Oracle® Linux Cloud Native Environment

Getting Started

F22532-10
September 2020

Oracle Legal Notices

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Table of Contents

Preface

This document contains information about Oracle Linux Cloud Native Environment. It includes information on installing and configuring Oracle Linux Cloud Native Environment and deploying a Kubernetes cluster.

Document generated on: 2020-09-17 (revision: 802)

Audience

This document is written for system administrators and developers who want to use Oracle Linux Cloud Native Environment. It is assumed that readers have a general understanding of the Oracle Linux operating system and container concepts.

Related Documents

The latest version of this document and other documentation for this product are available at:

https://docs.oracle.com/en/operating-systems/olcne/

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

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.

Chapter 1 Introduction to Oracle Linux Cloud Native Environment

Oracle Linux Cloud Native Environment is a fully integrated suite for the development and management of cloud-native applications. Based on the Open Container Initiative (OCI) and Cloud Native Computing Foundation (CNCF) standards, Oracle Linux Cloud Native Environment delivers a simplified framework for installations, updates, upgrades and configuration of key features for orchestrating microservices.

Kubernetes is used to deploy and manage containers. In addition to Kubernetes master and worker nodes, the Oracle Linux Cloud Native Environment Platform API Server (Platform API Server) must be installed on a server, and is needed to perform a deployment and manage modules. The term *module* in this document refers to a packaged software component that can be deployed to provide both core and optional cluster-wide functionality. The Kubernetes module for Oracle Linux Cloud Native Environment is the core module, and automatically installs and configures CRI-O, runC and Kata Containers.

The Kubernetes master and worker nodes run an Oracle Linux Cloud Native Environment Platform Agent (Platform Agent). The Platform Agent communicates with the Platform API Server to manage the deployment.

The Oracle Linux Cloud Native Environment Platform Command-Line Interface (Platform CLI) performs the validation and deployment of modules to the nodes, enabling easy deployment of modules such as the Kubernetes module. The required software for modules is configured by the Platform CLI, such as CRI-O, runC, Kata Containers, CoreDNS and Flannel.

This chapter provides introductory information about Oracle Linux Cloud Native Environment.

1.1 About the Oracle Linux Cloud Native Environment

The Oracle Linux Cloud Native Environment is a curated set of applications that deliver a production-ready cloud-native application environment, and includes deployment and management tools.

1.2 Oracle Linux Cloud Native Environment Components

This section contains information on the components that are used to create Oracle Linux Cloud Native Environment.

1.2.1 Container Runtimes

Containers are the fundamental infrastructure to deploy modern cloud applications. Oracle delivers the tools to create and provision Open Container Initiative (OCI)-compliant containers using CRI-O.

CRI-O, an implementation of the Kubernetes CRI (Container Runtime Interface) to enable using Open Container Initiative compatible runtimes, is included with Oracle Linux Cloud Native Environment. CRI-O allows you to run either runC or Kata Containers containers directly from Kubernetes, without any unnecessary code or tooling.

1.2.2 Container Orchestration

The Kubernetes module is based on the upstream Kubernetes project, and released under the CNCF Kubernetes Certified Conformance program. The Platform API Server simplifies the configuration and set up of the Kubernetes module, with support for backup and recovery. The Kubernetes module is developed for Oracle Linux and integrates with CRI-O to provide a comprehensive container and orchestration environment for the delivery of microservices and next-generation application development.

1.2.3 Cloud Native Networking

CNCF project Flannel provides the overlay network used by Kubernetes, and simplifies container-to-container networking.

The Container Network Interface (CNI) project, currently incubating under CNCF, seeks to simplify networking for container workloads by defining a common network interface for containers. The CNI plug-in is included with Oracle Linux Cloud Native Environment.

1.2.4 Cloud Native Storage

There are a number of storage projects associated with the CNCF foundation, and several providers are included by default in Kubernetes, including a plug-in for Gluster Storage for Oracle Linux.

Storage integration is provided through the use of plug-ins, referred to as the Container Storage Interface (CSI). The plug-ins adhere to a standard specification.

1.3 Oracle Linux Cloud Native Environment Architecture

Oracle Linux Cloud Native Environment is built from a number of discrete components. You interact with the environment directly using the Platform CLI. The Platform API Server interacts with the Platform Agent on each Kubernetes node. The Platform Agent is responsible for handling host-level operations on behalf of the Platform API Server.

1.3.1 Platform API Server

The Platform API Server performs the business logic. The Platform API Server manages all entities, from hosts to microservices. The Platform API Server is responsible for managing the state of the environment, including the deployment and configuration of modules to one or more nodes in a cluster.

1.3.2 Platform Agent

The Platform Agent runs on each host to proxy requests from the Platform API Server to small worker applications. The primary reason for this is to make sure the Platform Agent process uses as little memory as possible. In this book, the Platform Agent refers to the union of the Platform Agent process and associated worker applications.

The Platform Agent knows how to gather the state of resources on its host, and to change the state of those resources. That is, the Platform Agent knows if a firewall port is open or closed, or if a package is installed and at which version. It also knows how to close that port if it is open, upgrade the package if it is old, or install the package if it is not installed.

1.3.3 Platform CLI

The Platform CLI is used to communicate with the Platform API Server. The Platform CLI is a simple application (the `olcnectl` command) that converts the input to Platform API Server calls. No business logic takes place in the Platform CLI. The Platform CLI must be installed on an *operator* node. The operator node is used to deploy an environment.

Chapter 2 Oracle Linux Cloud Native Environment Prerequisites

This chapter describes the prerequisites for the systems to be used in an installation of Oracle Linux Cloud Native Environment. This chapter also discusses how to enable the repositories to install the Oracle Linux Cloud Native Environment packages.

2.1 Enabling Access to the Oracle Linux Cloud Native Environment Packages

The Oracle Linux Cloud Native Environment packages are available on the Oracle Linux yum server in the `ol7_olcne11` repository, or on the Unbreakable Linux Network (ULN) in the `ol7_x86_64_olcne11` channel, however there are also dependencies across other repositories and channels, and these must also be enabled on each system where Oracle Linux Cloud Native Environment is installed.

Warning

Oracle does not support Kubernetes on systems where the `ol7_preview`, `ol7_developer` or `ol7_developer_EPEL` yum repositories or ULN channels are enabled, or where software from these repositories or channels is currently installed on the systems where Kubernetes runs. Even if you follow the instructions in this document, you may render your platform unsupported if these repositories or channels are enabled or software from these channels or repositories is installed on your system.

2.1.1 Enabling Channels with ULN

If you are registered to use ULN, use the ULN web interface to subscribe the system to the appropriate channels.

To subscribe to the ULN channels:

1. Log in to https://linux.oracle.com with your ULN user name and password.
2. On the Systems tab, click the link named for the system in the list of registered machines.
3. On the System Details page, click **Manage Subscriptions**.
4. On the System Summary page, select each required channel from the list of available channels and click the right arrow to move the channel to the list of subscribed channels. Subscribe the system to the following channels:
   - `ol7_x86_64_olcne11`
   - `ol7_x86_64_kvm_utils`
   - `ol7_x86_64_addons`
   - `ol7_x86_64_latest`
   - `ol7_x86_64_UEKR5`
5. Click **Save Subscriptions**.

2.1.2 Enabling Repositories with the Oracle Linux Yum Server

If you are using the Oracle Linux yum server for system updates, enable the required yum repositories.

To enable the yum repositories:

1. Install the `oracle-olcne-release-el7` release package to install the Oracle Linux Cloud Native Environment yum repository configuration.

   ```
   $ sudo yum install oracle-olcne-release-el7
   ```

2. Enable the following yum repositories:

   - `ol7_olcne11`
   - `ol7_kvm_utils`
   - `ol7_addons`
   - `ol7_latest`
   - `ol7_UEKR5`

   Use the `yum-config-manager` tool to enable the yum repositories:

   ```
   $ sudo yum-config-manager --enable ol7_olcne11 ol7_kvm_utils ol7_addons ol7_latest ol7_UEKR5
   ```

3. Make sure the `ol7_olcne` yum repository is disabled:

   ```
   $ sudo yum-config-manager --disable ol7_olcne
   ```

2.2 Accessing the Container Registry

The container images that are deployed by the Platform CLI are hosted on the Oracle Container Registry at:

https://container-registry.oracle.com/

For more information about the Oracle Container Registry, see the *Oracle® Linux: Oracle Container Runtime for Docker User's Guide*.

For a deployment to use the Oracle Container Registry, each node within the environment must be provisioned with direct access to the Internet. In some cases, nodes within your environment may not be provisioned with direct access to the Internet. In these cases, you can set up a local container registry mirror. Each node requires direct access to the mirror host in this scenario.

Creating a Container Registry Mirror

1. Select a host to use for your container registry mirror service. The mirror host must have access to the Internet and should be able to pull images directly from the Oracle Container Registry, or alternately should have access to the correct image files stored locally. Ideally, the host should not be a node within your Oracle Linux Cloud Native Environment, but should be accessible to all of the nodes that are part of the environment.

   On the mirror host, install Oracle Container Runtime for Docker, and set up a Docker registry container, following the instructions in the *Oracle® Linux: Oracle Container Runtime for Docker User's Guide*.

2. On the mirror host, install the `olcne-utils` package, so you have access to the registry mirroring utility.

   ```
   $ sudo yum install olcne-utils
   ```

3. Copy the required container images from the Oracle Container Registry to the local Docker registry using the `registry-image-helper.sh` script with the required options:

```
$ registry-image-helper.sh --to host.example.com:5000/olcne
```

Where *host.example.com:5000* is the resolvable domain name and port on which your local Docker registry is available.

If the host where you are running the script does not have access to the Internet, you can replace the `--from` option with the `--local` option to load the container images directly from a local directory. The local directory which contains the images should be either `/usr/local/share/kubeadm/` or `/usr/local/share/olcne/`. The image files should be archives in TAR format. All TAR files in the directory are loaded into Docker when the script is run with this option.

You can use the `--version` option to specify the Kubernetes version you want to mirror. If not specified, the current release is used.

The script also accepts an `--images` option that can be used to specify a comma-separated list of image and tag names, if you need to mirror a custom set of images.

4. When you use a registry mirror, make sure you set the `--container-registry` option to the domain name and port for the registry mirror. You set this option when you create a Kubernetes module using the `olcnectl module create` command.

2.3 Software Requirements

The following sections describe the requirements that must be met to install and configure Oracle Linux Cloud Native Environment on Oracle Linux 7 systems.

2.3.1 Setting up a Network Time Service

As a clustering environment, Oracle Linux Cloud Native Environment requires that the system time is synchronized across each Kubernetes master and worker node within the cluster. Typically, this can be achieved by installing and configuring a Network Time Protocol (NTP) daemon on each node. Oracle recommends installing and setting up the chronyd daemon for this purpose.

To set up chronyd:

1. On each Kubernetes master and worker node, install the `chrony` package, if it is not already installed:

```
$ sudo yum install chrony
```

2. Edit the NTP configuration in `/etc/chrony.conf`. Your requirements may vary. If you are using DHCP to configure the networking for each node, it is possible to configure NTP servers automatically. If you have not got a locally configured NTP service that your systems can sync to, and your systems have Internet access, you can configure them to use the public `pool.ntp.org` service. See https://www.ntppool.org/.

3. Make sure NTP is enabled to restart at boot and that it is started before you proceed with the Oracle Linux Cloud Native Environment installation. For example:

```
$ sudo systemctl enable --now chronyd
```

For information on configuring a Network Time Service, see the *Oracle® Linux 7: Administrator's Guide*.

2.3.2 Disabling Swap

You must disable swap on the Kubernetes master and worker nodes. To disable swap, enter:

```
$ sudo swapoff -a
```

To make this permanent over reboots, edit the `/etc/fstab` file to remove or comment out any swap disks.

2.3.3 Setting SELinux to Permissive

The Platform CLI checks whether SELinux is set to enforcing mode on the Kubernetes master and worker nodes. If enforcing mode is enabled, the Platform CLI exits with an error requesting that you set SELinux to permissive mode. Setting SELinux to permissive mode allows containers to access the host file system, which is required by pod networks.

To disable SELinux enforcing mode, enter:

```
$ sudo sed -i 's/^SELINUX=enforcing/SELINUX=permissive/' /etc/selinux/config
$ sudo /usr/sbin/setenforce 0
```

2.3.4 Setting up the Firewall Rules

Oracle Linux 7 installs and enables `firewalld`, by default. The Platform CLI notifies you of any rules that you may need to add during the deployment of the Kubernetes module. The Platform CLI also provides the commands to run to modify your firewall configuration to meet the requirements.

Make sure that all required ports are open. The ports required for a Kubernetes deployment are:

- 2379/tcp: Kubernetes etcd server client API (on master nodes in multi-master deployments)
- 2380/tcp: Kubernetes etcd server client API (on master nodes in multi-master deployments)
- 6443/tcp: Kubernetes API server (master nodes)
- 8090/tcp: Platform Agent (master and worker nodes)
- 8091/tcp: Platform API Server (operator node)
- 8472/udp: Flannel overlay network, VxLAN backend (master and worker nodes)
- 10250/tcp: Kubernetes `kubelet` API server (master and worker nodes)
- 10251/tcp: Kubernetes `kube-scheduler` (on master nodes in multi-master deployments)
- 10252/tcp: Kubernetes `kube-controller-manager` (on master nodes in multi-master deployments)
- 10255/tcp: Kubernetes `kubelet` API server for read-only access with no authentication (master and worker nodes)

The commands to open the ports and to set up the firewall rules are provided below.

2.3.4.1 Single Master Firewall Rules

For a single master deployment, the following ports are required to be open in the firewall.

Operator Node

On the operator node, run:

```
$ sudo firewall-cmd --add-port=8091/tcp --permanent
```

Restart the firewall for these rules to take effect:

```
$ sudo systemctl restart firewalld
```

Worker Nodes

On the Kubernetes worker nodes run:

```
$ sudo firewall-cmd --zone=trusted --add-interface=cni0 --permanent
$ sudo firewall-cmd --add-port=8090/tcp --permanent
$ sudo firewall-cmd --add-port=10250/tcp --permanent
$ sudo firewall-cmd --add-port=10255/tcp --permanent
$ sudo firewall-cmd --add-port=8472/udp --permanent
```

Restart the firewall for these rules to take effect:

```
$ sudo systemctl restart firewalld
```

Master Nodes

On the Kubernetes master nodes run:

```
$ sudo firewall-cmd --zone=trusted --add-interface=cni0 --permanent
$ sudo firewall-cmd --add-port=8090/tcp --permanent
$ sudo firewall-cmd --add-port=10250/tcp --permanent
$ sudo firewall-cmd --add-port=10255/tcp --permanent
$ sudo firewall-cmd --add-port=8472/udp --permanent
$ sudo firewall-cmd --add-port=6443/tcp --permanent
```

Restart the firewall for these rules to take effect:

```
$ sudo systemctl restart firewalld
```

2.3.4.2 Multi-Master Firewall Rules

For a multi-master deployment, the following **additional** ports are required to be open in the firewall on the master node.

On the Kubernetes master nodes run:

```
$ sudo firewall-cmd --add-port=10251/tcp --permanent
$ sudo firewall-cmd --add-port=10252/tcp --permanent
$ sudo firewall-cmd --add-port=2379/tcp --permanent
$ sudo firewall-cmd --add-port=2380/tcp --permanent
```

Restart the firewall for these rules to take effect:

```
$ sudo systemctl restart firewalld
```

2.3.5 Setting up Other Network Options

This section contains information on other network related configuration that affects an Oracle Linux Cloud Native Environment deployment. You may not need to make changes from this section, but they are provided to help you understand any issues you may encounter related to network configuration.

2.3.5.1 Internet Access

The Platform CLI checks it is able to access the container registry, and possibly other Internet resources, to be able to pull any required container images. Unless you intend to set up a local registry mirror for container images, the systems where you intend to install Oracle Linux Cloud Native Environment must either have direct internet access, or must be configured to use a proxy.

2.3.5.2 Flannel Network

The Platform CLI configures a flannel network as the network fabric used for communications between Kubernetes pods. This overlay network uses VxLANs to facilitate network connectivity. For more information on flannel, see the upstream documentation at:

https://github.com/coreos/flannel

By default, the Platform CLI creates a network in the `10.244.0.0/16` range to host this network. The Platform CLI provides an option to set the network range to an alternate range, if required, during installation. Systems in an Oracle Linux Cloud Native Environment deployment must not have any network devices configured for this reserved IP range.

2.3.5.3 br_netfilter Module

The Platform CLI checks whether the `br_netfilter` module is loaded and exits if it is not available. This module is required to enable transparent masquerading and to facilitate Virtual Extensible LAN (VxLAN) traffic for communication between Kubernetes pods across the cluster. If you need to check whether it is loaded, run:

```
$ sudo lsmod|grep br_netfilter
br_netfilter           24576  0
bridge                155648  2 br_netfilter,ebtable_broute
```

If you see the output similar to shown, the `br_netfilter` module is loaded. Kernel modules are usually loaded as they are needed, and it is unlikely that you need to load this module manually. If necessary, you can load the module manually and add it as a permanent module by running:

```
$ sudo modprobe br_netfilter
$ sudo sh -c 'echo "br_netfilter" > /etc/modules-load.d/br_netfilter.conf'
```

2.3.5.4 Bridge Tunable Parameters

Kubernetes requires that packets traversing a network bridge are processed for filtering and for port forwarding. To achieve this, tunable parameters in the kernel bridge module are automatically set when the `kubeadm` package is installed and a sysctl file is created at `/etc/sysctl.d/k8s.conf` that contains the following lines:

```
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
```

If you modify this file, or create anything similar yourself, run the following command to load the bridge tunable parameters:

```
$ sudo /sbin/sysctl -p /etc/sysctl.d/k8s.conf
```

Chapter 3 Installing Oracle Linux Cloud Native Environment

This chapter discusses how to prepare the nodes to be used in an Oracle Linux Cloud Native Environment deployment. When the nodes are prepared, they must be installed with the Oracle Linux Cloud Native Environment software packages. When the nodes are set up with the software, you can use the Platform CLI to perform a deployment of a Kubernetes cluster. This chapter shows you how to perform the steps to set up the hosts and install the Oracle Linux Cloud Native Environment software, ready to perform a deployment.

3.1 Installation Overview

The high level overview of setting up Oracle Linux Cloud Native Environment is described in this section.

To install Oracle Linux Cloud Native Environment:

1. Prepare the operator node.

   An *operator* node is a host that is used to perform and manage the deployment of environments. The operator node must be set up with the Platform API Server, and the Platform CLI (`olcnectl`).

2. Prepare the Kubernetes nodes.

   The Kubernetes master and worker nodes must to be set up with the Platform Agent.

3. Set up a load balancer.

   If you are deploying a multi-master, highly available Kubernetes cluster, set up a load balancer. You can set up your own load balancer, or use the container-based load balancer deployed by the Platform CLI.

4. Set up X.509 Certificates.

   X.509 Certificates are used to provide secure communication between the Kubernetes nodes. You must set up the certificates before you create an environment and perform a deployment.

5. Start the Services.

   Start the Platform API Server and Platform Agent services on nodes using the X.509 Certificates.

3.2 Introduction to Environments

An *environment* is a namespace that encapsulates the software installed and managed by Oracle Linux Cloud Native Environment. Each environment contains at least the Kubernetes module.

The Platform CLI allows you to create and manage multiple deployments. Each deployment contains an environment, and each environment may potentially contain multiple modules.

3.3 Introduction to Modules

A *module* is a curated unit of software that can be installed and managed by Oracle Linux Cloud Native Environment. A module fulfills at least one specific role in a deployment. Modules that fulfill the same roles may be swapped out in a managed way. Modules may encapsulate other modules.

The available modules are:

- Kubernetes

- Istio
- Helm
- Prometheus

3.3.1 The Kubernetes Module

The core module in an Oracle Linux Cloud Native Environment deployment is Kubernetes. For more information about Kubernetes, see *Container Orchestration*.

The Kubernetes module includes sub-components, such as:

- **Flannel**: The default overlay network for a Kubernetes cluster.
- **CoreDNS**: The DNS server for a Kubernetes cluster.
- **CRI-O**: Manages the container runtime for a Kubernetes cluster.
- **runC**: The default container runtime.
- **Kata Containers**: An optional lightweight virtual machine runtime.

3.3.2 The Istio Module

Istio is a fully featured service mesh for microservices in Kubernetes clusters. Istio can handle most aspects of microservice management, for example, identity, authentication, transport security, metric scraping.

The Istio module for Oracle Linux Cloud Native Environment installs Istio into a Kubernetes module (cluster), and uses a Helm module for deployment.

For more information about installing and using the Istio module module, see *Service Mesh*.

3.3.3 The Helm Module

Helm is a package manager for Kubernetes. Helm simplifies the task of deploying and managing software inside Kubernetes clusters. Helm uses *charts* to manage the packages that it can deploy.

The Helm module for Oracle Linux Cloud Native Environment installs Helm into a Kubernetes module (cluster). The Helm module is used by the Platform API Server to install additional modules including the Istio and Prometheus modules.

Note

In this release, the Helm module should only be used in the context of an Istio module deployment.

3.3.4 The Prometheus Module

Prometheus is a systems monitoring and alerting toolkit that collects and stores metrics and other time series data from various sources and presents it in an easily retrievable manner.

The Prometheus module for Oracle Linux Cloud Native Environment is pre-configured with rich monitoring of important systems inside a Kubernetes cluster.

The Prometheus module is deployed by the Helm module into a Kubernetes cluster. The Prometheus module is required by the Istio module.

Note

In this release, the Prometheus module should only be used in the context of an Istio module deployment.

3.4 Setting up the Nodes

This section discusses setting up nodes to use in an Oracle Linux Cloud Native Environment. The nodes are used to form a Kubernetes cluster.

An *operator* node should be used to perform the deployment of the Kubernetes cluster using the Platform CLI and the Platform API Server. An operator node may be a node in the Kubernetes cluster, or a separate host. In examples in this book, the operator node is a separate host, and not part of the Kubernetes cluster.

On each Kubernetes node (both master and worker nodes) the Platform Agent must be installed. Before you set up the Kubernetes nodes, you must prepare them. For information on preparing the nodes, see Chapter 2, *Oracle Linux Cloud Native Environment Prerequisites*.

During the installation of the required packages on, an `olcne` user is created. This user is used to start the Platform API Server or Platform Agent services and has the minimum operating system privileges to perform that task. The `olcne` user should not be used for any other purpose.

3.4.1 Setting up the Operator Node

This section discusses setting up the operator node. The *operator* node is a host that is used to perform and manage the deployment of environments, including deploying the Kubernetes cluster.

To set up the operator node:

1. On the operator node, install the Platform CLI, Platform API Server, and utilities:

```
$ sudo yum install olcnectl olcne-api-server olcne-utils
```

2. Enable the `olcne-api-server` service, but do *not* start it. The `olcne-api-server` service is started when you configure the X.509 Certificates.

```
$ sudo systemctl enable olcne-api-server.service
```

For information on configuration options for the Platform API Server, see Section 6.1, “Configuring the Platform API Server”.

3.4.2 Setting up Kubernetes Nodes

This section discusses setting up the nodes to use in a Kubernetes cluster. Perform these steps on both Kubernetes master and worker nodes.

To set up the Kubernetes nodes:

1. On each node to be added to the Kubernetes cluster, install the Platform Agent package and utilities.

```
$ sudo yum install olcne-agent olcne-utils
```

2. Enable the `olcne-agent` service, but do *not* start it. The `olcne-agent` service is started when you configure the X.509 Certificates.

```
$ sudo systemctl enable olcne-agent.service
```

For information on configuration options for the Platform Agent, see Section 6.2, “Configuring the Platform Agent”.

3. If you use a proxy server, configure it with CRI-O. On each Kubernetes node, create a CRI-O `systemd` configuration directory:

```
$ sudo mkdir /etc/systemd/system/crio.service.d
```

   Create a file named `proxy.conf` in the directory, and add the proxy server information. For example:

```
[Service]
Environment="HTTP_PROXY=proxy.example.com:3128"
Environment="HTTPS_PROXY=proxy.example.com:3128"
Environment="NO_PROXY=mydomain.example.com"
```

4. If the `docker` service is running, stop and disable it.

```
$ sudo systemctl disable --now docker.service
```

5. If the `containerd` service is running, stop and disable it.

```
$ sudo systemctl disable --now containerd.service
```

3.4.3 Setting up a Load Balancer

There are two methods of setting up a load balancer to enable high availability of a multi-master Kubernetes cluster:

- Using your own external load balancer instance.
- Using the load balancer that can be deployed by the Platform CLI on the master nodes.

If you want to use your own load balancer implementation, it should be set up and ready to use before you perform a multi-master deployment. The load balancer hostname and port is entered as an option when you create the Kubernetes module. For more information on setting up your own load balancer, see the *Oracle® Linux 7: Administrator's Guide*, or *Oracle® Linux 8: Setting Up Load Balancing*.

If you want to use the in-built load balancer that can be deployed by the Platform CLI, you need to perform the following steps to prepare the master nodes. These steps should be performed on each master node.

To prepare master nodes for the load balancer deployed by the Platform CLI:

1. Set up the master nodes as described in Section 3.4.2, “Setting up Kubernetes Nodes”.
2. Nominate a virtual IP address that can be used for the primary master node. This IP address should not be in use on any node, and is assigned dynamically to the master node assigned the primary role by the load balancer. If the primary node fails, the load balancer reassigns the virtual IP address to another master node, and that, in turn, becomes the primary node. The virtual IP address used in examples in this documentation is `192.0.2.100`.
3. Open port 6444. When you use a virtual IP address, the Kubernetes API server port is changed from the default of 6443 to 6444. The load balancer listens on port 6443 and receives the requests and passes them to the Kubernetes API server.

```
$ sudo firewall-cmd --add-port=6444/tcp
$ sudo firewall-cmd --add-port=6444/tcp --permanent
```

4. Enable the Virtual Router Redundancy Protocol (VRRP) protocol:

```
$ sudo firewall-cmd --add-protocol=vrrp
$ sudo firewall-cmd --add-protocol=vrrp --permanent
```

5. If you use a proxy server, configure it with NGINX. On each Kubernetes master node, create an NGINX `systemd` configuration directory:

```
$ sudo mkdir /etc/systemd/system/olcne-nginx.service.d
```

Create a file named `proxy.conf` in the directory, and add the proxy server information. For example:

```
[Service]
Environment="HTTP_PROXY=proxy.example.com:3128"
Environment="HTTPS_PROXY=proxy.example.com:3128"
Environment="NO_PROXY=mydomain.example.com"
```

3.5 Setting up X.509 Certificates

Communication between the Kubernetes nodes is secured using X.509 certificates.

Before you deploy Kubernetes, you need to configure the X.509 certificates used to manage the communication between the nodes. There are a number of ways to manage and deploy the certificates. You can use:

- **Vault**: The certificates are managed using the HashiCorp Vault secrets manager. Certificates are created *during* the deployment of the Kubernetes module. You need to create a token authentication method for Oracle Linux Cloud Native Environment.
- **CA Certificates**: Use your own certificates, signed by a trusted Certificate Authority (CA), and copied to each Kubernetes node *before* the deployment of the Kubernetes module. These certificates are unmanaged and must be renewed and updated manually.
- **Private CA Certificates**: Using generated certificates, signed by a private CA you set up, and copied to each Kubernetes node *before* the deployment of the Kubernetes module. These certificates are unmanaged and must be renewed and updated manually. A script is provided to help you set this up.

A software-based secrets manager is recommended to manage these certificates. The HashiCorp Vault secrets manager can be used to generate, assign and manage the certificates. Oracle recommends you implement your own instance of Vault, setting up the appropriate security for your environment.

For more information on installing and setting up Vault, see the HashiCorp documentation at:

https://learn.hashicorp.com/vault/operations/ops-deployment-guide

If you do not want to use Vault, you can use your own certificates, signed by a trusted CA, and copied to each node. A script is provided to generate a private CA which allows you to generate certificates for each node. This script also gives you the commands needed to copy the certificates to the nodes.

3.5.1 Setting up Vault Authentication

To configure Vault for use with Oracle Linux Cloud Native Environment, set up a Vault token with the following properties:

- A PKI secret engine with a CA certificate or intermediate, located at `olcne_pki_intermediary`.
- A role under that PKI, named `olcne`, configured to not require a common name, and allow any name.
- A token authentication method and policy that attaches to the `olcne` role and can request certificates.

For information on setting up the Vault PKI secrets engine to generate dynamic X.509 certificates, see:

https://www.vaultproject.io/docs/secrets/pki/index.html

For information on creating Vault tokens, see:

https://www.vaultproject.io/docs/commands/token/create.html

3.5.2 Setting up CA Certificates

This section shows you how to use your own certificates, signed by a trusted CA, without using a secrets manager such as Vault. To use your own certificates, copy them to all Kubernetes nodes, and to the Platform API Server node.

To make sure the Platform Agent on each Kubernetes node, and the Platform API Server have access to certificates, make sure you copy them into the `/etc/olcne/certificates/` directory on each node. The path to the certificates is used when setting up the Platform Agent and Platform API Server, and when creating an environment.

The examples in this book use the `/etc/olcne/configs/certificates/production/` directory for certificates. For example:

- **CA Certificate**: `/etc/olcne/configs/certificates/production/ca.cert`
- **Node Key**: `/etc/olcne/configs/certificates/production/node.key`
- **Node Certificate**: `/etc/olcne/configs/certificates/production/node.cert`

3.5.3 Setting up Private CA Certificates

This section shows you how to create a private CA, and use that to generate signed certificates for the nodes.

To generate certificates using a private CA:

1. (Optional) You can set up keyless SSH between the operator node and the Kubernetes nodes to make it easier to copy the certificates to the nodes. For information on setting up keyless SSH, see *Oracle® Linux: Connecting to Remote Systems With OpenSSH*.
2. Use the `/etc/olcne/gen-certs-helper.sh` script to generate a private CA and certificates for the nodes.

Tip

The `gen-certs-helper.sh` script saves the certificate files to the directory from which you run the script. The `gen-certs-helper.sh` script also creates a script you can use to copy the certificates to each Kubernetes node (`olcne-transfer-certs.sh`). If you run the `gen-certs-helper.sh` script from the `/etc/olcne` directory, it uses the default certificate directory used in this book (`/etc/olcne/certificates/`) when creating the `olcne-transfer-certs.sh` script. This means you can start up the Platform API Server, and the Platform Agent on Kubernetes nodes, using the default certificate directory locations as shown in this book. You could also use the `--cert-dir` option to specify the location to save the certificates and transfer script.

Provide the nodes for which you want to create certificates using the `--nodes` option. You should create a certificate for each node that runs the Platform API Server or Platform Agent. That is, for the operator node, and each Kubernetes node. If you are deploying a multi-master Kubernetes deployment using a virtual IP address, you do not need to create a certificate for a virtual IP address.

Provide the private CA information using the `--cert-request*` options (some, but not all, of these options are shown in the example). You can get a list of all command options using the `gen-certs-helper.sh --help` command.

For example:

```
$ cd /etc/olcne
$ sudo ./gen-certs-helper.sh \
  --cert-request-organization-unit "My Company Unit" \
  --cert-request-organization "My Company" \
  --cert-request-locality "My Town" \
  --cert-request-state "My State" \
  --cert-request-country US \
  --cert-request-common-name cloud.example.com \
  --nodes operator.example.com,master1.example.com,worker1.example.com,worker2.example.com,
worker3.example.com
```

The certificates and keys for each node are generated and saved to the directory:

`/path/configs/certificates/tmp-olcne/node/`

Where *path* is the directory from which you ran the `gen-certs-helper.sh` script, or the location you set with the `--cert-dir` option; and *node* is the name of the node for which the certificate was generated.

The private CA certificate and key files are saved to the directory:

`/path/configs/certificates/production/`

3. Copy the certificate generated for a node from the `/path/configs/certificates/tmp-olcne/node/` directory to that node.

   To make sure the Platform Agent on each Kubernetes node, and the Platform API Server have access to certificates, make sure you copy them into the `/etc/olcne/certificates/` directory on each node. The path to the certificates is used when setting up the Platform Agent and Platform API Server, and when creating an environment.

   The examples in this book use the `/etc/olcne/configs/certificates/production/` directory as the location for certificates on nodes.

   A script is created to help you copy the certificates to the nodes, `/path/configs/certificates/olcne-transfer-certs.sh`. You can use this script and modify it to suit your needs, or transfer the certificates to the nodes using some other method.

   

   Important

   If you set up keyless SSH, change the `USER` variable in this script to the user you set up with keyless SSH.

   Run the script to copy the certificates to the nodes:

   ```
   $ bash -ex /path/configs/certificates/olcne-tranfer-certs.sh
   ```

4. Make sure the `olcne` user on each node that runs the Platform API Server or Platform Agent is able to read the directory in which you copy the certificates. If you used the default path for certificates of `/etc/olcne/certificates/`, the `olcne` user has read access.

   If you used a different path, check the `olcne` user can read the certificate path. On the operator node, and each Kubernetes node, run:

   ```
   $ sudo -u olcne ls /path/configs/certificates/production
   ca.cert  node.cert  node.key
   ```

   You should see a list of the certificates and key for the node.

3.6 Starting the Platform API Server and Platform Agent Services

This section discusses using certificates to set up secure communication between the Platform API Server and the Platform Agent on nodes in the cluster. You can set up secure communication using certificates managed by Vault, or using your own certificates copied to each node. You must configure the Platform API Server and the Platform Agent to use the certificates when you start the services.

For information on setting up the certificates with Vault, see Section 3.5, “Setting up X.509 Certificates”.

For information on creating a private CA to sign certificates that can be used during testing, see Section 3.5.3, “Setting up Private CA Certificates”.

3.6.1 Starting the Services Using Vault

This section shows you how to set up the Platform API Server and Platform Agent services to use certificates managed by Vault.

To set up and start the services using Vault:

1. On the operator node, use the `/etc/olcne/bootstrap-olcne.sh` script to configure the Platform API Server to retrieve and use a Vault certificate. Use the `bootstrap-olcne.sh --help` command for a list of options for this script. For example:

```
$ sudo /etc/olcne/bootstrap-olcne.sh \
    --secret-manager-type vault \
    --vault-token s.3QKNuRoTqLbjXaGBOmO6Psjh \
    --vault-address https://192.0.2.20:8200 \
    --force-download-certs \
    --olcne-component api-server
```

   The certificates are generated and downloaded from Vault. By default, the certificates are saved to the `/etc/olcne/certificates/` directory. You can alternatively specify a path for the certificates, for example, by including the following options in the `bootstrap-olcne.sh` command:

```
    --olcne-ca-path /etc/olcne/configs/certificates/production/ca.cert \
    --olcne-node-cert-path /etc/olcne/configs/certificates/production/node.cert \
    --olcne-node-key-path /etc/olcne/configs/certificates/production/node.key \
```

   The Platform API Server is configured to use the certificates, and started.

2. On each Kubernetes node, use the `/etc/olcne/bootstrap-olcne.sh` script to configure the Platform Agent to retrieve and use a certificate. For example:

```
$ sudo /etc/olcne/bootstrap-olcne.sh \
    --secret-manager-type vault \
    --vault-token s.3QKNuRoTqLbjXaGBOmO6Psjh \
    --vault-address https://192.0.2.20:8200 \
    --force-download-certs \
    --olcne-component agent
```

   The certificates are generated and downloaded from Vault.

   The Platform Agent is configured to use the certificates, and started.

3.6.2 Starting the Services Using Certificates

This section shows you how to set up the Platform API Server and Platform Agent services to use your own certificates, which have been copied to each node. This example assumes the certificates are available on all nodes in the `/etc/olcne/configs/certificates/production/` directory.

To set up and start the services using certificates:

1. On the operator node, use the `/etc/olcne/bootstrap-olcne.sh` script to configure the Platform API Server to use the certificates. Use the `bootstrap-olcne.sh --help` command for a list of options for this script. For example:

```
$ sudo /etc/olcne/bootstrap-olcne.sh \
    --secret-manager-type file \
    --olcne-node-cert-path /etc/olcne/configs/certificates/production/node.cert \
    --olcne-ca-path /etc/olcne/configs/certificates/production/ca.cert \
    --olcne-node-key-path /etc/olcne/configs/certificates/production/node.key \
    --olcne-component api-server
```

   The Platform API Server is configured to use the certificates, and started.

2. On each Kubernetes node, use the `/etc/olcne/bootstrap-olcne.sh` script to configure the Platform Agent to use the certificates. For example:

```
$ sudo /etc/olcne/bootstrap-olcne.sh \
    --secret-manager-type file \
    --olcne-node-cert-path /etc/olcne/configs/certificates/production/node.cert \
    --olcne-ca-path /etc/olcne/configs/certificates/production/ca.cert \
    --olcne-node-key-path /etc/olcne/configs/certificates/production/node.key \
    --olcne-component agent
```

   The Platform Agent is configured to use the certificates, and started.

Chapter 4 Creating and Managing a Kubernetes Cluster

This chapter shows you how to use the Platform CLI (`olcnectl`) to create an environment, add the Kubernetes module (`kubernetes`) to the environment, add the master and worker nodes, and perform the deployment to create a Kubernetes cluster.

For more information on the syntax for the `olcnectl` command, see Chapter 5, *Using the Platform CLI*.

4.1 Creating an Environment

The first step to deploying Oracle Linux Cloud Native Environment is to create an environment. You can create multiple environments, with each environment potentially containing multiple modules. Naming each environment and module makes it easier to manage the deployed components of Oracle Linux Cloud Native Environment. Use the `olcnectl environment create` command to create an environment.

This section shows you how to create an environment using Vault, and using your own certificates copied to the file system on each node.

For information on setting up the certificates with Vault, see Section 3.5, “Setting up X.509 Certificates”.

For information on creating a private CA to sign certificates that can be used during testing, see Section 3.5.3, “Setting up Private CA Certificates”.

4.1.1 Creating an Environment using Certificates Managed by Vault

This section shows you how to create an environment using Vault to provide and manage the certificates.

On the operator node, use the `olcnectl environment create` command to create an environment. For example, to create an environment named `myenvironment` using certificates generated from a Vault instance:

```
$ olcnectl --api-server 127.0.0.1:8091 environment create --environment-name myenvironment \
    --update-config \
    --vault-token s.3QKNuRoTqLbjXaGBOmO6Psjh \
    --secret-manager-type vault \
    --vault-address https://192.0.2.20:8200
```

The `--secret-manager-type vault` file option sets the certificate manager to Vault. Replace `--vault-token` with the token to access Vault. Replace `--vault-address` with the location of your Vault instance.

The `--update-config` option saves the certificate generated by Vault on the local host. When you use this option, you do not need to enter the certificate information again when managing the environment.

By default, the certificate is saved to `$HOME/.olcne/certificates/`*`environment_name`*`/`. If you want to specify a different location to save the certificate, use the `--olcne-node-cert-path`, `--olcne-ca-path`, and `--olcne-node-key-path` options. For example, add the following options to the `olcnectl environment create` command:

```
    --olcne-node-cert-path /etc/olcne/configs/certificates/production/node.cert \
    --olcne-ca-path /etc/olcne/configs/certificates/production/ca.cert \
    --olcne-node-key-path /etc/olcne/configs/certificates/production/node.key
```

4.1.2 Creating an Environment using Certificates

This section shows you how to create an environment using your own certificates, copied to each node. This example assumes the certificates are available on all nodes in the `/etc/olcne/configs/certificates/production/` directory.

On the operator node, create the environment using the `olcnectl environment create` command. For example:

```
$ olcnectl --api-server 127.0.0.1:8091 environment create --environment-name myenvironment \
    --update-config \
    --secret-manager-type file \
    --olcne-node-cert-path /etc/olcne/configs/certificates/production/node.cert \
    --olcne-ca-path /etc/olcne/configs/certificates/production/ca.cert \
    --olcne-node-key-path /etc/olcne/configs/certificates/production/node.key
```

The `--secret-manager-type file` file option sets the certificate manager to use file-based certificates.

You can optionally set the location for the certificate files using environment variables; `olcnectl` uses these if they are set.

The environment variables map to the `olcnectl environment create` command options:

- `$OLCNE_SM_CERT_PATH` sets the value used with the `--olcne-node-cert-path` option.
- `$OLCNE_SM_CA_PATH` sets the value used with the `--olcne-ca-path` option.
- `$OLCNE_SM_KEY_PATH` sets the value used with the `--olcne-node-key-path` option.

For example:

```
$ export OLCNE_SM_CA_PATH=/etc/olcne/configs/certificates/production/ca.cert
$ export OLCNE_SM_CERT_PATH=/etc/olcne/configs/certificates/production/node.cert
$ export OLCNE_SM_KEY_PATH=/etc/olcne/configs/certificates/production/node.key
$ olcnectl --api-server 127.0.0.1:8091 environment create --environment-name myenvironment \
    --update-config \
    --secret-manager-type file
```

4.2 Adding Kubernetes to an Environment

After you create the environment, you should add any modules you want in the deployment. A base deployment requires Kubernetes to be deployed to the environment. This is done by adding the `kubernetes` module to the environment.

You can see a list of the available modules for an environment using the `olcnectl module list` command. For example:

```
$ olcnectl --api-server 127.0.0.1:8091 module list --environment-name myenvironment
```

Use the `olcnectl module create` command to add modules to an environment. For example, to add the `kubernetes` module to the `myenvironment` environment, with one master and two worker nodes:

```
$ olcnectl --api-server 127.0.0.1:8091 module create --environment-name myenvironment \
  --module kubernetes --name mycluster \
  --container-registry container-registry.oracle.com/olcne \
  --apiserver-advertise-address 192.0.2.100 \
  --master-nodes master1.example.com:8090 \
  --worker-nodes worker1.example.com:8090,worker2.example.com:8090
```

The `--apiserver-advertise-address` option specifies the IP address of the interface on the master node to use when communicating with the worker nodes. This option makes sure that if there are multiple network interfaces on the master node, the one specified with this option is used.

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 a local registry, with

the Kubernetes images mirrored from the Oracle Container Registry. For information on creating a local registry, see Section 2.2, “Accessing the Container Registry”.

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

For the full list of the options available for the `kubernetes` module, see Section 5.1, “Platform CLI Syntax”.

For information on creating a multi-master highly available deployment, see Section 4.6, “Creating a Multi-Master (HA) Kubernetes Cluster”.

4.3 Validating the Kubernetes Module

When you have added the `kubernetes` module to an environment, you should validate the nodes are configured correctly to deploy the Kubernetes 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 --api-server 127.0.0.1:8091 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 --api-server 127.0.0.1:8091 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.

4.4 Deploying the Kubernetes Module

When you have added and validated a module, you can deploy it to the environment.

By using the `olcnectl module install` command, you can deploy the Kubernetes module to an environment to create a Kubernetes cluster.

As part of deploying the module:

- The Kubernetes packages are installed. 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.

For example, use the following command to deploy the `kubernetes` module named `mycluster` to the `myenvironment` environment:

```
$ olcnectl --api-server 127.0.0.1:8091 module install --environment-name myenvironment \
  --name mycluster
```

Kubernetes is deployed to the nodes.

Important

Deploying the Kubernetes module may take several minutes to complete.

4.5 Using the kubectl Command

When you have deployed a Kubernetes cluster to the nodes, you can use the `kubectl` command to create and manage pods. It is recommended that you use the `kubectl` command on a **master** node in the Kubernetes cluster.

Important

You must set up the `kubectl` command before you can use it.

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

For more information on setting up and using the `kubectl` command to manage pods, see *Container Orchestration*.

4.6 Creating a Multi-Master (HA) Kubernetes Cluster

This section discusses the differences between creating a single master deployment, and a multi-master deployment.

To deploy a multi-master Kubernetes cluster:

1. A number of additional ports are required to be open on master nodes in a multi-master deployment. For information on opening the required ports for a multi-master deployment, see Section 2.3.4.2, “Multi-Master Firewall Rules”.

2. A multi-master deployment needs a load balancer to provide high availability of master nodes. A load balancer can be deployed automatically when you perform a multi-master deployment, or you can use your own load balancer implementation.

   As part of using the load balancer deployed by the Platform CLI, NGINX and keepalived are installed on the master nodes to enable the container-based deployment of the load balancer. NGINX improves the resource availability and efficiency of your multi-master Kubernetes cluster. keepalived can be used to monitor services or systems and to failover automatically to a standby master node if problems occur. As part of deploying this load balancer for a multi-master deployment, the `olcne-nginx` and `keepalived` services are enabled and started on the master nodes.

   For information on preparing the master nodes to use the load balancer deployed by the Platform CLI, see Section 3.4.3, “Setting up a Load Balancer”.

3. Perform the same steps as a single master deployment, as discussed in Chapter 4, *Creating and Managing a Kubernetes Cluster*, using different options for the `olcne module create` command to specify either a virtual IP address for the load balancer deployed by the Platform CLI, or the hostname and port for your own load balancer instance.

To use the load balancer deployed by the Platform CLI, use the `--virtual-ip` option to set the virtual IP address to be used for the primary master node, for example, `--virtual-ip 192.0.2.100`.

Alternatively, you can set the hostname and port of your own load balancer implementation using the `--load-balancer` option, for example `--load-balancer lb.example.com:6443`.

You do not need to include the `--apiserver-advertise-address` option for a multi-master deployment.

The following example creates a multi-master deployment using the load balancer deployed by the Platform CLI. The virtual IP address of `192.0.2.100` is used for the primary master node.

```
$ olcnectl --api-server 127.0.0.1:8091 module create --environment-name myenvironment \
  --module kubernetes --name mycluster \
  --container-registry container-registry.oracle.com/olcne \
  --virtual-ip 192.0.2.100
  --master-nodes master1.example.com:8090,master2.example.com:8090,master3.example.com:8090 \
  --worker-nodes worker1.example.com:8090,worker2.example.com:8090,worker3.example.com:8090,
worker4.example.com:8090
```

The following example creates a multi-master deployment using your own load balancer, available on the host `lb.example.com` and running on port `6443`.

```
$ olcnectl --api-server 127.0.0.1:8091 module create --environment-name myenvironment \
  --module kubernetes --name mycluster \
  --container-registry container-registry.oracle.com/olcne \
  --load-balancer lb.example.com:6443
  --master-nodes master1.example.com:8090,master2.example.com:8090,master3.example.com:8090 \
  --worker-nodes worker1.example.com:8090,worker2.example.com:8090,worker3.example.com:8090,
worker4.example.com:8090
```

4. Continue with the module validation and install as described in in Chapter 4, *Creating and Managing a Kubernetes Cluster*.

4.7 Scaling a Kubernetes Cluster

A Kubernetes cluster may consist of either a single or multiple master 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.

You may want to replace one master or worker node in a Kubernetes cluster with another. So, first scale up to bring in the new node, and then scale down to remove the outdated 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.

If you used the `--apiserver-advertise-address` option when you created a Kubernetes module, then you cannot scale up from a single-master cluster to a multi-master cluster. However, if you used the

`--virtual-ip` or the `--load-balancer` options, then you can scale up, even if you have only a single-master cluster. For more information about the `--apiserver-advertise-address`, `--virtual-ip`, or `--load-balancer` options, see Section 4.6, “Creating a Multi-Master (HA) Kubernetes Cluster”.

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

1. A backup is taken of the cluster. In case something goes wrong during scaling up or scaling down, such as a failure occurring while scaling the cluster, 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 *Container Orchestration*.
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 master and worker nodes are added to or removed from the cluster.
4. The cluster is checked to ensure that all of the new and existing nodes are healthy. After validation of the cluster is completed, the cluster is scaled and you can access it.

In the following procedures, you learn how to scale up and scale down a Kubernetes cluster.

4.7.1 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 an Oracle Linux Cloud Native Environment. See Section 3.4.2, “Setting up Kubernetes Nodes”.
2. Set up a X.509 certificate for the node so that communication can happen between it and the other nodes of the Kubernetes cluster in a secure fashion. See Section 3.5, “Setting up X.509 Certificates”.
3. Start the Platform Agent service. See Section 3.6, “Starting the Platform API Server and Platform Agent Services”.

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 master node of the Kubernetes cluster, use the `kubectl get nodes` command to see the master and worker nodes of the cluster.

```
$ kubectl get nodes
NAME                  STATUS   ROLE     AGE     VERSION
master1.example.com   Ready    master   26h     v1.17.x+x.x.x.el7
master2.example.com   Ready    master   26h     v1.17.x+x.x.x.el7
master3.example.com   Ready    master   26h     v1.17.x+x.x.x.el7
worker1.example.com   Ready    <none>   26h     v1.17.x+x.x.x.el7
worker2.example.com   Ready    <none>   26h     v1.17.x+x.x.x.el7
worker3.example.com   Ready    <none>   26h     v1.17.x+x.x.x.el7
```

   For this example, there are three master nodes in the Kubernetes cluster:

   - `master1.example.com`
   - `master2.example.com`

- `master3.example.com`

There are also three worker nodes in the cluster:

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

2. To update a Kubernetes cluster by scaling it up, use the `olcnectl module update` command.

   For example, to add the `master4.example.com` master node, `worker4.example.com` worker node, and `worker5.example.com` worker node to the `kubernetes` module named `mycluster`, access the operator node of the Kubernetes cluster and execute the following command:

```
$ olcnectl --api-server 127.0.0.1:8091 module update --environment-name myenvironment \
  --name mycluster \
  --master-nodes master1.example.com:8090,master2.example.com:8090,master3.example.com:8090,\
master4.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
```

   In this example, you are scaling up a Kubernetes cluster so that it has four master nodes and five worker nodes. Make sure that if you are scaling up from a single-master to a multi-master cluster, you have specified a load balancer for the cluster. If you do not specify a load balancer, then you cannot scale up your master nodes.

   There are other options that you may find useful when scaling a Kubernetes cluster. The following example shows a more-complex method of scaling the cluster:

```
$ olcnectl --api-server 127.0.0.1:8091 module update --environment-name myenvironment \
  --name mycluster \
  --master-nodes master1.example.com:8090,master2.example.com:8090,master3.example.com:8090,\
master4.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 \
  --generate-scripts \
  --force
```

   The `--generate-scripts` 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`.

   The `--force` option suppresses the prompt displayed to confirm you want to continue with scaling the cluster.

3. Return to a master node of the Kubernetes cluster, and then use the `kubectl get nodes` command to verify that the cluster is scaled up to include the new master and worker nodes.

```
$ kubectl get nodes
NAME                 STATUS   ROLE     AGE     VERSION
master1.example.com  Ready    master   26h     v1.17.x+x.x.x.el7
master2.example.com  Ready    master   26h     v1.17.x+x.x.x.el7
master3.example.com  Ready    master   26h     v1.17.x+x.x.x.el7
master4.example.com  Ready    master   2m38s   v1.17.x+x.x.x.el7
worker1.example.com  Ready    <none>   26h     v1.17.x+x.x.x.el7
worker2.example.com  Ready    <none>   26h     v1.17.x+x.x.x.el7
worker3.example.com  Ready    <none>   26h     v1.17.x+x.x.x.el7
worker4.example.com  Ready    <none>   2m38s   v1.17.x+x.x.x.el7
```

```
worker5.example.com    Ready    <none>   2m38s   v1.17.x+x.x.x.el7
```

4.7.2 Scaling Down a Kubernetes Cluster

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

Warning

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

To scale down a Kubernetes cluster:

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

```
$ kubectl get nodes
NAME                  STATUS   ROLE     AGE     VERSION
master1.example.com   Ready    master   26h     v1.17.x+x.x.x.el7
master2.example.com   Ready    master   26h     v1.17.x+x.x.x.el7
master3.example.com   Ready    master   26h     v1.17.x+x.x.x.el7
master4.example.com   Ready    master   2m38s   v1.17.x+x.x.x.el7
worker1.example.com   Ready    <none>   26h     v1.17.x+x.x.x.el7
worker2.example.com   Ready    <none>   26h     v1.17.x+x.x.x.el7
worker3.example.com   Ready    <none>   26h     v1.17.x+x.x.x.el7
worker4.example.com   Ready    <none>   2m38s   v1.17.x+x.x.x.el7
worker5.example.com   Ready    <none>   2m38s   v1.17.x+x.x.x.el7
```

   For this example, there are four master nodes in the Kubernetes cluster:

   - `master1.example.com`
   - `master2.example.com`
   - `master3.example.com`
   - `master4.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. To update a Kubernetes cluster by scaling it down, use the `olcnectl module update` command.

   For example, to remove the `master4.example.com` master node, `worker4.example.com` worker node, and `worker5.example.com` worker node from the `kubernetes` module named `mycluster`, access the operator node of the Kubernetes cluster and execute the following command:

```
$ olcnectl --api-server 127.0.0.1:8091 module update --environment-name myenvironment \
  --name mycluster \
  --master-nodes master1.example.com:8090,master2.example.com:8090,master3.example.com:8090 \
```

```
  --worker-nodes worker1.example.com:8090,worker2.example.com:8090,worker3.example.com:8090
```

In this example, you are scaling down a Kubernetes cluster so that it has three master nodes and three worker nodes.

In this example, the `master4.example.com`, `worker4.example.com`, and `worker5.example.com` node are not specified. The Platform API Server keeps track of the nodes of the cluster that it is managing, and knows to scale down the cluster so that these nodes are removed from the cluster.

Because three master nodes and three worker nodes are specified in the `--master-nodes` and `--worker-nodes` options of the `olcnectl module update` command, the Platform API Server does not remove these nodes from the cluster.

3. Return to a master node of the Kubernetes cluster, and then use the `kubectl get nodes` command to verify that the cluster is scaled down.

```
$ kubectl get nodes
NAME                  STATUS   ROLE     AGE     VERSION
master1.example.com   Ready    master   26h     v1.17.x+x.x.x.el7
master2.example.com   Ready    master   26h     v1.17.x+x.x.x.el7
master3.example.com   Ready    master   26h     v1.17.x+x.x.x.el7
worker1.example.com   Ready    <none>   26h     v1.17.x+x.x.x.el7
worker2.example.com   Ready    <none>   26h     v1.17.x+x.x.x.el7
worker3.example.com   Ready    <none>   26h     v1.17.x+x.x.x.el7
```

4.8 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 --api-server 127.0.0.1:8091 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.

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

Chapter 5 Using the Platform CLI

This chapter discusses using the Platform CLI, the `olcnectl` command.

5.1 Platform CLI Syntax

The Platform CLI, `olcnectl`, is used to configure, deploy and manage the components of the Oracle Linux Cloud Native Environment. You interact with `olcnectl` by entering commands with a series of options.

The Platform CLI syntax is:

```
olcnectl [options] command [command_options]
```

Where *`options`* is one or more of:

- `{-a|--api-server} api_server_address:8091`

  The value of *`api_server_address`* is the IP address or hostname of the Platform API Server.

  If an Platform API Server is not specified, a local instance of an Platform API Server is created and used.

- `--olcne-ca-path ca_path`

  (Optional) The path to a predefined Certificate Authority certificate, or the destination of the certificate if using a secrets manager to download the certificate. The default is `/etc/olcne/certificates/ca.cert`, or gathered from the local configuration if the `--update-config` option is used.

- `--olcne-node-cert-path node_cert_path`

  (Optional) The path to a predefined key, or the destination of the key if using a secrets manager to download the key. The default is `/etc/olcne/certificates/node.key`, or gathered from the local configuration if the `--update-config` option is used.

- `--olcne-node-node-key-path node_key_path`

  (Optional) The path to a predefined certificate, or the a destination if using a secrets manager to download the certificate. The default is `/etc/olcne/certificates/node.cert`, or gathered from the local configuration if the `--update-config` option is used.

- `--secret-manager-type {file|vault}`

  The secrets manager type. The options are `file` or `vault`.

- `--update-config`

  When defined, the global arguments are written to a local configuration file and used for future calls to the Platform API Server. If this option has not been used previously, global arguments must be specified for every Platform API Server call.

  The global arguments configuration information is saved to `$HOME/.olcne/olcne.conf` on the local host.

  If you use Vault to generate certificates for nodes, the certificate is saved to `$HOME/.olcne/certificates/environment_name/` on the local host.

- `--vault-address vault_address`

The IP address of the Vault instance. The default is `https://127.0.0.1:8200`, or gathered from the local configuration if the `--update-config` option is used.

- `--vault-cert-sans vault_cert_sans`

  Subject Alternative Names (SANs) to pass to Vault to generate the Oracle Linux Cloud Native Environment certificate. The default is `127.0.0.1`, or gathered from the local configuration if the `--update-config` option is used.

- `--vault-token vault_token`

  The Vault authentication token.

And where *`command`* is one of:

- `environment {create|delete} {-E|--environment-name} environment_name`

  Creates an empty environment, or deletes an existing environment.

- `module list {-E|--environment-name} environment_name`

  Lists the available modules for an environment.

- `module create {-E|--environment-name} environment_name {-M|--module} module_name {-N|--name} name [module_args ...]`

  Adds and configures a module in an environment.

  The value of *`module_args`* is one or more arguments to configure a module in an environment.

  ***`module_args`* for the `kubernetes` module:**

  - `[-r|--container-registry] container_registry`

    The container registry that contains the Kubernetes images.

    If you do not provide this value, you are prompted for it by the Platform CLI.

  - `[-m|--master-nodes] nodes ...`

    A comma separated list of the hostnames or IP addresses of the Kubernetes master nodes, including the port number for the Platform Agent. The default port is `8090`. For example, `master1.example.com:8090,master2.example.com:8090`.

    If you do not provide this value, you are prompted for it by the Platform CLI.

  - `[-w|--worker-nodes] nodes ...`

    A comma separated list of the hostnames or IP addresses of the Kubernetes worker nodes, including the port number for the Platform Agent. The default port is `8090`. For example, `worker1.example.com:8090,worker2.example.com:8090`.

    If you do not provide this value, you are prompted for it by the Platform CLI.

  - `[-l|--load-balancer] load_balancer`

    The Kubernetes API server load balancer hostname or IP address, and port. The default port is `6443`. For example, `192.0.2.100:6443`.

- `[--virtual-ip] virtual_ip`

  The virtual IP address for the load balancer. This is optional.

  You should use this option if you do not specify your own load balancer using the `--load-balancer` option. When you specify a virtual IP address, it is used as the primary IP address for master nodes.

- `[--apiserver-bind-port-alt] port`

  The port on which the Kubernetes API server listens when you use a virtual IP address for the load balancer. The default is `6444`. This is optional.

  When you use a virtual IP address, the Kubernetes API server port is changed from the default of 6443 to 6444. The load balancer listens on port 6443 and receives the requests and passes them to

the Kubernetes API server. If you want to change the Kubernetes API server port in this situation from 6444, you can use this option to do so.

- `[--nginx-image] container_location`

  The location of an NGINX container image to use in a multi-master deployment. This is optional.

  You can use this option if you do not provide your own load balancer using the `--load-balancer` option. This option may be useful if you are using a mirrored container registry. For example:

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

  By default, `podman` is used to pull the NGINX image that is configured in `/usr/libexec/pull_olcne_nginx`. If you set the `--nginx-image` option to use another NGINX container image, the location of the image is written to `/etc/olcne-nginx/image`, and overrides the default image.

- `[-k|--kube-version] version`

  The Kubernetes version to deploy. The default is the latest version. For information on the latest version number, see *Release Notes*.

- `[-t|--kubeadm-token] token`

  The token to use for establishing bidirectional trust between Kubernetes nodes and control-plane nodes. The format is `[a-z0-9]{6}\.[a-z0-9]{16}`, for example, `abcdef.0123456789abcdef`.

- `[-o|--apiserver-advertise-address] IP_address`

  The IP address on which to advertise the Kubernetes API server to members of the Kubernetes cluster. This address must be reachable by the cluster nodes. If no value is provided, the master node's default interface is used.

  This option is not used in a multi-master, high availability deployment.

- `[-b|--apiserver-bind-port] port`

  The Kubernetes API server bind port. The default is `6443`.

- `[-s|--service-cidr] service_CIDR`

  The Kubernetes service CIDR. The default is `10.96.0.0/12`. This is the range from which each Kubernetes service is assigned an IP address.

- `[-p|--pod-cidr] pod_CIDR`

  The Kubernetes pod CIDR. The default is `10.244.0.0/16`. This is the range from which each Kubernetes pod network interface is assigned an IP address.

- `[-x|--kube-proxy-mode] {userspace|iptables|ipvs}`

  The routing mode for the Kubernetes proxy. The default is `iptables`. The available proxy modes are:

  - `userspace`: This is an older proxy mode.
  - `iptables`: This is the fastest proxy mode. This is the default mode.
  - `ipvs`: This is an experimental mode.

If no value is provided, the default of `iptables` is used. If the system's kernel or `iptables` version is insufficient, the `userspace` proxy is used.

- `[-e|--apiserver-cert-extra-sans] api_server_sans`

  The Subject Alternative Names (SANs) to use for the Kubernetes API server serving certificate. This value can contain both IP addresses and DNS names.

- `[-n|--pod-network] network_fabric`

  The network fabric for the Kubernetes cluster. The default is `flannel`.

***`module_args`* for the `helm` module:**

- `[--helm-kubernetes-module] kubernetes_module`

  The name of the `kubernetes` module instance that Helm should be installed into. Each instance of Kubernetes can have one instance of Helm associated with it.

- `[--helm-version] version`

  The version of Helm to deploy. The default is the latest version. For information on the latest version number, see *Release Notes*.

***`module_args`* for the `prometheus` module:**

- `[--prometheus-helm-module] helm_module`

  The name of the `helm` module instance that Prometheus should be associated with.

- `[--prometheus-image] container_registry`

  The container image registry and tag to use when deploying Prometheus. The default is `container-registry.oracle.com/olcne/prometheus`.

- `[--prometheus-version] version`

  The version of Prometheus to deploy. The default is the latest version. For information on the latest version number, see *Release Notes*.

- `[--prometheus-persistent-storage] {true|false}`

  If this value is `false`, Prometheus writes its data into an `emptydir` on the host where the pod is running. If the pod migrates, metric data is lost.

  If this value is `true`, Prometheus requisitions a Kubernetes `PersistentVolumeClaim` so that its data persists, despite destruction or migration of the pod.

  The default is `false`.

Important

Oracle Linux Cloud Native Environment does not yet have any modules that provide support for Kubernetes `PersistentVolumeClaim`s, so persistent storage must be manually set up.

***module_args* for the `istio` module:**

- `[--istio-helm-module] helm_module`

  The name of the `helm` module instance that Istio should be associated with. This instance of Helm is used to install the Istio module.

- `[--istio-container-registry] container_registry`

  The container image registry to use when deploying Istio. The default is `container-registry.oracle.com/olcne`.

- `[--istio-version] version`

  The version of Istio to deploy. The default is the latest version. For information on the latest version number, see *Release Notes*.

- `[--istio-mutual-tls] {true|false}`

  Sets whether to enable Mutual Transport Layer Security (mTLS) for communication between the control plane pods for Istio, and for any pods deployed into the Istio service mesh.

  The default is `true`.

  

  Important

  It is strongly recommended that this value is not set to `false`, especially in production environments.

- `module {validate|install} {-E|--environment-name} environment_name {-N|--name} name [--generate-scripts]`

  Validates or installs a module for an environment. When you validate a module, the nodes are checked to make sure they are set up correctly to run the module. If the nodes are not set up correctly, the commands required to fix each node is shown in the output.

  - `[--generate-scripts]`

    Generates a script which contains the commands required to fix any set up errors for the nodes in a module. A script is created for each node in the module, saved to the local directory, and named `hostname:8090.sh`.

- `module update {-E|--environment-name} environment_name {-N|--name} name [module_args ...]`

  Updates a module for an environment. The module configuration is automatically retrieved from the Platform API Server. This command can be used to update the version of Kubernetes and to scale a cluster.

  Where *`module_args`* is optionally one or more of:

  - `[-k|--kube-version]`

    Sets the Kubernetes version to upgrade to on the nodes. The default is the latest version. For information on the latest version number, see *Release Notes*.

    If this option is not provided when updating a Kubernetes module, any Kubernetes errata updates are installed.

  - `[-m|--master-nodes] nodes ...`

    A comma-separated list of the hostnames or IP addresses of the Kubernetes master nodes that should remain in or be added to the Kubernetes cluster, including the port number for the Platform Agent. Any master nodes not included in this list are removed from the cluster.

    The default port number for the Platform Agent is `8090`. For example, `master1.example.com:8090,master2.example.com:8090`.

  - `[-w|--worker-nodes] nodes ...`

    A comma-separated list of the hostnames or IP addresses of the Kubernetes worker nodes that should remain in or be added to the Kubernetes cluster, including the port number for the Platform Agent. Any worker nodes not included in this list are removed from the cluster.

    The default port number for the Platform Agent is `8090`. For example, `worker1.example.com:8090,worker2.example.com:8090`.

  - `[--nginx-image] container_location`

    The location of the NGINX container image to update. This is optional.

    This option pulls the NGINX container image from the container registry location you specify to update NGINX on the master nodes. For example:

    ```
    --nginx-image container-registry.oracle.com/olcne/nginx:1.17.7
    ```

  - `[--generate-scripts]`

    Generates a script which contains the commands required to fix any set up errors for the nodes in a module. A script is created for each node in the module, saved to the local directory, and named *`hostname`*`:8090.sh`.

  - `[--force]`

    Skips the confirmation prompt when scaling or updating a Kubernetes cluster.

- `module uninstall {-E|--environment-name} environment_name {-N|--name} name`

  Uninstalls a module for an environment.

- `module {backup|restore} {-E|--environment-name} environment_name {-N|--name} name`

  Backs up or restores a module for an environment.

- `module property list {-E|--environment-name} environment_name {-N|--name} name`

  Lists the available properties for a module in an environment.

- `module property get {-E|--environment-name} environment_name {-N|--name} name --property property_name`

  Lists the value of the property for a module in an environment.

And where *command_options* is one or more of:

- `{-E|--environment-name} environment_name`

  The value of *environment_name* is the name to use to identify an Oracle Linux Cloud Native Environment environment. An *environment* is a namespace that encapsulates software installed and managed by Oracle Linux Cloud Native Environment.

- `{-M|--module} module_name`

  The value of *module_name* is the name of an Oracle Linux Cloud Native Environment module.

  For information on available modules, see Section 3.3, “Introduction to Modules”.

- `{-N|--name} name`

  The value of *name* is the name to use to identify a module in an environment.

- `--property property_name`

  The value of *property-name* is the name used to identify a module property in an environment. You can get a list of the available properties for a module using the `olcnectl module property list` command.

5.2 Platform CLI Examples

This section shows you how to use the `olcnectl` command to create and manage an Oracle Linux Cloud Native Environment deployment. When you use the `olcnectl` command, you are prompted for any missing options. For details of the `olcnectl` command syntax, see Section 5.1, “Platform CLI Syntax”.

5.2.1 Creating an Environment

The first step to deploying Oracle Linux Cloud Native Environment is to create an empty environment.

You can create an environment using certificates provided by Vault, or using existing certificates on the nodes.

5.2.1.1 Creating an Environment using Vault

Use the `olcnectl environment create` command to create an environment. For example, to create an environment named `myenvironment` using certificates generated from a Vault instance:

```
$ olcnectl --api-server 127.0.0.1:8091 environment create --environment-name myenvironment \
  --update-config \
  --vault-token s.3QKNuRoTqLbjXaGBOmO6Psjh \
```

```
  --secret-manager-type vault \
  --vault-address https://192.0.2.20:8200
```

5.2.1.2 Creating an Environment using Certificates

Use the `olcnectl environment create` command with the `--secret-manager-type file` file option. For example:

```
$ olcnectl --api-server 127.0.0.1:8091 environment create --environment-name myenvironment \
  --update-config \
  --secret-manager-type file \
  --olcne-node-cert-path /etc/olcne/configs/certificates/production/node.cert \
  --olcne-ca-path /etc/olcne/configs/certificates/production/ca.cert \
  --olcne-node-key-path /etc/olcne/configs/certificates/production/node.key
```

5.2.2 Deleting an Environment

This section shows you how to delete an environment. You should uninstall and remove any modules from an environment before you delete it.

Use the `olcnectl environment delete` command to delete an environment. For example, to delete an environment named `myenvironment`:

```
$ olcnectl --api-server 127.0.0.1:8091 environment delete --environment-name myenvironment
```

5.2.3 Listing Available Modules in an Environment

This section shows you how to list the available modules for an environment.

Use the `olcnectl module list` command to list available modules for an environment. For example, to list the modules for an environment named `myenvironment`:

```
$ olcnectl --api-server 127.0.0.1:8091 module list --environment-name myenvironment
```

5.2.4 Adding Modules to an Environment

After you create an empty environment, you can add any modules you want to it. You can see a list of the available modules for an environment using the `olcnectl module list` command.

Use the `olcnectl module create` command to add and configure modules in an environment.

This example creates a Kubernetes cluster using the `kubernetes` module. For a full list of the *`module_args`* for the `kubernetes` module, see Section 5.1, “Platform CLI Syntax”.

To add the `kubernetes` module to an environment with one master and two worker nodes:

```
$ olcnectl --api-server 127.0.0.1:8091 module create --environment-name myenvironment \
  --module kubernetes --name mycluster \
  --container-registry container-registry.oracle.com/olcne \
  --apiserver-advertise-address 192.0.2.100 \
  --master-nodes master1.example.com:8090 \
  --worker-nodes worker1.example.com:8090,worker2.example.com:8090
```

5.2.5 Validating a Module

When you have added a module to an environment, you should validate it is configured correctly for a deployment before you perform the installation. Validating a module checks that it can be installed by verifying any relevant environment state. If the validation fails, a set of commands is returned to help you fix any issues.

Use the `olcnectl module validate` command to validate modules in an environment. For example, to validate the `kubernetes` module named `mycluster`:

```
$ olcnectl --api-server 127.0.0.1:8091 module validate --environment-name myenvironment \
  --name mycluster
```

5.2.6 Installing a Module

When you have added and validated a module, you can deploy it to the environment.

Use the `olcnectl module install` command to deploy modules to an environment. For example, to deploy the `kubernetes` module named `mycluster`:

```
$ olcnectl --api-server 127.0.0.1:8091 module install --environment-name myenvironment \
  --name mycluster
```

5.2.7 Scaling a Kubernetes Cluster

To scale a Kubernetes cluster up or down, use the `olcnectl module update` command. This command updates the Kubernetes cluster so that nodes can be added to it or removed from it.

For example, if the `kubernetes` module named `mycluster` has three master nodes `master1.example.com`, `master2.example.com`, and `master3.example.com` and three worker nodes `worker1.example.com`, `worker2.example.com`, and `worker3.example.com`, and you want to add the `master4.example.com` master node, `worker4.example.com` worker node, and `worker5.example.com` worker node to the module:

```
$ olcnectl --api-server 127.0.0.1:8091 module update --environment-name myenvironment \
  --name mycluster \
  --master-nodes master1.example.com:8090,master2.example.com:8090,master3.example.com:8090,\
master4.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
```

To remove the `master4.example.com` master node, `worker4.example.com` worker node, and `worker5.example.com` worker node from the `kubernetes` module named `mycluster`:

```
$ olcnectl --api-server 127.0.0.1:8091 module update --environment-name myenvironment \
  --name mycluster \
  --master-nodes master1.example.com:8090,master2.example.com:8090,master3.example.com:8090 \
  --worker-nodes worker1.example.com:8090,worker2.example.com:8090,worker3.example.com:8090
```

Because three master nodes and three worker nodes are specified in both the `--master-nodes` and `--worker-nodes` options of the `olcnectl module update` command, the Platform API Server does not remove these nodes from the cluster.

5.2.8 Updating the Kubernetes Release

Important

Before you update or upgrade the Kubernetes cluster, make sure you have updated or upgraded Oracle Linux Cloud Native Environment to the latest release. For information on updating or upgrading Oracle Linux Cloud Native Environment, see *Updates and Upgrades*.

Use the `olcnectl module update` command to update to the latest Kubernetes errata release, or to update to a new Kubernetes release. For example, to update the Kubernetes module named `mycluster` in the `myenvironment` environment to the latest Kubernetes errata release:

```
$ olcnectl --api-server 127.0.0.1:8091 module update --environment myenvironment \
  --name mycluster
```

The nodes in the environment are updated to the latest Kubernetes errata release.

Use the `olcnectl module update` command to upgrade to the latest Kubernetes release. For example, to upgrade to Kubernetes Release 1.17, enter:

```
$ olcnectl --api-server 127.0.0.1:8091 module update --environment-name myenvironment \
  --name mycluster \
  --kube-version 1.17.9
```

The `--kube-version` option specifies the release to which you want to upgrade. This example uses release number 1.17.9.

Important

Make sure you upgrade to the latest Kubernetes release. To get the version number of the latest Kubernetes release, see *Release Notes*.

The nodes in the environment are updated to Kubernetes Release 1.17.

5.2.9 Uninstalling a Module

Use the `olcnectl module uninstall` command to uninstall modules from an environment. Uninstalling the module also removes the module configuration from the Platform API Server. For example, to uninstall the `kubernetes` module named `mycluster`:

```
$ olcnectl --api-server 127.0.0.1:8091 module uninstall --environment-name myenvironment \
  --name mycluster
```

In this example, the Kubernetes containers are stopped and deleted on each node, and the Kubernetes cluster is removed.

5.2.10 Listing Module Properties

This section shows you how to list the available properties for a module in an environment.

Use the `olcnectl module property list` command to list the properties for a module in an environment. For example, to list the properties for the `kubernetes` module in an environment named `myenvironment`:

```
$ olcnectl --api-server 127.0.0.1:8091 module property list \
  --environment-name myenvironment --name mycluster
```

5.2.11 Listing the Value of a Module Property

This section shows you how to list the value of a property for a module in an environment.

Use the `olcnectl module property get` command to list the value for a property for a module in an environment. For example, to list the value of the `kubecfg` property for the `kubernetes` module in an environment named `myenvironment`:

```
$ olcnectl --api-server 127.0.0.1:8091 module property get \
  --environment-name myenvironment --name mycluster --property kubecfg
```

Chapter 6 Configuring Oracle Linux Cloud Native Environment Services

This chapter contains information about any configuration options for Oracle Linux Cloud Native Environment, including configuring the Platform API Server and Platform Agent services.

6.1 Configuring the Platform API Server

The Platform API Server runs as a Systemd service, named `olcne-api-server`. You can get logs for this service using:

```
$ sudo journalctl -u olcne-api-server
```

By default, the service runs on TCP port 8091. You can change this and other Platform API Server settings by editing the Systemd service unit file so that the binary is invoked to use additional options.

`olcne-api-server` options:

- `[-p|--port] port_number`

  Specifies the port that the Platform API Server binds to. Defaults to `8091` if unspecified.

- `[-i|--installables] installables_path`

  Specifies the path to the directory of installable modules. Defaults to `/etc/olcne/modules`.

- `[-x|--insecure]`

  Allows the gRPC server to accept clients that do not securely establish their identity.

To reconfigure the Platform API Server to use any of these options, you can edit the Systemd unit file at `/usr/lib/systemd/system/olcne-api-server.service` and append the option to the `ExecStart` line. For example:

```
[Unit]
Description=Platform API Server for Oracle Linux Cloud Native Environments
Wants=network.target
After=network.target

[Service]
ExecStart=/usr/libexec/olcne-api-server -i /etc/olcne/modules --port 9083
WorkingDirectory=/var/olcne
User=olcne
Group=olcne
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

If you edit the Systemd unit file, you must run the following commands for the changes to take effect:

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart olcne-api-server.service
```

Note

If you change the port value for this service, you should take this into account for all other instructions provided in the documentation.

6.2 Configuring the Platform Agent

The Platform Agent runs as a Systemd service, named `olcne-agent`. You can get logs for this service using:

```
$ sudo journalctl -u olcne-agent
```

By default, the service runs on TCP port 8090. You can change this and other Platform Agent settings by editing the Systemd service unit file so that the binary is invoked to use additional options.

Additional options available to `olcne-agent` include:

`olcne-agent` options:

- `[-p|--port] port-number`

  Specifies the port that the Platform Agent service binds to. Defaults to `8090` if unspecified.

- `[-x|--insecure]`

  Allows the gRPC server to accept clients that do not securely establish their identity.

To reconfigure the Platform Agent to use any of these options, you can edit the Systemd unit file at `/usr/lib/systemd/system/olcne-agent.service` and append the option to the `ExecStart` line.

If you edit the Systemd unit file, you must run the following commands for the changes to take effect:

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart olcne-agent.service
```

Note

If you change the port value for this service, you should take this into account for all other instructions provided in the documentation.

Terminology

Environment
: A namespace that encapsulates the software installed and managed by Oracle Linux Cloud Native Environment. Each environment contains at least the Kubernetes module.

Kubernetes Node
: A host in an Oracle Linux Cloud Native Environment that contains the Platform Agent and other required software to run as a Kubernetes cluster member, either as a master or worker node.

Module
: A module represents some installable object in an environment. The most common module is the Kubernetes module, as it is required for most other modules. A module may encapsulate other modules.

Node
: A host system that is a member of an Oracle Linux Cloud Native Environment, including hosts used as Kubernetes master and worker nodes, and hosts that run the Platform API Server and Platform CLI.

Operator Node
: A host in an Oracle Linux Cloud Native Environment that contains the Platform CLI. This node may also be a Platform API Server node.

Platform Agent
: The Oracle Linux Cloud Native Environment Platform Agent is a software agent installed on all nodes which is used by the Platform API Server to report and change state as directed by the Platform API Server.

Platform API Server
: The Oracle Linux Cloud Native Environment Platform API Server manages the state of one or more environments, including the deployment and configuration of modules to one or more nodes in a cluster.

Platform API Server Node
: A host in an Oracle Linux Cloud Native Environment that contains the Platform API Server. This node may also be an operator node.

Platform CLI
: The Oracle Linux Cloud Native Environment Platform Command-Line Interface used to create and manage deployments. The `olcnectl` command.