dkp detach cluster

Detach a cluster

```
dkp detach cluster CLUSTER_NAME [flags]
```

11.2.9.3.1 Options

```
  -h, --help               help for cluster
      --timeout duration   The length of time to wait before giving up on a detach,
defaults to wait forever (default 0s)
      --wait               If true, wait for resources to be gone before returning.
This waits for finalizers. (default true)
  -w, --workspace string   Name of the workspace of the attached cluster
```

11.2.9.3.2 Options inherited from parent commands

```
      --config string            Config file to use (default "/root/.kommander/
config")
      --context string           The name of the kubeconfig context to use
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

11.2.9.3.3 SEE ALSO

- dkp detach (see page 1489) - Detach one of [cluster]

11.2.10 dkp diagnose

Generate a support bundle

11.2.10.1 Synopsis

A support bundle is an archive of files, output, metrics and state from a server that can be used to assist when troubleshooting a Kubernetes cluster.

```
dkp diagnose [flags]
```

11.2.10.2 Options

```
      --allow-insecure-connections     When set, do not verify TLS certs when
retrieving spec and reporting results
      --as string                      Username to impersonate for the operation.
User could be a regular user or a service account in a namespace.
      --as-group stringArray           Group to impersonate for the operation, this
 flag can be repeated to specify multiple groups. (default [])
      --as-uid string                  UID to impersonate for the operation.
      --bootstrap-kubeconfig string    Path to the kubeconfig file to use for
 requests towards an additional bootstrap cluster
      --cache-dir string               Default cache directory (default "/root/.kube/
cache")
      --certificate-authority string   Path to a cert file for the certificate
authority
```

```
      --client-certificate string      Path to a client certificate file for TLS
      --client-key string              Path to a client key file for TLS
      --cluster string                 The name of the kubeconfig cluster to use
      --collect-without-permissions    Always generate a support bundle, even if it
some require additional permissions (default true)
      --context string                 The name of the kubeconfig context to use
      --disable-compression            If true, opt-out of response compression for
 all requests to the server
  -h, --help                           help for diagnose
      --insecure-skip-tls-verify       If true, the server's certificate will not be
checked for validity. This will make your HTTPS connections insecure
      --kubeconfig string              Path to the kubeconfig file to use for CLI
requests.
  -n, --namespace string               If present, the namespace scope for this CLI
request
      --redactors strings              Names of the additional redactors to use
(default [])
      --request-timeout string         The length of time to wait before giving up on
a single server request. Non-zero values should contain a corresponding time unit
(e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
  -s, --server string                  The address and port of the Kubernetes API
server
      --since string                   Force pod logs collectors to return logs newer
than a relative duration like 5s, 2m, or 3h.
      --since-time string              Force pod logs collectors to return logs after
a specific date (RFC3339)
      --tls-server-name string         Server name to use for server certificate
validation. If it is not provided, the hostname used to contact the server is used
      --token string                   Bearer token for authentication to the API
server
      --user string                    The name of the kubeconfig user to use
  -v, --verbose int                    Output verbosity
```

11.2.10.3 SEE ALSO

- dkp diagnose default-config (see page 1492) - Prints the default configuration of the diagnostics bundle collectors
- dkp diagnose ssh (see page 1491) - Collect node-level diagnostics data over SSH

11.2.10.4 dkp diagnose ssh

Collect node-level diagnostics data over SSH

```
dkp diagnose ssh path/to/inventory-file.yaml [flags]
```

11.2.10.4.1 Options

```
  -h, --help                help for ssh
      --redactors strings   Names of the additional redactors to use (default [])
      --timeout duration    Timeout for collecting bundle per node (default 5m0s)
```

11.2.10.4.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.10.4.3 SEE ALSO

- dkp diagnose (see page 1490) - Generate a support bundle

11.2.10.5 dkp diagnose default-config

Prints the default configuration of the diagnostics bundle collectors

```
dkp diagnose default-config [flags]
```

11.2.10.5.1 Options

```
  -h, --help   help for default-config
```

11.2.10.5.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.10.5.3 SEE ALSO

- dkp diagnose (see page 1490) - Generate a support bundle

11.2.11 dkp get

Get one of [appdeployments, chart, clusters, kubeconfig, nodepools, workspaces]

11.2.11.1 Options

```
  -h, --help   help for get
```

11.2.11.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.11.3 SEE ALSO

- dkp get appdeployments (see page 1494) - Get AppDeployments from a Workspace, Project, or all Workspaces and Projects
- dkp get chart (see page 1496) - Obtain information about charts stored in the repository
- dkp get clusters (see page 1497) - Get clusters from specified Workspace
- dkp get kubeconfig (see page 1493) - Retrieve cluster kubeconfig and modify local kubeconfig file
- dkp get nodepools (see page 1495) - Get nodepools for a given cluster
- dkp get workspaces (see page 1495) - Get Workspaces

11.2.11.4 dkp get kubeconfig

Retrieve cluster kubeconfig and modify local kubeconfig file

```
dkp get kubeconfig [flags]
```

11.2.11.4.1 Options

```
      --cluster string      Kommander Cluster to get kubeconfig for
  -c, --cluster-name name   Name used to prefix the cluster and all the created
resources.
  -h, --help                help for kubeconfig
      --kubeconfig string   Path to the kubeconfig for the management cluster. If
unspecified, default discovery rules apply.
```

```
  -n, --namespace string    If present, the namespace scope for this CLI request. (de
fault "default")
  -w, --workspace string    Name of the workspace to show clusters from
```

11.2.11.4.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.11.4.3 SEE ALSO

- dkp get (see page 1493) - Get one of [appdeployments, chart, clusters, kubeconfig, nodepools, workspaces]

11.2.11.5 dkp get appdeployments

Get AppDeployments from a Workspace, Project, or all Workspaces and Projects

11.2.11.5.1 Synopsis

Prints a table of the most important information about the specified resources. In case the AppDeployments are configured with cluster scoped specification, an optional CLUSTERS column (when printing in table format) will display enabled clusters for each AppDeployment.

```
dkp get appdeployments [APPDEPLOYMENT_NAME] [flags]
```

11.2.11.5.2 Options

```
  -A, --all-namespaces           If present, list the requested object(s) across all
namespaces.
      --config string            Config file to use (default "/root/.kommander/
config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for appdeployments
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
  -o, --output string            Output format. One of: table|yaml
  -p, --project string           Name of the project to show AppDeployments from.
Requires workspace flag (workspace that the project belongs to).
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

```
  -w, --workspace string         Name of the workspace to show AppDeployments from
```

11.2.11.5.3 SEE ALSO

- dkp get (see page 1493) - Get one of [appdeployments, chart, clusters, kubeconfig, nodepools, workspaces]

11.2.11.6 dkp get workspaces

Get Workspaces

```
dkp get workspaces [flags]
```

11.2.11.6.1 Options

```
  -A, --all-namespaces           If present, list the requested object(s) across all
namespaces.
      --config string            Config file to use (default "/root/.kommander/
config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for workspaces
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
  -o, --output string            Output format. One of: table|yaml
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

11.2.11.6.2 SEE ALSO

- dkp get (see page 1493) - Get one of [appdeployments, chart, clusters, kubeconfig, nodepools, workspaces]

11.2.11.7 dkp get nodepools

Get nodepools for a given cluster

```
dkp get nodepools [flags]
```

11.2.11.7.1 Options

```
  -c, --cluster-name name   Name used to prefix the cluster and all the created
resources.
  -h, --help                help for nodepools
      --kubeconfig string   Path to the kubeconfig for the management cluster. If
unspecified, default discovery rules apply.
  -n, --namespace string    If present, the namespace scope for this CLI request. (de
fault "default")
  -v, --verbose int         Output verbosity
```

11.2.11.7.2 SEE ALSO

- dkp get (see page 1493) - Get one of [appdeployments, chart, clusters, kubeconfig, nodepools, workspaces]

11.2.11.8 dkp get chart

Obtain information about charts stored in the repository

```
dkp get chart [chartName] [chartVersion] [flags]
```

11.2.11.8.1 Options

```
  -A, --all-namespaces           If present, list the requested object(s) across all
namespaces.
      --config string            Config file to use (default "/root/.kommander/
config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for chart
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
  -o, --output string            Output format. One of: table|yaml
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

11.2.11.8.2 SEE ALSO

- dkp get (see page 1493) - Get one of [appdeployments, chart, clusters, kubeconfig, nodepools, workspaces]

11.2.11.9 dkp get clusters

Get clusters from specified Workspace

```
dkp get clusters [CLUSTER_NAME] [flags]
```

11.2.11.9.1 Options

```
  -A, --all-namespaces           If present, list the requested object(s) across all
namespaces.
      --config string            Config file to use (default "/root/.kommander/
config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for clusters
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
  -o, --output string            Output format. One of: table|yaml
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
  -w, --workspace string         Name of the workspace to show clusters from
```

11.2.11.9.2 SEE ALSO

- dkp get (see page 1493) - Get one of [appdeployments, chart, clusters, kubeconfig, nodepools, workspaces]

11.2.12 dkp import

Import images from an image bundle into Containerd

11.2.12.1 Options

```
  -h, --help          help for import
  -v, --verbose int   Output verbosity
```

11.2.12.2 SEE ALSO

- dkp import image-bundle (see page 1498) - Import images from image bundles into Containerd

11.2.12.3 dkp import image-bundle

Import images from image bundles into Containerd

```
dkp import image-bundle [flags]
```

11.2.12.3.1 Options

```
      --containerd-namespace string   Containerd namespace to import images into (def
ault "k8s.io")
  -h, --help                          help for image-bundle
      --image-bundle strings          Tarball containing list of images to import.
Can also be a glob pattern. (default [])
```

11.2.12.3.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.12.3.3 SEE ALSO

- dkp import (see page 1497) - Import images from an image bundle into Containerd

11.2.13 dkp install

Install one of [kommander]

11.2.13.1 Options

```
      --config string            Config file to use (default "/root/.kommander/
config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for install
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

11.2.13.2 SEE ALSO

- dkp install kommander (see page 1499) - Install kommander

11.2.13.3 dkp install kommander

Install kommander

```
dkp install kommander [flags]
```

11.2.13.3.1 Options

```
      --airgapped                                  Enable airgapped mode.
      --bootstrap-repository string                git repository with bootstrap
definitions
      --charts-bundle stringArray                  Path to charts-bundle to upload to
chartmuseum, apart from parsing the kommander applications repository (default [])
      --disallow-charts-download                   make CLI rely solely on provided
chart bundles and do not try to download charts from the Internet
      --gitea-kommander-repository-name string     gitea kommander repository name
(default "kommander")
  -h, --help                                       help for kommander
      --init                                       Initialize default configuration
file, print it and exit without installing Kommander.
      --installer-config file                      Path to installation configuration
file
      --kommander-applications-repository string   git repository with application
definitions (default "v2.5.2")
  -o, --output string                              Output format for the
configuration file generated by --init. One of: yaml|json (default "yaml")
      --wait                                       Wait for all enabled applications
to be ready (default true)
      --wait-timeout duration                      Time to wait for all enabled
applications to be ready (30m, 1h, 2h) (default 1h0m0s)
```

11.2.13.3.2 Options inherited from parent commands

```
      --config string           Config file to use (default "/root/.kommander/
config")
      --context string          The name of the kubeconfig context to use
      --kubeconfig string       Path to the kubeconfig file to use for CLI requests.
```

```
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

11.2.13.3.3 SEE ALSO

- dkp install (see page 1498) - Install one of [kommander]

11.2.14 dkp move

Move one of [capi-resources]

11.2.14.1 Synopsis

Command "move" is deprecated, use "dkp move capi-resources" instead Move one of [capi-resources]

```
dkp move [flags]
```

11.2.14.2 Options

```
      --from-context string    Context to be used within the from-cluster's
kubeconfig file. If empty, current context will be used. (DEPRECATED: use "dkp move
capi-resources" instead)
      --from-kubeconfig file   Path to the kubeconfig for pivot's source cluster. If
unspecified, default discovery rules apply. (DEPRECATED: use "dkp move capi-
resources" instead)
  -h, --help                   help for move
  -n, --namespace string       If present, the namespace scope for this CLI request.
(default "default")
      --to-context string      Context to be used within the to-cluster's kubeconfig
file. If empty, current context will be used. (DEPRECATED: use "dkp move capi-
resources" instead)
      --to-kubeconfig file     Path to the kubeconfig for pivot's destination cluster
(DEPRECATED: use "dkp move capi-resources" instead)
      --to-namespace string    Resources are moved to this namespace in the to-
cluster.By default, the same as the from-cluster namespace.
  -v, --verbose int            Output verbosity
```

11.2.14.3 SEE ALSO

- dkp move capi-resources (see page 1501) - Move controllers and objects from one cluster to the other

11.2.14.4 dkp move capi-resources

Move controllers and objects from one cluster to the other

```
dkp move capi-resources [flags]
```

11.2.14.4.1 Options

```
      --from-context string    Context to be used within the from-cluster's
kubeconfig file. If empty, current context will be used.
      --from-kubeconfig file   Path to the kubeconfig for pivot's source cluster. If
unspecified, default discovery rules apply.
  -h, --help                   help for capi-resources
  -n, --namespace string       If present, the namespace scope for this CLI request.
(default "default")
      --to-context string      Context to be used within the to-cluster's kubeconfig
file. If empty, current context will be used.
      --to-kubeconfig file     Path to the kubeconfig for pivot's destination cluster
      --to-namespace string    Resources are moved to this namespace in the to-
cluster.By default, the same as the from-cluster namespace.
```

11.2.14.4.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.14.4.3 SEE ALSO

- dkp move (see page 1500) - Move one of [capi-resources]

11.2.15 dkp open

Open one of [dashboard]

11.2.15.1 Options

```
      --config string           Config file to use (default "/root/.kommander/
config")
      --context string          The name of the kubeconfig context to use
```

```
  -h, --help                     help for open
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

11.2.15.2 SEE ALSO

- dkp open dashboard (see page 1502) - Open DKP UI in your browser

11.2.15.3 dkp open dashboard

Open DKP UI in your browser

```
dkp open dashboard [flags]
```

11.2.15.3.1 Options

```
  -h, --help   help for dashboard
```

11.2.15.3.2 Options inherited from parent commands

```
      --config string            Config file to use (default "/root/.kommander/
config")
      --context string           The name of the kubeconfig context to use
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

11.2.15.3.3 SEE ALSO

- dkp open (see page 1501) - Open one of [dashboard]

11.2.16 dkp push

Push one of [chart, chart-bundle, image-bundle]

11.2.16.1 Options

```
  -h, --help   help for push
```

11.2.16.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.16.3 SEE ALSO

- dkp push chart (see page 1504) - Upload charts to the repository
- dkp push chart-bundle (see page 1503) - Upload chart bundles to the repository
- dkp push image-bundle (see page 1504) - Push images from an image bundle into an existing OCI registry

11.2.16.4 dkp push chart-bundle

Upload chart bundles to the repository

```
dkp push chart-bundle [chartsTarball]... [flags]
```

11.2.16.4.1 Options

```
      --config string           Config file to use (default "/root/.kommander/
config")
      --context string          The name of the kubeconfig context to use
  -h, --help                    help for chart-bundle
      --kubeconfig string       Path to the kubeconfig file to use for CLI requests.
      --request-timeout string  The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int             Output verbosity
```

11.2.16.4.2 SEE ALSO

- dkp push (see page 1502) - Push one of [chart, chart-bundle, image-bundle]

11.2.16.5 dkp push image-bundle

Push images from an image bundle into an existing OCI registry

```
dkp push image-bundle [flags]
```

11.2.16.5.1 Options

```
      --ecr-lifecycle-policy-file string       File containing ECR lifecycle policy
for newly created repositories (only applies if target registry is hosted on ECR,
ignored otherwise)
  -h, --help                                   help for image-bundle
      --image-bundle strings                   Tarball containing list of images to
push. Can also be a glob pattern. (default [])
      --to-registry string                     Registry to push images to. TLS
verification will be skipped when using an http:// registry.
      --to-registry-ca-cert-file string        CA certificate file used to verify TLS
verification of registry to push images to
      --to-registry-insecure-skip-tls-verify   Skip TLS verification of registry to
push images to (also use for non-TLS http registries)
      --to-registry-password string            Password to use to log in to
destination registry
      --to-registry-username string            Username to use to log in to
destination registry
  -v, --verbose int                            Output verbosity
```

11.2.16.5.2 SEE ALSO

- dkp push (see page 1502) - Push one of [chart, chart-bundle, image-bundle]

11.2.16.6 dkp push chart

Upload charts to the repository

```
dkp push chart [chartTarball]... [flags]
```

11.2.16.6.1 Options

```
      --config string           Config file to use (default "/root/.kommander/
config")
```

```
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for chart
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

11.2.16.6.2 SEE ALSO

- dkp push (see page 1502) - Push one of [chart, chart-bundle, image-bundle]

11.2.17 dkp scale

Scale one of [nodepool]

11.2.17.1 Options

```
  -h, --help          help for scale
  -v, --verbose int   Output verbosity
```

11.2.17.2 SEE ALSO

- dkp scale nodepool (see page 1505) - Scale a nodepool of a given cluster to the number of replicas

11.2.17.3 dkp scale nodepool

Scale a nodepool of a given cluster to the number of replicas

```
dkp scale nodepool name [flags]
```

11.2.17.3.1 Options

```
  -c, --cluster-name name         Name used to prefix the cluster and all the created
resources.
  -h, --help                      help for nodepool
      --kubeconfig string         Path to the kubeconfig for the management cluster.
If unspecified, default discovery rules apply.
  -n, --namespace string          If present, the namespace scope for this CLI
request. (default "default")
```

```
      --nodes-to-delete strings   A list of node names to mark for deletion when
scaling down a node pool. If left empty, the nodes to delete will be selected at
random. (default [])
      --replicas int              The new desired number of replicas.
```

11.2.17.3.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.17.3.3 SEE ALSO

- dkp scale (see page 1505) - Scale one of [nodepool]

11.2.18 dkp serve

Serve image or Helm chart bundles from an OCI registry

11.2.18.1 Options

```
  -h, --help          help for serve
  -v, --verbose int   Output verbosity
```

11.2.18.2 SEE ALSO

- dkp serve image-bundle (see page 1506) - Serve an OCI registry from image bundles

11.2.18.3 dkp serve image-bundle

Serve an OCI registry from image bundles

```
dkp serve image-bundle [flags]
```

11.2.18.3.1 Options

```
  -h, --help                        help for image-bundle
      --image-bundle strings        Tarball of images to serve. Can also be a glob
pattern. (default [])
```

```
      --listen-address string         Address to listen on (default "localhost")
      --listen-port uint16            Port to listen on (0 means use any free port)
      --tls-cert-file string          TLS certificate file
      --tls-private-key-file string   TLS private key file
```

11.2.18.3.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.18.3.3 SEE ALSO

- dkp serve (see page 1506) - Serve image or Helm chart bundles from an OCI registry

11.2.19 dkp update

Update one of [bootstrap (cluster), controlplane, nodepool]

11.2.19.1 Options

```
  -h, --help          help for update
  -v, --verbose int   Output verbosity
```

11.2.19.2 SEE ALSO

- dkp update bootstrap (see page 1519) - Update bootstrap cluster
- dkp update controlplane (see page 1513) - Update a Kubernetes cluster control plane, one of [aws, azure, eks, gcp, preprovisioned, vsphere]
- dkp update nodepool (see page 1507) - Upate a Kubernetes cluster node pool, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.19.3 dkp update nodepool

Upate a Kubernetes cluster node pool, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.19.3.1 Options

```
  -h, --help   help for nodepool
```

11.2.19.3.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.19.3.3 SEE ALSO

- dkp update (see page 1507) - Update one of [bootstrap (cluster), controlplane, nodepool]
- dkp update nodepool aws (see page 1509) - Update a Konvoy cluster node pool in AWS
- dkp update nodepool azure (see page 1508) - Update a Konvoy cluster node pool in Azure
- dkp update nodepool eks (see page 1511) - Update a Konvoy cluster node pool in EKS
- dkp update nodepool gcp (see page 1512) - Update a Konvoy cluster node pool in GCP
- dkp update nodepool preprovisioned (see page 1510) - Update a Konvoy cluster node pool in Preprovisioned
- dkp update nodepool vsphere (see page 1511) - Update a Konvoy cluster node pool in vSphere

11.2.19.3.4 dkp update nodepool azure

Update a Konvoy cluster node pool in Azure

```
dkp update nodepool azure [flags]
```

11.2.19.3.4.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
      --compute-gallery-id string   Compute Gallery ID of a custom image, e.g., '/
subscriptions/<subscription id>/resourceGroups/<resource group name>/providers/
Microsoft.Compute/galleries/<gallery name>/images/<image definition name>/versions/
<version id>' (replacing placeholders with the values used when creating the image)
  -h, --help                        help for azure
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
      --machine-size string         Worker machine size (ex. 'Standard_D2s_v3')
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --plan-offer string           The offer for a Marketplace image or a custom
image sourced from a Marketplace image requiring Plan information.
      --plan-publisher string       The publisher for a Marketplace image or a custom
image sourced from a Marketplace image requiring Plan information.
```

```
      --plan-sku string             The SKU for a Marketplace image or a custom image
sourced from a Marketplace image requiring Plan information.
      --use-context string          Use a specific context in a kubeconfig file.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

11.2.19.3.4.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.19.3.4.3 SEE ALSO

- dkp update nodepool (see page 1507) - Upate a Kubernetes cluster node pool, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.19.3.5 dkp update nodepool aws

Update a Konvoy cluster node pool in AWS

```
dkp update nodepool aws [flags]
```

11.2.19.3.5.1 Options

```
      --ami string                  AMI ID to use for machines
      --ami-base-os string          Base OS for Lookup search (ex. 'centos-7',
'ubuntu-18.04', 'ubuntu-20.04')
      --ami-format string           Lookup Format string to generate AMI search name
from
      --ami-owner string            Owner ID for AMI Lookup search
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
  -h, --help                        help for aws
      --instance-type string        Instance type to use for node pool machines
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --use-context string          Use a specific context in a kubeconfig file.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

11.2.19.3.5.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.19.3.5.3 SEE ALSO

- dkp update nodepool (see page 1507) - Upate a Kubernetes cluster node pool, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.19.3.6 dkp update nodepool preprovisioned

Update a Konvoy cluster node pool in Preprovisioned

```
dkp update nodepool preprovisioned [flags]
```

11.2.19.3.6.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
  -h, --help                        help for preprovisioned
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --use-context string          Use a specific context in a kubeconfig file.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

11.2.19.3.6.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.19.3.6.3 SEE ALSO

- dkp update nodepool (see page 1507) - Upate a Kubernetes cluster node pool, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.19.3.7 dkp update nodepool eks

Update a Konvoy cluster node pool in EKS

```
dkp update nodepool eks [flags]
```

11.2.19.3.7.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
  -h, --help                        help for eks
      --instance-type string        Instance type to use for node pool machines
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --use-context string          Use a specific context in a kubeconfig file.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

11.2.19.3.7.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.19.3.7.3 SEE ALSO

- dkp update nodepool (see page 1507) - Upate a Kubernetes cluster node pool, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.19.3.8 dkp update nodepool vsphere

Update a Konvoy cluster node pool in vSphere

```
dkp update nodepool vsphere [flags]
```

11.2.19.3.8.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
      --cpus int                    The number of virtual processors in a virtual
machine.
      --disk-size int               The size of a virtual machine's disk, in GB.
  -h, --help                        help for vsphere
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
      --memory int                  The size of a virtual machine's memory, in GB.
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --use-context string          Use a specific context in a kubeconfig file.
      --vm-template string          The virtual machine template to use for a virtual
machine.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

11.2.19.3.8.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.19.3.8.3 SEE ALSO

- dkp update nodepool (see page 1507) - Upate a Kubernetes cluster node pool, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.19.3.9 dkp update nodepool gcp

Update a Konvoy cluster node pool in GCP

```
dkp update nodepool gcp [flags]
```

11.2.19.3.9.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
  -h, --help                        help for gcp
```

```
      --image string                Full reference to an image to use for all nodes
(set either this or --image-family) (ex. 'projects/my-project/global/images/konvoy-
ubuntu-2004-1-99-99-1234567890')
      --image-family string         Full reference to an image family to use for all
nodes (set either this or --image) (ex. 'projects/my-project/global/images/family/
konvoy-ubuntu-2004-{{.K8sVersion}}')
      --instance-type string        Worker machine instance type (ex. "n2-standard-8")
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --use-context string          Use a specific context in a kubeconfig file.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

11.2.19.3.9.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.19.3.9.3 SEE ALSO

- dkp update nodepool (see page 1507) - Upate a Kubernetes cluster node pool, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.19.4 dkp update controlplane

Update a Kubernetes cluster control plane, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.19.4.1 Options

```
  -h, --help   help for controlplane
```

11.2.19.4.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.19.4.3 SEE ALSO

- dkp update (see page 1507) - Update one of [bootstrap (cluster), controlplane, nodepool]

- dkp update controlplane aws (see page 1514) - Update a Konvoy cluster control plane in AWS
- dkp update controlplane azure (see page 1515) - Update a Konvoy cluster control plane in Azure
- dkp update controlplane eks (see page 1516) - Update a Konvoy cluster control plane in EKS
- dkp update controlplane gcp (see page 1518) - Update a Konvoy cluster control plane in GCP
- dkp update controlplane preprovisioned (see page 1514) - Update a Konvoy cluster control plane in Preprovisioned
- dkp update controlplane vsphere (see page 1517) - Update a Konvoy cluster control plane in vSphere

11.2.19.4.4 dkp update controlplane preprovisioned

Update a Konvoy cluster control plane in Preprovisioned

```
dkp update controlplane preprovisioned [flags]
```

11.2.19.4.4.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
  -h, --help                        help for preprovisioned
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --use-context string          Use a specific context in a kubeconfig file.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

11.2.19.4.4.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.19.4.4.3 SEE ALSO

- dkp update controlplane (see page 1513) - Update a Kubernetes cluster control plane, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.19.4.5 dkp update controlplane aws

Update a Konvoy cluster control plane in AWS

```
dkp update controlplane aws [flags]
```

11.2.19.4.5.1 Options

```
      --ami string                  AMI ID to use for machines
      --ami-base-os string          Base OS for Lookup search (ex. 'centos-7',
'ubuntu-18.04', 'ubuntu-20.04')
      --ami-format string           Lookup Format string to generate AMI search name
from
      --ami-owner string            Owner ID for AMI Lookup search
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
  -h, --help                        help for aws
      --instance-type string        Instance type to use for control plane machines
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --use-context string          Use a specific context in a kubeconfig file.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

11.2.19.4.5.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.19.4.5.3 SEE ALSO

- dkp update controlplane (see page 1513) - Update a Kubernetes cluster control plane, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.19.4.6 dkp update controlplane azure

Update a Konvoy cluster control plane in Azure

```
dkp update controlplane azure [flags]
```

11.2.19.4.6.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
      --compute-gallery-id string   Compute Gallery ID of a custom image, e.g., '/
subscriptions/<subscription id>/resourceGroups/<resource group name>/providers/
Microsoft.Compute/galleries/<gallery name>/images/<image definition name>/versions/
<version id>' (replacing placeholders with the values used when creating the image)
  -h, --help                        help for azure
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
      --machine-size string         Worker machine size (ex. 'Standard_D2s_v3')
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --plan-offer string           The offer for a Marketplace image or a custom
image sourced from a Marketplace image requiring Plan information.
      --plan-publisher string       The publisher for a Marketplace image or a custom
image sourced from a Marketplace image requiring Plan information.
      --plan-sku string             The SKU for a Marketplace image or a custom image
sourced from a Marketplace image requiring Plan information.
      --use-context string          Use a specific context in a kubeconfig file.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

11.2.19.4.6.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.19.4.6.3 SEE ALSO

- dkp update controlplane (see page 1513) - Update a Kubernetes cluster control plane, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.19.4.7 dkp update controlplane eks

Update a Konvoy cluster control plane in EKS

```
dkp update controlplane eks [flags]
```

11.2.19.4.7.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
  -h, --help                        help for eks
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --use-context string          Use a specific context in a kubeconfig file.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

11.2.19.4.7.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.19.4.7.3 SEE ALSO

- dkp update controlplane (see page 1513) - Update a Kubernetes cluster control plane, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.19.4.8 dkp update controlplane vsphere

Update a Konvoy cluster control plane in vSphere

```
dkp update controlplane vsphere [flags]
```

11.2.19.4.8.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
      --cpus int                    The number of virtual processors in a virtual
machine.
      --disk-size int               The size of a virtual machine's disk, in GB.
  -h, --help                        help for vsphere
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
```

```
      --memory int                    The size of a virtual machine's memory, in GB.
  -n, --namespace string              If present, the namespace scope for this CLI
request. (default "default")
      --use-context string            Use a specific context in a kubeconfig file.
      --vm-template string            The virtual machine template to use for a virtual
machine.
      --wait                          If true, wait for operations to complete before
returning. (default true)
```

11.2.19.4.8.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.19.4.8.3 SEE ALSO

- dkp update controlplane (see page 1513) - Update a Kubernetes cluster control plane, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.19.4.9 dkp update controlplane gcp

Update a Konvoy cluster control plane in GCP

```
dkp update controlplane gcp [flags]
```

11.2.19.4.9.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
  -h, --help                        help for gcp
      --image string                Full reference to an image to use for all nodes
(set either this or --image-family) (ex. 'projects/my-project/global/images/konvoy-
ubuntu-2004-1-99-99-1234567890')
      --image-family string         Full reference to an image family to use for all
nodes (set either this or --image) (ex. 'projects/my-project/global/images/family/
konvoy-ubuntu-2004-{{.K8sVersion}}')
      --instance-type string        Control Plane machine instance type (ex. "n2-
standard-4")
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --use-context string          Use a specific context in a kubeconfig file.
```

```
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

11.2.19.4.9.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.19.4.9.3 SEE ALSO

- dkp update controlplane (see page 1513) - Update a Kubernetes cluster control plane, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.19.5 dkp update bootstrap

Update bootstrap cluster

11.2.19.5.1 Options

```
  -h, --help   help for bootstrap
```

11.2.19.5.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.19.5.3 SEE ALSO

- dkp update (see page 1507) - Update one of [bootstrap (cluster), controlplane, nodepool]
- dkp update bootstrap credentials (see page 1519) - Update credentials in the cluster

11.2.19.5.4 dkp update bootstrap credentials

Update credentials in the cluster

11.2.19.5.4.1 Options

```
  -h, --help   help for credentials
```

11.2.19.5.4.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.19.5.4.3 SEE ALSO

- dkp update bootstrap (see page 1519) - Update bootstrap cluster
- dkp update bootstrap credentials aws (see page 1520) - Update AWS credentials in the cluster and restart CAPA controllers
- dkp update bootstrap credentials azure (see page 1521) - Update Azure credentials in the cluster and restart CAPZ controllers
- dkp update bootstrap credentials gcp (see page 1521) - Update GCP credentials in the cluster and restart CAPG controllers
- dkp update bootstrap credentials vsphere (see page 1522) - Update VSphere credentials in the cluster and restart CAPV controllers

11.2.19.5.4.4 dkp update bootstrap credentials aws

Update AWS credentials in the cluster and restart CAPA controllers

```
dkp update bootstrap credentials aws [flags]
```

Options

```
      --context string      The name of the kubeconfig context to use
  -h, --help                help for aws
      --kubeconfig string   Path to the kubeconfig for the management cluster. If
unspecified, default discovery rules apply.
      --print-only          Print the credentials and exit the function. Without
modifying cluster
```

Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

SEE ALSO

- dkp update bootstrap credentials (see page 1519) - Update credentials in the cluster

11.2.19.5.4.5 dkp update bootstrap credentials azure

Update Azure credentials in the cluster and restart CAPZ controllers

```
dkp update bootstrap credentials azure [flags]
```

Options

```
      --context string      The name of the kubeconfig context to use
  -h, --help                help for azure
      --kubeconfig string   Path to the kubeconfig for the management cluster. If
unspecified, default discovery rules apply.
      --print-only          Print the credentials and exit the function. Without
modifying cluster
```

Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

SEE ALSO

- dkp update bootstrap credentials (see page 1519) - Update credentials in the cluster

11.2.19.5.4.6 dkp update bootstrap credentials gcp

Update GCP credentials in the cluster and restart CAPG controllers

```
dkp update bootstrap credentials gcp [flags]
```

Options

```
      --context string      The name of the kubeconfig context to use
  -h, --help                help for gcp
      --kubeconfig string   Path to the kubeconfig for the management cluster. If
unspecified, default discovery rules apply.
      --print-only          Print the credentials and exit the function. Without
modifying cluster
```

Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

SEE ALSO

- dkp update bootstrap credentials (see page 1519) - Update credentials in the cluster

11.2.19.5.4.7 dkp update bootstrap credentials vsphere

Update VSphere credentials in the cluster and restart CAPV controllers

```
dkp update bootstrap credentials vsphere [flags]
```

Options

```
      --context string      The name of the kubeconfig context to use
  -h, --help                help for vsphere
      --kubeconfig string   Path to the kubeconfig for the management cluster. If
unspecified, default discovery rules apply.
      --print-only          Print the credentials and exit the function. Without
modifying cluster
```

Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

SEE ALSO

- dkp update bootstrap credentials (see page 1519) - Update credentials in the cluster

11.2.20 dkp upgrade

Upgrade one of [addons, capi-components, catalogapp, kommander, workspace]

11.2.20.1 Options

```
  -h, --help   help for upgrade
```

11.2.20.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.20.3 SEE ALSO

- dkp upgrade addons (see page 1527) - Upgrade the core Addons in a cluster, one of [aws, azure, eks, gcp, preprovisioned, vsphere]
- dkp upgrade capi-components (see page 1524) - Upgrade the CAPI components in the cluster
- dkp upgrade catalogapp (see page 1525) - Upgrade a Catalog Application to a newer version
- dkp upgrade kommander (see page 1523) - Upgrade the Kommander version of the targeted cluster
- dkp upgrade workspace (see page 1526) - Upgrade all platform applications in the given workspace and its projects to the same version as platform applications running on the management cluster

11.2.20.4 dkp upgrade kommander

Upgrade the Kommander version of the targeted cluster

11.2.20.4.1 Synopsis

Upgrades all Kommander components and platform applications running on the targeted cluster. No attached clusters and applications running on them are affected by this action.

```
dkp upgrade kommander [flags]
```

11.2.20.4.2 Options

```
      --charts-bundle stringArray                 Path to charts-bundle to upload to
chartmuseum, apart from parsing the kommander applications repository (default [])
      --config string                             Config file to use (default "/
root/.kommander/config")
      --context string                            The name of the kubeconfig context
to use
```

```
      --core-app-timeout duration                  Timeout to wait for upgrade of
each kommander core application (default 20m0s)
      --disable-appdeployments strings             List of AppDeployments to be
disabled during upgrade (default [])
      --disallow-charts-download                   make CLI rely solely on provided
chart bundles and do not try to download charts from the Internet
      --gitea-kommander-repository-name string     gitea kommander repository name
(default "kommander")
  -h, --help                                       help for kommander
      --kommander-applications-repository string   git repository with application
definitions (default "v2.5.2")
      --kommander-charts-version string            Kommander helm charts version to
download. Default: download all available versions
      --kubeconfig string                          Path to the kubeconfig file to use
for CLI requests.
      --platform-apps-timeout duration             Timeout to wait for upgrade of the
set of platform applications (default 30m0s)
      --request-timeout string                     The length of time to wait before
giving up on a single server request. Non-zero values should contain a corresponding
time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int                                Output verbosity
```

11.2.20.4.3 SEE ALSO

- dkp upgrade (see page 1522) - Upgrade one of [addons, capi-components, catalogapp, kommander, workspace]

11.2.20.5 dkp upgrade capi-components

Upgrade the CAPI components in the cluster

```
dkp upgrade capi-components [flags]
```

11.2.20.5.1 Options

```
      --aws-service-endpoints string     Custom AWS service endpoints in a semi-colon
separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=${URL};$
{SigningRegion2}...
  -h, --help                             help for capi-components
      --http-proxy string                HTTP proxy for CAPI controllers
      --https-proxy string               HTTPS proxy for CAPI controllers
      --kubeconfig string                Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --no-proxy strings                 No Proxy list for CAPI controllers (default
 [])
```

```
      --timeout duration                The length of time to wait before giving up.
Zero means wait forever (e.g. 300s, 30m, 3h). (default 10m0s)
  -v, --verbose int                     Output verbosity
      --wait                            If true, wait for operations to complete
before returning. (default true)
      --with-aws-bootstrap-credentials  Set true to use AWS bootstrap credentials
from your environment. When false, the instance profile of the EC2 instance where the
CAPA controller is scheduled on will be used instead.
      --with-gcp-bootstrap-credentials  Set true to use GCP bootstrap credentials
from your environment. When false, the service account of the VM instance where the
CAPG controller is scheduled on will be used instead.
```

11.2.20.5.2 SEE ALSO

- dkp upgrade (see page 1522) - Upgrade one of [addons, capi-components, catalogapp, kommander, workspace]

11.2.20.6 dkp upgrade catalogapp

Upgrade a Catalog Application to a newer version

```
dkp upgrade catalogapp CATALOGAPP_NAME --to-version VERSION [--workspace WORKSPACE |
--project PROJECT] [flags]
```

11.2.20.6.1 Options

```
      --config string                   Config file to use (default "/
root/.kommander/config")
      --context string                  The name of the kubeconfig context to use
      --core-app-timeout duration       Timeout to wait for upgrade of each
kommander core application (default 20m0s)
      --disable-appdeployments strings  List of AppDeployments to be disabled during
upgrade (default [])
  -h, --help                            help for catalogapp
      --kubeconfig string               Path to the kubeconfig file to use for CLI
requests.
      --platform-apps-timeout duration  Timeout to wait for upgrade of the set of
platform applications (default 30m0s)
      --project string                  Name of the Project to upgrade the Catalog
App in
      --request-timeout string          The length of time to wait before giving up
on a single server request. Non-zero values should contain a corresponding time unit
(e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
      --to-version string               Version the Catalog App should be upgraded
to
```

```
  -v, --verbose int                    Output verbosity
  -w, --workspace string               Name of the Workspace to upgrade the Catalog
App in
```

11.2.20.6.2 SEE ALSO

- dkp upgrade (see page 1522) - Upgrade one of [addons, capi-components, catalogapp, kommander, workspace]

11.2.20.7 dkp upgrade workspace

Upgrade all platform applications in the given workspace and its projects to the same version as platform applications running on the management cluster

```
dkp upgrade workspace WORKSPACE_NAME [--dry-run] [flags]
```

11.2.20.7.1 Options

```
      --config string                  Config file to use (default "/
root/.kommander/config")
      --context string                 The name of the kubeconfig context to use
      --core-app-timeout duration      Timeout to wait for upgrade of each
kommander core application (default 20m0s)
      --disable-appdeployments strings   List of AppDeployments to be disabled during
upgrade (default [])
      --dry-run                        Do not upgrade, just list the AppDeployments
that would be upgraded
  -h, --help                           help for workspace
      --kubeconfig string              Path to the kubeconfig file to use for CLI
requests.
      --platform-apps-timeout duration   Timeout to wait for upgrade of the set of
platform applications (default 30m0s)
      --request-timeout string         The length of time to wait before giving up
on a single server request. Non-zero values should contain a corresponding time unit
(e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int                    Output verbosity
```

11.2.20.7.2 SEE ALSO

- dkp upgrade (see page 1522) - Upgrade one of [addons, capi-components, catalogapp, kommander, workspace]

11.2.20.8 dkp upgrade addons

Upgrade the core Addons in a cluster, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.20.8.1 Options

```
  -h, --help          help for addons
  -v, --verbose int   Output verbosity
```

11.2.20.8.2 SEE ALSO

- dkp upgrade (see page 1522) - Upgrade one of [addons, capi-components, catalogapp, kommander, workspace]
- dkp upgrade addons aws (see page 1527) - Upgrade the core Addons in a AWS cluster
- dkp upgrade addons azure (see page 1528) - Upgrade the core Addons in a Azure cluster
- dkp upgrade addons eks (see page 1530) - Upgrade the core Addons in a EKS cluster
- dkp upgrade addons gcp (see page 1532) - Upgrade the core Addons in a GCP cluster
- dkp upgrade addons preprovisioned (see page 1529) - Upgrade the core Addons in a Preprovisioned cluster
- dkp upgrade addons vsphere (see page 1531) - Upgrade the core Addons in a vSphere cluster

11.2.20.8.3 dkp upgrade addons aws

Upgrade the core Addons in a AWS cluster

```
dkp upgrade addons aws [flags]
```

11.2.20.8.3.1 Options

```
      --allow-missing-template-keys   If true, ignore any errors in templates when a
field or map key is missing in the template. Only applies to golang and jsonpath
output formats. (default true)
  -c, --cluster-name name             Name used to prefix the cluster and all the
created resources.
      --dry-run                       Only print the objects that would be created,
without creating them.
  -h, --help                          help for aws
      --kubeconfig string             Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
```

```
  -n, --namespace string              If present, the namespace scope for this CLI
request. (default "default")
  -o, --output string                 Output format. One of: (json, yaml, name, go-
template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string       Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --show-managed-fields           If true, keep the managedFields when printing
objects in JSON or YAML format.
      --template string               Template string or path to template file to use
when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
```

11.2.20.8.3.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.20.8.3.3 SEE ALSO

- dkp upgrade addons (see page 1527) - Upgrade the core Addons in a cluster, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.20.8.4 dkp upgrade addons azure

Upgrade the core Addons in a Azure cluster

```
dkp upgrade addons azure [flags]
```

11.2.20.8.4.1 Options

```
      --allow-missing-template-keys   If true, ignore any errors in templates when a
field or map key is missing in the template. Only applies to golang and jsonpath
output formats. (default true)
  -c, --cluster-name name             Name used to prefix the cluster and all the
created resources.
      --dry-run                       Only print the objects that would be created,
without creating them.
  -h, --help                          help for azure
      --kubeconfig string             Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
  -n, --namespace string              If present, the namespace scope for this CLI
request. (default "default")
```

```
  -o, --output string                 Output format. One of: (json, yaml, name, go-
template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string       Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --show-managed-fields           If true, keep the managedFields when printing
objects in JSON or YAML format.
      --template string               Template string or path to template file to use
when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
```

11.2.20.8.4.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.20.8.4.3 SEE ALSO

- dkp upgrade addons (see page 1527) - Upgrade the core Addons in a cluster, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.20.8.5 dkp upgrade addons preprovisioned

Upgrade the core Addons in a Preprovisioned cluster

```
dkp upgrade addons preprovisioned [flags]
```

11.2.20.8.5.1 Options

```
      --allow-missing-template-keys   If true, ignore any errors in templates when a
field or map key is missing in the template. Only applies to golang and jsonpath
output formats. (default true)
  -c, --cluster-name name             Name used to prefix the cluster and all the
created resources.
      --dry-run                       Only print the objects that would be created,
without creating them.
  -h, --help                          help for preprovisioned
      --kubeconfig string             Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
  -n, --namespace string              If present, the namespace scope for this CLI
request. (default "default")
  -o, --output string                 Output format. One of: (json, yaml, name, go-
template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
```

```
      --output-directory string       Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --show-managed-fields           If true, keep the managedFields when printing
objects in JSON or YAML format.
      --template string               Template string or path to template file to use
when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
```

11.2.20.8.5.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.20.8.5.3 SEE ALSO

- dkp upgrade addons (see page 1527) - Upgrade the core Addons in a cluster, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.20.8.6 dkp upgrade addons eks

Upgrade the core Addons in a EKS cluster

```
dkp upgrade addons eks [flags]
```

11.2.20.8.6.1 Options

```
      --allow-missing-template-keys   If true, ignore any errors in templates when a
field or map key is missing in the template. Only applies to golang and jsonpath
output formats. (default true)
  -c, --cluster-name name             Name used to prefix the cluster and all the
created resources.
      --dry-run                       Only print the objects that would be created,
without creating them.
  -h, --help                          help for eks
      --kubeconfig string             Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
  -n, --namespace string              If present, the namespace scope for this CLI
request. (default "default")
  -o, --output string                 Output format. One of: (json, yaml, name, go-
template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string       Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
```

```
      --show-managed-fields           If true, keep the managedFields when printing
objects in JSON or YAML format.
      --template string               Template string or path to template file to use
when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
```

11.2.20.8.6.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.20.8.6.3 SEE ALSO

- dkp upgrade addons (see page 1527) - Upgrade the core Addons in a cluster, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.20.8.7 dkp upgrade addons vsphere

Upgrade the core Addons in a vSphere cluster

```
dkp upgrade addons vsphere [flags]
```

11.2.20.8.7.1 Options

```
      --allow-missing-template-keys   If true, ignore any errors in templates when a
field or map key is missing in the template. Only applies to golang and jsonpath
output formats. (default true)
  -c, --cluster-name name             Name used to prefix the cluster and all the
created resources.
      --dry-run                       Only print the objects that would be created,
without creating them.
  -h, --help                          help for vsphere
      --kubeconfig string             Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
  -n, --namespace string              If present, the namespace scope for this CLI
request. (default "default")
  -o, --output string                 Output format. One of: (json, yaml, name, go-
template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string       Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --show-managed-fields           If true, keep the managedFields when printing
objects in JSON or YAML format.
```

```
      --template string                Template string or path to template file to use
when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
```

11.2.20.8.7.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.20.8.7.3 SEE ALSO

- dkp upgrade addons (see page 1527) - Upgrade the core Addons in a cluster, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.20.8.8 dkp upgrade addons gcp

Upgrade the core Addons in a GCP cluster

```
dkp upgrade addons gcp [flags]
```

11.2.20.8.8.1 Options

```
      --allow-missing-template-keys   If true, ignore any errors in templates when a
field or map key is missing in the template. Only applies to golang and jsonpath
output formats. (default true)
  -c, --cluster-name name             Name used to prefix the cluster and all the
created resources.
      --dry-run                       Only print the objects that would be created,
without creating them.
  -h, --help                          help for gcp
      --kubeconfig string             Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
  -n, --namespace string              If present, the namespace scope for this CLI
request. (default "default")
  -o, --output string                 Output format. One of: (json, yaml, name, go-
template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string       Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --show-managed-fields           If true, keep the managedFields when printing
objects in JSON or YAML format.
      --template string               Template string or path to template file to use
when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
```

11.2.20.8.8.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.20.8.8.3 SEE ALSO

- dkp upgrade addons (see page 1527) - Upgrade the core Addons in a cluster, one of [aws, azure, eks, gcp, preprovisioned, vsphere]

11.2.21 dkp edit

Edit a resource on the server

11.2.21.1 Synopsis

Edit a resource from the default editor.

The edit command allows you to directly edit any API resource you can retrieve via the command-line tools. It will open the editor defined by your KUBE_EDITOR, or EDITOR environment variables, or fall back to 'vi' for Linux or 'notepad' for Windows. You can edit multiple objects, although changes are applied one at a time. The command accepts file names as well as command-line arguments, although the files you point to must be previously saved versions of resources.

```
dkp edit (RESOURCE/NAME | -f FILENAME)
```

11.2.21.2 Options

```
      --allow-missing-template-keys   If true, ignore any errors in templates when a
field or map key is missing in the template. Only applies to golang and jsonpath
output formats. (default true)
      --config string                 Config file to use (default "/root/.kommander/
config")
      --context string                The name of the kubeconfig context to use
      --field-manager string          Name of the manager used to track field
ownership. (default "kommander-cli")
  -f, --filename strings              Filename, directory, or URL to files to use to
edit the resource (default [])
  -h, --help                          help for edit
      --kubeconfig string             Path to the kubeconfig file to use for CLI
requests.
  -k, --kustomize string              Process the kustomization directory. This flag
can't be used together with -f or -R.
```

```
  -n, --namespace string              namespace of the resource (default "default")
  -o, --output string                 Output format. One of: (json, yaml, name, go-
template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-patch                  Output the patch if the resource is edited.
  -R, --recursive                     Process the directory used in -f, --filename
recursively. Useful when you want to manage related manifests organized within the
same directory.
      --request-timeout string        The length of time to wait before giving up on
a single server request. Non-zero values should contain a corresponding time unit
(e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
      --save-config                   If true, the configuration of current object
will be saved in its annotation. Otherwise, the annotation will be unchanged. This
flag is useful when you want to perform kubectl apply on this object in the future.
      --show-managed-fields           If true, keep the managedFields when printing
objects in JSON or YAML format.
      --subresource string            If specified, edit will operate on the
subresource of the requested object. Must be one of [status]. This flag is alpha and
may change in the future.
      --template string               Template string or path to template file to use
when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
      --validate string[="strict"]    Must be one of: strict (or true), warn, ignore
(or false).
                                            "true" or "strict" will use a schema to
validate the input and fail the request if invalid. It will perform server side
validation if ServerSideFieldValidation is enabled on the api-server, but will fall
back to less reliable client-side validation if not.
                                            "warn" will warn about unknown or
duplicate fields without blocking the request if server-side field validation is
enabled on the API server, and behave as "ignore" otherwise.
                                            "false" or "ignore" will not perform any
schema validation, silently dropping any unknown or duplicate fields. (default
"strict")
  -v, --verbose int                   Output verbosity
      --windows-line-endings          Defaults to the line ending native to your
platform.
```

11.2.22 dkp version

Print version information

11.2.22.1 Synopsis

Print version information

```
dkp version [flags]
```

11.2.22.2 Options

```
  -h, --help            help for version
      --long            If true, print additional version information.
  -o, --output string   One of 'yaml' or 'json'.
```

11.2.22.3 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

11.2.23 dkp cluster

Get cluster information

11.2.23.1 Options

```
      --config string            Config file to use (default "/root/.kommander/
config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for cluster
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

11.2.23.2 SEE ALSO

- dkp cluster type (see page 1535) - Retrieve cluster type

11.2.23.3 dkp cluster type

Retrieve cluster type

```
dkp cluster type [flags]
```

11.2.23.3.1 Options

```
  -h, --help   help for type
```

11.2.23.3.2 Options inherited from parent commands

```
      --config string            Config file to use (default "/root/.kommander/
config")
      --context string           The name of the kubeconfig context to use
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

11.2.23.3.3 SEE ALSO

- dkp cluster (see page 1535) - Get cluster information

12 Release Notes

View release-specific information for DKP

Download DKP

> You must be a registered user and logged on to the support portal to download this product. New customers must contact their sales representative or sales@d2iq.com[1123] before attempting to download or install DKP.

Release Notes for specific releases:

- DKP 2.5.0 Release Notes (see page 1537)
- DKP 2.5.1 Release Notes (see page 1563)
- DKP 2.5.2 Release Notes (see page 1580)

12.1 DKP 2.5.0 Release Notes

DKP® version 2.5 was released on April 12, 2023.

Download DKP

> You must be a registered user and logged on to the support portal to download this product. New customers must contact their sales representative or sales@d2iq.com[1124] before attempting to download or install DKP.

12.1.1 Release Summary

Welcome to D2iQ Kubernetes Platform (DKP) 2.5.0! This release focuses on **MultiCluster - MultiCloud** capabilities. Also this release provides fixes to reported issues, integrates changes from previous releases, and maintains compatibility and support for other packages used in DKP. Below, you will find information about the following:

- DKP 2.5.0 Features and Enhancements (see page 1538)
- DKP 2.5.0 Supported Kubernetes Versions (see page 1543)
- DKP 2.5.0 Kubernetes major updates and deprecations (see page 1544)

1123 mailto:sales@d2iq.com
1124 mailto:sales@d2iq.com

- DKP 2.5.0 Components and Applications (see page 1544)
- DKP 2.5.0 Known Issues and Limitations (see page 1552)
- DKP 2.5.0 Customer Incidents (see page 1562)

12.1.2 Additional resources

- For more information about working with native Kubernetes, see the Kubernetes documentation[1125].
- For a full list of attributed 3rd party software, see http://d2iq.com/legal/3rd .

12.1.3 DKP 2.5.0 Features and Enhancements

The following improvements are included in this release.

12.1.3.1 Video Overview

Here are a couple videos highlighting some of the new features in DKP 2.5.

https://www.youtube.com/watch?v=HYJ4W_O_DW8https://youtu.be/GUl7zEED19k

12.1.3.2 Expand a DKP Essential Cluster to a DKP Enterprise Managed Cluster

Enterprise Gov Advanced

You can now expand your standalone DKP Essential clusters to be centrally managed under a DKP Enterprise Management cluster. This allows you to manage all your Kubernetes clusters centrally, existing and future, through a single pane of glass with the industry's best Cloud Native platform.

For more information, see Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 732).

12.1.3.3 Downloadable PDF for Air-gapped Environments

A PDF for each version of the DKP documentation set (see page 1600) is now available for download for use in secure environments that do not have access to the external network.

12.1.3.4 Support for Rocky Linux 9.1

Rocky Linux is now supported (see page 62) as an alternative, open-source operating system to CentOS for pre-provisioned, AWS, Azure, and vSphere environments! Rocky Linux is currently the leading open-source and community-supported alternative to CentOS and Red Hat Enterprise Linux, designed to be compatible with the upstream CentOS source. Both air-gapped and non-air-gapped environments are now supported as well as creating images with Konvoy Image Builder (see page 1282).

1125 https://kubernetes.io/docs/home/

12.1.3.5 DKP Insights Enhancements

This release provides CIS compliance with `kube-bench`, extended Insight alert details, and enables users to activate DKP Insights with a licensing key. It also maintains compatibility and support for other packages used in Insights.

12.1.3.5.1 Add-on Licensing in DKP UI

If you are enrolled in the Technical Preview of Insights, you can now activate Insights with an Add-on license in the DKP UI.

See (2.5) DKP Insights Activating a License Key[11261127] for more information.

> If you want to upgrade *DKP* or *DKP Insights* to a new version, you must uninstall Insights, upgrade DKP, and re-install a compatible Insights version[11281129] to continue using it.

12.1.3.5.2 CIS Compliance with Kube-bench Scanning

This version of Insights comes pre-configured with an additional 3rd-party scanning tool, kube-bench[1130] by Aqua Security[1131]. `Kube-bench` verifies that your Kubernetes clusters run securely, by examining your clusters' compliance with the CIS Kubernetes Benchmark[1132].

See (2.5) Kube-bench[11331134] for more information on how this tool has been set up in Insights.

12.1.3.5.3 Extended Root Cause Analysis for Polaris Insights

This version of DKP Insights comes with extended alerts for Polaris[11351136]-reported issues. The alert details now include information on the root cause (RCA) and suggestions for solutions.

1126 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/213322715/%282.5%29+DKP+Insights+Activating+a+License+Key
1127 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/213322715/%282.5%29+DKP+Insights+Activating+a+License+Key
1128 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/213322705/%282.5%29+DKP+and+Insights+Compatibility
1129 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/213322705/%282.5%29+DKP+and+Insights+Compatibility
1130 https://aquasecurity.github.io/kube-bench/latest/
1131 https://www.aquasec.com/products/kubernetes-security/
1132 https://www.cisecurity.org/benchmark/kubernetes
1133 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/213355056/%282.5%29+Kube-bench
1134 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/213355056/%282.5%29+Kube-bench
1135 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/212994561/2.5%2BPolaris
1136 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/212994561/2.5%2BPolaris

12.1.3.5.4 Supported Kubernetes Versions

Insights supports the same Kubernetes versions as the DKP platform[11371138]. For an overview of compatible DKP and Insights versions, see (2.5) DKP and Insights Compatibility[11391140].

12.1.3.6 Additional DKP Enhancements

12.1.3.6.1 AKS Lifecycle Management through the DKP UI

Users can now create, delete, provision and read Microsoft AKS clusters within the DKP UI. For more information, see Create a new AKS Cluster via UI (see page 1077) .

12.1.3.6.2 Konvoy Image Builder uses HCL-based Templates over JSON

Previous versions of Konvoy Image Builder (KIB) (see page 1282) used JSON templates when building images using Packer. With this release, KIB switches to using HCL templates instead. This keeps KIB in alignment with the upstream Packer project, which has started transitioning away from JSON templates. JSON will still work, but will not receive further updates.

12.1.3.6.3 Support for Ubuntu on vSphere

Ubuntu 20.04 (see page 62) is now supported for vSphere environments! Both air-gapped and non-air-gapped environments are now supported as well as creating images with Konvoy Image Builder (see page 1282).

12.1.3.6.4 New Cluster Creation Flag to Simplify Output YAML

Currently the DKP CLI allows users to print cluster and node pool resources to standard output with `dkp create cluster ... --dry-run -o yaml > cluster.yaml`. However, this dumps thousands of lines of text to the standard output. Given the size of the output, it is difficult to manage and edit. DKP has released a new flag `--output-directory` for `create cluster` and `create nodepool` to output smaller (see page 0) and more manageable manifest files.

12.1.3.6.5 Scenario-Based Installs

All variables for a provider specific installation have been combined into individual sections of documentation for install for a seamless start to using DKP. If you are using a cloud provider and FIPS, that combination has been assembled for you step-by-step! See the Day 1 - Basic Installs by Infrastructure (see page 124) section of the Documentation to find the combination you need.

1137 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/213354216/2.5+DKP+2.5.0+Supported+Kubernetes+Versions
1138 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/213354216/2.5+DKP+2.5.0+Supported+Kubernetes+Versions
1139 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/213322705/%282.5%29+DKP+and+Insights+Compatibility
1140 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/213322705/%282.5%29+DKP+and+Insights+Compatibility

You will find that each provider-specific section parallels the other sections, making it easier to find the installation scenario you need. There are, of course, additional steps and procedures where needed and depending on the specific provider. In addition, we added some tips and warnings to these procedures to help avoid some common problems associated with creating managed (see page 97) clusters.

Making the structures parallel is part of a larger “targeted headings” effort to improve Search and find-ability of information in the documentation. Stay tuned for more improvements in this area!

12.1.3.6.6 Added Procedures for Managed vSphere Clusters from the DKP CLI

Instructions for creating managed (see page 97) vSphere clusters is now available at the end of each of the installation scenarios. DKP supports network-connected, managed vSphere clusters as well as air-gapped,FIPS, and FIPS air-gapped managed clusters.

12.1.3.6.7 Improved Installation Times for Kommander

This version of DKP includes an updated application deployment order for the Kommander component, significantly decreasing installation times.

12.1.3.6.8 Improved `kube-prometheus-stack` Override Control on the Management Cluster

In DKP versions 2.4.x and earlier, `kube-prometheus-stack` was installed on the Management cluster with a default configuration using an override ConfigMap called `kube-prometheus-stack-overrides`. To add a custom configuration, you had to edit this ConfigMap, and add to the existing set of values. In DKP versions 2.5.x and later, these default overrides are stored in a new ConfigMap called `kube-prometheus-stack-mgmt-overrides`. With this improvement, the Management cluster override values are now automatically updated during DKP upgrades. You can still modify the `kube-prometheus-stack-overrides` ConfigMap for custom configurations.

Customizations applied in `kube-prometheus-stack-overrides` ConfigMap **take precedence** over the `kube-prometheus-stack-mgmt-overrides` ConfigMap.

During the upgrade from DKP 2.4.x to DKP 2.5.x, the `kube-prometheus-stack-overrides` ConfigMap on the Management cluster is automatically updated to remove the Management-cluster-specific default override values that were applied at installation. If no custom configuration was added to this ConfigMap, then the ConfigMap is removed, and the `kube-prometheus-stack` AppDeployment is updated to remove the `.spec.configOverrides` field.

A backup of `kube-prometheus-stack-overrides` is saved to a ConfigMap called `kube-prometheus-stack-overrides-backup` to ensure you can look up any previous values.

If you want to update, add, or remove service monitors from the `.prometheus.additionalServiceMonitors` section of the values, you must copy and paste the entire list to the overrides ConfigMap and edit it. This is because list values are not merged across multiple configuration files, but replaced. See Configure Alerts Using AlertManager (see page 880) for more details on how to add service monitors.

12.1.3.6.9 Temporary kubeconfig Applied with Self-Managed Flag Use

The `dkp create cluster` command was enhanced to use a temporary kubeconfig file when using the `--self-managed` flag.

12.1.3.6.10 Apple M1 Macbook Support

DKP now supports ARM64 machines! Previously, machines with an M1 CPU prevented KIND bootstraps. Now you use the DKP CLI on an Apple Macbook M1 machine in the same way that it's currently supported on Linux and Mac arm64 CPU architectures.

12.1.3.6.11 Istio Support

Istio is now a fully-supported platform application in DKP. For more information, see Deploy Istio Using DKP (see page 858).

12.1.3.6.12 External Load Balancer Support

If you want to use a non-DKP load balancer (see page 854) for external traffic, you can now Install Kommander with an External Load Balancer (see page 1242).

12.1.3.6.13 Konvoy Image Builder (KIB) Release v2.2.6[1141]

- feat: HCL instead of JSON packer templates
- feat: allow submaps in images files
- feat: Install specific cri-tools package on Debian derivatives
- feat: build Rocky Linux 9.1 images and build vSphere template for Rocky Linux 9.1
- feat: support building Ubuntu 20.04 with offline OS packages bundle and add images file for ubuntu 20.04
- feat: update Containerd to 1.6.17
- Install only the required Ansible collections to reduce size

1141 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v2.2.6

12.1.4 DKP 2.5.0 Supported Kubernetes Versions

12.1.4.1 Deploying Cluster Versions

In DKP 2.5, the Kubernetes version installed by default is **1.25.4**. The newest and oldest Kubernetes versions used with DKP must be within one minor version.

- latest supported Kubernetes version is at **1.25.4 for deploying clusters** for all providers except EKS.
- Kubernetes versions are supported at **1.24.x for deploying clusters in EKS**

Kubernetes 1.25.4, enables you to benefit from the latest features and security fixes in upstream Kubernetes. This release comes with approximately 40 enhancements that you can benefit from like pod security admission[1142], cgroups V2[1143], among others. To read more about major features in this release, visit https://kubernetes.io/blog/2022/08/23/kubernetes-v1-25-release/.

12.1.4.2 Attaching Clusters Versions

DKP 2.5.0 supports attaching clusters with the following Kubernetes versions:

Product	Compatible Kubernetes Versions
EKS	1.24.x
AKS	1.25.x
GKE	1.25.x

Attaching Kubernetes clusters with versions greater than n-1 is not recommended.

When attempting to attach a cluster with a lower Kubernetes version than the DKP default, you need to build and use an AMI, vSphere template, etc with that lower version of Kubernetes. See Konvoy Image Builder (see page 1282) for documentation on building images. Failure to do this could result in an error.

1142 https://kubernetes.io/docs/concepts/security/pod-security-admission/
1143 https://kubernetes.io/docs/concepts/architecture/cgroups/

12.1.5 DKP 2.5.0 Kubernetes major updates and deprecations

Before upgrading, we **strongly recommend** verifying your current setup against the information on this page and reading more about Kubernetes' new features in Kubernetes urgent upgrade notes[1144].

12.1.5.1 Deprecated API Services

With this version, Kubernetes stopped serving several API versions for CronJob, EndpointSlice, Event, HorizontalPodAutoscaler, PodDisruptionBudget, **PodSecurityPolicy**, and RuntimeClass services. For more information on the deprecated API versions for each of these services, refer to https://kubernetes.io/docs/reference/using-api/deprecation-guide/#v1-25.

12.1.5.2 New OCI Image Registry

DKP now uses the upstream `registry.k8s.io` registry instead of the previous `k8s.gcr.io` registry for all new and upgraded clusters.

Ensure you add `registry.k8s.io` to any firewall or proxy allowlists.

12.1.6 DKP 2.5.0 Components and Applications

The following are component and application versions for DKP.

12.1.6.1 Components

Component Name	Version
Cluster API Core (CAPI)	1.3.3-d2iq.2
Cluster API AWS Infrastructure Provider (CAPA)	1.5.5
Cluster API Google Cloud Infrastructure Provider (CAPG)	1.2.1

1144 https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.25.md#urgent-upgrade-notes

Component Name	Version
Cluster API Pre-provisioned Infrastructure Provider (CAPPP)	0.14.4
Cluster API vSphere Infrastructure Provider (CAPV)	1.5.2
Cluster API Azure Infrastructure Provider (CAPZ)	1.7.1
Konvoy Image Builder (KIB)	2.2.6[1145]
containerd	1.6.17-d2iq.1
etcd	3.5.5
Calico	3.25.1[1146]
Cluster Autoscaler	1.25.x
CSI_VERSION	Default Storage Providers in DKP (see page 103) See this page for versions and driver table.
Metal LB	0.12.1
Node Feature Version	0.12.1

1145 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v2.2.6
1146 https://github.com/projectcalico/calico/blob/v3.25.1/calico/_includes/release-notes/v3.25.1-release-notes.md

12.1.6.2 Applications

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Centralized Grafana	centralized-grafana	44.2.1	• chart: 44.2.1[1147] • prometheus-operator: 0.62.0 • grafana: 9.3.1	Link[1148]	Link[1149]
Centralized Kubecost	centralized-kubecost	0.33.1	• chart: 0.33.1[1150] • kubecost: 1.100.2	Link[1151]	Link[1152]
Chartmuseum	chartmuseum	3.9.0	• chart: 3.9.0[1153] • chartmuseum: 3.9.0	Link[1154]	Link[1155]
Dex	dex	2.11.1	• chart: 2.11.1[1156] • dex: 2.35.1-d2iq.1	Link[1157]	Link[1158]
Dex K8s Authenticator	dex-k8s-authenticator	1.2.14	• chart: 1.2.14[1159] • dex-k8s-authenticator: 1.2.4	Link[1160]	Link[1161]

1147 https://artifacthub.io/packages/helm/mesosphere/kube-prometheus-stack/44.2.1
1148 https://artifacthub.io/packages/helm/mesosphere/kube-prometheus-stack/44.2.1?modal=values
1149 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/centralized-grafana/44.2.1/defaults/cm.yaml
1150 https://artifacthub.io/packages/helm/mesosphere-stable/kubecost/0.33.1
1151 https://artifacthub.io/packages/helm/mesosphere-stable/kubecost/0.33.1?modal=values
1152 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/centralized-kubecost/0.33.1/defaults/cm.yaml
1153 https://artifacthub.io/packages/helm/chartmuseum/chartmuseum/3.9.0
1154 https://artifacthub.io/packages/helm/chartmuseum/chartmuseum/3.9.0?modal=values
1155 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/chartmuseum/3.9.0/defaults/cm.yaml
1156 https://artifacthub.io/packages/helm/mesosphere-stable/dex/2.11.1
1157 https://artifacthub.io/packages/helm/mesosphere-stable/dex/2.11.1?modal=values
1158 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/dex/2.11.1/defaults/cm.yaml
1159 https://artifacthub.io/packages/helm/mesosphere/dex-k8s-authenticator/1.2.14
1160 https://artifacthub.io/packages/helm/mesosphere/dex-k8s-authenticator/1.2.14?modal=values
1161 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/dex-k8s-authenticator/1.2.14/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
DKP Insights Management	dkp-insights-management	0.4.1	• chart: 0.4.1 • dkp-insights-management: 0.4.1	N/A	Link[1162]
External DNS	external-dns	6.13.2	• chart: 6.13.2[1163] • external-dns: 0.13.2	Link[1164]	Link[1165]
Fluent Bit	fluent-bit	0.21.6	• chart: 0.21.6[1166] • fluent-bit: 2.0.6	Link[1167]	Link[1168]
Gatekeeper	gatekeeper	3.11.0	• chart: 3.11.0[1169] • gatekeeper: 3.11.0	Link[1170]	Link[1171]
Gitea	gitea	7.0.2	• chart: 7.0.2[1172] • gitea: 1.18.3	Link[1173]	Link[1174]
Grafana Logging	grafana-logging	6.38.1	• chart: 6.38.1[1175] • grafana: 9.1.5	Link[1176]	Link[1177]

1162 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/dkp-insights-management/0.4.1/defaults/cm.yaml
1163 https://artifacthub.io/packages/helm/bitnami/external-dns/6.13.2
1164 https://artifacthub.io/packages/helm/bitnami/external-dns/6.13.2?modal=values
1165 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/external-dns/6.13.2/defaults/cm.yaml
1166 https://artifacthub.io/packages/helm/fluent/fluent-bit/0.21.6
1167 https://artifacthub.io/packages/helm/fluent/fluent-bit/0.21.6?modal=values
1168 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/fluent-bit/0.21.6/defaults/cm.yaml
1169 https://artifacthub.io/packages/helm/gatekeeper/gatekeeper/3.11.0
1170 https://artifacthub.io/packages/helm/gatekeeper/gatekeeper/3.11.0?modal=values
1171 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/gatekeeper/3.11.0/defaults/cm.yaml
1172 https://artifacthub.io/packages/helm/gitea/gitea/7.0.2
1173 https://artifacthub.io/packages/helm/gitea/gitea/7.0.2?modal=values
1174 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/gitea/7.0.2/defaults/cm.yaml
1175 https://artifacthub.io/packages/helm/grafana/grafana/6.38.1
1176 https://artifacthub.io/packages/helm/grafana/grafana/6.38.1?modal=values
1177 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/grafana-logging/6.38.1/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Grafana Loki	grafana-loki	0.69.4	• chart: 0.69.4[1178] • loki: 2.7.3	Link[1179]	Link[1180]
Istio	istio	1.16.2	• chart: 1.16.2[1181] • istio: 1.16.2	Link[1182]	Link[1183]
Jaeger	jaeger	2.40.0	• chart: 2.40.0[1184] • jaeger: 1.39.0	Link[1185]	Link[1186]
Karma	karma	2.0.1	• chart: 2.0.1[1187] • karma: 0.70	Link[1188]	Link[1189]
Kiali	kiali	1.63.2	• chart: 1.63.2[1190] • kiali: 1.63.2	Link[1191]	Link[1192]
Knative	knative	0.5.2	• chart: 0.5.2[1193] • knative: 0.22.3	Link[1194]	Link[1195]

1178 https://artifacthub.io/packages/helm/grafana/loki-distributed/0.69.4
1179 https://artifacthub.io/packages/helm/grafana/loki-distributed/0.69.4?modal=values
1180 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/grafana-loki/0.69.4/defaults/cm.yaml
1181 https://artifacthub.io/packages/helm/mesosphere/istio/1.16.2
1182 https://artifacthub.io/packages/helm/mesosphere/istio/1.16.2?modal=values
1183 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/istio/1.16.2/defaults/cm.yaml
1184 https://artifacthub.io/packages/helm/jaegertracing/jaeger-operator/2.40.0
1185 https://artifacthub.io/packages/helm/jaegertracing/jaeger-operator/2.40.0?modal=values
1186 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/jaeger/2.40.0/defaults/cm.yaml
1187 https://artifacthub.io/packages/helm/mesosphere-stable/karma/2.0.1
1188 https://artifacthub.io/packages/helm/mesosphere-stable/karma/2.0.1?modal=values
1189 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/karma/2.0.1/defaults/cm.yaml
1190 https://artifacthub.io/packages/helm/kiali/kiali-operator/1.63.2
1191 https://artifacthub.io/packages/helm/kiali/kiali-operator/1.63.2?modal=values
1192 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/kiali/1.63.2/defaults/cm.yaml
1193 https://artifacthub.io/packages/helm/mesosphere/knative/0.5.2
1194 https://artifacthub.io/packages/helm/mesosphere/knative/0.5.2?modal=values
1195 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/knative/0.5.2/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Flux	kommander-flux	0.40.2	• chart: N/A • flux: 0.40.2	N/A	N/A
Kube OIDC Proxy	kube-oidc-proxy	0.3.2	• chart: 0.3.1[1196] • kube-oidc-proxy: 0.3.0	Link[1197]	Link[1198]
Kube Prometheus Stack	kube-prometheus-stack	44.2.1	• chart: 44.2.1[1199] • prometheus-operator: 0.62.0 • grafana: 9.3.1 • prometheus: 2.41.0 • prometheus-alertmanager: 0.25.0	Link[1200]	Link[1201]
Kubecost	kubecost	0.33.1	• chart: 0.33.1[1202] • kubecost: 1.100.2	Link[1203]	Link[1204]
Kubefed	kubefed	0.10.0	• chart: 0.10.0[1205] • kubefed: 0.10.0	Link[1206]	Link[1207]

1196 https://artifacthub.io/packages/helm/mesosphere/kube-oidc-proxy/0.3.1
1197 https://artifacthub.io/packages/helm/mesosphere/kube-oidc-proxy/0.3.1?modal=values
1198 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/kube-oidc-proxy/0.3.2/defaults/cm.yaml
1199 https://artifacthub.io/packages/helm/mesosphere/kube-prometheus-stack/44.2.1
1200 https://artifacthub.io/packages/helm/mesosphere/kube-prometheus-stack/44.2.1?modal=values
1201 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/kube-prometheus-stack/44.2.1/defaults/cm.yaml
1202 https://artifacthub.io/packages/helm/mesosphere-stable/kubecost/0.33.1
1203 https://artifacthub.io/packages/helm/mesosphere-stable/kubecost/0.33.1?modal=values
1204 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/kubecost/0.33.1/defaults/cm.yaml
1205 https://artifacthub.io/packages/helm/kubefed/kubefed/0.10.0
1206 https://artifacthub.io/packages/helm/kubefed/kubefed/0.10.0?modal=values
1207 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/kubefed/0.10.0/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Kubernetes Dashboard	kubernetes-dashboard	6.0.0	• chart: 6.0.0[1208] • kubernetes-dashboard: 2.7.0	Link[1209]	Link[1210]
Kubetunnel	kubetunnel	0.0.15	• chart: 0.0.15 • kubetunnel: 0.0.15	N/A	Link[1211]
Logging Operator	logging-operator	3.17.10	• chart: 3.17.10[1212] • logging-operator: 3.17.10 • logging-operator-logging: 3.17.10	Link[1213]	Link[1214]
NFS Server Provisioner	nfs-server-provisioner	0.6.0	• chart: 0.6.0[1215] • nfs-server-provisioner: 2.3.0	Link[1216]	Link[1217]
NVIDIA GPU Operator	nvidia-gpu-operator	22.9.1	• chart: 22.9.1[1218] • nvidia-gpu-operator: 22.9.1	Link[1219]	Link[1220]

1208 https://artifacthub.io/packages/helm/k8s-dashboard/kubernetes-dashboard/6.0.0
1209 https://artifacthub.io/packages/helm/k8s-dashboard/kubernetes-dashboard/6.0.0?modal=values
1210 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/kubernetes-dashboard/6.0.0/defaults/cm.yaml
1211 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/kubetunnel/0.0.15/defaults/cm.yaml
1212 https://artifacthub.io/packages/helm/banzaicloud-stable/logging-operator/3.17.10
1213 https://artifacthub.io/packages/helm/banzaicloud-stable/logging-operator/3.17.10?modal=values
1214 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/logging-operator/3.17.10/defaults/cm.yaml
1215 https://artifacthub.io/packages/helm/mesosphere/nfs-server-provisioner/0.6.0
1216 https://artifacthub.io/packages/helm/mesosphere/nfs-server-provisioner/0.6.0?modal=values
1217 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/nfs-server-provisioner/0.6.0/defaults/cm.yaml
1218 https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/getting-started.html#chart-customization-options
1219 https://raw.githubusercontent.com/NVIDIA/gpu-operator/v22.9.1/deployments/gpu-operator/values.yaml
1220 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/nvidia-gpu-operator/22.9.1/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Grafana (project)	project-grafana-logging	6.38.1	• chart: 6.38.1[1221] • grafana: 9.1.5	Link[1222]	Link[1223]
Grafana Loki (project)	project-grafana-loki	0.69.4	• chart: 0.69.4[1224] • loki: 2.7.3	Link[1225]	Link[1226]
Prometheus Adapter	prometheus-adapter	4.0.2	• chart: 4.0.2[1227] • prometheus-adapter: 0.10.0	Link[1228]	Link[1229]
Reloader	reloader	1.0.5	• chart: 1.0.5[1230] • reloader: 1.0.5	Link[1231]	Link[1232]
Rook Ceph	rook-ceph	1.10.11	• chart: 1.10.11[1233] • rook-ceph: 1.10.11	Link[1234]	Link[1235]

1221 https://artifacthub.io/packages/helm/grafana/grafana/6.38.1
1222 https://artifacthub.io/packages/helm/grafana/grafana/6.38.1?modal=values
1223 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/project-grafana-logging/6.38.1/defaults/cm.yaml
1224 https://artifacthub.io/packages/helm/grafana/loki-distributed/0.69.4
1225 https://artifacthub.io/packages/helm/grafana/loki-distributed/0.69.4?modal=values
1226 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/project-grafana-loki/0.69.4/defaults/cm.yaml
1227 https://artifacthub.io/packages/helm/prometheus-community/prometheus-adapter/4.0.2
1228 https://artifacthub.io/packages/helm/prometheus-community/prometheus-adapter/4.0.2?modal=values
1229 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/prometheus-adapter/4.0.2/defaults/cm.yaml
1230 https://github.com/stakater/Reloader/tree/v1.0.5/README.md#helm-charts
1231 https://artifacthub.io/packages/helm/stakater/reloader/1.0.5?modal=values
1232 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/reloader/1.0.5/defaults/cm.yaml
1233 https://github.com/rook/rook/tree/v1.10.11/Documentation/Helm-Charts/operator-chart.md
1234 https://raw.githubusercontent.com/rook/rook/v1.10.11/deploy/charts/rook-ceph/values.yaml
1235 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/rook-ceph/1.10.11/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Rook Ceph Cluster	rook-ceph-cluster	1.10.11	• chart: 1.10.11[1236] • rook-ceph: 1.10.11 • rook-ceph-cluster: 17.2.5	Link[1237]	Link[1238]
Thanos	thanos	0.4.8	• chart: 0.4.6[1239] • thanos: 0.17.1	Link[1240]	Link[1241]
Traefik	traefik	20.8.0	• chart: 20.8.0[1242] • traefik: 2.9.6	Link[1243]	Link[1244]
Traefik ForwardAuth	traefik-forward-auth	0.3.8	• chart: 0.3.8[1245] • traefik-forward-auth: 3.1.0	Link[1246]	Link[1247]
Velero	velero	3.4.0	• chart: 3.1.2[1248] • velero: 1.10.0	Link[1249]	Link[1250]

12.1.7 DKP 2.5.0 Known Issues and Limitations

The following items are known issues with this release.

1236 https://github.com/rook/rook/tree/v1.10.11/Documentation/Helm-Charts/ceph-cluster-chart.md
1237 https://raw.githubusercontent.com/rook/rook/v1.10.11/deploy/charts/rook-ceph-cluster/values.yaml
1238 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/rook-ceph-cluster/1.10.11/defaults/cm.yaml
1239 https://artifacthub.io/packages/helm/banzaicloud-stable/thanos/0.4.6
1240 https://artifacthub.io/packages/helm/banzaicloud-stable/thanos/0.4.6?modal=values
1241 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/thanos/0.4.8/defaults/cm.yaml
1242 https://artifacthub.io/packages/helm/traefik/traefik/20.8.0
1243 https://artifacthub.io/packages/helm/traefik/traefik/20.8.0?modal=values
1244 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/traefik/20.8.0/defaults/cm.yaml
1245 https://artifacthub.io/packages/helm/mesosphere/traefik-forward-auth/0.3.8
1246 https://artifacthub.io/packages/helm/mesosphere/traefik-forward-auth/0.3.8?modal=values
1247 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/traefik-forward-auth/0.3.8/defaults/cm.yaml
1248 https://artifacthub.io/packages/helm/vmware-tanzu/velero/3.1.2
1249 https://artifacthub.io/packages/helm/vmware-tanzu/velero/3.1.2?modal=values
1250 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.0/services/velero/3.4.0/defaults/cm.yaml

12.1.7.1 Use Static Credentials to Provision an Azure Cluster

Only static credentials can be used when provisioning an Azure cluster (see page 1041).

12.1.7.2 Containerd 1.4.13 File Limit Issue

In this version of DKP, we introduced containerd 1.6.17. The systemd unit for containerd 1.6.17 provided upstream removes all file number limits (`LimitNOFILE=infinity`). In our testing, we found that removing these limits broke some IO sensitive applications like Rook Ceph and HAProxy. Because of this, the KIB version included in this release sets the `LimitNOFILE` value in the containerd systemd unit to the value (`1048576`) that was used in previous containerd 1.4.13 version releases.

12.1.7.3 Intermittent Error Status when Creating EKS Clusters in the UI

When provisioning an EKS cluster through the UI, you may receive a brief error state because the EKS cluster may sporadically lose connectivity with the management cluster which results in the following symptoms:

- The UI shows the cluster is in an error state.
- The kubeconfig generated and retrieved from Kommander ceases to work.
- Applications created on the management cluster may not be immediately federated to managed EKS clusters.

After a few moments, the error will resolve, without any action on your part. A new kubeconfig generated and retrieved from Kommander then works properly, and the UI shows that it is working again. In the meantime, you can continue to use the UI to work on the cluster such as deploy applications, create projects, and add roles.

12.1.7.4 Installation Issue in Pre-provisioned Environments

An issue with Rook Ceph's deployment prevents pre-provisioned environments from installing this DKP version. To solve this issue, you must set up a minimum of 40 GB of raw storage for your worker nodes and customize your Rook Ceph installation as indicated in Install Kommander in a Pre-provisioned Environment (see page 1271).

12.1.7.5 Resolve issues with failed HelmReleases

An issue with the Flux helm-controller[1251] can cause HelmReleases to fail with the error message *Helm upgrade failed: another operation (install/upgrade/rollback) is in progress*. This can happen when the helm-controller is restarted while a HelmRelease is still upgrading, or installing.

1251 https://github.com/fluxcd/helm-controller/issues/149

12.1.7.5.1 Workaround

To ensure the HelmRelease error was caused by the helm-controller restarting, first try to suspend/resume the HelmRelease:

```
kubectl -n <namespace> patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n <namespace> patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

This might resolve the issue. If not, continue with the following steps:

You should see the HelmRelease attempting to reconcile, and then it either succeeds (with status: *Release reconciliation succeeded*) or it fails with the same error as before.

If the HelmRelease is still in the failed state, it is likely related to the helm-controller restarting. For example, if the 'reloader' HelmRelease is the one that is stuck.

To resolve the issue, follow these steps:

1. List secrets containing the affected HelmRelease name:

   ```
   kubectl get secrets -n ${NAMESPACE} | grep reloader
   ```

   The output should look like this:

   ```
   kommander-reloader-reloader-token-9qd8b                          kubernetes.io/
   service-account-token   3      171m
   sh.helm.release.v1.kommander-reloader.v1                         helm.sh/
   release.v1                     1      171m
   sh.helm.release.v1.kommander-reloader.v2                         helm.sh/
   release.v1                     1      117m
   ```

   In this example, `sh.helm.release.v1.kommander-reloader.v2` is the most recent revision.

2. Find and delete the most recent revision secret, for example, `sh.helm.release.v1.*.<revision>`:

   ```
   kubectl delete secret -n <namespace> <most recent helm revision secret name>
   ```

3. Suspend and resume the HelmRelease to trigger a reconciliation:

   ```
   kubectl -n <namespace> patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[
   {"op": "replace", "path": "/spec/suspend", "value": true}]'
   kubectl -n <namespace> patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[
   {"op": "replace", "path": "/spec/suspend", "value": false}]'
   ```

You should see the HelmRelease is reconciled and eventually the upgrade and install succeeds.

12.1.7.6 Limitations to Disk Resizing in vSphere

The DKP CLI flags `--control-plane-disk-size` and `--worker-disk-size` are unable to resize the root file system of VMs created using OS images. The flags work by resizing the primary disk of the VM. When the VM boots, the root file system is expanded to fill the disk, but that expansion does not work for some file systems, for example, for file systems contained in an LVM Logical Volume. To ensure your root file system has the size you expect, please see https://d2iq.atlassian.net/wiki/spaces/DENT/pages/212993627/2.5%2BCreate%2Ba%2BvSphere%2BBase%2BOS%2BImage#Disk-Size (see page 0).

12.1.7.7 Error Status in Grafana Logging Dashboard with EKS Clusters

Currently, it is not possible to use FluentBit to collect Admin-level logs (see page 798) on a managed EKS cluster.

If you have these logs enabled, the following message appears when you access the Kubernetes Audit Dashboard in the Grafana Logging Dashboard:

```
Cannot read properties of undefined (reading '0')
```

12.1.7.8 Logging Operator Upgrade Error

There is a race condition that could result in the `logging-operator-logging-fluentd` using the incorrect image tag during upgrade from DKP 2.4.0 to DKP 2.5.0.

The image tag is eventually corrected by the `logging-operator`, however due to the nature of StatefulSets, the failing pod needs to be removed in order for the StatefulSet to continue rolling out the required updates.

1. Run this command to find if any Fluentd pods are in the `ImagePullBackOff` state post-upgrade:

   ```
   kubectl get pod -l app.kubernetes.io/name=fluentd,app.kubernetes.io/managed-
   by=logging-operator-logging,app.kubernetes.io/component=fluentd -n kommander
   ```

2. If `ImagePullBackOff` is present in the output like in the example below, you will need to continue with these steps to resolve the issue.

   ```
   NAME                                READY   STATUS             RESTARTS   AGE
   logging-operator-logging-fluentd-0  3/3     Running            0
   6m41s
   logging-operator-logging-fluentd-1  2/3     ImagePullBackOff   0
   2m33s
   ```

3. Delete the Fluentd pod that is in an `ImagePullBackOff` state. In this case, it is `logging-operator-logging-fluentd-1`:

```
kubectl delete pod -n kommander logging-operator-logging-fluentd-1
```

4. The upgrade of the `logging-operator-logging-fluentd` StatefulSet now proceeds as normal.

12.1.7.9 Nodepools Update Error with Knative

The following only applies to your environment if you have Knative installed and if its deployment is scaled to less than 5 pods.
An issue with the `PodDisruptionBudget` resource blocks the deletion of old nodes when upgrading from DKP 2.4.0 to DKP 2.5.0, which results in a failure of the DKP nodepools upgrade.

1. If the `dkp update nodepool` command fails, check to see if `PodDisruptionBudget` with `ALLOWED DISRUPTIONS` equals 0 using the following command:

```
kubectl get pdb -n knative-serving
NAME                            MIN AVAILABLE   MAX UNAVAILABLE   ALLOWED
DISRUPTIONS   AGE
activator-pdb                   80%             N/A               0
22h
webhook-pdb                     80%             N/A               0
22h
```

2. Obtain the list of pods in the knative-serving namespace containing PDB with the following command:

```
kubectl get pods -n knative-serving -l 'app in (webhook, activator)'
```

3. The output should look similar to the following:

```
NAME                          READY   STATUS    RESTARTS   AGE
webhook-XXXXXXXXX-XXXX        2/2     Running   0          5d21h
activator-XXXXXXXXX-XXXX      2/2     Running   0          4d23h
```

4. Delete the pods that contains the `PodDisruptionBudget` resource:

```
kubectl delete pod -n knative-serving activator-XXXX-XXX webhook-XXXXX-XXXX
```

5. The upgrade of DKP and Knative now proceeds as normal. Re-run the `dkp update nodepool` command.

12.1.7.10 Rook Ceph Install Error

An issue may emerge when installing `rook-ceph` on vSphere clusters using RHEL operating systems.

This issue occurs during initial installation of rook-ceph, causing the object store used by Velero and Grafana Loki, to be unavailable. If the installation of Kommander component of DKP is unsuccessful due to `rook-ceph` failing, you might need to apply the following workaround.

1. Run the following command to see if the cluster is affected by this issue.

```
kubectl describe CephObjectStores dkp-object-store -n kommander
```

2. If the following output appears, this workaround needs to be applied:

```
Name:         dkp-object-store
Namespace:    kommander
...
  Warning  ReconcileFailed     7m55s (x19 over 52m)
  rook-ceph-object-controller  failed to reconcile CephObjectStore
  "kommander/dkp-object-store". failed to create object store deployments:
failed
  to configure multisite for object store: failed create ceph multisite for
  object-store ["dkp-object-store"]: failed to commit config changes after
  creating multisite config for CephObjectStore "kommander/dkp-object-store":
  failed to commit RGW configuration period changes%!(EXTRA []string=[]):
signal: interrupt
```

3. Kubectl exec into the `rook-ceph-tools` pod.

```
export WORKSPACE_NAMESPACE=<workspace namespace>
CEPH_TOOLS_POD=$(kubectl get pods -l app=rook-ceph-tools -n $
{WORKSPACE_NAMESPACE} -o name)
kubectl exec -it -n ${WORKSPACE_NAMESPACE} $CEPH_TOOLS_POD bash
```

4. Run the following commands to set `dkp-object-store` as the default zonegroup.
   NOTE: The `period update` command may take a few minutes to complete

```
radosgw-admin zonegroup default --rgw-zonegroup=dkp-object-store
radosgw-admin period update --commit
```

5. Next, restart the `rook-ceph-operator` deployment for the `CephobjectStore` to be reconciled.

```
kubectl rollout restart deploy -n${WORKSPACE_NAMESPACE} rook-ceph-operator
```

6. After running the commands above, the `CephObjectStore` should be `Connected` once the `rook-ceph` operator reconciles the object (this may take some time).

```
kubectl wait CephObjectStore --for=jsonpath='{.status.phase}'=Connected dkp-
object-store -n ${WORKSPACE_NAMESPACE} --timeout 10m
```

12.1.7.11 Post Upgrade, Volume Cannot Attach to a Node Already Attached to Another Node

Due to an upstream issue[1252], when you bring a new node up during a Kubernetes version upgrade and then delete the old node, an existing volume might not attach to the new node.

You will see this when a new pod that uses a volume does not become ready in the new node, and then an event that says something such as `Volume <pvc/pv-id> is already exclusively attached to one node and can't be attached to another`.

This will be fixed[1253] in a future Kubernetes release[1254], for example this is described in vSphere here[1255]. Different methods might be needed to resolve this manually[1256], including this method to resolve on vSphere[1257].

12.1.7.11.1 DKP 2.4.0 to DKP 2.50 `rook-ceph-cluster` Helm Release Upgrade Error

1. If you see the `rook-ceph-cluster` HelmRelease with an error similar to this:

```
status:
  conditions:
  - lastTransitionTime: "2023-05-10T13:56:28Z"
    message: "Helm rollback failed: cannot patch \\"dkp-ceph-cluster\\" with
kind CephCluster:
      Internal error occurred: failed calling webhook \\"cephcluster-wh-rook-
ceph-admission-controller-kommander.rook.io\\":
```

1252 https://discuss.kubernetes.io/t/multi-attach-error-for-volume-pvc-volume-is-already-exclusively-attached-to-one-node-and-cant-be-attached-to-another/18951
1253 https://github.com/kubernetes/enhancements/pull/1116
1254 https://kubernetes.io/docs/reference/command-line-tools-reference/feature-gates/?highlight=Feature#:~:text=1.27-,NodeOutOfServiceVolumeDetach,-false
1255 https://github.com/kubernetes-sigs/vsphere-csi-driver/issues/125
1256 https://discuss.kubernetes.io/t/multi-attach-error-for-volume-pvc-volume-is-already-exclusively-attached-to-one-node-and-cant-be-attached-to-another/18951/3
1257 https://kb.vmware.com/s/article/70519

```
      failed to call webhook: Post \\"<https://rook-ceph-admission-
controller.kommander.svc:443/validate-ceph-rook-io-v1-cephcluster?timeout=5s\
\">:>
      dial tcp 10.99.16.90:443: connect: connection refused && cannot patch \\"
dkp-object-store\\"
      with kind CephObjectStore: Internal error occurred: failed calling
webhook \\"cephobjectstore-wh-rook-ceph-admission-controller-kommander.rook.io\
\":
      failed to call webhook: Post \\"<https://rook-ceph-admission-
controller.kommander.svc:443/validate-ceph-rook-io-v1-cephobjectstore?
timeout=5s\\">:>
      dial tcp 10.99.16.90:443: connect: connection refused\\n\\nLast Helm
logs:\\n\\nPatch
      CephCluster \\"dkp-ceph-cluster\\" in namespace kommander\\nerror
updating the
      resource \\"dkp-ceph-cluster\\":\\n\\t cannot patch \\"dkp-ceph-cluster\
\" with kind
      CephCluster: Internal error occurred: failed calling webhook \
\"cephcluster-wh-rook-ceph-admission-controller-kommander.rook.io\\":
      failed to call webhook: Post \\"<https://rook-ceph-admission-
controller.kommander.svc:443/validate-ceph-rook-io-v1-cephcluster?timeout=5s\
\">:>
      dial tcp 10.99.16.90:443: connect: connection refused\\nPatch
CephObjectStore
      \\"dkp-object-store\\" in namespace kommander\\nerror updating the
resource \\"dkp-object-store\\":\\n\\t
      cannot patch \\"dkp-object-store\\" with kind CephObjectStore: Internal
error
      occurred: failed calling webhook \\"cephobjectstore-wh-rook-ceph-
admission-controller-kommander.rook.io\\":
      failed to call webhook: Post \\"<https://rook-ceph-admission-
controller.kommander.svc:443/validate-ceph-rook-io-v1-cephobjectstore?
timeout=5s\\">:>
      dial tcp 10.99.16.90:443: connect: connection refused\\nwarning: Rollback
\\"rook-ceph-cluster\\"
      failed: cannot patch \\"dkp-ceph-cluster\\" with kind CephCluster:
Internal error
      occurred: failed calling webhook \\"cephcluster-wh-rook-ceph-admission-
controller-kommander.rook.io\\":
      failed to call webhook: Post \\"<https://rook-ceph-admission-
controller.kommander.svc:443/validate-ceph-rook-io-v1-cephcluster?timeout=5s\
\">:>
      dial tcp 10.99.16.90:443: connect: connection refused && cannot patch \\"
dkp-object-store\\"
      with kind CephObjectStore: Internal error occurred: failed calling
webhook \\"cephobjectstore-wh-rook-ceph-admission-controller-kommander.rook.io\
\":
      failed to call webhook: Post \\"<https://rook-ceph-admission-
controller.kommander.svc:443/validate-ceph-rook-io-v1-cephobjectstore?
timeout=5s\\">:>
      dial tcp 10.99.16.90:443: connect: connection refused"
    reason: RollbackFailed
```

```
    status: "False"
    type: Ready
  - lastTransitionTime: "2023-05-10T13:56:26Z"
    message: "Helm upgrade failed: cannot patch \\"dkp-object-store\\" with
kind CephObjectStore:
      Internal error occurred: failed calling webhook \\"cephobjectstore-wh-
rook-ceph-admission-controller-kommander.rook.io\\":
      failed to call webhook: Post \\"<https://rook-ceph-admission-
controller.kommander.svc:443/validate-ceph-rook-io-v1-cephobjectstore?
timeout=5s\\":>
      dial tcp 10.99.16.90:443: connect: connection refused\\n\\nLast Helm
logs:\\n\\nPatch
      Ingress \\"dkp-ceph-cluster-dashboard\\" in namespace kommander\\nPatch
CephCluster
      \\"dkp-ceph-cluster\\" in namespace kommander\\nPatch CephObjectStore \\"
dkp-object-store\\"
      in namespace kommander\\nerror updating the resource \\"dkp-object-store\
\":\\n\\t
      cannot patch \\"dkp-object-store\\" with kind CephObjectStore: Internal
error
      occurred: failed calling webhook \\"cephobjectstore-wh-rook-ceph-
admission-controller-kommander.rook.io\\":
      failed to call webhook: Post \\"<https://rook-ceph-admission-
controller.kommander.svc:443/validate-ceph-rook-io-v1-cephobjectstore?
timeout=5s\\":>
      dial tcp 10.99.16.90:443: connect: connection refused\\nwarning: Upgrade
\\"rook-ceph-cluster\\"
      failed: cannot patch \\"dkp-object-store\\" with kind CephObjectStore:
Internal
      error occurred: failed calling webhook \\"cephobjectstore-wh-rook-ceph-
admission-controller-kommander.rook.io\\":
      failed to call webhook: Post \\"<https://rook-ceph-admission-
controller.kommander.svc:443/validate-ceph-rook-io-v1-cephobjectstore?
timeout=5s\\":>
      dial tcp 10.99.16.90:443: connect: connection refused"
    reason: UpgradeFailed
    status: "False"
    type: Released
  - lastTransitionTime: "2023-05-10T13:56:28Z"
    message: "Helm rollback failed: cannot patch \\"dkp-ceph-cluster\\" with
kind CephCluster:
      Internal error occurred: failed calling webhook \\"cephcluster-wh-rook-
ceph-admission-controller-kommander.rook.io\\":
      failed to call webhook: Post \\"<https://rook-ceph-admission-
controller.kommander.svc:443/validate-ceph-rook-io-v1-cephcluster?timeout=5s\
\":>
      dial tcp 10.99.16.90:443: connect: connection refused && cannot patch \\"
dkp-object-store\\"
      with kind CephObjectStore: Internal error occurred: failed calling
webhook \\"cephobjectstore-wh-rook-ceph-admission-controller-kommander.rook.io\
\":
```

```
      failed to call webhook: Post \\"<https://rook-ceph-admission-
controller.kommander.svc:443/validate-ceph-rook-io-v1-cephobjectstore?
timeout=5s\\">:>
      dial tcp 10.99.16.90:443: connect: connection refused\\n\\nLast Helm
logs:\\n\\nPatch
      CephCluster \\"dkp-ceph-cluster\\" in namespace kommander\\nerror
updating the
      resource \\"dkp-ceph-cluster\\":\\n\\t cannot patch \\"dkp-ceph-cluster\
\" with kind
      CephCluster: Internal error occurred: failed calling webhook \
\"cephcluster-wh-rook-ceph-admission-controller-kommander.rook.io\\":
      failed to call webhook: Post \\"<https://rook-ceph-admission-
controller.kommander.svc:443/validate-ceph-rook-io-v1-cephcluster?timeout=5s\
\">:>
      dial tcp 10.99.16.90:443: connect: connection refused\\nPatch
CephObjectStore
      \\"dkp-object-store\\" in namespace kommander\\nerror updating the
resource \\"dkp-object-store\\":\\n\\t
      cannot patch \\"dkp-object-store\\" with kind CephObjectStore: Internal
error
      occurred: failed calling webhook \\"cephobjectstore-wh-rook-ceph-
admission-controller-kommander.rook.io\\":
      failed to call webhook: Post \\"<https://rook-ceph-admission-
controller.kommander.svc:443/validate-ceph-rook-io-v1-cephobjectstore?
timeout=5s\\">:>
      dial tcp 10.99.16.90:443: connect: connection refused\\nwarning: Rollback
\\"rook-ceph-cluster\\"
      failed: cannot patch \\"dkp-ceph-cluster\\" with kind CephCluster:
Internal error
      occurred: failed calling webhook \\"cephcluster-wh-rook-ceph-admission-
controller-kommander.rook.io\\":
      failed to call webhook: Post \\"<https://rook-ceph-admission-
controller.kommander.svc:443/validate-ceph-rook-io-v1-cephcluster?timeout=5s\
\">:>
      dial tcp 10.99.16.90:443: connect: connection refused && cannot patch \\"
dkp-object-store\\"
      with kind CephObjectStore: Internal error occurred: failed calling
webhook \\"cephobjectstore-wh-rook-ceph-admission-controller-kommander.rook.io\
\":
      failed to call webhook: Post \\"<https://rook-ceph-admission-
controller.kommander.svc:443/validate-ceph-rook-io-v1-cephobjectstore?
timeout=5s\\">:>
      dial tcp 10.99.16.90:443: connect: connection refused"
    reason: RollbackFailed
    status: "False"
    type: Remediated
```

2. Reconcile the `rook-ceph-cluster` HelmRelease once the `rook-ceph` HelmRelease becomes ready using the commands below. The upgrade now proceeds as normal.

```
export WORKSPACE_NAMESPACE=<workspace namespace>
kubectl -n ${WORKSPACE_NAMESPACE} patch helmrelease rook-ceph-cluster --
type='json' -p='[{"op": "replace", "path": "/spec/suspend", "value": true}]'
kubectl -n ${WORKSPACE_NAMESPACE} patch helmrelease rook-ceph-cluster --
type='json' -p='[{"op": "replace", "path": "/spec/suspend", "value": false}]'
```

12.1.8 DKP 2.5.0 Customer Incidents

The following resolved incidents are included in this release:

12.1.8.1 Misconfiguration of OIDC Identity Provider Crashes Dex

D2IQ-95131

In previous versions of DKP, if you misconfigured an Identify Provider, you could lose all access to the DKP UI and have to fix the problem using the command line. In this release, the `dex` component has been modified so that it tolerates this situation better.

12.1.8.2 DKP Missing Required Images for kubetunnel in Air-gapped Environments

D2IQ-95830

When attempting to attach an air-gapped cluster with the “network restricted” configuration, the managed cluster remains stuck in a “Pending” status do to a missing image.

The missing image has been added to the air-gapped bundle, resolving the issue.

12.1.8.3 Velero 'backup logs' Command Doesn't Work

D2IQ-95509

The `velero backup logs <backup-name>` command did not work in DKP 2.4 as a result of the change from Minio to Rook-Ceph storage backends. This problem has been corrected.

12.1.8.4 Microsoft Azure Upgrade Breaks Volume Attachments for Pods

D2IQ-95191

As a result of a bug in the upstream Azure CSI provider, volumes could remain attached to Virtual Machines that were deleted, preventing them from being attached to replacement machines. This prevented a successful upgrade to a new DKP version, as the upgrade process replaces machine instances. The problem has been fixed upstream, and the new Azure CSI provider is included in this release.

12.1.8.5 License Counts the Control Plane Cores after the 2.4 Upgrade

D2IQ-95188

Due to an upstream change, the licensing code was not correctly excluding Control plane nodes when calculating the number of cores in use. This problem has been corrected.

12.1.8.6 Calico Not Updated during DKP Upgrade on Flatcar

D2IQ-94862

When upgrading a cluster using Flatcar OS, the Calico resources were not properly upgraded due to a missing `osHint` label on some cluster resources. The appropriate `osHint` labels are now applied on cluster creation and upgrade.

12.2 DKP 2.5.1 Release Notes

DKP® version 2.5.1 was released on June 2, 2023

Download DKP

> You must be a registered user and logged on to the support portal to download this product. New customers must contact their sales representative or sales@d2iq.com[1258] before attempting to download or install DKP.

12.2.1 Release Summary

Welcome to D2iQ Kubernetes Platform (DKP) 2.5.1! This release provides fixes to reported issues, integrates changes from previous releases, and maintains compatibility and support for other packages used in DKP.

- DKP 2.5.1 Features and Enhancements (see page 1564)
- DKP 2.5.1 Supported Kubernetes Versions (see page 1564)
- DKP 2.5.1 Kubernetes major updates and deprecations (see page 1565)
- DKP 2.5.1 Components and Applications (see page 1565)
- DKP 2.5.1 Known Issues and Limitations (see page 1573)
- DKP 2.5.1 Customer Incidents (see page 1580)

12.2.2 Additional resources

- For more information about working with native Kubernetes, see the Kubernetes documentation[1259].
- For a full list of attributed 3rd party software, see http://d2iq.com/legal/3rd .

1258 mailto:sales@d2iq.com
1259 https://kubernetes.io/docs/home/

12.2.3 DKP 2.5.1 Features and Enhancements

The following improvements are included in this release.

12.2.3.1 Podman vs. Docker

Podman is now supported by DKP! If you choose to use Podman, it works with Linux on DKP within the following parameters:

- Version 4.0 of Podman[1260] or higher for Linux.
- Host requirements found here: https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
- Set[1261] `alias docker=podman` - In DKP documentation we use `Docker` command line syntax in some procedures. If you are using Podman, we suggest you set `alias docker=podman`.

12.2.4 DKP 2.5.1 Supported Kubernetes Versions

12.2.4.1 Deploying Cluster Versions

In DKP 2.5, the Kubernetes version installed by default is **1.25.4**. The newest and oldest Kubernetes versions used with DKP must be within one minor version.

- latest supported Kubernetes version is at **1.25.4 for deploying clusters** for all providers except EKS.
- other Kubernetes versions are supported at **1.24.x for deploying clusters in EKS**

Kubernetes 1.25.4, enables you to benefit from the latest features and security fixes in upstream Kubernetes. This release comes with approximately 40 enhancements that you can benefit from like pod security admission[1262], cgroups V2[1263], among others. To read more about major features in this release, visit https://kubernetes.io/blog/2022/08/23/kubernetes-v1-25-release/.

12.2.4.2 Attaching Clusters Versions

DKP 2.5.0 supports **attaching clusters** with the following Kubernetes versions:

Product	Compatible Kubernetes Versions
EKS	1.24.x
AKS	1.25.x

1260 https://podman.io/getting-started/installation
1261 https://podman.io/whatis.html
1262 https://kubernetes.io/docs/concepts/security/pod-security-admission/
1263 https://kubernetes.io/docs/concepts/architecture/cgroups/

Product	Compatible Kubernetes Versions
GKE	1.25.x

When attempting to attach a cluster with a lower Kubernetes version than the DKP default, you need to build and use an image with that lower version of Kubernetes. See Konvoy Image Builder (see page 1282) for documentation on building images. Failure to do this could result in an error.

12.2.5 DKP 2.5.1 Kubernetes major updates and deprecations

Before upgrading, we **strongly recommend** verifying your current setup against the information on this page and reading more about Kubernetes' new features in Kubernetes urgent upgrade notes[1264].

12.2.5.1 Deprecated API Services

With this version, Kubernetes stopped serving several API versions for CronJob, EndpointSlice, Event, HorizontalPodAutoscaler, PodDisruptionBudget, **PodSecurityPolicy**, and RuntimeClass services. For more information on the deprecated API versions for each of these services, refer to https://kubernetes.io/docs/reference/using-api/deprecation-guide/#v1-25.

12.2.5.2 New OCI Image Registry

DKP now uses the upstream `registry.k8s.io` registry instead of the previous `k8s.gcr.io` registry for all new and upgraded clusters.

Ensure you add `registry.k8s.io` to any firewall or proxy allowlists.

12.2.6 DKP 2.5.1 Components and Applications

The following are component and application versions for DKP.

1264 https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.25.md#urgent-upgrade-notes

12.2.6.1 Components

Component Name	Version
Cluster API Core (CAPI)	1.3.3-d2iq.2
Cluster API AWS Infrastructure Provider (CAPA)	1.5.5
Cluster API Google Cloud Infrastructure Provider (CAPG)	1.2.1
Cluster API Pre-provisioned Infrastructure Provider (CAPPP)	0.14.4
Cluster API vSphere Infrastructure Provider (CAPV)	1.5.2
Cluster API Azure Infrastructure Provider (CAPZ)	1.7.1
Konvoy Image Builder (KIB)	2.2.7[1265]
containerd	1.6.17-d2iq.1
etcd	3.5.5
Calico	3.25.1[1266]
Cluster Autoscaler	1.25.x
CSI_VERSION	Default Storage Providers in DKP (see page 103) See this page for versions and driver table.
Metal LB	0.12.1
Node Feature Version	0.12.1

1265 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v2.2.7
1266 https://github.com/projectcalico/calico/blob/v3.25.1/calico/_includes/release-notes/v3.25.1-release-notes.md

12.2.6.2 Applications

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Centralized Grafana	centralized-grafana	44.2.1	• chart: 44.2.1[1267] • prometheus-operator: 0.62.0 • grafana: 9.3.1	Link[1268]	Link[1269]
Centralized Kubecost	centralized-kubecost	0.33.1	• chart: 0.33.1[1270] • kubecost: 1.100.2	Link[1271]	Link[1272]
Chartmuseum	chartmuseum	3.9.0	• chart: 3.9.0[1273] • chartmuseum: 3.9.0	Link[1274]	Link[1275]
Dex	dex	2.11.1	• chart: 2.11.1[1276] • dex: 2.35.1-d2iq.1	Link[1277]	Link[1278]
Dex K8s Authenticator	dex-k8s-authenticator	1.2.14	• chart: 1.2.14[1279] • dex-k8s-authenticator: 1.2.4	Link[1280]	Link[1281]

1267 https://artifacthub.io/packages/helm/mesosphere/kube-prometheus-stack/44.2.1
1268 https://artifacthub.io/packages/helm/mesosphere/kube-prometheus-stack/44.2.1?modal=values
1269 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/centralized-grafana/44.2.1/defaults/cm.yaml
1270 https://artifacthub.io/packages/helm/mesosphere-stable/kubecost/0.33.1
1271 https://artifacthub.io/packages/helm/mesosphere-stable/kubecost/0.33.1?modal=values
1272 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/centralized-kubecost/0.33.1/defaults/cm.yaml
1273 https://artifacthub.io/packages/helm/chartmuseum/chartmuseum/3.9.0
1274 https://artifacthub.io/packages/helm/chartmuseum/chartmuseum/3.9.0?modal=values
1275 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/chartmuseum/3.9.0/defaults/cm.yaml
1276 https://artifacthub.io/packages/helm/mesosphere-stable/dex/2.11.1
1277 https://artifacthub.io/packages/helm/mesosphere-stable/dex/2.11.1?modal=values
1278 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/dex/2.11.1/defaults/cm.yaml
1279 https://artifacthub.io/packages/helm/mesosphere/dex-k8s-authenticator/1.2.14
1280 https://artifacthub.io/packages/helm/mesosphere/dex-k8s-authenticator/1.2.14?modal=values
1281 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/dex-k8s-authenticator/1.2.14/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
DKP Insights Management	dkp-insights-management	0.4.1	• chart: 0.4.1 • dkp-insights-management: 0.4.1	N/A	Link[1282]
External DNS	external-dns	6.13.2	• chart: 6.13.2[1283] • external-dns: 0.13.2	Link[1284]	Link[1285]
Fluent Bit	fluent-bit	0.21.6	• chart: 0.21.6[1286] • fluent-bit: 2.0.6	Link[1287]	Link[1288]
Gatekeeper	gatekeeper	3.11.0	• chart: 3.11.0[1289] • gatekeeper: 3.11.0	Link[1290]	Link[1291]
Gitea	gitea	7.0.2	• chart: 7.0.2[1292] • gitea: 1.18.3	Link[1293]	Link[1294]
Grafana Logging	grafana-logging	6.38.1	• chart: 6.38.1[1295] • grafana: 9.1.5	Link[1296]	Link[1297]

1282 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/dkp-insights-management/0.4.1/defaults/cm.yaml
1283 https://artifacthub.io/packages/helm/bitnami/external-dns/6.13.2
1284 https://artifacthub.io/packages/helm/bitnami/external-dns/6.13.2?modal=values
1285 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/external-dns/6.13.2/defaults/cm.yaml
1286 https://artifacthub.io/packages/helm/fluent/fluent-bit/0.21.6
1287 https://artifacthub.io/packages/helm/fluent/fluent-bit/0.21.6?modal=values
1288 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/fluent-bit/0.21.6/defaults/cm.yaml
1289 https://artifacthub.io/packages/helm/gatekeeper/gatekeeper/3.11.0
1290 https://artifacthub.io/packages/helm/gatekeeper/gatekeeper/3.11.0?modal=values
1291 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/gatekeeper/3.11.0/defaults/cm.yaml
1292 https://artifacthub.io/packages/helm/gitea/gitea/7.0.2
1293 https://artifacthub.io/packages/helm/gitea/gitea/7.0.2?modal=values
1294 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/gitea/7.0.2/defaults/cm.yaml
1295 https://artifacthub.io/packages/helm/grafana/grafana/6.38.1
1296 https://artifacthub.io/packages/helm/grafana/grafana/6.38.1?modal=values
1297 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/grafana-logging/6.38.1/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Grafana Loki	grafana-loki	0.69.4	• chart: 0.69.4[1298] • loki: 2.7.3	Link[1299]	Link[1300]
Istio	istio	1.16.2	• chart: 1.16.2[1301] • istio: 1.16.2	Link[1302]	Link[1303]
Jaeger	jaeger	2.40.0	• chart: 2.40.0[1304] • jaeger: 1.39.0	Link[1305]	Link[1306]
Karma	karma	2.0.1	• chart: 2.0.1[1307] • karma: 0.70	Link[1308]	Link[1309]
Kiali	kiali	1.63.2	• chart: 1.63.2[1310] • kiali: 1.63.2	Link[1311]	Link[1312]
Knative	knative	0.5.2	• chart: 0.5.2[1313] • knative: 0.22.3	Link[1314]	Link[1315]

1298 https://artifacthub.io/packages/helm/grafana/loki-distributed/0.69.4
1299 https://artifacthub.io/packages/helm/grafana/loki-distributed/0.69.4?modal=values
1300 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/grafana-loki/0.69.4/defaults/cm.yaml
1301 https://artifacthub.io/packages/helm/mesosphere/istio/1.16.2
1302 https://artifacthub.io/packages/helm/mesosphere/istio/1.16.2?modal=values
1303 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/istio/1.16.2/defaults/cm.yaml
1304 https://artifacthub.io/packages/helm/jaegertracing/jaeger-operator/2.40.0
1305 https://artifacthub.io/packages/helm/jaegertracing/jaeger-operator/2.40.0?modal=values
1306 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/jaeger/2.40.0/defaults/cm.yaml
1307 https://artifacthub.io/packages/helm/mesosphere-stable/karma/2.0.1
1308 https://artifacthub.io/packages/helm/mesosphere-stable/karma/2.0.1?modal=values
1309 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/karma/2.0.1/defaults/cm.yaml
1310 https://artifacthub.io/packages/helm/kiali/kiali-operator/1.63.2
1311 https://artifacthub.io/packages/helm/kiali/kiali-operator/1.63.2?modal=values
1312 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/kiali/1.63.2/defaults/cm.yaml
1313 https://artifacthub.io/packages/helm/mesosphere/knative/0.5.2
1314 https://artifacthub.io/packages/helm/mesosphere/knative/0.5.2?modal=values
1315 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/knative/0.5.2/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Flux	kommander-flux	0.40.2	• chart: N/A • flux: 0.40.2	N/A	N/A
Kube OIDC Proxy	kube-oidc-proxy	0.3.2	• chart: 0.3.1[1316] • kube-oidc-proxy: 0.3.0	Link[1317]	Link[1318]
Kube Prometheus Stack	kube-prometheus-stack	44.2.1	• chart: 44.2.1[1319] • prometheus-operator: 0.62.0 • grafana: 9.3.1 • prometheus: 2.41.0 • prometheus-alertmanager: 0.25.0	Link[1320]	Link[1321]
Kubecost	kubecost	0.33.1	• chart: 0.33.1[1322] • kubecost: 1.100.2	Link[1323]	Link[1324]
Kubefed	kubefed	0.10.0	• chart: 0.10.0[1325] • kubefed: 0.10.0	Link[1326]	Link[1327]

1316 https://artifacthub.io/packages/helm/mesosphere/kube-oidc-proxy/0.3.1
1317 https://artifacthub.io/packages/helm/mesosphere/kube-oidc-proxy/0.3.1?modal=values
1318 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/kube-oidc-proxy/0.3.2/defaults/cm.yaml
1319 https://artifacthub.io/packages/helm/mesosphere/kube-prometheus-stack/44.2.1
1320 https://artifacthub.io/packages/helm/mesosphere/kube-prometheus-stack/44.2.1?modal=values
1321 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/kube-prometheus-stack/44.2.1/defaults/cm.yaml
1322 https://artifacthub.io/packages/helm/mesosphere-stable/kubecost/0.33.1
1323 https://artifacthub.io/packages/helm/mesosphere-stable/kubecost/0.33.1?modal=values
1324 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/kubecost/0.33.1/defaults/cm.yaml
1325 https://artifacthub.io/packages/helm/kubefed/kubefed/0.10.0
1326 https://artifacthub.io/packages/helm/kubefed/kubefed/0.10.0?modal=values
1327 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/kubefed/0.10.0/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Kubernetes Dashboard	kubernetes-dashboard	6.0.0	• chart: 6.0.0[1328] • kubernetes-dashboard: 2.7.0	Link[1329]	Link[1330]
Kubetunnel	kubetunnel	0.0.15	• chart: 0.0.15 • kubetunnel: 0.0.15	N/A	Link[1331]
Logging Operator	logging-operator	3.17.10	• chart: 3.17.10[1332] • logging-operator: 3.17.10 • logging-operator-logging: 3.17.10	Link[1333]	Link[1334]
NFS Server Provisioner	nfs-server-provisioner	0.6.0	• chart: 0.6.0[1335] • nfs-server-provisioner: 2.3.0	Link[1336]	Link[1337]
NVIDIA GPU Operator	nvidia-gpu-operator	22.9.1	• chart: 22.9.1[1338] • nvidia-gpu-operator: 22.9.1	Link[1339]	Link[1340]

1328 https://artifacthub.io/packages/helm/k8s-dashboard/kubernetes-dashboard/6.0.0
1329 https://artifacthub.io/packages/helm/k8s-dashboard/kubernetes-dashboard/6.0.0?modal=values
1330 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/kubernetes-dashboard/6.0.0/defaults/cm.yaml
1331 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/kubetunnel/0.0.15/defaults/cm.yaml
1332 https://artifacthub.io/packages/helm/banzaicloud-stable/logging-operator/3.17.10
1333 https://artifacthub.io/packages/helm/banzaicloud-stable/logging-operator/3.17.10?modal=values
1334 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/logging-operator/3.17.10/defaults/cm.yaml
1335 https://artifacthub.io/packages/helm/mesosphere/nfs-server-provisioner/0.6.0
1336 https://artifacthub.io/packages/helm/mesosphere/nfs-server-provisioner/0.6.0?modal=values
1337 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/nfs-server-provisioner/0.6.0/defaults/cm.yaml
1338 https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/getting-started.html#chart-customization-options
1339 https://raw.githubusercontent.com/NVIDIA/gpu-operator/v22.9.1/deployments/gpu-operator/values.yaml
1340 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/nvidia-gpu-operator/22.9.1/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Grafana (project)	project-grafana-logging	6.38.1	• chart: 6.38.1[1341] • grafana: 9.1.5	Link[1342]	Link[1343]
Grafana Loki (project)	project-grafana-loki	0.69.4	• chart: 0.69.4[1344] • loki: 2.7.3	Link[1345]	Link[1346]
Prometheus Adapter	prometheus-adapter	4.0.2	• chart: 4.0.2[1347] • prometheus-adapter: 0.10.0	Link[1348]	Link[1349]
Reloader	reloader	1.0.5	• chart: 1.0.5[1350] • reloader: 1.0.5	Link[1351]	Link[1352]
Rook Ceph	rook-ceph	1.10.11	• chart: 1.10.11[1353] • rook-ceph: 1.10.11	Link[1354]	Link[1355]

1341 https://artifacthub.io/packages/helm/grafana/grafana/6.38.1
1342 https://artifacthub.io/packages/helm/grafana/grafana/6.38.1?modal=values
1343 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/project-grafana-logging/6.38.1/defaults/cm.yaml
1344 https://artifacthub.io/packages/helm/grafana/loki-distributed/0.69.4
1345 https://artifacthub.io/packages/helm/grafana/loki-distributed/0.69.4?modal=values
1346 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/project-grafana-loki/0.69.4/defaults/cm.yaml
1347 https://artifacthub.io/packages/helm/prometheus-community/prometheus-adapter/4.0.2
1348 https://artifacthub.io/packages/helm/prometheus-community/prometheus-adapter/4.0.2?modal=values
1349 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/prometheus-adapter/4.0.2/defaults/cm.yaml
1350 https://github.com/stakater/Reloader/tree/v1.0.5/README.md#helm-charts
1351 https://artifacthub.io/packages/helm/stakater/reloader/1.0.5?modal=values
1352 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/reloader/1.0.5/defaults/cm.yaml
1353 https://github.com/rook/rook/tree/v1.10.11/Documentation/Helm-Charts/operator-chart.md
1354 https://raw.githubusercontent.com/rook/rook/v1.10.11/deploy/charts/rook-ceph/values.yaml
1355 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/rook-ceph/1.10.11/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Rook Ceph Cluster	rook-ceph-cluster	1.10.11	• chart: 1.10.11[1356] • rook-ceph: 1.10.11 • rook-ceph-cluster: 17.2.5	Link[1357]	Link[1358]
Thanos	thanos	0.4.8	• chart: 0.4.6[1359] • thanos: 0.17.1	Link[1360]	Link[1361]
Traefik	traefik	20.8.0	• chart: 20.8.0[1362] • traefik: 2.9.6	Link[1363]	Link[1364]
Traefik ForwardAuth	traefik-forward-auth	0.3.8	• chart: 0.3.8[1365] • traefik-forward-auth: 3.1.0	Link[1366]	Link[1367]
Velero	velero	3.4.0	• chart: 3.1.2[1368] • velero: 1.10.0	Link[1369]	Link[1370]

12.2.7 DKP 2.5.1 Known Issues and Limitations

The following items are known issues with this release.

1356 https://github.com/rook/rook/tree/v1.10.11/Documentation/Helm-Charts/ceph-cluster-chart.md
1357 https://raw.githubusercontent.com/rook/rook/v1.10.11/deploy/charts/rook-ceph-cluster/values.yaml
1358 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/rook-ceph-cluster/1.10.11/defaults/cm.yaml
1359 https://artifacthub.io/packages/helm/bitnami/thanos/0.4.6
1360 https://artifacthub.io/packages/helm/bitnami/thanos/0.4.6?modal=values
1361 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/thanos/0.4.8/defaults/cm.yaml
1362 https://artifacthub.io/packages/helm/traefik/traefik/20.8.0
1363 https://artifacthub.io/packages/helm/traefik/traefik/20.8.0?modal=values
1364 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/traefik/20.8.0/defaults/cm.yaml
1365 https://artifacthub.io/packages/helm/mesosphere/traefik-forward-auth/0.3.8
1366 https://artifacthub.io/packages/helm/mesosphere/traefik-forward-auth/0.3.8?modal=values
1367 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/traefik-forward-auth/0.3.8/defaults/cm.yaml
1368 https://artifacthub.io/packages/helm/vmware-tanzu/velero/3.1.2
1369 https://artifacthub.io/packages/helm/vmware-tanzu/velero/3.1.2?modal=values
1370 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.1/services/velero/3.4.0/defaults/cm.yaml

12.2.7.1 Use Static Credentials to Provision an Azure Cluster

Only static credentials can be used when provisioning an Azure cluster (see page 1041).

12.2.7.2 Containerd 1.4.13 File Limit Issue

In this version of DKP, we introduced containerd 1.6.17. The systemd unit for containerd 1.6.17 provided upstream removes all file number limits (`LimitNOFILE=infinity`). In our testing, we found that removing these limits broke some IO sensitive applications like Rook Ceph and HAProxy. Because of this, the KIB version included in this release sets the `LimitNOFILE` value in the containerd systemd unit to the value (`1048576`) that was used in previous containerd 1.4.13 version releases.

12.2.7.3 Intermittent Error Status when Creating EKS Clusters in the UI

When provisioning an EKS cluster through the UI, you may receive a brief error state because the EKS cluster may sporadically lose connectivity with the management cluster which results in the following symptoms:

- The UI shows the cluster is in an error state.
- The kubeconfig generated and retrieved from Kommander ceases to work.
- Applications created on the management cluster may not be immediately federated to managed EKS clusters.

After a few moments, the error will resolve, without any action on your part. A new kubeconfig generated and retrieved from Kommander then works properly, and the UI shows that it is working again. In the meantime, you can continue to use the UI to work on the cluster such as deploy applications, create projects, and add roles.

12.2.7.4 Installation Issue in Pre-provisioned Environments

An issue with Rook Ceph's deployment prevents pre-provisioned environments from installing this DKP version. To solve this issue, you must set up a minimum of 40 GB of raw storage for your worker nodes and customize your Rook Ceph installation as indicated in Install Kommander in a Pre-provisioned Environment (see page 1271).

12.2.7.5 Resolve issues with failed HelmReleases

An issue with the Flux helm-controller[1371] can cause HelmReleases to fail with the error message *Helm upgrade failed: another operation (install/upgrade/rollback) is in progress*. This can happen when the helm-controller is restarted while a HelmRelease is still upgrading, or installing.

1371 https://github.com/fluxcd/helm-controller/issues/149

12.2.7.5.1 Workaround

To ensure the HelmRelease error was caused by the helm-controller restarting, first try to suspend/resume the HelmRelease:

```
kubectl -n <namespace> patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n <namespace> patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

This might resolve the issue. If not, continue with the following steps:

You should see the HelmRelease attempting to reconcile, and then it either succeeds (with status: *Release reconciliation succeeded*) or it fails with the same error as before.

If the HelmRelease is still in the failed state, it is likely related to the helm-controller restarting. For example, if the 'reloader' HelmRelease is the one that is stuck.

To resolve the issue, follow these steps:

1. List secrets containing the affected HelmRelease name:

   ```
   kubectl get secrets -n ${NAMESPACE} | grep reloader
   ```

   The output should look like this:

   ```
   kommander-reloader-reloader-token-9qd8b                          kubernetes.io/
   service-account-token   3      171m
   sh.helm.release.v1.kommander-reloader.v1                         helm.sh/
   release.v1                     1      171m
   sh.helm.release.v1.kommander-reloader.v2                         helm.sh/
   release.v1                     1      117m
   ```

   In this example, `sh.helm.release.v1.kommander-reloader.v2` is the most recent revision.

2. Find and delete the most recent revision secret, for example, `sh.helm.release.v1.*.<revision>`:

   ```
   kubectl delete secret -n <namespace> <most recent helm revision secret name>
   ```

3. Suspend and resume the HelmRelease to trigger a reconciliation:

   ```
   kubectl -n <namespace> patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[
   {"op": "replace", "path": "/spec/suspend", "value": true}]'
   kubectl -n <namespace> patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[
   {"op": "replace", "path": "/spec/suspend", "value": false}]'
   ```

You should see the HelmRelease is reconciled and eventually the upgrade and install succeeds.

12.2.7.6 Limitations to Disk Resizing in vSphere

The DKP CLI flags `--control-plane-disk-size` and `--worker-disk-size` are unable to resize the root file system of VMs created using OS images. The flags work by resizing the primary disk of the VM. When the VM boots, the root file system is expanded to fill the disk, but that expansion does not work for some file systems, for example, for file systems contained in an LVM Logical Volume. To ensure your root file system has the size you expect, please see https://d2iq.atlassian.net/wiki/spaces/DENT/pages/212993627/2.5%2BCreate%2Ba%2BvSphere%2BBase%2BOS%2BImage#Disk-Size (see page 0).

12.2.7.7 Error Status in Grafana Logging Dashboard with EKS Clusters

Currently, it is not possible to use FluentBit to collect Admin-level logs (see page 798) on a managed EKS cluster.

If you have these logs enabled, the following message appears when you access the Kubernetes Audit Dashboard in the Grafana Logging Dashboard:

```
Cannot read properties of undefined (reading '0')
```

12.2.7.8 Logging Operator Upgrade Error

There is a race condition that could result in the `logging-operator-logging-fluentd` using the incorrect image tag during upgrade from DKP 2.4.0 to DKP 2.5.0.

The image tag is eventually corrected by the `logging-operator`, however due to the nature of StatefulSets, the failing pod needs to be removed in order for the StatefulSet to continue rolling out the required updates.

1. Run this command to find if any Fluentd pods are in the `ImagePullBackOff` state post-upgrade:

   ```
   kubectl get pod -l app.kubernetes.io/name=fluentd,app.kubernetes.io/managed-
   by=logging-operator-logging,app.kubernetes.io/component=fluentd -n kommander
   ```

2. If `ImagePullBackOff` is present in the output like in the example below, you will need to continue with these steps to resolve the issue.

   ```
   NAME                                READY   STATUS             RESTARTS   AGE
   logging-operator-logging-fluentd-0  3/3     Running            0
   6m41s
   logging-operator-logging-fluentd-1  2/3     ImagePullBackOff   0
   2m33s
   ```

3. Delete the Fluentd pod that is in an `ImagePullBackOff` state. In this case, it is `logging-operator-logging-fluentd-1`:

```
kubectl delete pod -n kommander logging-operator-logging-fluentd-1
```

4. The upgrade of the `logging-operator-logging-fluentd` StatefulSet now proceeds as normal.

12.2.7.9 Nodepools Update Error with Knative

The following only applies to your environment if you have Knative installed and if its deployment is scaled to less than 5 pods.
An issue with the `PodDisruptionBudget` resource blocks the deletion of old nodes when upgrading from DKP 2.4.0 to DKP 2.5.0, which results in a failure of the DKP nodepools upgrade.

1. If the `dkp update nodepool` command fails, check to see if `PodDisruptionBudget` with `ALLOWED DISRUPTIONS` equals 0 using the following command:

```
kubectl get pdb -n knative-serving
NAME                            MIN AVAILABLE   MAX UNAVAILABLE   ALLOWED
DISRUPTIONS   AGE
activator-pdb                   80%             N/A               0
22h
webhook-pdb                     80%             N/A               0
22h
```

2. Obtain the list of pods in the knative-serving namespace containing PDB with the following command:

```
kubectl get pods -n knative-serving -l 'app in (webhook, activator)'
```

3. The output should look similar to the following:

```
NAME                          READY   STATUS    RESTARTS   AGE
webhook-XXXXXXXXX-XXXX        2/2     Running   0          5d21h
activator-XXXXXXXXX-XXXX      2/2     Running   0          4d23h
```

4. Delete the pods that contains the `PodDisruptionBudget` resource:

```
kubectl delete pod -n knative-serving activator-XXXX-XXX webhook-XXXXX-XXXX
```

5. The upgrade of DKP and Knative now proceeds as normal. Re-run the `dkp update nodepool` command.

12.2.7.10 Rook Ceph Install Error

An issue may emerge when installing `rook-ceph` on vSphere clusters using RHEL operating systems.

This issue occurs during initial installation of rook-ceph, causing the object store used by Velero and Grafana Loki, to be unavailable. If the installation of Kommander component of DKP is unsuccessful due to `rook-ceph` failing, you might need to apply the following workaround.

1. Run the following command to see if the cluster is affected by this issue.

```
kubectl describe CephObjectStores dkp-object-store -n kommander
```

2. If the following output appears, this workaround needs to be applied:

```
Name:         dkp-object-store
Namespace:    kommander
...
  Warning  ReconcileFailed     7m55s (x19 over 52m)
  rook-ceph-object-controller  failed to reconcile CephObjectStore
  "kommander/dkp-object-store". failed to create object store deployments:
failed
  to configure multisite for object store: failed create ceph multisite for
  object-store ["dkp-object-store"]: failed to commit config changes after
  creating multisite config for CephObjectStore "kommander/dkp-object-store":
  failed to commit RGW configuration period changes%!(EXTRA []string=[]):
signal: interrupt
```

3. Kubectl exec into the `rook-ceph-tools` pod.

```
export WORKSPACE_NAMESPACE=<workspace namespace>
CEPH_TOOLS_POD=$(kubectl get pods -l app=rook-ceph-tools -n $
{WORKSPACE_NAMESPACE} -o name)
kubectl exec -it -n ${WORKSPACE_NAMESPACE} $CEPH_TOOLS_POD bash
```

4. Run the following commands to set `dkp-object-store` as the default zonegroup.
   NOTE: The `period update` command may take a few minutes to complete

```
radosgw-admin zonegroup default --rgw-zonegroup=dkp-object-store
radosgw-admin period update --commit
```

5. Next, restart the `rook-ceph-operator` deployment for the `CephobjectStore` to be reconciled.

```
kubectl rollout restart deploy -n${WORKSPACE_NAMESPACE} rook-ceph-operator
```

6. After running the commands above, the `CephObjectStore` should be `Connected` once the `rook-ceph` operator reconciles the object (this may take some time).

```
kubectl wait CephObjectStore --for=jsonpath='{.status.phase}'=Connected dkp-
object-store -n ${WORKSPACE_NAMESPACE} --timeout 10m
```

12.2.7.11 Post Upgrade, Volume Cannot Attach to a Node Already Attached to Another Node

Due to an upstream issue[1372], when you bring a new node up during a Kubernetes version upgrade and then delete the old node, an existing volume might not attach to the new node.

You will see this when a new pod that uses a volume does not become ready in the new node, and then an event that says something such as `Volume <pvc/pv-id> is already exclusively attached to one node and can't be attached to another`.

This will be fixed[1373] in a future Kubernetes release[1374], for example this is described in vSphere here[1375]. Different methods might be needed to resolve this manually[1376], including this method to resolve on vSphere[1377].

12.2.7.12 Control Plane Upgrade - Oracle Linux node-feature-discovery Fails

When upgrading the Oracle Linux node-feature-discovery fails to terminate, so the control-plane upgrade process can freeze and fail after the time out.

Follow these steps to avoid this issue:

1. View the machines for your cluster:

```
kubectl get machines
```

2. Identify the machine that has the faulty `node-feature-discovery` pod, and then delete that machine:

1372 https://discuss.kubernetes.io/t/multi-attach-error-for-volume-pvc-volume-is-already-exclusively-attached-to-one-node-and-cant-be-attached-to-another/18951
1373 https://github.com/kubernetes/enhancements/pull/1116
1374 https://kubernetes.io/docs/reference/command-line-tools-reference/feature-gates/?highlight=Feature#:~:text=1.27-,NodeOutOfServiceVolumeDetach,-false
1375 https://github.com/kubernetes-sigs/vsphere-csi-driver/issues/125
1376 https://discuss.kubernetes.io/t/multi-attach-error-for-volume-pvc-volume-is-already-exclusively-attached-to-one-node-and-cant-be-attached-to-another/18951/3
1377 https://kb.vmware.com/s/article/70519

```
kubectl delete machine <faulty-machine-name-here>
```

A replacement machine generates and reconciles on your cluster.

12.2.8 DKP 2.5.1 Customer Incidents

The following resolved incident is included in this release:

12.2.8.1 DKP 2.5.0 Image Seed Fails with a Docker Error

D2IQ-96939

When attempting to seed a local Harbor registry with DKP 2.5.0 images, with a `Docker registry panic` error. This issue is corrected.

12.3 DKP 2.5.2 Release Notes

DKP® version 2.5.2 was released on September 13, 2023

Download DKP

> You must be a registered user and logged on to the support portal to download this product. New customers must contact their sales representative or sales@d2iq.com[1378] before attempting to download or install DKP.

12.3.1 Release Summary

Welcome to D2iQ Kubernetes Platform (DKP) 2.5.2! This release provides fixes to reported issues, integrates changes from previous releases, and maintains compatibility and support for other packages used in DKP.

- DKP 2.5.2 Supported Kubernetes Versions (see page 1581)
- DKP 2.5.2 Kubernetes Major Updates and Deprecations (see page 1582)
- DKP 2.5.2 Components and Applications (see page 1582)
- DKP 2.5.2 Known Issues and Limitations (see page 1590)
- DKP 2.5.2 Customer Incidents (see page 1597)

1378 mailto:sales@d2iq.com

12.3.2 Additional resources

- For more information about working with native Kubernetes, see the Kubernetes documentation[1379].
- For a full list of attributed 3rd party software, see http://d2iq.com/legal/3rd .

12.3.3 DKP 2.5.2 Supported Kubernetes Versions

12.3.3.1 Deploying Cluster Versions

In DKP 2.5, the Kubernetes version installed by default is **1.25.4**. The newest and oldest Kubernetes versions used with DKP must be within one minor version.

- latest supported Kubernetes version is at **1.25.4 for deploying clusters** for all providers except EKS.
- other Kubernetes versions are supported at **1.24.x for deploying clusters in EKS**

Kubernetes 1.25.4, enables you to benefit from the latest features and security fixes in upstream Kubernetes. This release comes with approximately 40 enhancements that you can benefit from like pod security admission[1380], cgroups V2[1381], among others. To read more about major features in this release, visit https://kubernetes.io/blog/2022/08/23/kubernetes-v1-25-release/.

12.3.3.2 Attaching Clusters Versions

DKP 2.5.0 supports **attaching clusters** with the following Kubernetes versions:

Product	Compatible Kubernetes Versions
EKS	1.24.x
AKS	1.25.x
GKE	1.25.x

When attempting to attach a cluster with a lower Kubernetes version than the DKP default, you need to build and use an image with that lower version of Kubernetes. See Konvoy Image Builder (see page 1282) for documentation on building images. Failure to do this could result in an error.

1379 https://kubernetes.io/docs/home/
1380 https://kubernetes.io/docs/concepts/security/pod-security-admission/
1381 https://kubernetes.io/docs/concepts/architecture/cgroups/

12.3.4 DKP 2.5.2 Kubernetes Major Updates and Deprecations

Before upgrading, we **strongly recommend** verifying your current setup against the information on this page and reading more about Kubernetes' new features in Kubernetes urgent upgrade notes[1382].

12.3.4.1 Deprecated API Services

With this version, Kubernetes stopped serving several API versions for CronJob, EndpointSlice, Event, HorizontalPodAutoscaler, PodDisruptionBudget, **PodSecurityPolicy**, and RuntimeClass services. For more information on the deprecated API versions for each of these services, refer to https://kubernetes.io/docs/reference/using-api/deprecation-guide/#v1-25.

12.3.4.2 New OCI Image Registry

DKP now uses the upstream `registry.k8s.io` registry instead of the previous `k8s.gcr.io` registry for all new and upgraded clusters.

Ensure you add `registry.k8s.io` to any firewall or proxy allowlists.

12.3.5 DKP 2.5.2 Components and Applications

The following are component and application versions for this version of DKP.

12.3.5.1 Components

Component Name	Version
Cluster API Core (CAPI)	1.3.3-d2iq.2
Cluster API AWS Infrastructure Provider (CAPA)	1.5.5
Cluster API Google Cloud Infrastructure Provider (CAPG)	1.2.1

1382 https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.25.md#urgent-upgrade-notes

Component Name	Version
Cluster API Pre-provisioned Infrastructure Provider (CAPPP)	0.14.10
Cluster API vSphere Infrastructure Provider (CAPV)	1.5.2
Cluster API Azure Infrastructure Provider (CAPZ)	1.7.1
Konvoy Image Builder (KIB)	2.2.12[1383]
containerd	1.6.17-d2iq.1
etcd	3.5.5
Calico	3.25.1[1384]
Cluster Autoscaler	1.25.x
CSI_VERSION	Default Storage Providers in DKP (see page 103) See this page for versions and driver table.
Metal LB	0.12.1
Node Feature Version	0.12.1

1383 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v2.2.12
1384 https://github.com/projectcalico/calico/blob/v3.25.1/calico/_includes/release-notes/v3.25.1-release-notes.md

12.3.5.2 Applications

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Centralized Grafana	centralized-grafana	44.2.2	• chart: 44.2.1[1385] • prometheus-operator: 0.62.0 • grafana: 9.3.1	Link[1386]	Link[1387]
Centralized Kubecost	centralized-kubecost	0.33.2	• chart: 0.33.1[1388] • kubecost: 1.100.2	Link[1389]	Link[1390]
Cert Manager	cert-manager	1.11.0	• chart: 1.11.0[1391] • cert-manager: 1.11.0	Link[1392]	Link[1393]
Chartmuseum	chartmuseum	3.9.0	• chart: 3.9.0[1394] • chartmuseum: 3.9.0	Link[1395]	Link[1396]
Dex	dex	2.11.1	• chart: 2.11.1[1397] • dex: 2.35.1-d2iq.1	Link[1398]	Link[1399]

1385 https://artifacthub.io/packages/helm/mesosphere/kube-prometheus-stack/44.2.1
1386 https://artifacthub.io/packages/helm/mesosphere/kube-prometheus-stack/44.2.1?modal=values
1387 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/centralized-grafana/44.2.2/defaults/cm.yaml
1388 https://artifacthub.io/packages/helm/mesosphere-stable/kubecost/0.33.1
1389 https://artifacthub.io/packages/helm/mesosphere-stable/kubecost/0.33.1?modal=values
1390 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/centralized-kubecost/0.33.2/defaults/cm.yaml
1391 https://artifacthub.io/packages/helm/cert-manager/cert-manager/1.11.0
1392 https://artifacthub.io/packages/helm/cert-manager/cert-manager/1.11.0?modal=values
1393 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/cert-manager/1.11.0/defaults/cm.yaml
1394 https://artifacthub.io/packages/helm/chartmuseum/chartmuseum/3.9.0
1395 https://artifacthub.io/packages/helm/chartmuseum/chartmuseum/3.9.0?modal=values
1396 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/chartmuseum/3.9.0/defaults/cm.yaml
1397 https://artifacthub.io/packages/helm/mesosphere-stable/dex/2.11.1
1398 https://artifacthub.io/packages/helm/mesosphere-stable/dex/2.11.1?modal=values
1399 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/dex/2.11.1/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Dex K8s Authenticator	dex-k8s-authenticator	1.2.14	• chart: 1.2.14[1400] • dex-k8s-authenticator: 1.2.4	Link[1401]	Link[1402]
DKP Insights Management	dkp-insights-management	0.4.1	• chart: 0.4.1 • dkp-insights-management: 0.4.1	N/A	Link[1403]
External DNS	external-dns	6.13.2	• chart: 6.13.2[1404] • external-dns: 0.13.2	Link[1405]	Link[1406]
Fluent Bit	fluent-bit	0.21.6	• chart: 0.21.6[1407] • fluent-bit: 2.0.6	Link[1408]	Link[1409]
Gatekeeper	gatekeeper	3.11.1	• chart: 3.11.0[1410] • gatekeeper: 3.11.0	Link[1411]	Link[1412]
Gitea	gitea	7.0.2	• chart: 7.0.2[1413] • gitea: 1.18.3	Link[1414]	Link[1415]

1400 https://artifacthub.io/packages/helm/mesosphere/dex-k8s-authenticator/1.2.14
1401 https://artifacthub.io/packages/helm/mesosphere/dex-k8s-authenticator/1.2.14?modal=values
1402 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/dex-k8s-authenticator/1.2.14/defaults/cm.yaml
1403 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/dkp-insights-management/0.4.1/defaults/cm.yaml
1404 https://artifacthub.io/packages/helm/bitnami/external-dns/6.13.2
1405 https://artifacthub.io/packages/helm/bitnami/external-dns/6.13.2?modal=values
1406 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/external-dns/6.13.2/defaults/cm.yaml
1407 https://artifacthub.io/packages/helm/fluent/fluent-bit/0.21.6
1408 https://artifacthub.io/packages/helm/fluent/fluent-bit/0.21.6?modal=values
1409 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/fluent-bit/0.21.6/defaults/cm.yaml
1410 https://artifacthub.io/packages/helm/gatekeeper/gatekeeper/3.11.0
1411 https://artifacthub.io/packages/helm/gatekeeper/gatekeeper/3.11.0?modal=values
1412 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/gatekeeper/3.11.1/defaults/cm.yaml
1413 https://artifacthub.io/packages/helm/gitea/gitea/7.0.2
1414 https://artifacthub.io/packages/helm/gitea/gitea/7.0.2?modal=values
1415 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/gitea/7.0.2/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Grafana Logging	grafana-logging	6.38.1	• chart: 6.38.1[1416] • grafana: 9.1.5	Link[1417]	Link[1418]
Grafana Loki	grafana-loki	0.69.5	• chart: 0.69.4[1419] • loki: 2.7.3	Link[1420]	Link[1421]
Istio	istio	1.16.2	• chart: 1.16.2[1422] • istio: 1.16.2	Link[1423]	Link[1424]
Jaeger	jaeger	2.40.0	• chart: 2.40.0[1425] • jaeger: 1.39.0	Link[1426]	Link[1427]
Karma	karma	2.0.1	• chart: 2.0.1[1428] • karma: 0.70	Link[1429]	Link[1430]
Kiali	kiali	1.63.2	• chart: 1.63.2[1431] • kiali: 1.63.2	Link[1432]	Link[1433]

1416 https://artifacthub.io/packages/helm/grafana/grafana/6.38.1
1417 https://artifacthub.io/packages/helm/grafana/grafana/6.38.1?modal=values
1418 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/grafana-logging/6.38.1/defaults/cm.yaml
1419 https://artifacthub.io/packages/helm/grafana/loki-distributed/0.69.4
1420 https://artifacthub.io/packages/helm/grafana/loki-distributed/0.69.4?modal=values
1421 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/grafana-loki/0.69.5/defaults/cm.yaml
1422 https://artifacthub.io/packages/helm/mesosphere/istio/1.16.2
1423 https://artifacthub.io/packages/helm/mesosphere/istio/1.16.2?modal=values
1424 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/istio/1.16.2/defaults/cm.yaml
1425 https://artifacthub.io/packages/helm/jaegertracing/jaeger-operator/2.40.0
1426 https://artifacthub.io/packages/helm/jaegertracing/jaeger-operator/2.40.0?modal=values
1427 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/jaeger/2.40.0/defaults/cm.yaml
1428 https://artifacthub.io/packages/helm/mesosphere-stable/karma/2.0.1
1429 https://artifacthub.io/packages/helm/mesosphere-stable/karma/2.0.1?modal=values
1430 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/karma/2.0.1/defaults/cm.yaml
1431 https://artifacthub.io/packages/helm/kiali/kiali-operator/1.63.2
1432 https://artifacthub.io/packages/helm/kiali/kiali-operator/1.63.2?modal=values
1433 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/kiali/1.63.2/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Knative	knative	0.5.2	• chart: 0.5.2[1434] • knative: 0.22.3	Link[1435]	Link[1436]
Flux	kommander-flux	0.40.2	• chart: N/A • flux: 0.40.2	N/A	N/A
Kube OIDC Proxy	kube-oidc-proxy	0.3.2	• chart: 0.3.1[1437] • kube-oidc-proxy: 0.3.0	Link[1438]	Link[1439]
Kube Prometheus Stack	kube-prometheus-stack	44.2.2	• chart: 44.2.1[1440] • prometheus-operator: 0.62.0 • grafana: 9.3.1 • prometheus: 2.41.0 • prometheus-alertmanager: 0.25.0	Link[1441]	Link[1442]
Kubecost	kubecost	0.33.2	• chart: 0.33.1[1443] • kubecost: 1.100.2	Link[1444]	Link[1445]
Kubefed	kubefed	0.10.0	• chart: 0.10.0[1446] • kubefed: 0.10.0	Link[1447]	Link[1448]

1434 https://artifacthub.io/packages/helm/mesosphere/knative/0.5.2
1435 https://artifacthub.io/packages/helm/mesosphere/knative/0.5.2?modal=values
1436 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/knative/0.5.2/defaults/cm.yaml
1437 https://artifacthub.io/packages/helm/mesosphere/kube-oidc-proxy/0.3.1
1438 https://artifacthub.io/packages/helm/mesosphere/kube-oidc-proxy/0.3.1?modal=values
1439 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/kube-oidc-proxy/0.3.2/defaults/cm.yaml
1440 https://artifacthub.io/packages/helm/mesosphere/kube-prometheus-stack/44.2.1
1441 https://artifacthub.io/packages/helm/mesosphere/kube-prometheus-stack/44.2.1?modal=values
1442 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/kube-prometheus-stack/44.2.2/defaults/cm.yaml
1443 https://artifacthub.io/packages/helm/mesosphere-stable/kubecost/0.33.1
1444 https://artifacthub.io/packages/helm/mesosphere-stable/kubecost/0.33.1?modal=values
1445 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/kubecost/0.33.2/defaults/cm.yaml
1446 https://artifacthub.io/packages/helm/kubefed/kubefed/0.10.0
1447 https://artifacthub.io/packages/helm/kubefed/kubefed/0.10.0?modal=values
1448 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/kubefed/0.10.0/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Kubernetes Dashboard	kubernetes-dashboard	6.0.0	• chart: 6.0.0[1449] • kubernetes-dashboard: 2.7.0	Link[1450]	Link[1451]
Kubetunnel	kubetunnel	0.0.15	• chart: 0.0.15 • kubetunnel: 0.0.15	N/A	Link[1452]
Logging Operator	logging-operator	3.17.11	• chart: 3.17.10[1453] • logging-operator: 3.17.10 • logging-operator-logging: 3.17.10	Link[1454]	Link[1455]
NFS Server Provisioner	nfs-server-provisioner	0.6.0	• chart: 0.6.0[1456] • nfs-server-provisioner: 2.3.0	Link[1457]	Link[1458]
NVIDIA GPU Operator	nvidia-gpu-operator	22.9.1	• chart: 22.9.1[1459] • nvidia-gpu-operator: 22.9.1	Link[1460]	Link[1461]

1449 https://artifacthub.io/packages/helm/k8s-dashboard/kubernetes-dashboard/6.0.0
1450 https://artifacthub.io/packages/helm/k8s-dashboard/kubernetes-dashboard/6.0.0?modal=values
1451 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/kubernetes-dashboard/6.0.0/defaults/cm.yaml
1452 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/kubetunnel/0.0.15/defaults/cm.yaml
1453 https://artifacthub.io/packages/helm/banzaicloud-stable/logging-operator/3.17.10
1454 https://artifacthub.io/packages/helm/banzaicloud-stable/logging-operator/3.17.10?modal=values
1455 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/logging-operator/3.17.11/defaults/cm.yaml
1456 https://artifacthub.io/packages/helm/mesosphere/nfs-server-provisioner/0.6.0
1457 https://artifacthub.io/packages/helm/mesosphere/nfs-server-provisioner/0.6.0?modal=values
1458 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/nfs-server-provisioner/0.6.0/defaults/cm.yaml
1459 https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/getting-started.html#chart-customization-options
1460 https://raw.githubusercontent.com/NVIDIA/gpu-operator/v22.9.1/deployments/gpu-operator/values.yaml
1461 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/nvidia-gpu-operator/22.9.1/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Grafana (project)	project-grafana-logging	6.38.1	• chart: 6.38.1[1462] • grafana: 9.1.5	Link[1463]	Link[1464]
Grafana Loki (project)	project-grafana-loki	0.69.5	• chart: 0.69.4[1465] • loki: 2.7.3	Link[1466]	Link[1467]
Prometheus Adapter	prometheus-adapter	4.0.2	• chart: 4.0.2[1468] • prometheus-adapter: 0.10.0	Link[1469]	Link[1470]
Reloader	reloader	1.0.5	• chart: 1.0.5[1471] • reloader: 1.0.5	Link[1472]	Link[1473]
Rook Ceph	rook-ceph	1.10.11	• chart: 1.10.11[1474] • rook-ceph: 1.10.11	Link[1475]	Link[1476]

1462 https://artifacthub.io/packages/helm/grafana/grafana/6.38.1
1463 https://artifacthub.io/packages/helm/grafana/grafana/6.38.1?modal=values
1464 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/project-grafana-logging/6.38.1/defaults/cm.yaml
1465 https://artifacthub.io/packages/helm/grafana/loki-distributed/0.69.4
1466 https://artifacthub.io/packages/helm/grafana/loki-distributed/0.69.4?modal=values
1467 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/project-grafana-loki/0.69.5/defaults/cm.yaml
1468 https://artifacthub.io/packages/helm/prometheus-community/prometheus-adapter/4.0.2
1469 https://artifacthub.io/packages/helm/prometheus-community/prometheus-adapter/4.0.2?modal=values
1470 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/prometheus-adapter/4.0.2/defaults/cm.yaml
1471 https://github.com/stakater/Reloader/tree/v1.0.5/README.md#helm-charts
1472 https://artifacthub.io/packages/helm/stakater/reloader/1.0.5?modal=values
1473 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/reloader/1.0.5/defaults/cm.yaml
1474 https://github.com/rook/rook/tree/v1.10.11/Documentation/Helm-Charts/operator-chart.md
1475 https://raw.githubusercontent.com/rook/rook/v1.10.11/deploy/charts/rook-ceph/values.yaml
1476 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/rook-ceph/1.10.11/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Rook Ceph Cluster	rook-ceph-cluster	1.10.11	• chart: 1.10.11[1477] • rook-ceph: 1.10.11 • rook-ceph-cluster: 17.2.5	Link[1478]	Link[1479]
Thanos	thanos	0.4.8	• chart: 0.4.6[1480] • thanos: 0.17.1	Link[1481]	Link[1482]
Traefik	traefik	20.8.0	• chart: 20.8.0[1483] • traefik: 2.9.6	Link[1484]	Link[1485]
Traefik ForwardAuth	traefik-forward-auth	0.3.10	• chart: 0.3.10[1486] • traefik-forward-auth: 3.1.0	Link[1487]	Link[1488]
Velero	velero	3.4.0	• chart: 3.1.2[1489] • velero: 1.10.0	Link[1490]	Link[1491]

12.3.6 DKP 2.5.2 Known Issues and Limitations

The following items are known issues with this release.

1477 https://github.com/rook/rook/tree/v1.10.11/Documentation/Helm-Charts/ceph-cluster-chart.md
1478 https://raw.githubusercontent.com/rook/rook/v1.10.11/deploy/charts/rook-ceph-cluster/values.yaml
1479 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/rook-ceph-cluster/1.10.11/defaults/cm.yaml
1480 https://artifacthub.io/packages/helm/bitnami/thanos/0.4.6
1481 https://artifacthub.io/packages/helm/bitnami/thanos/0.4.6?modal=values
1482 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/thanos/0.4.8/defaults/cm.yaml
1483 https://artifacthub.io/packages/helm/traefik/traefik/20.8.0
1484 https://artifacthub.io/packages/helm/traefik/traefik/20.8.0?modal=values
1485 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/traefik/20.8.0/defaults/cm.yaml
1486 https://artifacthub.io/packages/helm/mesosphere/traefik-forward-auth/0.3.10
1487 https://artifacthub.io/packages/helm/mesosphere/traefik-forward-auth/0.3.10?modal=values
1488 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/traefik-forward-auth/0.3.10/defaults/cm.yaml
1489 https://artifacthub.io/packages/helm/vmware-tanzu/velero/3.1.2
1490 https://artifacthub.io/packages/helm/vmware-tanzu/velero/3.1.2?modal=values
1491 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.5.2/services/velero/3.4.0/defaults/cm.yaml

12.3.6.1 Use Static Credentials to Provision an Azure Cluster

Only static credentials can be used when provisioning an Azure cluster (see page 1041).

12.3.6.2 Containerd 1.4.13 File Limit Issue

In this version of DKP, we introduced containerd 1.6.17. The systemd unit for containerd 1.6.17 provided upstream removes all file number limits (`LimitNOFILE=infinity`). In our testing, we found that removing these limits broke some IO sensitive applications like Rook Ceph and HAProxy. Because of this, the KIB version included in this release sets the `LimitNOFILE` value in the containerd systemd unit to the value (`1048576`) that was used in previous containerd 1.4.13 version releases.

12.3.6.3 Intermittent Error Status when Creating EKS Clusters in the UI

When provisioning an EKS cluster through the UI, you may receive a brief error state because the EKS cluster may sporadically lose connectivity with the management cluster which results in the following symptoms:

- The UI shows the cluster is in an error state.
- The kubeconfig generated and retrieved from Kommander ceases to work.
- Applications created on the management cluster may not be immediately federated to managed EKS clusters.

After a few moments, the error will resolve, without any action on your part. A new kubeconfig generated and retrieved from Kommander then works properly, and the UI shows that it is working again. In the meantime, you can continue to use the UI to work on the cluster such as deploy applications, create projects, and add roles.

12.3.6.4 Installation Issue in Pre-provisioned Environments

An issue with Rook Ceph's deployment prevents pre-provisioned environments from installing this DKP version. To solve this issue, you must set up a minimum of 40 GB of raw storage for your worker nodes and customize your Rook Ceph installation as indicated in Install Kommander in a Pre-provisioned Environment (see page 1271).

12.3.6.5 Resolve Issues with Failed HelmReleases

An issue with the Flux helm-controller[1492] can cause HelmReleases to fail with the error message *Helm upgrade failed: another operation (install/upgrade/rollback) is in progress*. This can happen when the helm-controller is restarted while a HelmRelease is still upgrading, or installing.

1492 https://github.com/fluxcd/helm-controller/issues/149

12.3.6.5.1 Workaround

To ensure the HelmRelease error was caused by the helm-controller restarting, first try to suspend/resume the HelmRelease:

```
kubectl -n <namespace> patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n <namespace> patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

This might resolve the issue. If not, continue with the following steps:

You should see the HelmRelease attempting to reconcile, and then it either succeeds (with status: *Release reconciliation succeeded*) or it fails with the same error as before.

If the HelmRelease is still in the failed state, it is likely related to the helm-controller restarting. For example, if the 'reloader' HelmRelease is the one that is stuck.

To resolve the issue, follow these steps:

1. List secrets containing the affected HelmRelease name:

   ```
   kubectl get secrets -n ${NAMESPACE} | grep reloader
   ```

   The output should look like this:

   ```
   kommander-reloader-reloader-token-9qd8b                          kubernetes.io/
   service-account-token   3      171m
   sh.helm.release.v1.kommander-reloader.v1                         helm.sh/
   release.v1                     1      171m
   sh.helm.release.v1.kommander-reloader.v2                         helm.sh/
   release.v1                     1      117m
   ```

   In this example, `sh.helm.release.v1.kommander-reloader.v2` is the most recent revision.

2. Find and delete the most recent revision secret, for example, `sh.helm.release.v1.*.<revision>`:

   ```
   kubectl delete secret -n <namespace> <most recent helm revision secret name>
   ```

3. Suspend and resume the HelmRelease to trigger a reconciliation:

   ```
   kubectl -n <namespace> patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[
   {"op": "replace", "path": "/spec/suspend", "value": true}]'
   kubectl -n <namespace> patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[
   {"op": "replace", "path": "/spec/suspend", "value": false}]'
   ```

You should see the HelmRelease is reconciled and eventually the upgrade and install succeeds.

12.3.6.6 Limitations to Disk Resizing in vSphere

The DKP CLI flags `--control-plane-disk-size` and `--worker-disk-size` are unable to resize the root file system of VMs created using OS images. The flags work by resizing the primary disk of the VM. When the VM boots, the root file system is expanded to fill the disk, but that expansion does not work for some file systems, for example, for file systems contained in an LVM Logical Volume. To ensure your root file system has the size you expect, please see https://d2iq.atlassian.net/wiki/spaces/DENT/pages/212993627/2.5%2BCreate%2Ba%2BvSphere%2BBase%2BOS%2BImage#Disk-Size (see page 0).

12.3.6.7 Error Status in Grafana Logging Dashboard with EKS Clusters

Currently, it is not possible to use FluentBit to collect Admin-level logs (see page 798) on a managed EKS cluster.

If you have these logs enabled, the following message appears when you access the Kubernetes Audit Dashboard in the Grafana Logging Dashboard:

```
Cannot read properties of undefined (reading '0')
```

12.3.6.8 Logging Operator Upgrade Error

There is a race condition that could result in the `logging-operator-logging-fluentd` using the incorrect image tag during upgrade from DKP 2.4.0 to DKP 2.5.0.

The image tag is eventually corrected by the `logging-operator`, however due to the nature of StatefulSets, the failing pod needs to be removed in order for the StatefulSet to continue rolling out the required updates.

1. Run this command to find if any Fluentd pods are in the `ImagePullBackOff` state post-upgrade:

   ```
   kubectl get pod -l app.kubernetes.io/name=fluentd,app.kubernetes.io/managed-
   by=logging-operator-logging,app.kubernetes.io/component=fluentd -n kommander
   ```

2. If `ImagePullBackOff` is present in the output like in the example below, you will need to continue with these steps to resolve the issue.

   ```
   NAME                                READY   STATUS             RESTARTS   AGE
   logging-operator-logging-fluentd-0  3/3     Running            0
   6m41s
   logging-operator-logging-fluentd-1  2/3     ImagePullBackOff   0
   2m33s
   ```

3. Delete the Fluentd pod that is in an `ImagePullBackOff` state. In this case, it is `logging-operator-logging-fluentd-1`:

```
kubectl delete pod -n kommander logging-operator-logging-fluentd-1
```

4. The upgrade of the `logging-operator-logging-fluentd` StatefulSet now proceeds as normal.

12.3.6.9 Nodepools Update Error with Knative

The following only applies to your environment if you have Knative installed and if its deployment is scaled to less than 5 pods.
An issue with the `PodDisruptionBudget` resource blocks the deletion of old nodes when upgrading from DKP 2.4.0 to DKP 2.5.0, which results in a failure of the DKP nodepools upgrade.

1. If the `dkp update nodepool` command fails, check to see if `PodDisruptionBudget` with `ALLOWED DISRUPTIONS` equals 0 using the following command:

```
kubectl get pdb -n knative-serving
NAME                            MIN AVAILABLE   MAX UNAVAILABLE   ALLOWED
DISRUPTIONS   AGE
activator-pdb                   80%             N/A               0
22h
webhook-pdb                     80%             N/A               0
22h
```

2. Obtain the list of pods in the knative-serving namespace containing PDB with the following command:

```
kubectl get pods -n knative-serving -l 'app in (webhook, activator)'
```

3. The output should look similar to the following:

```
NAME                          READY   STATUS    RESTARTS   AGE
webhook-XXXXXXXXX-XXXX        2/2     Running   0          5d21h
activator-XXXXXXXXX-XXXX      2/2     Running   0          4d23h
```

4. Delete the pods that contains the `PodDisruptionBudget` resource:

```
kubectl delete pod -n knative-serving activator-XXXX-XXX webhook-XXXXX-XXXX
```

5. The upgrade of DKP and Knative now proceeds as normal. Re-run the `dkp update nodepool` command.

12.3.6.10 Rook Ceph Install Error

An issue may emerge when installing `rook-ceph` on vSphere clusters using RHEL operating systems.

This issue occurs during initial installation of rook-ceph, causing the object store used by Velero and Grafana Loki, to be unavailable. If the installation of Kommander component of DKP is unsuccessful due to `rook-ceph` failing, you might need to apply the following workaround.

1. Run the following command to see if the cluster is affected by this issue.

```
kubectl describe CephObjectStores dkp-object-store -n kommander
```

2. If the following output appears, this workaround needs to be applied:

```
Name:         dkp-object-store
Namespace:    kommander
...
  Warning  ReconcileFailed     7m55s (x19 over 52m)
  rook-ceph-object-controller  failed to reconcile CephObjectStore
  "kommander/dkp-object-store". failed to create object store deployments:
failed
  to configure multisite for object store: failed create ceph multisite for
  object-store ["dkp-object-store"]: failed to commit config changes after
  creating multisite config for CephObjectStore "kommander/dkp-object-store":
  failed to commit RGW configuration period changes%!(EXTRA []string=[]):
signal: interrupt
```

3. Kubectl exec into the `rook-ceph-tools` pod.

```
export WORKSPACE_NAMESPACE=<workspace namespace>
CEPH_TOOLS_POD=$(kubectl get pods -l app=rook-ceph-tools -n $
{WORKSPACE_NAMESPACE} -o name)
kubectl exec -it -n ${WORKSPACE_NAMESPACE} $CEPH_TOOLS_POD bash
```

4. Run the following commands to set `dkp-object-store` as the default zonegroup.
   NOTE: The `period update` command may take a few minutes to complete

```
radosgw-admin zonegroup default --rgw-zonegroup=dkp-object-store
radosgw-admin period update --commit
```

5. Next, restart the `rook-ceph-operator` deployment for the `CephobjectStore` to be reconciled.

```
kubectl rollout restart deploy -n${WORKSPACE_NAMESPACE} rook-ceph-operator
```

6. After running the commands above, the `CephObjectStore` should be `Connected` once the `rook-ceph` operator reconciles the object (this may take some time).

```
kubectl wait CephObjectStore --for=jsonpath='{.status.phase}'=Connected dkp-
object-store -n ${WORKSPACE_NAMESPACE} --timeout 10m
```

12.3.6.11 Post Upgrade, Volume Cannot Attach to a Node Already Attached to Another Node

Due to an upstream issue[1493], when you bring a new node up during a Kubernetes version upgrade and then delete the old node, an existing volume might not attach to the new node.

You will see this when a new pod that uses a volume does not become ready in the new node, and then an event that says something such as `Volume <pvc/pv-id> is already exclusively attached to one node and can't be attached to another`.

This will be fixed[1494] in a future Kubernetes release[1495], for example this is described in vSphere here[1496]. Different methods might be needed to resolve this manually[1497], including this method to resolve on vSphere[1498].

12.3.6.12 Control Plane Upgrade - Oracle Linux node-feature-discovery Fails

When upgrading the Oracle Linux node-feature-discovery fails to terminate, so the control-plane upgrade process can freeze and fail after the time out.

Follow these steps to avoid this issue:

1. View the machines for your cluster:

```
kubectl get machines
```

2. Identify the machine that has the faulty `node-feature-discovery` pod, and then delete that machine:

1493 https://discuss.kubernetes.io/t/multi-attach-error-for-volume-pvc-volume-is-already-exclusively-attached-to-one-node-and-cant-be-attached-to-another/18951
1494 https://github.com/kubernetes/enhancements/pull/1116
1495 https://kubernetes.io/docs/reference/command-line-tools-reference/feature-gates/?highlight=Feature#:~:text=1.27-,NodeOutOfServiceVolumeDetach,-false
1496 https://github.com/kubernetes-sigs/vsphere-csi-driver/issues/125
1497 https://discuss.kubernetes.io/t/multi-attach-error-for-volume-pvc-volume-is-already-exclusively-attached-to-one-node-and-cant-be-attached-to-another/18951/3
1498 https://kb.vmware.com/s/article/70519

```
kubectl delete machine <faulty-machine-name-here>
```

A replacement machine generates and reconciles on your cluster.

12.3.7 DKP 2.5.2 Customer Incidents

The following resolved incidents are included in this release:

12.3.7.1 LOG_LEVEL Hardcoded in the traefik-forward-auth Chart

D2IQ-98651

The chart for the traefik-forward-auth service hardcoded the LOG_LEVEL environment variable to TRACE, which prevented users from setting a custom LOG_LEVEL. The chart is corrected and now honors customized LOG_LEVEL settings.

12.3.7.2 Duplicate apiserver scrape targets in Prometheus

D2IQ-98647

The default values in the kube-prometheus-stack chart included a duplicate scrape target for the Kubernetes API server. The duplicate scrape target is removed.

12.3.7.3 Disappearing text in GitOps source forms

D2IQ-98291

In the Kommander UI dashboard, when viewing *Projects* > *Continuous Deployment* > *GitOps Source* form, the value of the field *Path* disappeared after leaving the text box. The web form is now corrected and the *Path* field remains viewable.

12.3.7.4 Kommander UI Navigation Issues

D2IQ-97403

The Kommander UI dashboard could not navigate to an attached cluster when the cluster name had capitalized letters. This issue is corrected.

12.3.7.5 Unable to disable installation of Gatekeeper

D2IQ-97260

After uninstalling or deactivating Gatekeeper in a workspace and then creating a new cluster, Gatekeeper was continuing to deploy in the workspace. This issue is corrected and Gatekeeper remains uninstalled from new cluster workspaces.

12.3.7.6 Read-only View of the Kommander UI enabled

D2IQ-98522

For users without access to perform actions, the Kommander Dashboard UI can now be viewed in read-only status.

12.3.7.7 Adding Custom Prometheus Jobs Removes some Default Jobs

D2IQ-98044

When adding additional jobs to Prometheus, some default jobs were being removed. This issue is corrected.

13 Access Documentation

The following sections describe how to access other versions of documentation:

13.1 Supported Documentation

You can access all n-2 supported documentation at D2iQ Help Center[1499].

13.2 Archived Documentation

In accordance with our version support policy[15001501], we regularly archive older, unsupported versions of our documentation. At this time, this includes documentation for:

- DKP and DKP Insights 2.3 documentation at D2iQ Documentation Archive[1502].
- Konvoy, Kommander, Kaptain, DC/OS and Dispatch at https://github.com/mesosphere/dcos-docs-site/tags.

1499 http://docs.d2iq.com/
1500 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29923883/Version+Support+Policy
1501 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29923883/Version+Support+Policy
1502 https://archive-docs.d2iq.com/

14 Download Documentation

This PDF contains the entire DKP documentation space. The file name contains the version and date created.

(see page 1600)

15 Version support policy

D2iQ® supports N-2 of the latest `MAJOR.MINOR` version of DKP. For example, if the current GA version of DKP® is 2.5, then D2iQ supports all patch versions of DKP 2.4, and 2.3.

When the next version releases, support continues for 2.5, and 2.4. Support for DKP version 2.3.x expires. You should upgrade DKP with every new release to stay up-to-date with the latest features and bug fixes.

15.1 Supported Kubernetes Versions

Each DKP release supports a range of Kubernetes versions. Details for supported Kubernetes versions on DKP can be found in the Release Notes (see page 1537) on the DKP 2.5.0 Supported Kubernetes Versions (see page 1543) page.

15.2 Supported Operating Systems

Details for supported operating systems on DKP can be found in Supported Operating Systems (see page 62).

15.3 Features in Patches

Occasionally, to make new features available at a faster rate, D2iQ releases features as part of a patch release. If the Release Notes indicate a feature you need and do not yet have, consider upgrading to the latest version to take full advantage of new features and functions.

15.4 Experimental Status

“Experimental” means software, features, functionality, sample configurations, or other speculative content that is still under exploration, development, or testing by D2iQ. Experimental components carry no guarantee of eventual release as GA and therefore must not be used in Production Environments. Experimental components qualify for limited, Severity 4 support only and may be discontinued at any time, with or without notice.

Since Experimental components are not intended for Production Environment use, D2iQ cannot assume any responsibility for errors occurring during their use in Production. We can offer only these services in a commercially-reasonable manner, based on the availability of relevant subject matter experts (SMEs):

- General operational guidance for the Experimental component.
- Identifying and diagnosing of errors in configuration or implementation, if possible.
- Advice on preventing and recovering from failures and troubleshooting, as available.

Support for Experimental components is provided on a Standard level, Severity 4 basis only.

This software is provided “as is” and without any express or implied warranties including, without limitation, the implied warranties of merchantability and fitness for a particular purpose.

15.5 Technical Preview

We provide Technical Preview, or Tech Preview, features to showcase capabilities that might be added to future versions of the product. As they are not yet production-ready, the support terms are the same as defined for experimental features. Technical Preview features are not guaranteed to move forward, so could be removed from future versions of the product.

15.6 Support Definitions

15.6.1 Secondary Support

The following section describes D2iQ’s support for secondary applications, such as platform applications. All platform applications that D2iQ ships with DKP products are covered under secondary support:

Type	Scope Example	Support Offered
Configuration	• Guidance for base technology and DKP interoperability configuration questions and troubleshooting for different components of the DKP platform. • No support for base technology’s configuration that is unrelated to its integration with DKP. • No support for performance issues with the base technology that is unrelated to its integration with DKP.	Supported with severity 3 & 4 support terms

Type	Scope Example	Support Offered
Failure Assistance	• Assistance with installation, and upgrade failures of the service as captured in the supported DKP product upgrade pathway. • Assistance with service failures due to platform issues. For example: DKP Enterprise or DKP Essential • Support is limited to troubleshooting for root cause up to DKP product limit. Root causes that are identified to be beyond this limit will need to be pursued by the company who creates the platform application base technology. Note, if the RCA for the failure is due to a non-standard configuration or non-DKP use of the platform application, we will be unable to provide assistance beyond basic identification. • No assistance for base technology's failures that is unrelated to its integration with DKP.	Supported with all severities
Bug Fixes	• Bug fixes for service integration with DKP. • Upstream bug fixes to identified issues in the base technology of the platform application on a best effort basis. • No guarantees that our changes to upstream will be accepted. • No commitment to maintaining forks upstream.	RCA supported with all severities, Fix supported with severity 3 & 4 support terms

Type	Scope Example	Support Offered
Documentation errors	• Documentation fixes for life cycle management of services and integration with DKP. • Upstream documentation fixes to reported and identified issues in base technology of the platform application via a best effort basis. • No guarantees that our changes will be accepted.	Supported with severity 4 support terms

15.6.2 Standard Level & Severity Definitions

To view our severity level and support terms refer to D2iQ Support and Maintenance Terms[1503].

1503 https://d2iq.com/legal/support-terms

16 Legal Notices

D2iQ® and its licensors are the owners of all right, title, and interest in and to D2iQ software products, the documentation, all updates, upgrades, and derivative works thereto, and all intellectual property rights therein. D2iQ software products may additionally include third-party open source software.

See the D2iQ Legal page[1504] for additional details.

1504 https://d2iq.com/legal/3rd