Oracle Cloud Native Environment

Container Orchestration for Release 1.6

F75589-08
January 2024

Oracle Cloud Native Environment Container Orchestration for Release 1.6,

F75589-08

Copyright © 2022, 2024, Oracle and/or its affiliates.

Contents

Preface

This book describes how to use Kubernetes, which is an implementation of the open-source, containerized application management platform from the upstream Kubernetes release. Oracle provides additional tools, testing and support to deliver this technology with confidence. Kubernetes integrates with container products to handle more complex deployments where clustering may be used to improve the scalability, performance and availability of containerized applications. Detail is provided on the advanced features of Kubernetes and how it can be installed, configured and used as a component of Oracle Cloud Native Environment.

This document describes functionality and usage available in the most current release of the product.

Documentation License

The content in this document is licensed under the Creative Commons Attribution–Share Alike 4.0 (CC-BY-SA) license. In accordance with CC-BY-SA, if you distribute this content or an adaptation of it, you must provide attribution to Oracle and retain the original copyright notices.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

For information about the accessibility of the Oracle Help Center, see the Oracle Accessibility Conformance Report at https://www.oracle.com/corporate/accessibility/templates/t2-11535.html.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1

Introduction to Kubernetes

Kubernetes is an open-source system for automating the deployment, scaling and management of containerized applications. Primarily, Kubernetes provides the tools to easily create a cluster of systems across which containerized applications can be deployed and scaled as required.

The Kubernetes project is maintained at:

https://kubernetes.io/

Kubernetes is fully tested on Oracle Linux 8 and Oracle Linux 7 and includes additional tools developed at Oracle to ease configuration and deployment of a Kubernetes cluster.

For more information on Kubernetes releases, hardware and software requirements, new and notable features, and known issues, see Release Notes.

Kubernetes Components

You are likely to encounter the following common components when you start working with Kubernetes on Oracle Linux. The descriptions provided are brief, and largely intended to help provide a glossary of terms and an overview of the architecture of a typical Kubernetes environment. Upstream documentation can be found at:

https://kubernetes.io/docs/concepts/

Nodes

Kubernetes Node architecture is described in detail at:

https://kubernetes.io/docs/concepts/architecture/nodes/

Control Plane Node

The control plane node is responsible for cluster management and for providing the API that is used to configure and manage resources within the Kubernetes cluster. Kubernetes control plane node components can be run within Kubernetes itself, as a set of containers within a dedicated pod. These components can be replicated to provide highly available (HA) control plane node functionality.

The following components are required for a control plane node:

- **API Server** (`kube-apiserver`): The Kubernetes REST API is exposed by the API Server. This component processes and validates operations and then updates information in the Cluster State Store to trigger operations on the worker nodes. The API is also the gateway to the cluster.
- **Cluster State Store** (`etcd`): Configuration data relating to the cluster state is stored in the Cluster State Store, which can roll out changes to the coordinating components like the Controller Manager and the Scheduler. It is essential to have a backup plan in place for the data stored in this component of your cluster.

- **Cluster Controller Manager** (`kube-controller-manager`): This manager is used to perform many of the cluster-level functions, as well as application management, based on input from the Cluster State Store and the API Server.
- **Scheduler** (`kube-scheduler`): The Scheduler handles automatically determining where containers should be run by monitoring availability of resources, quality of service and affinity and anti-affinity specifications.

The control plane node is also usually configured as a worker node within the cluster. Therefore, the control plane node also runs the standard node services: the kubelet service, the container runtime and the kube proxy service. Note that it is possible to taint a node to prevent workloads from running on an inappropriate node. The `kubeadm` utility automatically taints the control plane node so that no other workloads or containers can run on this node. This helps to ensure that the control plane node is never placed under any unnecessary load and that backup and restore of the control plane node for the cluster is simplified.

If the control plane node becomes unavailable for a period, cluster functionality is suspended, but the worker nodes continue to run container applications without interruption.

For single node clusters, when the control plane node is offline, the API is unavailable, so the environment is unable to respond to node failures and there is no way to perform new operations like creating new resources or editing or moving existing resources.

A high availability cluster with multiple control plane nodes ensures that more requests for control plane node functionality can be handled, and with the assistance of control plane replica nodes, uptime is significantly improved.

Control Plane Replica Nodes

Control plane replica nodes are responsible for duplicating the functionality and data contained on control plane nodes within a Kubernetes cluster configured for high availability. To benefit from increased uptime and resilience, you can host control plane replica nodes in different zones, and configure them to load balance for your Kubernetes cluster.

Replica nodes are designed to mirror the control plane node configuration and the current cluster state in real time so that if the control plane nodes become unavailable the Kubernetes cluster can fail over to the replica nodes automatically whenever they are needed. In the event that a control plane node fails, the API continues to be available, the cluster can respond automatically to other node failures and you can still perform regular operations for creating and editing existing resources within the cluster.

Worker Nodes

Worker nodes within the Kubernetes cluster are 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. The worker nodes perform any actions triggered via the Kubernetes API, which runs on the control plane node.

All nodes within a Kubernetes cluster must run the following services:

- **Kubelet Service**: The agent that allows each worker node to communicate with the API Server running on the control plane node. This agent is also responsible

for setting up pod requirements, such as mounting volumes, starting containers and reporting status.

- **Container Runtime**: An environment where containers can be run. In this release, the container runtimes are either runC or Kata Containers. For more information about the container runtimes, see Container Runtimes.
- **Kube Proxy Service**: A service that programs rules to handle port forwarding and IP redirects to ensure that network traffic from outside the pod network can be transparently proxied to the pods in a service.

In all cases, these services are run from `systemd` as inter-dependent daemons.

Pods

Kubernetes introduces the concept of "pods", which are groupings of one or more containers and their shared storage, and any specific options on how these should be run together. Pods are used for tightly coupled applications that would typically run on the same logical host and which may require access to the same system resources. Typically, containers in a pod share the same network and memory space and can access shared volumes for storage. These shared resources allow the containers in a pod to communicate internally in a seamless way as if they were installed on a single logical host.

You can easily create or destroy pods as a set of containers. This makes it possible to do rolling updates to an application by controlling the scaling of the deployment. It also allows you to scale up or down easily by creating or removing replica pods. For more information on pods, see the upstream documentation at:

https://kubernetes.io/docs/concepts/workloads/pods/

ReplicaSet, Deployment, StatefulSet Controllers

Kubernetes provides a variety of controllers that you can use to define how pods are set up and deployed within the Kubernetes cluster. These controllers can be used to group pods together according to their runtime needs and define pod replication and pod start up ordering.

You can define a set of pods that should be replicated with a *ReplicaSet*. This allows you to define the exact configuration for each of the pods in the group and which resources they should have access to. Using ReplicaSets not only caters to the easy scaling and rescheduling of an application, but also allows you to perform rolling or multi track updates to an application. For more information on ReplicaSets, see the upstream documentation at:

https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/

You can use a *Deployment* to manage pods and *ReplicaSet*s. *Deployment*s are useful when you need to roll out changes to ReplicaSets. By using a *Deployment* to manage a *ReplicaSet*, you can easily rollback to an earlier *Deployment* revision. A *Deployment* allows you to create a newer revision of a *ReplicaSet* and then migrate existing pods from a previous *ReplicaSet* into the new revision. The *Deployment* can then manage the cleanup of older unused *ReplicaSet*s. For more information on Deployments, see the upstream documentation at:

https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

You can use *StatefulSet*s to create pods that guarantee start up order and unique identifiers, which are then used to ensure that the pod maintains its identity across the lifecycle of the *StatefulSet*. This feature makes it possible to run stateful applications within Kubernetes, as typical persistent components such as storage and networking are guaranteed. Furthermore,

when you create pods they are always created in the same order and allocated identifiers that are applied to host names and the internal cluster DNS. Those identifiers ensure there are stable and predictable network identities for pods in the environment. For more information on StatefulSets, see the upstream documentation at:

https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

Services

You can use services to expose access to one or more mutually interchangeable pods. Since pods can be replicated for rolling updates and for scalability, clients accessing an application must be directed to a pod running the correct application. Pods may also need access to applications outside of Kubernetes. In either case, you can define a service to make access to these facilities transparent, even if the actual backend changes.

Typically, services consist of port and IP mappings. How services function in network space is defined by the service type when it is created.

The default service type is the `ClusterIP`, and you can use this to expose the service on the internal IP of the cluster. This option makes the service only reachable from within the cluster. Therefore, you should use this option to expose services for applications that need to be able to access each other from within the cluster.

Frequently, clients outside of the Kubernetes cluster may need access to services within the cluster. You can achieve this by creating a `NodePort` service type. This service type enables you to take advantage of the *Kube Proxy* service that runs on every worker node and reroute traffic to a `ClusterIP`, which is created automatically along with the `NodePort` service. The service is exposed on each node IP at a static port, called the `NodePort`. The Kube Proxy routes traffic destined to the `NodePort` into the cluster to be serviced by a pod running inside the cluster. This means that if a `NodePort` service is running in the cluster, it can be accessed via any node in the cluster, regardless of where the pod is running.

Building on top of these service types, the `LoadBalancer` service type makes it possible for you to expose the service externally by using a cloud provider's load balancer. This allows an external load balancer to handle redirecting traffic to pods directly in the cluster via the Kube Proxy. A `NodePort` service and a `ClusterIP` service are automatically created when you set up the `LoadBalancer` service.

> **Important:**
>
> As you add services for different pods, you must ensure that your network is properly configured to allow traffic to flow for each service declaration. If you create a `NodePort` or `LoadBalancer` service, any of the ports exposed must also be accessible through any firewalls that are in place.
>
> If you are running `firewalld` on any of your nodes, make sure you add rules to allow traffic for the external facing ports of the services that you create.

For more information on services, see the upstream documentation at:

https://kubernetes.io/docs/concepts/services-networking/service/

Volumes

In Kubernetes, a *volume* is storage that persists across the containers within a pod for the lifespan of the pod itself. When a container within the pod is restarted, the data in the Kubernetes volume is preserved. Furthermore, Kubernetes volumes can be shared across containers within the pod, providing a file store that different containers can access locally.

Kubernetes supports a variety of volume types that define how the data is stored and how persistent it is, which are described in detail in the upstream documentation at:

https://kubernetes.io/docs/concepts/storage/volumes/

Kubernetes volumes typically have a lifetime that matches the lifetime of the pod, and data in a volume persists for as long as the pod using that volume exists. Containers can be restarted within the pod, but the data remains persistent. If the pod is destroyed, the data is usually destroyed with it.

In some cases, you may require even more persistence to ensure the lifecycle of the volume is decoupled from the lifecycle of the pod. Kubernetes introduces the concepts of the *PersistentVolume* and the *PersistentVolumeClaim*. *PersistentVolume*s are similar to *Volume*s except that they exist independently of a pod. They define how to access a storage resource type, such as NFS or iSCSI. You can configure a *PersistentVolumeClaim* to make use of the resources available in a *PersistentVolume*, and the *PersistentVolumeClaim* will specify the quota and access modes that should be applied to the resource for a consumer. A pod you have created can then make use of the *PersistentVolumeClaim* to gain access to these resources with the appropriate access modes and size restrictions applied.

For more information about volumes and setting up and using persistent storage with Kubernetes applications, see Storage.

Namespaces

Kubernetes implements and maintains strong separation of resources through the use of namespaces. Namespaces effectively run as virtual clusters backed by the same physical cluster and are intended for use in environments where Kubernetes resources must be shared across use cases.

Kubernetes takes advantage of namespaces to separate cluster management and specific Kubernetes controls from any other user-specific configuration. Therefore, all of the pods and services specific to the Kubernetes system are found within the `kube-system` namespace. A `default` namespace is also created to run all other deployments for which no namespace has been set.

For more information on namespaces, see the upstream documentation at:

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

About CRI-O

When you deploy Kubernetes worker nodes, CRI-O is also deployed. CRI-O is an implementation of the Kubernetes Container Runtime Interface (CRI) to enable using Open Container Initiative (OCI) compatible runtimes. It is a lightweight alternative to using Docker as the runtime for Kubernetes. CRI-O allows Kubernetes to use any OCI-compliant runtime as the container runtime for running pods.

CRI-O delegates containers to run on appropriate nodes, based on the configuration set in pod files. *Privileged* pods can be run using the runC runtime engine (`runc`), and *unprivileged* pods can be run using the Kata Containers runtime engine (`kata-runtime`). Defining whether containers are trusted or untrusted is set in the Kubernetes pod or deployment file.

For information on how to set the container runtime, see Container Runtimes.

2
Creating a Kubernetes Cluster

This chapter shows you how to use the Platform CLI (`olcnectl`) to create a Kubernetes cluster. This chapter assumes you have installed the Oracle Cloud Native Environment software packages on the nodes, configured them to be used in a cluster and created an environment in which to install the Kubernetes module, as discussed in Getting Started.

The high level steps to create a Kubernetes cluster are:

- Create a Kubernetes module to specify information about the cluster.
- Validate the Kubernetes module to make sure Kubernetes can be installed on the nodes.
- Install the Kubernetes module to install the Kubernetes packages on the nodes and create the cluster.

The `olcnectl` command is used to perform these steps. For more information on the syntax for the `olcnectl` command, see Platform Command-Line Interface.

> **Tip:**
>
> You can also use a configuration file to create modules. The configuration file is a YAML file that contains the information about the environments and modules you want to deploy. Using a configuration file reduces the information you need to provide with `olcnectl` commands. For information on creating and using a configuration file, see Platform Command-Line Interface.

Setting the Kubernetes Pod Network

You can use the following pod networking technologies with your Kubernetes cluster:

- **Flannel**. The default networking option when you create a Kubernetes module.
- **Calico**. Calico can be set up when you create the Kubernetes module, or afterwards as the Calico module, using your own configuration.
- **Multus**. Multus can be set up as the Multus module after the Kubernetes module is installed. Multus is installed as a module on top of either Flannel or Calico.

Flannel Networking

Flannel is the default networking option when you create a Kubernetes module. You do not need to set any command options to use Flannel, it is installed by default.

Calico Networking

Calico is an optional pod networking technology. For more information on Calico, see the upstream documentation.

Prerequisites

This section contains the prerequisite information you need to set up Calico.

Disabling firewalld Service

To use Calico, you must disable the **firewalld** service on each Kubernetes node. To disable the **firewalld** service, enter:

```
sudo systemctl stop firewalld.service
sudo systemctl disable firewalld.service
```

Updating Proxy Configuration

If you are using a proxy server in your environment, edit the CRI-O proxy configuration file and add the Kubernetes service IP (the default is `10.96.0.1`) to the `NO_PROXY` variable, for example, on each Kubernetes node, edit the `/etc/systemd/system/crio.service.d/proxy.conf` file:

```
[Service]
Environment="HTTP_PROXY=http://proxy.example.com:3128"
Environment="HTTPS_PROXY=https://proxy.example.com:3128"
Environment="NO_PROXY=mydomain.example.com,10.96.0.1"
```

Reload the configuration file and restart the **crio** service:

```
sudo systemctl daemon-reload
sudo systemctl restart crio.service
```

> **Note:**
>
> You don't need to perform this step if you're using the `olcnectl provision` command to perform a quick installation. This is set up for you automatically when using that installation method and you provide any proxy information.

Creating a Calico Configuration File

If you are installing the Calico module, you should create a Calico configuration file to configure Calico to your requirements. This file should contain the `spec` portion of an `operator.tigera.io/v1/Installation`. This file should be available on the operator node.

For information on the Calico configuration file, see the upstream documentation.

An example Calico configuration file is:

```
installation:
  cni:
    type: Calico
  calicoNetwork:
```

```
    bgp: Disabled
    ipPools:
    - cidr: 198.51.100.0/24
      encapsulation: VXLAN
  registry: container-registry.oracle.com
  imagePath: olcne
```

Deploying Calico with the Kubernetes Module

If you want to use the default configuration for Calico, you can set this as an option when you create the Kubernetes module using the `--pod-network calico` option of the `olcnectl module create --module kubernetes` command. This sets Calico as the networking option instead of Flannel.

A minimum default configuration is used for Calico with this installation method. If you want to configure Calico, you should install the Calico module.

Deploying the Calico Module

You can optionally install the Calico module. This allows you to use your own configuration file for Calico.

To use this method, you must create the Kubernetes module using the `--pod-network none` option. This option sets no networking for pods in the Kubernetes cluster. You then install the Calico module to configure the pod networking.

For the syntax to use to create a Calico module, see the `calico` option of the `olcnectl module create` command in Platform Command-Line Interface.

To deploy the Calico module:

1. Create and install a Kubernetes module using the `--pod-network none` option of the `olcnectl module create --module kubernetes` command. This option sets no networking for pods in the Kubernetes cluster. The name of the Kubernetes module in this example is `mycluster`.

   > **Note:**
   >
   > When you install the Kubernetes module with the `--pod-network none` option, all `kube-system` pods are marked as `pending` until you install the Calico module. When Calico is installed, these `kube-system` pods are marked as `running`.

2. Create a Calico module and associate it with the Kubernetes module named `mycluster` using the `--calico-kubernetes-module` option. In this example, the Calico module is named `mycalico`.

   ```
   olcnectl module create \
   --environment-name myenvironment \
   --module calico \
   --name mycalico \
   ```

```
--calico-kubernetes-module mycluster \
--calico-installation-config calico-config.yaml
```

The `--module` option sets the module type to create, which is `calico`. You define the name of the Calico module using the `--name` option, which in this case is `mycalico`.

The `--calico-kubernetes-module` option sets the name of the Kubernetes module.

The `--calico-installation-config` option sets the location for the Calico configuration file. This file must be available on the operator node under the provided path. For information on creating this configuration file, see Creating a Calico Configuration File.

If you do not include all the required options when adding the module, you are prompted to provide them.

3. Use the `olcnectl module validate` command to validate the Calico module can be deployed to the nodes. For example:

```
olcnectl module validate \
--environment-name myenvironment \
--name mycalico
```

4. Use the `olcnectl module install` command to install the Calico module. For example:

```
olcnectl module install \
--environment-name myenvironment \
--name mycalico
```

The Calico module is deployed into the Kubernetes cluster.

Multus Networking

Multus is an optional pod networking technology that creates a networking bridge to either Flannel or Calico. For more information on Multus, see the upstream documentation.

You can install Multus as a module to create a network bridge to Flannel or Calico. You can create a Multus module using the default configuration, or write a configuration file to suit your own requirements.

Prerequisites

This section contains the prerequisite information you need to set up Multus.

Updating Proxy Configuration

If you are using a proxy server in your environment, edit the CRI-O proxy configuration file and add the Kubernetes service IP (the default is `10.96.0.1`) to the `NO_PROXY` variable, for example, on each Kubernetes node, edit the `/etc/systemd/system/crio.service.d/proxy.conf` file:

```
[Service]
Environment="HTTP_PROXY=http://proxy.example.com:3128"
```

```
Environment="HTTPS_PROXY=https://proxy.example.com:3128"
Environment="NO_PROXY=mydomain.example.com,10.96.0.1"
```

Reload the configuration file and restart the **crio** service:

```
sudo systemctl daemon-reload
sudo systemctl restart crio.service
```

> **Note:**
>
> You don't need to perform this step if you're using the `olcnectl provision` command to perform a quick installation. This is set up for you automatically when using that installation method and you provide any proxy information.

Creating a Multus Configuration File

If you are installing the Multus module, it is recommended you create a configuration file to set up the networking to suit your requirements. The default configuration of Multus is not recommended for a production environment.

The configuration file should contain zero or more Kubernetes `NetworkAttachmentDefinition`s Custom Resource Definitions. These definitions set up the network attachment, that is, the secondary interface for the pods. This file should be available on the operator node.

For information on creating the Multus configuration file, see the upstream documentation.

An example Multus configuration file is:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bridge-conf
spec:
  config: '{
      "cniVersion": "0.3.1",
      "type": "bridge",
      "bridge": "mybr0",
      "ipam": {
          "type": "host-local",
          "subnet": "192.168.12.0/24",
          "rangeStart": "192.168.12.10",
          "rangeEnd": "192.168.12.200"
      }
    }'
```

Deploying the Multus Module

This section contains the information on how to install the Multus module. You must have a Kubernetes module installed before you install Multus. The Kubernetes module can use either Flannel or Calico as the Kubernetes pod networking technology.

For the syntax to use to create a Multus module, see the `multus` option of the `olcnectl module create` command in Platform Command-Line Interface.

To deploy the Multus module:

1. Create and install a Kubernetes module using either Flannel or Calico. The name of the Kubernetes module in this example is `mycluster`.
2. Create a Multus module and associate it with the Kubernetes module named `mycluster` using the `--multus-kubernetes-module` option. In this example, the Multus module is named `mymultus`.

   ```
   olcnectl module create \
   --environment-name myenvironment \
   --module multus \
   --name mymultus \
   --multus-kubernetes-module mycluster \
   --multus-config multus-config.conf
   ```

   The `--module` option sets the module type to create, which is `multus`. You define the name of the Multus module using the `--name` option, which in this case is `mymultus`.

   The `--multus-kubernetes-module` option sets the name of the Kubernetes module.

   The `--multus-config` option sets the location for the Multus configuration file. This file must be available on the operator node under the provided path. For information on creating this configuration file, see Creating a Multus Configuration File.

   If you do not include all the required options when adding the module, you are prompted to provide them.
3. Use the `olcnectl module validate` command to validate the Multus module can be deployed to the nodes. For example:

   ```
   olcnectl module validate \
   --environment-name myenvironment \
   --name mymultus
   ```
4. Use the `olcnectl module install` command to install the Multus module. For example:

   ```
   olcnectl module install \
   --environment-name myenvironment \
   --name mymultus
   ```

   The Multus module is deployed into the Kubernetes cluster.

Creating a Kubernetes Module

The Kubernetes module can be set up to create a:

- Highly available (HA) cluster with an external load balancer
- HA cluster with an internal load balancer
- Cluster with a single control plane node (non-HA cluster)

To create an HA cluster you need at least three control plane nodes and two worker nodes.

For information on setting up an external load balancer, or for information on preparing the control plane nodes to use the internal load balancer installed by the Platform CLI, see Getting Started.

A number of additional ports are required to be open on control plane nodes in an HA cluster. For information on opening the required ports for an HA cluster, see Getting Started.

Use the `olcne module create` command to create a Kubernetes module. If you do not include all the required options when using this command, you are prompted to provide them. For the full list of the options available for the Kubernetes module, see Platform Command-Line Interface.

Creating an HA Cluster with External Load Balancer

This section shows you how to create a Kubernetes module to create an HA cluster using an external load balancer.

The following example creates an HA cluster using your own load balancer, available on the host `lb.example.com` and running on port `6443`.

```
olcnectl module create \
--environment-name myenvironment \
--module kubernetes \
--name mycluster \
--container-registry container-registry.oracle.com/olcne \
--load-balancer lb.example.com:6443 \
--control-plane-nodes
control1.example.com:8090,control2.example.com:8090,control3.example.com:8090 \
--worker-nodes
worker1.example.com:8090,worker2.example.com:8090,worker3.example.com:8090,worker4.exam
ple.com:8090 \
--selinux enforcing \
--restrict-service-externalip-ca-cert /etc/olcne/certificates/restrict_external_ip/
ca.cert \
--restrict-service-externalip-tls-cert /etc/olcne/certificates/restrict_external_ip/
node.cert \
--restrict-service-externalip-tls-key /etc/olcne/certificates/restrict_external_ip/
node.key
```

The `--environment-name` sets the name of the environment in which to create the Kubernetes module. This example sets it to `myenvironment`.

The `--module` option sets the module type to create. To create a Kubernetes module this must be set to `kubernetes`.

The `--name` option sets the name used to identify the Kubernetes module. This example sets it to `mycluster`.

The `--container-registry` option specifies the container registry from which to pull the Kubernetes images. This example uses the Oracle Container Registry, but you may also use an Oracle Container Registry mirror, or a local registry with the Kubernetes images mirrored from the Oracle Container Registry. For information on using an Oracle Container Registry mirror, or creating a local registry, see Getting Started.

However, you can set a new default container registry value during an update or upgrade of the Kubernetes module.

The `--load-balancer` option sets the hostname and port of an external load balancer. This example sets it to `lb.example.com:6443`.

The `--control-plane-nodes` option includes a comma separated list of the hostnames or IP addresses of the control plane nodes to be included in the cluster and the port number on which the Platform Agent is available. The default port number is `8090`.

> **Note:**
>
> You can create a cluster that uses an external load balancer with a single control plane node. However, HA and failover features are not available until you reach at least three control plane nodes in the cluster. To increase the number of control plane nodes, scale up the cluster. For information on scaling up the cluster, see Scaling Up a Kubernetes Cluster.

The `--worker-nodes` option includes a comma separated list of the hostnames or IP addresses of the worker nodes to be included in the cluster and the port number on which the Platform Agent is available. If a worker node is behind a NAT gateway, use the public IP address for the node. The worker node's interface behind the NAT gateway must have an public IP address using the /32 subnet mask that is reachable by the Kubernetes cluster. The /32 subnet restricts the subnet to one IP address, so that all traffic from the Kubernetes cluster flows through this public IP address (for more information about configuring NAT, see Getting Started). The default port number is `8090`.

If SELinux is set to `enforcing` mode (the operating system default and the recommended mode) on the control plane node and worker nodes, you must also use the `--selinux enforcing` option when you create the Kubernetes module.

You must also include the location of the certificates for the `externalip-validation-webhook-service` Kubernetes service. These certificates must be located on the operator node. The `--restrict-service-externalip-ca-cert` option sets the location of the CA certificate. The `--restrict-service-externalip-tls-cert` sets the location of the node certificate. The `--restrict-service-externalip-tls-key` option sets the location of the node key. For information on setting up these certificates, see Getting Started.

You can optionally use the `--restrict-service-externalip-cidrs` option to set the external IP addresses that can be accessed by Kubernetes services. For example:

```
--restrict-service-externalip-cidrs 192.0.2.0/24,198.51.100.0/24
```

In this example, the IP ranges that are allowed are within the `192.0.2.0/24` and `198.51.100.0/24` CIDR blocks.

The default pod networking uses Flannel. You can optionally set the pod networking technology to Calico or to none. Set the pod networking using the `--pod-network` option. Using `--pod-network calico` sets Calico to be the CNI for pods instead of Flannel. Using `--pod-network none` sets no CNI, which allows you to use the Calico module to install Calico with a configuration file that suits your pod networking requirements. For more information on pod networking options, see Setting the Kubernetes Pod Network.

You can optionally set the network interface to use for the Kubernetes data plane (the interface used by the pods running on Kubernetes). By default, the interface used by the the Platform Agent (set with the `--control-plane-nodes` and `--worker-nodes` options) is used for both the Kubernetes control plane node and the data plane. If you want to specify a separate network interface to use for the data plane, include the `--pod-network-iface` option. For example, `--pod-network-iface ens1`. This results in the control plane node using the network interface used by the Platform Agent, and the data plane using a separate network interface, which in this example is `ens1`.

> **Note:**
>
> You can also use a regex expression with the `--pod-network-iface` option. For example:
>
> `--pod-network-iface "ens[1-5]|eth5"`
>
> If you use regex to set the interface name, the first matching interface returned by the kernel is used.

Creating an HA Cluster with Internal Load Balancer

This section shows you how to create a Kubernetes module to create an HA cluster using an internal load balancer, installed by the Platform CLI on the control plane nodes.

This example creates an HA cluster using the internal load balancer installed by the Platform CLI.

```
olcnectl module create \
--environment-name myenvironment \
--module kubernetes \
--name mycluster \
--container-registry container-registry.oracle.com/olcne \
--virtual-ip 192.0.2.100 \
--control-plane-nodes
control1.example.com:8090,control2.example.com:8090,control3.example.com:8090 \
--worker-nodes
worker1.example.com:8090,worker2.example.com:8090,worker3.example.com:8090,worker4.exam
ple.com:8090 \
--selinux enforcing \
--restrict-service-externalip-ca-cert /etc/olcne/certificates/restrict_external_ip/
ca.cert \
--restrict-service-externalip-tls-cert /etc/olcne/certificates/restrict_external_ip/
node.cert \
--restrict-service-externalip-tls-key /etc/olcne/certificates/restrict_external_ip/
node.key
```

The `--virtual-ip` option sets the virtual IP address to be used for the primary control plane node, for example, `192.0.2.100`. This IP address should be available on the network and should not be assigned to any hosts on the network. This IP address is dynamically assigned to the control plane node assigned as the primary controller by the load balancer.

If you are using a container registry mirror, you must also set the location of the NGINX image using the `--nginx-image` option. This option must be set to the location of your registry mirror in the format:

registry:*port*/olcne/nginx:*version*

For example:

```
--nginx-image myregistry.example.com:5000/olcne/nginx:1.17.7
```

All other options used in this example are described in Creating an HA Cluster with External Load Balancer.

Creating a Cluster with a Single Control Plane Node

This section shows you how to create Kubernetes module to create a cluster with a single control plane node. No load balancer is used or required with this type of cluster.

This example creates a cluster with a single control plane node.

```
olcnectl module create \
--environment-name myenvironment \
--module kubernetes --name mycluster \
--container-registry container-registry.oracle.com/olcne \
--control-plane-nodes control1.example.com:8090 \
--worker-nodes worker1.example.com:8090,worker2.example.com:8090 \
--selinux enforcing \
--restrict-service-externalip-ca-cert /etc/olcne/certificates/
restrict_external_ip/ca.cert \
--restrict-service-externalip-tls-cert /etc/olcne/certificates/
restrict_external_ip/node.cert \
--restrict-service-externalip-tls-key /etc/olcne/certificates/
restrict_external_ip/node.key
```

The `--control-plane-nodes` option should contain only one node.

All other options used in this example are described in Creating an HA Cluster with External Load Balancer.

Validating a Kubernetes Module

When you have created a Kubernetes module in an environment, you should validate the nodes are configured correctly to install the module.

Use the `olcnectl module validate` command to validate the nodes are configured correctly. For example, to validate the Kubernetes module named `mycluster` in the `myenvironment` environment:

```
olcnectl module validate \
--environment-name myenvironment \
--name mycluster
```

If there are any validation errors, the commands required to fix the nodes are provided in the output. If you want to save the commands as scripts, use the `--generate-scripts` option. For example:

```
olcnectl module validate \
--environment-name myenvironment \
--name mycluster \
--generate-scripts
```

A script is created for each node in the module, saved to the local directory, and named *hostname*:8090.sh. You can copy the script to the appropriate node, and run it to fix any validation errors.

Installing a Kubernetes Module

When you have created and validated a Kubernetes module, you use it to install Kubernetes on the nodes and create a cluster.

Use the `olcnectl module install` command to install Kubernetes on the nodes to create a cluster.

As part of installing the Kubernetes module:

- The Kubernetes packages are installed on the nodes. The `kubeadm` package installs the packages required to run CRI-O and Kata Containers. CRI-O is needed to delegate containers to a runtime engine (either `runc` or `kata-runtime`). For more information about container runtimes, see Container Runtimes.
- The `crio` and `kubelet` services are enabled and started.
- If you are installing an internal load balancer, the `olcne-nginx` and `keepalived` services are enabled and started on the control plane nodes.

For example, use the following command to use the Kubernetes module named `mycluster` in the `myenvironment` environment to create a cluster:

```
olcnectl module install \
--environment-name myenvironment \
--name mycluster
```

The Kubernetes module is used to install Kubernetes on the nodes and the cluster is started and validated for health.

> **Important:**
>
> Installing Kubernetes may take several minutes to complete.

Reporting Information about the Kubernetes Module

When you have installed a Kubernetes module, you can review information about the Kubernetes module and its properties.

Use the `olcnectl module report` command to review information about the module.

For example, use the following command to review the Kubernetes module named `mycluster` in `myenvironment`:

```
olcnectl module report \
--environment-name myenvironment \
--name mycluster \
--children
```

For more information on the syntax for the `olcnectl module report` command, see Platform Command-Line Interface.

3

Setting up the Kubernetes Command-Line Interface (kubectl)

This chapter describes how to set up the Kubernetes Command-Line Interface (`kubectl`). The `kubectl` command is part of Kubernetes and is used to create and manage the containerized applications you deploy on the Kubernetes cluster.

The `kubectl` utility is a command line tool that interfaces with the Kubernetes API server to run commands against the Kubernetes cluster. The `kubectl` command is typically run on the *control plane* node of the cluster (the recommended option), although you can set up `kubectl` access on an external non-cluster node if required. The `kubectl` utility effectively grants full administrative rights to the cluster and all of the nodes in the cluster.

This chapter discusses setting up the `kubectl` command to access a Kubernetes cluster from either a control plane node or an external node (not part of the Kubernetes cluster).

Setting up kubectl on a Control Plane Node

To set up the `kubectl` command on a control plane node, copy and paste these commands to a terminal in your home directory on a **control plane** node:

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
export KUBECONFIG=$HOME/.kube/config
echo 'export KUBECONFIG=$HOME/.kube/config' >> $HOME/.bashrc
```

Verify that you can use the `kubectl` command using any `kubectl` command such as:

```
kubectl get pods --all-namespaces
```

The output looks similar to:

```
NAMESPACE              NAME                          READY   STATUS
RESTARTS   AGE
externalip-validat...  externalip-validation-...     1/1     Running
0          1h
kube-system            coredns-...                   1/1     Running
0          1h
kube-system            coredns-...                   1/1     Running
0          1h
kube-system            etcd-...                      1/1     Running
2          1h
kube-system            etcd-...                      1/1     Running
2          1h
kube-system            kube-apiserver-...            1/1     Running
```

```
2          1h
kube-system            kube-apiserver-...            1/1     Running
2          1h
kube-system            kube-controller-manager-...   1/1     Running
5 (1h ago) 1h
kube-system            kube-controller-manager-...   1/1     Running
2          1h
kube-system            kube-flannel-...              1/1     Running
0          1h
kube-system            kube-flannel-...              1/1     Running
0          1h
kube-system            kube-flannel-...              1/1     Running
0          1h
kube-system            kube-proxy-...                1/1     Running
0          1h
kube-system            kube-proxy-...                1/1     Running
0          1h
kube-system            kube-proxy-...                1/1     Running
0          1h
kube-system            kube-scheduler-...            1/1     Running
5 (1h ago) 1h
kube-system            kube-scheduler-...            1/1     Running
2          1h
kubernetes-dashboard   kubernetes-dashboard-...      1/1     Running
0          1h
```

Setting up kubectl on a Non-Cluster Node

Oracle Cloud Native Environment allows you to create multiple environments from the operator node. With this in mind, it is recommended that you use the `kubectl` command on a control plane node in the Kubernetes cluster. If you use the `kubectl` command from outside the cluster, and you have multiple environments deployed, you may inadvertently manage an unexpected Kubernetes cluster. However, if you need to set up the `kubectl` command to run from outside the cluster, you need to configure it.

The following example shows you how to set up a non-cluster host with `kubectl` command-line access to a Kubernetes cluster.

> **Note:**
>
> The following example assumes the operating system of the non-cluster node is Oracle Linux. However, you can also set up `kubectl` on macOS and Microsoft Windows hosts by leveraging the Kubernetes community package. For Microsoft Windows hosts you would also need to install Windows Subsystem for Linux 2 (WLS 2).

1. **Create a copy of the Kubernetes configuration file.**

   On the operator node, use the `olcnectl module property get` command to get the Kubernetes configuration file for the cluster:

```
olcnectl module property get \
--environment-name myenvironment \
--name mycluster \
--property kubecfg | base64 -d > kubeconfig.yaml
```

A file named `kubeconfig.yaml` is created that contains the Kubernetes configuration information required to access the cluster.

2. **Set up the Kubernetes file on the non-cluster node:**
   a. Log onto the non-cluster node.
   b. Copy the `kubeconfig.yaml` from the operator node to a local directory on the non-cluster node.

> **Caution:**
>
> It is important to follow security best practices when copying a configuration file with sensitive information between hosts.

   c. Create a subdirectory named `.kube` in your home directory:

```
mkdir -p $HOME/.kube
```

   d. Copy the `kubeconfig.yaml` file to the `.kube` directory:

```
cp /path_to_file/kubeconfig.yaml $HOME/.kube/config
```

   e. Export the path to the file for the `KUBECONFIG` environment variable:

```
export KUBECONFIG=$HOME/.kube/config
```

   f. To permanently set this environment variable, add it to your `.bashrc` file:

```
echo 'export KUBECONFIG=$HOME/.kube/config' >> $HOME/.bashrc
```

3. **Install `kubectl` on the non-cluster node:**
   a. Set up the node with the required access to Oracle Cloud Native Environment packages by enabling repositories or channels as required. See Getting Started for more information.
   b. Install `kubectl` by running the following command:

```
sudo dnf install kubectl
```

4. **Verify `kubectl` access from the non-cluster node:**

   Verify that you can use the `kubectl` command:

```
kubectl get pods -n kube-system
NAME                                     READY   STATUS    RESTARTS   AGE
coredns-5bc65d7f4b-qzfcc                 1/1     Running   0          23h
coredns-5bc65d7f4b-z64f2                 1/1     Running   0          23h
etcd-control1.example.com                1/1     Running   0          23h
kube-apiserver-control1.example.com      1/1     Running   0          23h
```

```
kube-controller-control1.example.com    1/1     Running   0          23h
...
```

4
Using Kubernetes

This chapter describes how to get started using Kubernetes to deploy, maintain and scale your containerized applications. In this chapter, we describe basic usage of the `kubectl` command to get you started creating and managing containers and services within your environment.

The `kubectl` utility is fully documented in the upstream documentation at:

https://kubernetes.io/docs/reference/kubectl/

About Runtime Engines

`runc` is the default runtime engine when you create containers. You can also use the `kata-runtime` runtime engine to create Kata containers. For information on Kata containers and how to create them, see Container Runtimes.

Getting Information about Nodes

To get a listing of all of the nodes in a cluster and the status of each node, use the `kubectl get` command. This command can be used to obtain listings of any kind of resource that Kubernetes supports. In this case, the `nodes` resource:

```
kubectl get nodes
NAME                  STATUS    ROLES          AGE      VERSION
control.example.com   Ready     control-plane  1h       v1.21.x+x.x.x.el8
worker1.example.com   Ready     <none>         1h       v1.21.x+x.x.x.el8
worker2.example.com   Ready     <none>         1h       v1.21.x+x.x.x.el8
```

You can get more detailed information about any resource using the `kubectl describe` command. If you specify the name of the resource, the output is limited to information about that resource alone; otherwise, full details of all resources are also printed to screen. For example:

```
kubectl describe nodes worker1.example.com
Name:               worker1.example.com
Roles:              <none>
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=worker1.example.com
                    kubernetes.io/os=linux
Annotations:        flannel.alpha.coreos.com/backend-data:
{"VtepMAC":"fe:78:5f:ea:7c:c0"}
                    flannel.alpha.coreos.com/backend-type: vxlan
                    flannel.alpha.coreos.com/kube-subnet-manager: true
                    flannel.alpha.coreos.com/public-ip: 192.0.2.11
                    kubeadm.alpha.kubernetes.io/cri-socket: /var/run/crio/crio.sock
                    node.alpha.kubernetes.io/ttl: 0
                    volumes.kubernetes.io/controller-managed-attach-detach: true
...
```

Running an Application in a Pod

To create a pod with a single running container, you can use the `kubectl create` command. For example:

```
kubectl create deployment --image nginx hello-world
deployment.apps/hello-world created
```

Substitute `nginx` with a container image. Substitute `hello-world` with a name for your deployment. Your pods are named by using the deployment name as a prefix.

> **Tip:**
>
> Deployment, pod and service names conform to a requirement to match a DNS-1123 label. These must consist of lower case alphanumeric characters or -, and must start and end with an alphanumeric character. The regular expression that is used to validate names is '`[a-z0-9]([-a-z0-9]*[a-z0-9])?`'. If you use a name for your deployment that does not validate, an error is returned.

There are many additional optional parameters that can be used when you run a new application within Kubernetes. For instance, at run time, you can specify how many replica pods should be started, or you might apply a label to the deployment to make it easier to identify pod components. To see a full list of options available to you, run `kubectl run --help`.

To check that your new application deployment has created one or more pods, use the `kubectl get pods` command:

```
kubectl get pods
NAME                           READY     STATUS    RESTARTS   AGE
hello-world-5f55779987-wd857   1/1       Running   0          1m
```

Use `kubectl describe` to show a more detailed view of your pods, including which containers are running and what image they are based on, as well as which node is currently hosting the pod:

```
kubectl describe pods
Name:               hello-world-5f55779987-wd857
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               worker1.example.com/192.0.2.11
Start Time:         Fri, 16 Aug 2019 08:48:33 +0100
Labels:             app=hello-world
                    pod-template-hash=5f55779987
Annotations:        <none>
Status:             Running
IP:                 10.244.1.3
Controlled By:      ReplicaSet/hello-world-5f55779987
Containers:
  nginx:
    Container ID:   cri-o://
417b4b59f7005eb4b1754a1627e01f957e931c0cf24f1780cd94fa9949be1d31
```

```
    Image:          nginx
    Image ID:       docker-pullable://
nginx@sha256:5d32f60db294b5deb55d078cd4feb410ad88e6fe7...
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Mon, 10 Dec 2018 08:25:25 -0800
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-s8wj4 (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-s8wj4:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-s8wj4
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
....
```

Scaling a Pod Deployment

To change the number of instances of the same pod that you are running, you can use the `kubectl scale deployment` command. For example:

```
kubectl scale deployment --replicas=3 hello-world
deployment.apps/hello-world scaled
```

You can check that the number of pod instances has been scaled appropriately:

```
kubectl get pods
NAME                           READY     STATUS    RESTARTS   AGE
hello-world-5f55779987-tswmg   1/1       Running   0          18s
hello-world-5f55779987-v8w5h   1/1       Running   0          26m
hello-world-5f55779987-wd857   1/1       Running   0          18s
```

Exposing a Service Object for an Application

Typically, while many applications only need to communicate internally within a pod, or even across pods, you might need to expose your application externally so that clients outside of the Kubernetes cluster can interface with the application. You can do this by creating a service definition for the deployment.

> **Note:**
>
> **Prerequisite for `LoadBalancer` service:**
>
> The **Oracle Cloud Infrastructure Cloud Controller Manager** module (`oci-ccm`) is used to create and manage Oracle Cloud Infrastructure load balancers for Kubernetes applications. Hence, the following example assumes you have completed installation of the `oci-ccm` module as described in Application Load Balancers.

To expose a deployment using a service object, you must define the service type that should be used. The following example shows how you might use the `kubectl expose deployment` command to expose the application via a `LoadBalancer` service:

```
kubectl expose deployment hello-world --port 80 --type=LoadBalancer
service/hello-world exposed
```

Use `kubectl get services` to list the different services that the cluster is running as shown in the following example. Note that the `EXTERNAL-IP` field of the `LoadBalancer` service initially shows as `<pending>` whilst the setup of the service is still in progress:

```
kubectl get services
NAME          TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
hello-world   LoadBalancer   10.102.42.160   <pending>     80:31847/TCP   3s
kubernetes    ClusterIP      10.96.0.1       <none>        443/TCP        5h13m
```

You can see the load balancer in the Oracle Cloud Infrastructure console. Initially, its state in the console is shown as **Creating**.

Wait a few minutes for the setup of the service to complete. Run the `kubectl get services` command again, and note that the `EXTERNAL-IP` field is now populated with the IP address assigned to the `LoadBalancer` service:

```
kubectl get services
NAME          TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)          AGE
hello-world   LoadBalancer   10.102.42.160   192.0.2.250    80:31847/TCP     85s
kubernetes    ClusterIP      10.96.0.1       <none>         443/TCP          5h15m
```

In the preceding sample output the `PORT(s)` field contains the following ports:

- **Port `80`:** The port at which the `LoadBalancer` service can be accessed. In this example, the service would be accessed at the following URL:

  ```
  http://192.0.2.250
  ```

- **Port `31847`:** The port assigned to the `NodePort` service. The `NodePort` service enables the application to be accessed using URL format *worker_node*`:`*NodePort*, for example:

  ```
  http://worker1.example.com:31847/
  ```

> **Note:**
>
> Kubernetes creates the `NodePort` service as part of its `LoadBalancer` setup.

You can verify the services have been set up successfully by running `curl` commands as shown in the following examples:

- For the `LoadBalancer` service:

```
curl http://192.0.2.250

    <html>
        <head>
            <title>Welcome to this servicer</title>
        </head>
        <body>
                <h1>Welcome to this service</h1>
                ...
        </body>
    </html>
```

- For each worker node, verify the `NodePort` service by running a `curl` command as illustrated below:

```
curl http://worker1.example.com:31847/

     <html>
        <head>
            <title>Welcome to this servicer</title>
            ...
        </head>
        <body>
                <h1>Welcome to this service</h1>
                ...
        </body>
    </html>
```

Deleting a Service or Deployment

Objects can be deleted easily within Kubernetes so that your environment can be cleaned. Use the `kubectl delete` command to remove an object.

To delete a service, specify the services object and the name of the service that you want to remove. For example:

```
kubectl delete services hello-world
service "hello-world" deleted
```

To delete an entire deployment, and all of the pod replicas running for that deployment, specify the deployment object and the name that you used to create the deployment:

```
kubectl delete deployment hello-world
deployment.extensions "hello-world" deleted
```

Working With Namespaces

Namespaces can be used to further separate resource usage and to provide limited environments for particular use cases. By default, Kubernetes configures a namespace for Kubernetes system components and a standard namespace to be used for all other deployments for which no namespace is defined.

To view existing namespaces, use the `kubectl get namespaces` and `kubectl describe namespaces` commands.

The `kubectl` command only displays resources in the `default` namespace, unless you set the namespace specifically for a request. Therefore, if you need to view the pods specific to the Kubernetes system, you would use the `--namespace` option to set the namespace to `kube-system` for the request. For example, in a cluster with a single control plane node:

```
kubectl get pods --namespace=kube-system
NAME                                    READY   STATUS    RESTARTS   AGE
coredns-5bc65d7f4b-qzfcc                1/1     Running   0          23h
coredns-5bc65d7f4b-z64f2                1/1     Running   0          23h
etcd-control1.example.com               1/1     Running   0          23h
kube-apiserver-control1.example.com     1/1     Running   0          23h
kube-controller-control1.example.com    1/1     Running   0          23h
kube-flannel-ds-2sjbx                   1/1     Running   0          23h
kube-flannel-ds-njg9r                   1/1     Running   0          23h
kube-proxy-m2rt2                        1/1     Running   0          23h
kube-proxy-tbkxd                        1/1     Running   0          23h
kube-scheduler-control1.example.com     1/1     Running   0          23h
kubernetes-dashboard-7646bf6898-d6x2m   1/1     Running   0          23h
```

Using Deployment Files

To simplify the creation of pods and their related requirements, you can create a deployment file that define all of the elements that comprise the deployment. This deployment defines which images should be used to generate the containers within the pod, along with any runtime requirements, as well as Kubernetes networking and storage requirements in the form of services that should be configured and volumes that may need to be mounted.

Deployments are described in detail at:

https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

5
Accessing the Kubernetes Dashboard

The Kubernetes Dashboard container is created as part of the `kubernetes-dashboard` namespace. You can also start the Dashboard using the `kubectl-proxy` service. The Dashboard provides an intuitive graphical user interface to a Kubernetes cluster that can be accessed using a standard web browser.

The Kubernetes Dashboard is described in the upstream Kubernetes documentation at:

https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/

This chapter shows you how to start and connect to the Kubernetes Dashboard.

Starting the Dashboard

To start the Dashboard, you can run a proxy service that allows traffic on the node where it is running to reach the internal pod where the Dashboard application is running. This is achieved by running the `kubectl proxy` service:

```
kubectl proxy
Starting to serve on 127.0.0.1:8001
```

The Dashboard is available on the node where the proxy is running for as long as the proxy runs. To exit the proxy, use `Ctrl+C`.

You can run this as a `systemd` service and enable it so that it is always available after subsequent reboots:

```
sudo systemctl enable --now kubectl-proxy.service
```

This `systemd` service requires that the `/etc/kubernetes/admin.conf` is present to run. If you want to change the port that is used for the proxy service, or you want to add other proxy configuration parameters, you can configure this by editing the `systemd` drop-in file at `/etc/systemd/system/kubectl-proxy.service.d/10-kubectl-proxy.conf`. You can get more information about the configuration options available for the `kubectl proxy` service by running:

```
kubectl proxy --help
```

Connecting to the Dashboard

To access the Dashboard, open a web browser on the node where the `kubectl proxy` service is running and navigate to:

http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/

To log in, you must authenticate using a token. For more information on authentication tokens, see the upstream documentation at:

https://github.com/kubernetes/dashboard/blob/master/docs/user/access-control/README.md

If you are running Kubernetes 1.24.*x* or later, you can set up a token for the `admin-user` using:

```
kubectl -n kubernetes-dashboard create token admin-user
```

If you are running Kubernetes 1.23.*x*, and have not set up specific tokens for this purpose, you can use a token allocated to a service account, such as the `namespace-controller`. Run the following command to obtain the token value for the `namespace-controller`:

```
      kubectl -n kube-system describe $(kubectl -n kube-system \
get secret -n kube-system -o name | grep namespace) | grep token:
```

Copy and paste the entire value of the token output by either of these commands into the token field on the log in page to authenticate.

Connecting to the Dashboard Remotely

If you need to access the Dashboard remotely, you can use SSH tunneling to do port forwarding from your localhost to the node running the `kubectl proxy` service. The easiest option is to use SSH tunneling to forward a port on your local system to the port configured for the `kubectl proxy` service on the node that you want to access. This method retains some security as the HTTP connection is encrypted by virtue of the SSH tunnel and authentication is handled by your SSH configuration. For example, on your local system run:

```
ssh -L 8001:127.0.0.1:8001 192.0.2.10
```

Substitute `192.0.2.10` with the IP address of the host where the `kubectl proxy` service is running. When the SSH connection is established, you can open a browser on your localhost and navigate to:

http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/

You should see the Dashboard log in screen for the remote Kubernetes cluster. Use the same token information to authenticate as if you were connecting to the Dashboard locally.

Connecting to the Dashboard Container

You do not need to start the Dashboard using the `kubectl-proxy` service as it is already running as a container when you install the Kubernetes module. This is another method to access the Dashboard. To verify the container is running, enter:

```
kubectl get pods --namespace kubernetes-dashboard
NAME                                    READY   STATUS    RESTARTS
AGE
kubernetes-dashboard-785945dc77-c8l72   1/1     Running   0
19m
```

There is also a Kubernetes Dashboard service. You can show that service using:

```
kubectl get svc --namespace kubernetes-dashboard
NAME                   TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
kubernetes-dashboard   ClusterIP   10.100.29.246   <none>        443/TCP
20m
```

To access this service, you should assign an external IP address to the ClusterIP, or patch the service to assign an IP address using a NodePort. When you have assigned an external IP address, you can connect to the service using a web browser that has access to that network.

6
Scaling a Kubernetes Cluster

A Kubernetes cluster may consist of either a single or multiple control plane node and worker nodes. The more applications that you run in a cluster, the more resources (nodes) that you need. So, what do you do if you need additional resources to handle a high amount of workload or traffic, or if you want to deploy more services to the cluster? You add additional nodes to the cluster. Or, what happens if there are faulty nodes in your cluster? You remove them.

Scaling a Kubernetes cluster is updating the cluster by adding nodes to it or removing nodes from it. When you add nodes to a Kubernetes cluster, you are scaling up the cluster, and when you remove nodes from the cluster, you are scaling down the cluster.

If you want to replace a node in a cluster, first scale up the cluster (add the new node) and then scale down the cluster (remove the old node).

> **Note:**
>
> Oracle recommends that you should not scale the cluster up and down at the same time. You should scale up, then scale down, in two separate commands. To avoid split-brain scenarios, scale your Kubernetes cluster control plane nodes in odd numbers. For example, 3, 5, or 7 control plane nodes ensures the reliability of your cluster.

If you used the `--apiserver-advertise-address` option when you created a Kubernetes module, then you cannot scale up from a cluster with a single control plane node to a highly available (HA) cluster with multiple control plane nodes. However, if you used the `--virtual-ip` or the `--load-balancer` options, then you can scale up, even if you have only a single control plane node cluster.

> **Important:**
>
> The `--apiserver-advertise-address` option has been deprecated. Use the `--control-plane-nodes` option.

When you scale a Kubernetes cluster, the following actions are completed:

1. A back up is taken of the cluster. In case something goes wrong during scaling up or scaling down, you can revert to the previous state so that you can restore the cluster. For more information about backing up and restoring a Kubernetes cluster, see Backing up and Restoring a Kubernetes Cluster.
2. Any nodes that you want to add to the cluster are validated. If the nodes have any validation issues, such as firewall issues, then the update to the cluster cannot proceed,

and the nodes cannot be added to the cluster. You are prompted for what to do to resolve the validation issues so that the nodes can be added to the cluster.

3. The control plane node and worker nodes are added to or removed from the cluster.
4. The cluster is checked to make sure all nodes are healthy. After validation of the cluster is completed, the cluster is scaled and you can access it.

> **Tip:**
>
> The examples in this chapter show you how to scale up and down by changing the control plane node and worker nodes at the same time by providing all the nodes to be included in the cluster using the `--control-plane-nodes` and `--worker-nodes` options. If you only want to scale control plane nodes, you only need to provide the list of control plane nodes to include in the cluster using the `--control-plane-nodes` option (you do not need to provide all worker nodes). Similarly, if you only want to scale worker nodes, you only need to provide the list of worker nodes using the `--worker-nodes` option.

Scaling Up a Kubernetes Cluster

Before you scale up a Kubernetes cluster, set up the new nodes so they can be added to the cluster.

To prepare a node:

1. Set up the node so it can be added to a Kubernetes cluster. For information on setting up a Kubernetes node see Getting Started.
2. If you are using private X.509 certificates for nodes, you need to copy the certificates to the node. You do not need to do anything if you are using Vault to provide certificates for nodes. For information using X.509 certificates see Getting Started.
3. Start the Platform Agent service. For information on starting the Platform Agent, see Getting Started.

After completing these actions, use the instructions in this procedure to add nodes to a Kubernetes cluster.

To scale up a Kubernetes cluster:

1. From a control plane node of the Kubernetes cluster, use the `kubectl get nodes` command to see the control plane node and worker nodes of the cluster.

```
kubectl get nodes
NAME                  STATUS   ROLE            AGE     VERSION
control1.example.com   Ready    control-plane   26h     v1.21.x+x.x.x.el8
control2.example.com   Ready    control-plane   26h     v1.21.x+x.x.x.el8
control3.example.com   Ready    control-plane   26h     v1.21.x+x.x.x.el8
worker1.example.com    Ready    <none>          26h     v1.21.x+x.x.x.el8
worker2.example.com    Ready    <none>          26h     v1.21.x+x.x.x.el8
worker3.example.com    Ready    <none>          26h     v1.21.x+x.x.x.el8
```

   In this example, there are three control plane nodes in the Kubernetes cluster:

- control1.example.com
- control2.example.com
- control3.example.com

There are also three worker nodes in the cluster:

- worker1.example.com
- worker2.example.com
- worker3.example.com

2. Use the `olcnectl module update` command to scale up a Kubernetes cluster.

   In this example, the Kubernetes cluster is scaled up so that it has four control plane nodes and five worker nodes. This example adds a new control plane node (`control.example.com`) and two new workers nodes (`worker4.example.com` and `worker5.example.com`) to the Kubernetes module named `mycluster`. From the operator node run:

   ```
   olcnectl module update \
   --environment-name myenvironment \
   --name mycluster \
   --control-plane-nodes
   control1.example.com:8090,control2.example.com:8090,control3.example.com:8090,\
   control4.example.com:8090 \
   --worker-nodes
   worker1.example.com:8090,worker2.example.com:8090,worker3.example.com:8090,\
   worker4.example.com:8090,worker5.example.com:8090
   ```

   Make sure that if you are scaling up from a single control plane node to a highly available cluster, you have specified a load balancer for the cluster. If you do not specify a load balancer, then you cannot scale up your control plane nodes. That is, you cannot move from a single control plane node to a highly available cluster without a load balancer.

   You can optionally include the `--generate-scripts` option. This option generates scripts you can run for each node in the event of any validation failures encountered during scaling. A script is created for each node in the module, saved to the local directory, and named *`hostname`*`:8090.sh`.

   You can also optionally included the `--force` option to suppress the prompt displayed to confirm you want to continue with scaling the cluster.

3. On a control plane node of the Kubernetes cluster, use the `kubectl get nodes` command to verify the cluster is scaled up to include the new control plane node and worker nodes.

   ```
   kubectl get nodes
   NAME                  STATUS   ROLE           AGE     VERSION
   control1.example.com   Ready   control-plane   26h     v1.21.x+x.x.x.el8
   control2.example.com   Ready   control-plane   26h     v1.21.x+x.x.x.el8
   control3.example.com   Ready   control-plane   26h     v1.21.x+x.x.x.el8
   control4.example.com   Ready   control-plane   2m38s   v1.21.x+x.x.x.el8
   worker1.example.com    Ready   <none>          26h     v1.21.x+x.x.x.el8
   worker2.example.com    Ready   <none>          26h     v1.21.x+x.x.x.el8
   worker3.example.com    Ready   <none>          26h     v1.21.x+x.x.x.el8
   worker4.example.com    Ready   <none>          2m38s   v1.21.x+x.x.x.el8
   worker5.example.com    Ready   <none>          2m38s   v1.21.x+x.x.x.el8
   ```

Scaling Down a Kubernetes Cluster

This procedure shows you how to remove nodes from a Kubernetes cluster.

> **NOT_SUPPORTED:**
>
> Be careful if you are scaling down the control plane nodes of your cluster. If you have two control plane nodes and you scale down to have only one control plane node, then you would have only a single point of failure.

To scale down a Kubernetes cluster:

1. From a control plane node of the Kubernetes cluster, use the `kubectl get nodes` command to see the control plane node and worker nodes of the cluster.

   ```
   kubectl get nodes
   NAME                  STATUS   ROLE           AGE     VERSION
   control1.example.com  Ready    control-plane  26h     v1.21.x+x.x.x.el8
   control2.example.com  Ready    control-plane  26h     v1.21.x+x.x.x.el8
   control3.example.com  Ready    control-plane  26h     v1.21.x+x.x.x.el8
   control4.example.com  Ready    control-plane  2m38s   v1.21.x+x.x.x.el8
   worker1.example.com   Ready    <none>         26h     v1.21.x+x.x.x.el8
   worker2.example.com   Ready    <none>         26h     v1.21.x+x.x.x.el8
   worker3.example.com   Ready    <none>         26h     v1.21.x+x.x.x.el8
   worker4.example.com   Ready    <none>         2m38s   v1.21.x+x.x.x.el8
   worker5.example.com   Ready    <none>         2m38s   v1.21.x+x.x.x.el8
   ```

   In this example, there are four control plane nodes in the Kubernetes cluster:

   - `control1.example.com`
   - `control2.example.com`
   - `control3.example.com`
   - `control4.example.com`

   There are also five worker nodes in the cluster:

   - `worker1.example.com`
   - `worker2.example.com`
   - `worker3.example.com`
   - `worker4.example.com`
   - `worker5.example.com`

2. Use the `olcnectl module update` command to scale down a Kubernetes cluster.

   In this example, the Kubernetes cluster is scaled down so that it has three control plane nodes and three worker nodes. This example removes a control plane node (`control4.example.com`) and two workers nodes (`worker4.example.com` and `worker5.example.com`) from the Kubernetes module named `mycluster`. As the nodes are no longer listed in the `--control-plane-nodes` or `--worker-nodes` options, they are removed from the cluster. From the operator node run:

```
olcnectl module update \
--environment-name myenvironment \
--name mycluster \
--control-plane-nodes
control1.example.com:8090,control2.example.com:8090,control3.example.com:8090 \
--worker-nodes
worker1.example.com:8090,worker2.example.com:8090,worker3.example.com:8090
```

3. On a control plane node of the Kubernetes cluster, use the `kubectl get nodes` command to verify the cluster is scaled down to remove the control plane node and worker nodes.

```
kubectl get nodes
NAME                  STATUS   ROLE           AGE     VERSION
control1.example.com  Ready    control-plane  26h     v1.21.x+x.x.x.el8
control2.example.com  Ready    control-plane  26h     v1.21.x+x.x.x.el8
control3.example.com  Ready    control-plane  26h     v1.21.x+x.x.x.el8
worker1.example.com   Ready    <none>         26h     v1.21.x+x.x.x.el8
worker2.example.com   Ready    <none>         26h     v1.21.x+x.x.x.el8
worker3.example.com   Ready    <none>         26h     v1.21.x+x.x.x.el8
```

7

Backing up and Restoring a Kubernetes Cluster

This chapter discusses how to back up and restore a Kubernetes cluster in Oracle Cloud Native Environment.

Backing up Control Plane Nodes

Adopting a back up strategy to protect your Kubernetes cluster against control plane node failures is important, particularly for clusters with only one control plane node. High availability clusters with multiple control plane nodes also need a fallback plan if the resilience provided by the replication and failover functionality has been exceeded.

You do not need to bring down the cluster to perform a back up as part of your disaster recovery plan. On the operator node, use the `olcnectl module backup` command to back up the key containers and manifests for all the control plane nodes in your cluster.

> **Important:**
>
> Only the key containers required for the Kubernetes control plane node are backed up. No application containers are backed up.

For example:

```
olcnectl module backup \
--environment-name myenvironment \
--name mycluster
```

The back up files are stored in the `/var/olcne/backups` directory on the operator node. The files are saved to a timestamped folder that follows the pattern:

`/var/olcne/backups/`*`environment-name`*`/kubernetes/`*`module-name`*`/`*`timestamp`*

You can interact with the directory and the files it contains just like any other, for example:

```
sudo ls /var/olcne/backups/myenvironment/kubernetes/mycluster/20191007040013
control1.example.com.tar control2.example.com.tar control3.example.com.tar  etcd.tar
```

Restoring Control Plane Nodes

These restore steps are intended for use when a Kubernetes cluster needs to be reconstructed as part of a planned disaster recovery scenario. Unless there is a total cluster failure you do not need to manually recover individual control plane nodes in a high availability cluster that is able to self-heal with replication and failover.

In order to restore a control plane node, you must have a pre-existing Oracle Cloud Native Environment, and have deployed the Kubernetes module. You cannot restore to a non-existent environment.

To restore a control plane node:

1. Make sure the Platform Agent is running correctly on the control plane nodes before proceeding:

   ```
   systemctl status olcne-agent.service
   ```

2. On the operator node, use the `olcnectl module restore` command to restore the key containers and manifests for the control plane nodes in your cluster. For example:

   ```
   olcnectl module restore \
   --environment-name myenvironment \
   --name mycluster
   ```

   The files from the latest timestamped folder from `/var/olcne/backups/`*`environment-name`*`/kubernetes/`*`module-name`*`/` are used to restore the cluster to its previous state.

   You may be prompted by the Platform CLI to perform additional set up steps on your control plane nodes to fulfill the prerequisite requirements. If that happens, follow the instructions and run the `olcnectl module restore` command again.

3. You can verify the restore operation was successful using the `kubectl` command on a control plane node. For example:

   ```
   kubectl get nodes
   NAME                    STATUS   ROLES           AGE     VERSION
   control1.example.com    Ready    control-plane   9m27s   v1.21.x+x.x.x.el8
   worker1.example.com     Ready    <none>          8m53s   v1.21.x+x.x.x.el8

   kubectl get pods -n kube-system
   NAME                                    READY   STATUS    RESTARTS   AGE
   coredns-5bc65d7f4b-qzfcc                1/1     Running   0          9m
   coredns-5bc65d7f4b-z64f2                1/1     Running   0          9m
   etcd-control1.example.com               1/1     Running   0          9m
   kube-apiserver-control1.example.com     1/1     Running   0          9m
   kube-controller-control1.example.com    1/1     Running   0          9m
   kube-flannel-ds-2sjbx                   1/1     Running   0          9m
   kube-flannel-ds-njg9r                   1/1     Running   0          9m
   kube-proxy-m2rt2                        1/1     Running   0          9m
   kube-proxy-tbkxd                        1/1     Running   0          9m
   kube-scheduler-control1.example.com     1/1     Running   0          9m
   kubernetes-dashboard-7646bf6898-d6x2m   1/1     Running   0          9m
   ```

8

Setting Access to externalIPs in Kubernetes Services

This chapter discusses setting access to `externalIPs` in Kubernetes services. For more information on `externalIPs`, see the upstream documentation at:

https://kubernetes.io/docs/concepts/services-networking/service/#external-ips

When you deploy Kubernetes, a service is deployed to the cluster that controls access to `externalIPs` in Kubernetes services. The service is named `externalip-validation-webhook-service` and runs in the `externalip-validation-system` namespace.

After Kubernetes is deployed, you can see the service is running using:

```
kubectl get services --namespace externalip-validation-system
NAME                                      TYPE        CLUSTER-IP      EXTERNAL-IP
PORT(S)   AGE
externalip-validation-webhook-service   ClusterIP   10.100.79.236   <none>
443/TCP   15m
```

This Kubernetes service requires X.509 certificates be set up prior to deploying Kubernetes. You can use certificates generated by Vault, your own certificates, or generate certificates using the `gen-certs-helper.sh` script. For information on setting up these certificates, see Getting Started.

When you deploy Kubernetes, you need to provide the location of these certificates in the `olcnectl module create` command. Examples of creating a Kubernetes module and setting the certificate locations are shown in Creating a Kubernetes Module.

Enabling Access to CIDR Blocks

You can optionally set the external IP addresses that can be accessed by Kubernetes services when you create the module. You use the `--restrict-service-externalip-cidrs` option of the `olcnectl module create` command to set this. In this example, the IP ranges that are allowed are within the `192.0.2.0/24` and `198.51.100.0/24` CIDR blocks.

```
olcnectl module create \
--environment-name myenvironment \
--module kubernetes \
--name mycluster \
...
--restrict-service-externalip-ca-cert /etc/olcne/certificates/restrict_external_ip/
ca.cert \
--restrict-service-externalip-tls-cert /etc/olcne/certificates/restrict_external_ip/
node.cert \
--restrict-service-externalip-tls-key /etc/olcne/certificates/restrict_external_ip/
node.key \
--restrict-service-externalip-cidrs 192.0.2.0/24,198.51.100.0/24
```

Modifying Access to CIDR Blocks

If you have a Kubernetes module that has CIDR blocks configured to be allowed, you can modify this configuration using the `--restrict-service-externalip-cidrs` option of the `olcnectl module update` command. This allows you to change the CIDRS that are configured. For example, to set the CIDR block that can be accessed to `192.0.2.0/24` for an existing Kubernetes module:

```
olcnectl module update \
--environment-name myenvironment \
--name mycluster \
--restrict-service-externalip-cidrs 192.0.2.0/24
```

To remove access to any CIDR blocks, which means no access to `externalIPs` is allowed, set `--restrict-service-externalip-cidrs` option to null, for example:

```
olcnectl module update \
--environment-name myenvironment \
--name mycluster \
--restrict-service-externalip-cidrs ""
```

Disabling Access to externalIPs

If you want to restrict Kubernetes services from accessing any `externalIPs`, do not you set any CIDR blocks that are allowed when you create the Kubernetes module. That is, do not use the `--restrict-service-externalip-cidrs` option of the `olcnectl module create` command. The `externalip-validation-webhook-service` Kubernetes service is deployed, but does not allow access to any `externalIPs`. For example:

```
olcnectl module create \
--environment-name myenvironment \
--module kubernetes \
--name mycluster \
...
--restrict-service-externalip-ca-cert /etc/olcne/certificates/
restrict_external_ip/ca.cert \
--restrict-service-externalip-tls-cert /etc/olcne/certificates/
restrict_external_ip/node.cert \
--restrict-service-externalip-tls-key /etc/olcne/certificates/
restrict_external_ip/node.key
```

If you have an existing Kubernetes module and you want to remove access to all CIDR blocks that may have been configured, update the module and set the `--restrict-service-externalip-cidrs` option to null as shown in Modifying Access to CIDR Blocks.

Enabling Access to all externalIPs

If you want all Kubernetes services to be able to access all `externalIPs`, you can disable this feature using the `--restrict-service-externalip false` option of the `olcnectl module create` command. Disabling this feature means that all Kubernetes services have access to all `externalIPs` in the cluster.

If you disable this feature, the `externalip-validation-webhook-service` Kubernetes service is not deployed to the cluster, which means no validation of external IP addresses is performed for Kubernetes services, and access is allowed for all CIDR blocks. For example, when you create a Kubernetes module, include the `--restrict-service-externalip false` option:

```
olcnectl module create \
--environment-name myenvironment \
--module kubernetes \
--name mycluster \
...
--restrict-service-externalip false
```

You can disable this feature in a Kubernetes cluster by using the `--restrict-service-externalip false` option of the `olcnectl module update` command. Modifying a Kubernetes module in this way removes the `externalip-validation-webhook-service` Kubernetes service from the cluster, so validation is not performed. For example:

```
olcnectl module update \
--environment-name myenvironment \
--name mycluster \
--restrict-service-externalip false
```

Conversely, if you enable this feature in a Kubernetes cluster by using the `--restrict-service-externalip true` option of the `olcnectl module update` command, the `externalip-validation-webhook-service` Kubernetes service is deployed to the cluster, so validation is then performed. For example:

```
olcnectl module update \
--environment-name myenvironment \
--name mycluster \
--restrict-service-externalip true
```

9
Using Operators with Kubernetes

This chapter discusses how to install and use the Operator Lifecycle Manager module for Oracle Cloud Native Environment to install and manage operators in a Kubernetes cluster.

A Kubernetes operator is a design pattern that allows you to write code to automate tasks and extend Kubernetes. It is a set of concepts you can use to define a service for Kubernetes and helps to automate administrative services in Kubernetes.

The Operator Lifecycle Manager module installs an instance Operator Lifecycle Manager into a Kubernetes cluster, which you can use to manage the installation and lifecycle management of operators in a Kubernetes cluster. The Operator Lifecycle Manager is essentially a package manager that interacts with operator registries. For more information about the Operator Lifecycle Manager, see the upstream documentation at:

https://olm.operatorframework.io/

OperatorHub is an operator registry that contains upstream Kubernetes operators that you can use to deploy operators in your cluster. The OperatorHub is at:

https://operatorhub.io/

Operator Lifecycle Manager in many ways performs the same tasks as Helm. A major additional feature that Operator Lifecycle Manager provides is that it has built-in support to validate Custom Resource Definitions (CRDs) inside Kubernetes software. Operators with CRDs can use these to make sure dependencies are met and no interfaces are duplicated. Otherwise, Operator Lifecycle Manager manages deployments in a similar way to Helm.

Installing the Operator Lifecycle Manager Module

The Operator Lifecycle Manager is installed into a Kubernetes cluster as an Oracle Cloud Native Environment module.

To install the Operator Lifecycle Manager module:

1. Use the `olcnectl module create` command to create the Operator Lifecycle Manager module. Specify the name of the Kubernetes module with the `--olm-kubernetes-module` option.

   ```
   olcnectl module create \
   --environment-name myenvironment \
   --module operator-lifecycle-manager \
   --name myolm \
   --olm-kubernetes-module mycluster
   ```

2. Use the `olcnectl module install` command to install the Operator Lifecycle Manager module:

   ```
   olcnectl module install \
   --environment-name myenvironment \
   --name myolm
   ```

The Operator Lifecycle Manager module is deployed and the required containers are running in the `operator-lifecycle-manager` namespace.

Verifying the Operator Lifecycle Manager Module Deployment

You can verify the Operator Lifecycle Manager module is deployed and the required deployments are running in the `operator-lifecycle-manager` namespace. To verify the containers are deployed, use the `kubectl` command on a control plane node.

To verify the required containers are running, list the deployments running in the `operator-lifecycle-manager` namespace. You should see similar results to those shown here:

```
kubectl get deploy -n operator-lifecycle-manager
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
catalog-operator   1/1     1            1           2m36s
olm-operator       1/1     1            1           2m36s
packageserver      2/2     2            2           2m30s
```

You can also review information about the Operator Lifecycle Manager module and its properties.

Use the `olcnectl module report` command to review information about the module.

For example, use the following command to review the Operator Lifecycle Manager module named `myolm` in `myenvironment`:

```
olcnectl module report \
--environment-name myenvironment \
--name myolm \
--children
```

For more information on the syntax for the `olcnectl module report` command, see Platform Command-Line Interface.

Listing Operator Registries

You can show the available operator registries using the `kubectl` command on a control plane node:

```
kubectl get catalogsource -n operator-lifecycle-manager
NAME                   DISPLAY               TYPE   PUBLISHER        AGE
operatorhubio-catalog  Community Operators   grpc   OperatorHub.io   3m35s
```

The OperatorHub registry is shown in the output. This is the default operator registry.

The OperatorHub provides examples of the text to use for your operator manifest files. On each operator's page on OperatorHub, there are example YAML files to create operator manifest files.

Installing Operators

To see all the operators that can be installed, use the `kubectl` command on a control plane node:

```
kubectl get packagemanifest
NAME                                     CATALOG               AGE
vault                                    Community Operators   3m22s
submariner                               Community Operators   3m22s
credstash-operator                       Community Operators   3m22s
eunomia                                  Community Operators   3m22s
ibm-block-csi-operator-community         Community Operators   3m22s
...
```

A list of the upstream operators available on OperatorHub are displayed. These are all available to be installed by the Operator Lifecycle Manager.

When you have decided on the operator name and catalog, you need to create the Kubernetes resources that tell Operator Lifecycle Manager how to install the operator. Two resources must be created: an `OperatorGroup` and a `Subscription`. If a new namespace is being created, you can create the `Namespace` in the same operator manifest file.

You can download starter operator manifest files for operators from the OperatorHub.

This example shows you how to create an `etcd` operator which is pulled from the OperatorHub.

To create an operator:

1. In a web browser, go to the OperatorHub and find the name of the operator you want to install. The OperatorHub is at:

   https://operatorhub.io/

   This example uses the `etcd` operator at:

   https://operatorhub.io/operator/etcd

   Click **Install**.

   A dialog is displayed that shows the `kubectl create` command to deploy the operator. For example:

   ```
   kubectl create -f https://operatorhub.io/install/etcd.yaml
   ```

   Copy the URL in this command that contains the operator manifest YAML file.

2. On a control plane node, download the `etcd` operator manifest YAML file from the OperatorHub:

   ```
   curl --remote-name https://operatorhub.io/install/etcd.yaml
   ```

3. You can edit this manifest YAML file to suit your needs. At the time of writing, this file contained the information required to create a `Namespace`, `OperatorGroup` and `Subscription` for the `etcd` operator:

   ```
   apiVersion: v1
   kind: Namespace
   metadata:
     name: my-etcd
   ---
   ```

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: operatorgroup
  namespace: my-etcd
spec:
  targetNamespaces:
  - my-etcd
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: my-etcd
  namespace: my-etcd
spec:
  channel: singlenamespace-alpha
  name: etcd
  source: operatorhubio-catalog
  sourceNamespace: olm
```

Edit this file to change the `sourceNamespace` from `olm` to `operator-lifecycle-manager` in the `Subscription` section so that it works properly with Operator Lifecycle Manager. Operator Lifecycle Manager runs in the `operator-lifecycle-manager` namespace, which is different to the upstream namespace.

```
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: my-etcd
  namespace: my-etcd
spec:
  channel: singlenamespace-alpha
  name: etcd
  source: operatorhubio-catalog
  sourceNamespace: operator-lifecycle-manager
```

4. Use the `kubectl apply` command to deploy the `etcd` operator.

```
kubectl apply -f etcd.yaml
namespace/my-etcd created
operatorgroup.operators.coreos.com/operatorgroup created
subscription.operators.coreos.com/my-etcd created
```

The operator is deployed into the namespace set in the operator manifest file, which in this example is `my-etcd`.

5. You can see the operator's `ClusterServiceVersion` information using:

```
kubectl get csv -n my-etcd
NAME                 DISPLAY   VERSION   REPLACES              PHASE
etcdoperator.v0.9.4  etcd      0.9.4     etcdoperator.v0.9.2   Succeeded
```

6. You can see the operator pods are running using:

```
kubectl get pods -n my-etcd
NAME                             READY   STATUS    RESTARTS   AGE
etcd-operator-75fb7df8b5-42k7b   3/3     Running   0          5m45s
```

Removing Operators

To remove an operator and uninstall it, delete the Kubernetes resources. For example, on a control plane node, use the `kubectl delete` command to delete the operator:

```
kubectl delete -f etcd.yaml
namespace "my-etcd" deleted
operatorgroup.operators.coreos.com "operatorgroup" deleted
subscription.operators.coreos.com "my-etcd" deleted
```

Uninstalling the Operator Lifecycle Manager Module

To uninstall the Operator Lifecycle Manager module, uninstall the module using the `olcnectl module uninstall` command. For example:

```
olcnectl module uninstall \
--environment-name myenvironment \
--name myolm
```

10
Removing a Kubernetes Cluster

If you want to remove a Kubernetes cluster, use the `olcnectl module uninstall` command. For example, to uninstall the Kubernetes module named `mycluster`:

```
olcnectl module uninstall \
--environment-name myenvironment \
--name mycluster
```

On each node, the Kubernetes containers are stopped and deleted, the Kubernetes cluster is removed, and the `kubelet` service is stopped.

Uninstalling a module also removes the module configuration from the Platform API Server. If you uninstall a module and want to reinstall it, you need to create the module again using the `olcnectl module create` command.

> **Tip:**
>
> If you reinstall a Kubernetes module on hosts that were previously used in a Kubernetes cluster, you may need to run the `sudo kubeadm reset -f` command on each node before you redeploy the module.