Oracle Cloud Native Environment

Kubernetes Module for Release 1.9

F93851-01
May 2024

Oracle Cloud Native Environment Kubernetes Module for Release 1.9,

F93851-01

Contents

9 Removing a Kubernetes Cluster

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 extra tools, testing, and support to deliver this technology with confidence. Kubernetes integrates with container products to handle more complex deployments where clustering might 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

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/.

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 and includes extra tools developed at Oracle to ease configuration and deployment of a Kubernetes cluster.

Kubernetes Components

You're likely to meet the following common components when you start working with Kubernetes on Oracle Cloud Native Environment. 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's 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 such as the Controller Manager and the Scheduler. Ensure you have a backup plan in place for the data stored in this component of the cluster.

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

The control plane node can be 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's 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 no new operations, such as creating new resources or editing or moving existing resources, can be performed.

A high availability cluster with many control plane nodes ensures that more requests for control plane node functionality can be handled, and with the help of control plane replica nodes, uptime is 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 the 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're needed. If 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 occur. The worker nodes perform any actions triggered by the Kubernetes API, which runs on the control plane node.

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

- **Kubelet Service** (`kubelet`): 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** (`kube-proxy`): 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 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 are to be run together. Pods are used for tightly coupled applications that would typically run on the same logical host and which might 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. You can scale up or down easily by creating or removing replica pods. For more information on pods, see the upstream Kubernetes documentation.

ReplicaSet, Deployment, StatefulSet Controllers

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

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* lets you 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 Kubernetes documentation.

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're always created in the same order and allocated identifiers that are applied to host names and the internal cluster DNS. Those identifiers ensure stable and predictable network identities for pods in the environment. For more information on StatefulSets, see the upstream Kubernetes documentation.

Services

You can use services to expose access to one or more mutually interchangeable pods. As 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 might 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's 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, use this option to expose services for applications that need to access each other from within the cluster.

Often, clients outside of the Kubernetes cluster might need access to services within the cluster. You can achieve this by creating a `NodePort` service type. This service type lets 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 from 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. An external load balancer can handle redirecting traffic to pods directly in the cluster from 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 the network is 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're running `firewalld` on any of the nodes, ensure 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 Kubernetes documentation.

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 provides various volume types that define how the data is stored and how it's persisted, which are described in detail in the upstream Kubernetes documentation.

Kubernetes volumes typically have a lifetime that matches the lifetime of the pod, and data in a volume persists for while 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.

Sometimes, you might 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 use the resources available in a *PersistentVolume*, and the *PersistentVolumeClaim* specifies the quota and access modes to be applied to the resource for a consumer. A pod you have created can then use 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 Oracle Cloud Infrastructure Cloud Controller Manager Module and Rook Module.

Namespaces

Kubernetes implements and maintains strong separation of resources by using 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 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 Kubernetes documentation.

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. CRI-O 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 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 Installation.

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 ensure 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 CNI

You can use the following Kubernetes Container Network Interface (CNI) plugins to manage pod network traffic:

- **Flannel**: Flannel is the default CNI when you create a Kubernetes module. You don't need to set any command options to use Flannel as it's installed by default.
- **Calico**: Calico is an optional CNI you can use instead of Flannel. You can set Calico as the CNI 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 Kubernetes CNI instead of Flannel. A minimum default configuration is used for Calico with this installation method. You can optionally install Calico as a module. To install Calico as a module, you set `--pod-network none` when you create the Kubernetes module so that no CNI is set up when you deploy Kubernetes. For more information on Calico, and how to install it as a module, see Calico Module.

> **Important:**
>
> To set Calico as the Kubernetes CNI, you must first perform the prerequisites in Calico Module

- **Multus**: Multus is an optional CNI that creates a networking bridge to either Flannel or Calico. Multus can be set up using the Multus module after the Kubernetes module is installed. For more information on Multus and how to install the module, see Multus Module.

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 (no HA).

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 Installation.

Extra 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 Installation.

Use the `olcne module create` command to create a Kubernetes module. If you don't include all the required options when using this command, you're 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 a load balancer available on the host `lb.example.com` and listening 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.co
m:8090 \
--worker-nodes
worker1.example.com:8090,worker2.example.com:8090,worker3.example.com:8
090,worker4.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 `--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 might 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 Installation.

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. HA and failover features aren't 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's 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 Installation). The default port number is `8090`.

If SELinux is set to `enforcing` mode (the OS default and the recommended mode) on the control plane node and worker nodes, you must also include 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 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 Installation.

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 Kubernetes CNI for pods is Flannel. You can optionally set the CNI to Calico or to none. Set the pod networking using the `--pod-network` option. Using `--pod-network calico` sets Calico to be the CNI instead of Flannel. Using `--pod-network none` sets no CNI, which lets you install the Calico module. For more information on Calico, see Calico Module.

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. 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,wo
rker4.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 `--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 must be available on the network and must 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're 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 the 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 must 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, 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
```

You can optionally use the `--log-level` option to set the level of logging displayed in the command output. By default, error messages are displayed. For example, you can set the logging level to show all messages when you include:

```
--log-level debug
```

The log messages are also saved as an operation log. You can view operation logs as commands are running, or when they've completed. For more information using operation logs, see Platform Command-Line Interface.

If any validation errors are returned, the commands required to fix the nodes are provided in the output. 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're 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
```

You can optionally use the `--log-level` option to set the level of logging displayed in the command output. By default, error messages are displayed. For example, you can set the logging level to show all messages when you include:

```
--log-level debug
```

The log messages are also saved as an operation log. You can view operation logs as commands are running, or when they've completed. For more information using operation logs, see Platform Command-Line Interface.

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

> **Important:**
>
> Installing Kubernetes might 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 CLI (`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 a *control plane* node of the cluster (the recommended option), although you can set up `kubectl` access on an external node that's not in the cluster, if required. The `kubectl` utility effectively grants full administrative rights to the cluster and all 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 the 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 deployments --all-namespaces
```

The output looks similar to:

```
NAMESPACE                      NAME                            READY   UP-TO-
DATE   AVAILABLE   AGE
externalip-validation-system   externalip-validation-webhook   1/1
1            1           29m
kube-system                    coredns                         2/2
2            2           30m
kubernetes-dashboard           kubernetes-dashboard            1/1
1            1           29m
ocne-modules                   ocne-module-operator            1/1
1            1           29m
```

Setting up kubectl on a Non-Cluster Node

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

The following example shows you how to set up a host that's not in the cluster with `kubectl` to access to a Kubernetes cluster.

> **Note:**
>
> The following example assumes the OS of the 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 also need to install Windows Subsystem for Linux 2 (WLS 2).
>
> To set up `kubectl` on a host that's not in the cluster:

1. 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 host. Log in to the host and copy the `kubeconfig.yaml` from the operator node to a local directory on the host.

   > **Caution:**
   >
   > Follow security best practices when copying a configuration file with sensitive information between hosts.

   a. Create a subdirectory named `.kube` in the home directory:

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

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

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

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

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

d. To permanently set this environment variable, add it to the `.bashrc` file:

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

3. Install `kubectl` on the host.

   Set up the node with the required access to Oracle Cloud Native Environment packages by enabling repositories or channels as required. See Installation for more information.

   Install `kubectl`:

```
sudo dnf install kubectl
```

4. Verify you can use the `kubectl` command:

```
kubectl get deployments --all-namespaces
```

   The output looks similar to:

```
NAMESPACE                      NAME                              READY   UP-
TO-DATE   AVAILABLE   AGE
externalip-validation-system   externalip-validation-webhook     1/1
1            1           29m
kube-system                    coredns                           2/2
2            2           30m
kubernetes-dashboard           kubernetes-dashboard              1/1
1            1           29m
ocne-modules                   ocne-module-operator              1/1
1            1           29m
```

4
Using Kubernetes

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

The `kubectl` utility is fully documented in the upstream Kubernetes documentation.

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 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 Kubernetes resource. In this case, the `nodes` resource:

```
kubectl get nodes
```

The output looks similar to:

```
NAME                  STATUS    ROLES          AGE      VERSION
control.example.com   Ready     control-plane  1h       version
worker1.example.com   Ready     <none>         1h       version
worker2.example.com   Ready     <none>         1h       version
```

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
```

The output looks similar to:

```
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
```

Substitute `nginx` with a container image. Substitute `hello-world` with a name for the deployment. The 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 lowercase alphanumeric characters or `-`, and must start and end with an alphanumeric character. The regular expression that's used to validate names is '`[a-z0-9]([-a-z0-9]*[a-z0-9])?`'. If you use a name for the deployment that doesn't validate, an error is returned.

Many more optional parameters can be used when you run a new application within Kubernetes. For example, at run time, you can specify how many replica pods are to 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 a new application deployment has created one or more pods, use the `kubectl get pods` command:

```
kubectl get pods
```

The output looks similar to:

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

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

```
kubectl describe pods
```

The output looks similar to:

```
Name:               hello-world-5f55779987-wd857
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               worker1.example.com/192.0.2.11
Start Time:         <date> 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're running, you can use the `kubectl scale deployment` command. For example:

```
kubectl scale deployment --replicas=3 hello-world
```

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

```
kubectl get pods
```

The output looks similar to:

```
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 an 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:**
>
> The Oracle Cloud Infrastructure Cloud Controller Manager module is used to create and manage Oracle Cloud Infrastructure load balancers for Kubernetes applications. The following example assumes you have installed this module as described in Oracle Cloud Infrastructure Cloud Controller Manager Module.

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

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

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
```

The output looks similar to:

```
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
```

The output looks similar to:

```
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
  ```

  The output looks similar to:

  ```
  <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:

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

  The output looks similar to:

  ```
  <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 the environment can be cleaned up. 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
```

To delete an entire deployment, and all 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
```

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 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:

```
kubectl get pods --namespace kube-system
```

The output looks similar to:

```
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 elements that consist of the deployment. This deployment defines which images are to be used to generate the containers within the pod, along with any runtime requirements, and Kubernetes networking, and storage requirements in the form of services to be configured and volumes that might need to be mounted.

Deployments are described in detail in the upstream Kubernetes documentation.

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.

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

Starting the Dashboard

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

```
kubectl proxy
```

The output looks similar to:

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

The Dashboard is available on the node where the proxy is running. To exit the proxy, use `Ctrl+C`. When you exit the proxy, it ends the application, and the Dashboard is no longer available.

You can run this as a `systemd` service and enable it so that it's always available after OS reboots:

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

This `systemd` service requires that the `/etc/kubernetes/admin.conf` is present to run. To change the port that's 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 Kubernetes documentation.

Set up a token for the `admin-user` using:

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

Copy and paste the entire value of the token output 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 the localhost to the node running the `kubectl proxy` service. The easiest option is to use SSH tunneling to forward a port on the 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 the SSH configuration. For example, on the 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 the localhost and navigate to:

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

The Dashboard log in screen is displayed 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 don't need to start the Dashboard using the `kubectl-proxy` service as it's 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
```

The output looks similar to:

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

A Kubernetes Dashboard service is also deployed. You can show that service using:

```
kubectl get svc --namespace kubernetes-dashboard
```

The output looks similar to:

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

To access this service, 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

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

Reasons for scaling up a cluster might include the need to handle a larger workload, increased network traffic, or the need to run more applications in the cluster. Reasons for scaling down a cluster might include temporarily removing a node for maintenance or troubleshooting.

Before adding a new node, you need to set up the node to meet all the necessary requirements for it to be part of the Kubernetes cluster. For information on setting up a Kubernetes node, see Installation. Depending upon the type of nodes you're adding to the cluster, and the system setup, you might also need to add the new nodes to the load balancer configured for the cluster.

An environment with a single control plane node (created for example, for testing purposes) can be scaled up to a highly available (HA) cluster with many control plane nodes, providing it was created with the `--load-balancer` or `--virtual-ip` options.

When you scale a Kubernetes cluster:

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 can't proceed, and the nodes can't be added to the cluster. You're prompted for what to do to resolve the validation issues so that the nodes can be added to the cluster.
3. The control plane nodes and worker nodes are added to or removed from the cluster.
4. The cluster is checked to ensure all nodes are healthy. After validation of the cluster is completed, the cluster is scaled and you can access it.

Best Practices for Scaling a Kubernetes Cluster

The following list describes best practices to be followed when scaling a Kubernetes cluster in a production environment:

Scale Up and Down in Separate Steps
We recommend that you don't scale the cluster up and down in one step: scale up, and then scale down, in two separate commands.

Scaling Control Plane Nodes
To avoid split-brain scenarios, the number of control plane nodes in a cluster must always be an odd number equal to or greater than three, for example, 3, 5, or 7. Thus, control nodes must be scaled up and down two nodes at a time to maintain cluster quorum.

We recommend that clusters are provisioned with a minimum of five control plane nodes in case two nodes need to be removed during a maintenance operation.

Scaling Worker Nodes
Replace worker nodes in the cluster one node at a time to let applications running on the node to migrate to other nodes.
The cluster must always have a minimum of three worker nodes. Thus, we recommend that clusters are provisioned with a minimum of four worker nodes in case a node is removed during a maintenance operation.

> **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 don't need to provide the worker node list). 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, you must set up the new nodes so they can be added to the cluster. Also, depending upon the type of nodes you're adding to the cluster, and the system setup, you might also need to add the new nodes to the load balancer configured for the cluster.

Setting up the New Kubernetes Nodes

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 Installation.
2. If you're using private X.509 certificates for nodes, you need to generate and copy the certificates to the node. You don't need to do anything if you're using Vault to provide certificates for nodes. For information using X.509 certificates see Installation.
3. Start the Platform Agent service. For information on starting the Platform Agent, see Installation.

Adding New Nodes to the Load Balancer

If you're using an external load balancer for the Kubernetes cluster (set with the `--load-balancer` option when you created the Kubernetes module), add any new control plane nodes to it. If you're using an Oracle Cloud Infrastructure load balancer, add any new control plane nodes to the appropriate backend set and set the port for the control plane nodes to `6443`. If you're using the load balancer deployed by the Platform CLI (set with the `--virtual-ip option` when you created the Kubernetes module), you

don't need to add the control plane nodes to it. This is done automatically when you scale the nodes into the cluster.

If you have the Istio module installed and set up with a load balancer for the Istio ingress gateway, and you're adding new worker nodes, add the new worker nodes to the Istio egress load balancer. If you're using an Oracle Cloud Infrastructure load balancer, add any new worker nodes to the appropriate backend set.

Adding New Nodes to the Kubernetes Cluster

After completing the preparatory steps in the preceding sections, 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 nodes and worker nodes of the cluster.

   ```
   kubectl get nodes
   ```

   The output looks similar to:

   ```
   NAME                   STATUS   ROLE           AGE     VERSION
   control1.example.com    Ready    control-plane   26h     version
   control2.example.com    Ready    control-plane   26h     version
   control3.example.com    Ready    control-plane   26h     version
   worker1.example.com     Ready    <none>          26h     version
   worker2.example.com     Ready    <none>          26h     version
   worker3.example.com     Ready    <none>          26h     version
   ```

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

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

   Three worker nodes are also 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 the recommended minimum of five control plane nodes and four worker nodes. This example adds two new control plane nodes (`control4.example.com` and `control5.example.com`) and one new worker node (`worker4.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:8
   ```

```
090,\
control4.example.com:8090,control5.example.com:8090 \
--worker-nodes
worker1.example.com:8090,worker2.example.com:8090,worker3.example.co
m:8090,\
worker4.example.com:8090
```

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 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.

You can optionally use the `--log-level` option to set the level of logging displayed in the command output. By default, error messages are displayed. For example, you can set the logging level to show all messages when you include:

```
--log-level debug
```

The log messages are also saved as an operation log. You can view operation logs as commands are running, or when they've completed. For more information using operation logs, see Platform Command-Line Interface.

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

```
kubectl get nodes
```

The output looks similar to:

```
NAME                  STATUS   ROLE           AGE     VERSION
control1.example.com   Ready   control-plane   26h     version
control2.example.com   Ready   control-plane   26h     version
control3.example.com   Ready   control-plane   26h     version
control4.example.com   Ready   control-plane   2m38s   version
control5.example.com   Ready   control-plane   2m38s   version
worker1.example.com    Ready   <none>          26h     version
worker2.example.com    Ready   <none>          26h     version
worker3.example.com    Ready   <none>          26h     version
worker4.example.com    Ready   <none>          2m38s   version
```

Scaling Down a Kubernetes Cluster

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

NOT_SUPPORTED:

Be careful if you're scaling down the control plane nodes of the 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 nodes and worker nodes of the cluster.

   ```
   kubectl get nodes
   ```

   The output looks similar to:

   ```
   NAME                  STATUS   ROLE           AGE     VERSION
   control1.example.com  Ready    control-plane  26h     version
   control2.example.com  Ready    control-plane  26h     version
   control3.example.com  Ready    control-plane  26h     version
   control4.example.com  Ready    control-plane  2m38s   version
   control5.example.com  Ready    control-plane  2m38s   version
   worker1.example.com   Ready    <none>         26h     version
   worker2.example.com   Ready    <none>         26h     version
   worker3.example.com   Ready    <none>         26h     version
   worker4.example.com   Ready    <none>         2m38s   version
   ```

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

   - `control1.example.com`
   - `control2.example.com`
   - `control3.example.com`
   - `control4.example.com`
   - `control5.example.com`

   Four worker nodes are also in the cluster:

   - `worker1.example.com`
   - `worker2.example.com`
   - `worker3.example.com`
   - `worker4.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 two control plane nodes (`control4.example.com` and `control5.example.com`) and one worker node

(`worker4.example.com`) from 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 \
--worker-nodes
worker1.example.com:8090,worker2.example.com:8090,worker3.example.co
m:8090
```

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

   ```
   kubectl get nodes
   ```

   The output looks similar to:

   ```
   NAME                  STATUS   ROLE           AGE     VERSION
   control1.example.com  Ready    control-plane  26h     version
   control2.example.com  Ready    control-plane  26h     version
   control3.example.com  Ready    control-plane  26h     version
   worker1.example.com   Ready    <none>         26h     version
   worker2.example.com   Ready    <none>         26h     version
   worker3.example.com   Ready    <none>         26h     version
   ```

4. The removed nodes can be added back into the cluster in a scale-up operation after any necessary maintenance has been completed. However, if the nodes are to be replaced by new ones, then you might need to remove the old nodes from the load balancer. For information on load balancers, see Adding New Nodes to the Load Balancer.

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 a Kubernetes cluster against control plane node failures is important, especially for clusters with only one control plane node. High availability clusters with many control plane nodes also need a fallback plan if the resilience provided by the replication and failover functionality has been exceeded.

You don't need to bring down the cluster to perform a back up as part of a 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 the 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`*

Restoring Control Plane Nodes

These restore steps are intended for use when a Kubernetes cluster must be reconstructed as part of a planned disaster recovery scenario. Unless a total cluster failure occurs, you don't need to manually recover individual control plane nodes in a high availability cluster as it can self-heal with replication and failover.

To restore a control plane node, you must have an existing Oracle Cloud Native Environment, and have deployed the Kubernetes module. You can't restore to an environment that doesn't exist.

To restore a control plane node:

1. Ensure 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 the 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 might be prompted by the Platform CLI to perform extra set up steps on the control plane nodes to fulfill the prerequisite requirements. Follow any 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, to list the nodes, use:

   ```
   kubectl get nodes
   ```

   And to list the pods running in the `kube-system` namespace, use:

   ```
   kubectl get pods --namespace kube-system
   ```

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 Kubernetes documentation.

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
```

The output looks similar to:

```
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 before deploying Kubernetes. You can use certificates generated by Vault, CA Certificates, or generate certificates using the `gen-certs-helper.sh` script. For information on setting up these certificates, see Installation.

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
```

Changing Access to CIDR Blocks

If you have a Kubernetes module that has CIDR blocks configured to be allowed, you can change this configuration using the `--restrict-service-externalip-cidrs` option of the `olcnectl module update` command. This lets you 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

To restrict Kubernetes services from accessing any `externalIPs`, don't you set any CIDR blocks that are allowed when you create the Kubernetes module. So, don't use the `--restrict-service-externalip-cidrs` option of the `olcnectl module create` command. The `externalip-validation-webhook-service` Kubernetes service is deployed, but doesn't 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 configured CIDR blocks, you update the module and set the `--restrict-service-externalip-cidrs` option to null as shown in Changing Access to CIDR Blocks.

Enabling Access to all externalIPs

If you want all Kubernetes services 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 isn't 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. Changing a Kubernetes module in this way removes the `externalip-validation-webhook-service` Kubernetes service from the cluster, so validation isn't 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
Removing a Kubernetes Cluster

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 used in a previous Kubernetes cluster, run the `sudo kubeadm reset -f` command on each node before you redeploy the module.