ORACLE

Oracle Cloud Native Environment on Oracle Private Cloud Appliance

Best Practices of deployment of an HA Kubernetes cluster on Oracle Private Cloud Appliance and deployment of Weblogic Server

November 17, 2021 | Version 2.01

PURPOSE STATEMENT

This document describes best practices and demonstrates installing Oracle Cloud Native Environment and deploying WebLogic and Traefik on the Oracle Private Cloud Appliance Release Release , and describes best practices. While key concepts are described, this paper is not a replacement for the official documentation on Oracle Private Cloud Appliance, Oracle Cloud Native Environment, Kubernetes, WebLogic or Traefik. Further documentation is linked in this paper.

DISCLAIMER

This document is for informational purposes only and is intended solely to assist you in planning for the implementation and upgrade of the product features described. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle.

Due to the nature of the product architecture, it may not be possible to safely include all features described in this document without risking significant destabilization of the code.

TABLE OF CONTENTS

INTRODUCTION

Oracle Private Cloud Appliance (PCA) is an on-premises converged infrastructure platform for on-premises clouds that allows customers to efficiently consolidate business critical middleware and application workloads.

Oracle Private Cloud Appliance fully supports Oracle Cloud Native Environment (OCNE), a curated set of open-source Cloud Native Computing Foundation (CNCF) projects. It can be easily deployed, has been tested for interoperability, and offers enterprise-grade support. With the Oracle Cloud Native Environment, Oracle provides features for customers to develop microservices-based applications that can be deployed in environments that support open standards and specifications. This document will describe best practices and illustrate deploying an OCNE environment with WebLogic and Traefik on a PCA system.

ORACLE PRIVATE CLOUD APPLIANCE - INTEGRATED PLATFORM FOR CLOUD

The Oracle Private Cloud Appliance (PCA) is an Oracle Engineered System designed for the application tier. It is an integrated hardware and software system that reduces infrastructure complexity and deployment time for virtualized workloads in private clouds. It is a complete platform for a wide range of application types and workloads, with built-in management, compute, storage and networking resources.

Oracle Private Cloud Appliance Benefits

PCA enables rapid deployment of converged compute, network, and storage technologies for hosting workloads in guest operating systems. Private Cloud Appliance provides 'quick time to value' for a robust virtualization platform, going from first power-up to starting VMs in a matter of hours. PCA automatically discovers hardware components and configures them to work with one another, reducing design and administrative effort, eliminating potential errors, and speeding time to application deployment. PCA's automated configuration implements Oracle best practices for optimal performance and availability

PCA is cost effective, has high performance levels across a broad range of application types, easy to manage, and delivers better 'time to value' than disparate build-your-own solutions. Oracle Private Cloud Appliance together with Oracle Exadata provides powerful, single-vendor, application and database platforms for today's data driven enterprise.

Oracle Private Cloud Appliance offers an optimized platform to consolidate enterprise mission-critical workloads and modern cloud-native containerized workloads. It provides the simplest path to modernize your workloads and help you accelerate the digital transformation to meet your changing business needs.

Its built-in secure multi tenancy, zero downtime upgradability, capacity on demand and single pane of glass management make it the ideal infrastructure for rapid deployment of mission critical workloads. Oracle Private Cloud Appliance together with Oracle Cloud Infrastructure provides customers with a complete solution to securely maintain workloads on both private and public clouds.

Oracle Private Cloud Appliance Virtualization and cloud platform

The Private Cloud Appliance provides virtualization life cycle management using Oracle VM. Oracle VM consists of two parts: Oracle VM Manager and Oracle VM Server. These are automatically pre-installed on PCA without additional license fees, representing a substantial cost savings over other virtualization platforms. Oracle Enterprise Manager is optional but highly recommended, and is available without additional cost for providing cloud capabilities.

Oracle VM Manager is an advanced and widely used virtualization management product for controlling multiple servers, VMs, networks and storage resources under a graphical browser user interface. Oracle VM Manager also provides programmable REST APIs and command line interfaces to permit automation. Each PCA has one active instance of Oracle VM Manager used as a central control plane to administer the entire PCA.

Oracle VM Server is a high-performance hypervisor that runs virtual machines, based on commands sent from Oracle VM Manager. Oracle VM Server is automatically installed and configured on every compute node, both when a PCA rack is installed, and when additional compute nodes are added. Oracle VM efficiently runs virtual machines, which may run Oracle Linux, Oracle Solaris, other Linux versions, and Microsoft Windows, and the applications these operating systems support.

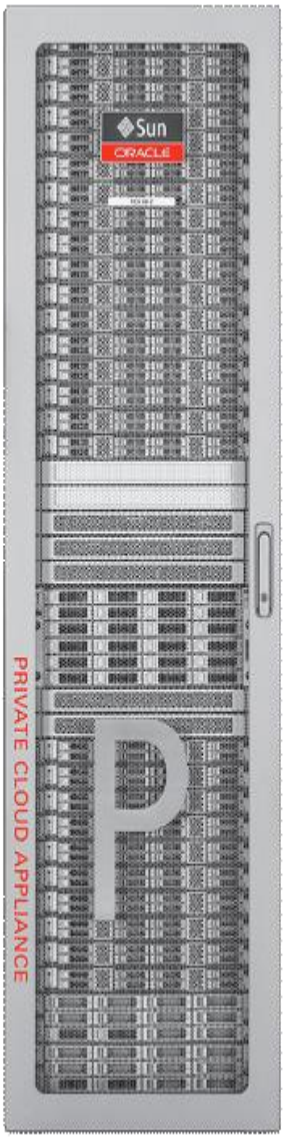
Oracle VM provides advanced functions that benefit applications.

- Administrators can set anti-affinity rules to ensure that virtual machines comprising a clustered application such as WebLogic Server do not run on the same physical server, insulating them from single points of failure. Using anti-affinity is a best practice for deploying OLCNE.
- High Availability (HA) settings can automatically reboot VMs if they or the server they are running on crash. For example, HA can bring up a WebLogic administrative instance or Kubernetes master after an outage.
- Live migration allows you to move running VMs from one server to another without stopping the VM and interrupting its applications. This is typically used for load balancing and to perform compute node maintenance without virtual machine outage,
- Dynamic Resource Scheduling (DRS) policies can provide load balancing among server based on several parameters, including CPU and network utilization according to policy settings. DRS automatically live migrates VMs from heavily loaded servers to less loaded ones to permit optimal performance.
- Resource management policies control CPU allocation for differential service between applications sharing the same servers, including CPU caps, and share-based priorities. This permits higher degrees of CPU oversubscription without affecting the service level objectives for production applications running alongside less-critical applications.
- Administrators can define tenant groups to dedicate PCA compute nodes to different clients. This ensures dedicated resources for applications or departments. PCA tenant groups leverage standard Oracle VM server pools, with additional automation and default configuration.

PCA customers can optionally use Oracle Enterprise Manager 13c to provide Infrastructure as a Service (IaaS) cloud capabilities, Oracle Enterprise Manager provides comprehensive management capabilities, including role-based access control, monitoring and chargeback. It is a best practice for production environments with different user and role categories. PCA can also leverage Ansible to provide “infrastructure as code”.

Oracle Private Cloud Appliance X8-2 Hardware Platform

Oracle Private Cloud Appliance is an easy-to-deploy, “turnkey” converged infrastructure solution that integrates compute, network, and storage resources in a software-defined fabric.



Oracle Private Cloud Appliance rack consists of the following main hardware components:

- **Management Nodes**: PCA uses two latest generation Oracle servers X8-2 function as the management nodes for Oracle Private Cloud Appliance X8. They serve as an active-passive cluster for management operations, providing resiliency in case of planned outage or server failure. Oracle VM Manager and other management functions run on the active management node. When a management node assumes the active role it takes over a virtual IP address (VIP) address, so clients of the management interface do not need to know which management node is currently active.
- **Compute Nodes**. Compute nodes are Oracle Server X8-2 or X9-2 systems powered by two Intel® Xeon® Processors. X8-2 servers have 24 cores per socket and can be ordered in three different memory configurations – 384GB, 768GB and 1.5 TB. X9-2 servers have 32 cores per socket and can be ordered in three different memory configurations – 512GB, 1.0TB, and 2.0TB. Each compute node runs Oracle VM Server for x86 to provide server virtualization. Compute nodes may be added or removed from the Oracle Private Cloud Appliance configurations without any downtime. A

Private Cloud Appliance rack can support up to 1,200 compute cores when using X8-2 servers, and up to 1,280 compute cores with X9-2 servers.

- **Switches**. Ethernet switches used for the data network and management network in a Private Cloud Appliance, configured to provide "wire once" Software Defined Networking (SDN). This permits multiple isolated virtual networks to be created on the same physical network hardware components.
- . The different types of switches used are:
  - Leaf Switches - (2) 36 port 100GbE switches used for high-speed internal communication between the internal hardware components (Compute Nodes, system disk, management servers) in a Private Cloud Appliance solution
  - Spine Switches - (2) 36 port 100GbE switches used for high-speed communication between the Private Cloud Appliance and other Engineered Systems, storage or the data center network. Spine switches form the backbone of the network and perform routing tasks.
  - Management Switch - (1) 48 port switch used to provide easy management of all internal hardware components (Compute Nodes, system disk, fabric interconnects, management servers) in a Private Cloud Appliance. High speed low latency SDN is implemented on top of 100GbE leaf and spine switches. These offer 100GbE connectivity for all communication between internal-rack components and allow flexible 10/25/40 or 100 GbE connectivity to customer datacenter.
- **Integrated Storage**. Oracle Private Cloud Appliance features a fully integrated, enterprise-grade Oracle ZFS Storage Appliance ZS7-2 MR ("ZFSSA") providing extreme performance and superior efficiency required by demanding enterprise applications running in VMs. This storage subsystem is designed to be fully redundant for maximum fault tolerance and serviceability in production. The Oracle Private Cloud Appliance X8-2 storage subsystem is loaded with high-performance DIMM and flash memory for optimal read/write performance under the most demanding file storage workloads. The storage capacity of Oracle Private Cloud Appliance X8-2 can be expanded beyond the initial configuration by adding storage trays. Storage can also be expanded by adding data center racks containing external Oracle ZFS Storage Appliances.

These elements provide a resilient management plane, network and storage resources, and scalable compute.

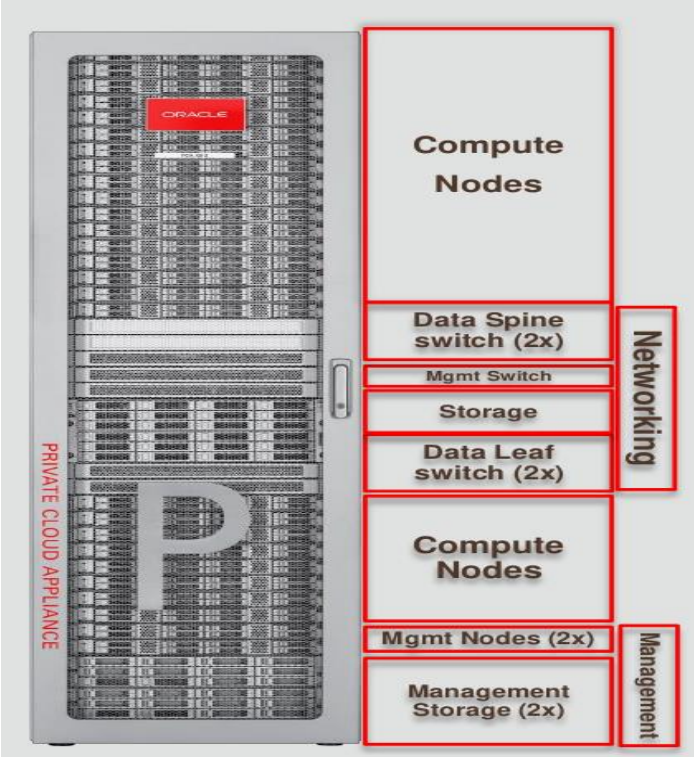


Image Caption 1. Oracle PCA X8-2 Software Environment

OVERVIEW OF ORACLE CLOUD NATIVE ENVIRONMENT (OCNE)

Oracle Cloud Native Environment (OCNE) is a fully integrated suite for the development and management of cloud-native applications. It provides a framework for orchestrating microservices through Kubernetes, consistent with open standards and practices emphasized by the Open Container Initiative (OCI), the Cloud Native Computing Foundation (CNCF), and others.

The Kubernetes module is the core module. It is used to deploy and manage containers and automatically install and configure CRI-O, runC and Kata Containers. CRI-O manages the container runtime for a Kubernetes cluster. The runtime may be either runC or Kata Containers. Among OCNE properties, it is

- Very well suited for scalable microservices-based production deployments.
- Incorporates features needed by modern K8S ecosystem.
- Has a predictable roadmap based on key open source component releases

OCNE provides a framework for orchestrating microservices through Kubernetes, based on the Open Container Initiative (OCI) and Cloud Native Computing Foundation (CNCF).

Oracle Cloud Native Environment Core Concepts:

An OCNE environment is a namespace that encapsulates OCNE software modules, including at a minimum the Kubernetes module. An OCNE module is essentially a unit of software that OCNE can install and manage within an environment, such as the Kubernetes module, Istio module, and the Operator Lifecycle Manager module.

Node types

OCNE makes use of the following types of Kubernetes node**:**

- **Operator node** performs and manages the deployment of environments, including deploying the Kubernetes cluster. An operator node may be a node in the Kubernetes cluster, or a separate host. The operator node calls on Kubernetes APIs to communicate with Kubernetes agents on target nodes.
- **Control plane nodes** run management services. There can be a single node in a non-resilient configuration, but the recommended configuration and best practice uses multiple nodes to avoid single points of failure.
- **Worker nodes** run the applications. There are a minimum of two worker nodes, sized according to application requirements.

The following diagram provides a high-level view of the OCNE architecture. The Platform CLI is used to communicate with the Platform API server and uses the `olcnectl` command. The Platform API server is responsible for managing the overall environment. The Platform Agent runs on each host and receives requests from the Platform API server. The operator node runs the Platform CLI and Platform API, and the Kubernetes control plane and worker nodes run the Platform Agent.

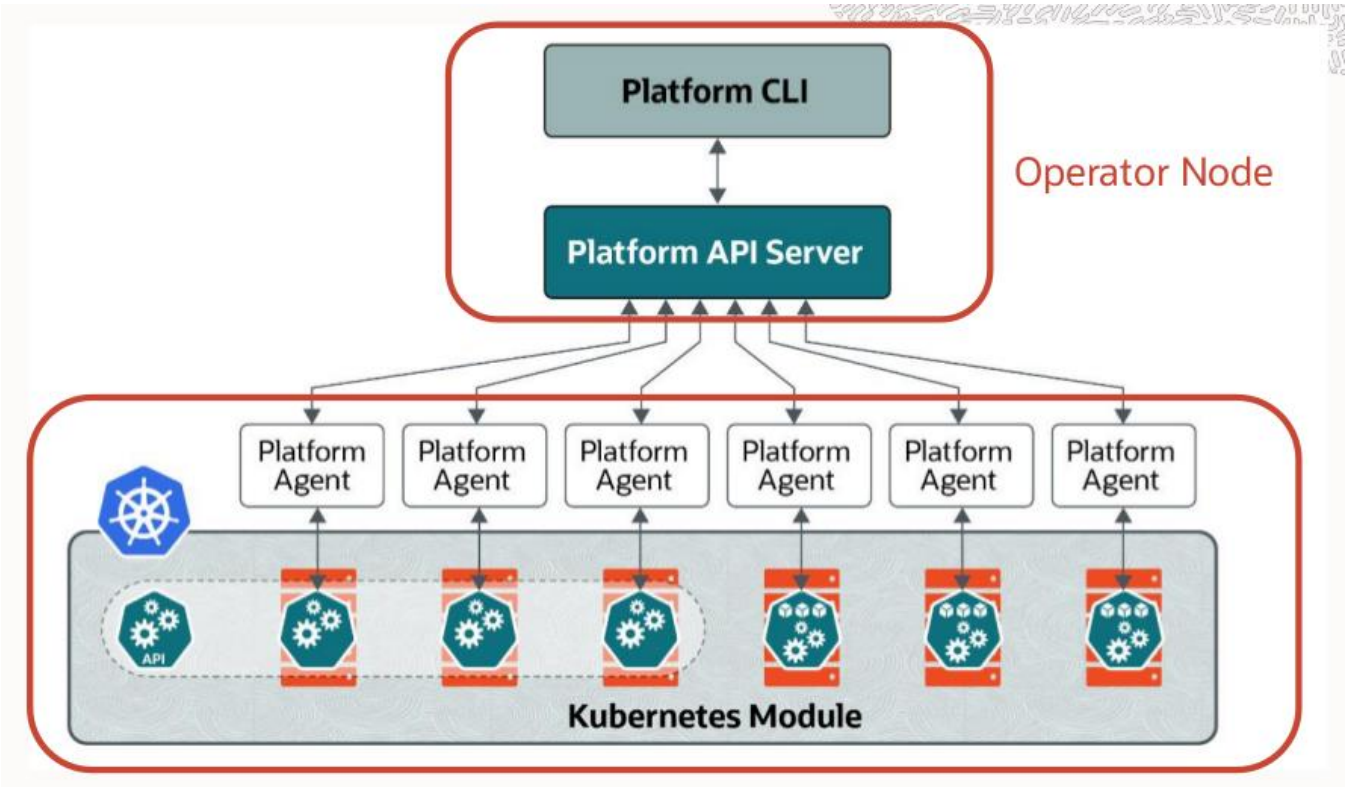


Image Caption 2. Architectural overview of HA Kubernetes clusters on Oracle Private Cloud Appliance / Private Cloud at Customer

A Kubernetes cluster configured this way is Highly Available (HA), avoiding Single Point of Failure (SPOF), as it has:

- 3 Kubernetes Master Nodes
- Multiple Kubernetes Worker Nodes (scaled as needed)
- Integrated Load Balancer support

ORACLE VERRAZZANO ENTERPRISE CONTAINER PLATFORM

Oracle Verrazzano Enterprise Container Platform is an important technology related to OCNE, which enhances Oracle's extensive portfolio of standards-based open software technologies for cloud native application development, deployment, and lifecycle management in Oracle Cloud, other clouds, and on-premises.

Many enterprises have an investment in custom applications. Many of these applications are critical to their mission and business; some of these applications are traditional WebLogic Server applications, and some are not. As the industry focus shifts to the cloud, these enterprises are looking for solutions that enable them to flexibly adopt cloud-native technologies to improve productivity and innovation, to modernize their existing applications, and to run their applications where they choose.

You can use your Oracle Verrazzano Enterprise Container Platform subscription to run Verrazzano workloads on systems where Oracle Linux is running, such as Oracle Private Cloud Appliance. The Oracle Verrazzano Enterprise Container Platform is an ideal fit for the Oracle Private Cloud Appliance, the premiere Oracle Engineered System for application tier.

OVERVIEW OF WEBLOGIC SERVER AND TRAEFIK

This paper uses WebLogic and Traefik to demonstrate using Oracle Cloud Native on the PCA.

Oracle WebLogic Server

Oracle WebLogic Server is the industry leading application server for building enterprise applications using Java, and deploying them on a reliable, scalable runtime with low cost of ownership.

A WebLogic domain is a basic unit of WebLogic server, which includes the administration server and managed servers, as well as related resources and services. An administration server manages the domain configuration and hosts the Administration Console, and Managed Servers hosts the actual application and associated resources. From a developer perspective, WebLogic Server is Java middleware. WebLogic Server has been updated to support modern Cloud deployment patterns in Kubernetes.

Oracle WebLogic Server: Application server to build and run enterprise application.

- **WebLogic Domain**: A basic administrative unit of WebLogic server. It includes administration server and managed servers, as well as related resources and services.
- **Administration Server**: Manages domain configuration and hosts the Administration Console.
- **Managed Server**: Hosts the application and associated resources.

Traefik

Traefik is a modern reverse proxy and load balancer supporting dynamic configuration through service discovery. In this demonstration, Traefik is used for load balancing across Kubernetes nodes hosting WebLogic server instances.

Requirements for on-premises Load Balancer

A load balancer must distribute load across active nodes in a Kubernetes cluster and must adjust to recognize when a node has been added or removed from a cluster.

Kubernetes service type `LoadBalancer` is a specification describing this capability, not an implementation. In public cloud use, the Kubernetes cloud-controller-manager (CCM) interfaces with the public cloud provider, which provides a load balancer implementation. Kubernetes service type `LoadBalancer` has a corresponding load balancer implementation in the CCM to match the specification to the cloud-provided service.

For on-premises use, Kubernetes `cloud-controller-manager` does not exist, therefore no implementation is defined for the `LoadBalancer` service type and one must be provided. An implementation can either use a load balancer specifically designed for on-premises Kubernetes and other applications, such as F5 BigIP, or manually configure load balancers with other forms of ingress. This demo employs a manually configured external load balancer (`haproxy`) and only represents Traefik's use as an ingress controller.

For production, we recommend, as a best practice, the use of an implementation conforming to the Kubernetes `LoadBalancer` on ingress sautomated. Examples include NGINX Ingress Controller and F5 BigIP Controller for Kubernetes.

The following diagram shows the WebLogic and Traefik perspective of the demo environment. We have manually configured `haproxy`, the reverse proxy in this diagram, to handle network traffic.

In this demo `haproxy` is configured so that the host name `ca-pca-vm10` routes to the node port 30701, which exposes the WebLogic administration server to allow access to the WebLogic Administration console. The host name `ca-pca-vm20` routes to node port 30305 of the Traefik ingress controller, which will deliver the request to the Managed Server hosting the application in the WebLogic domain. Therefore, the host name `ca-pca-vm20` will allow access to the application itself.

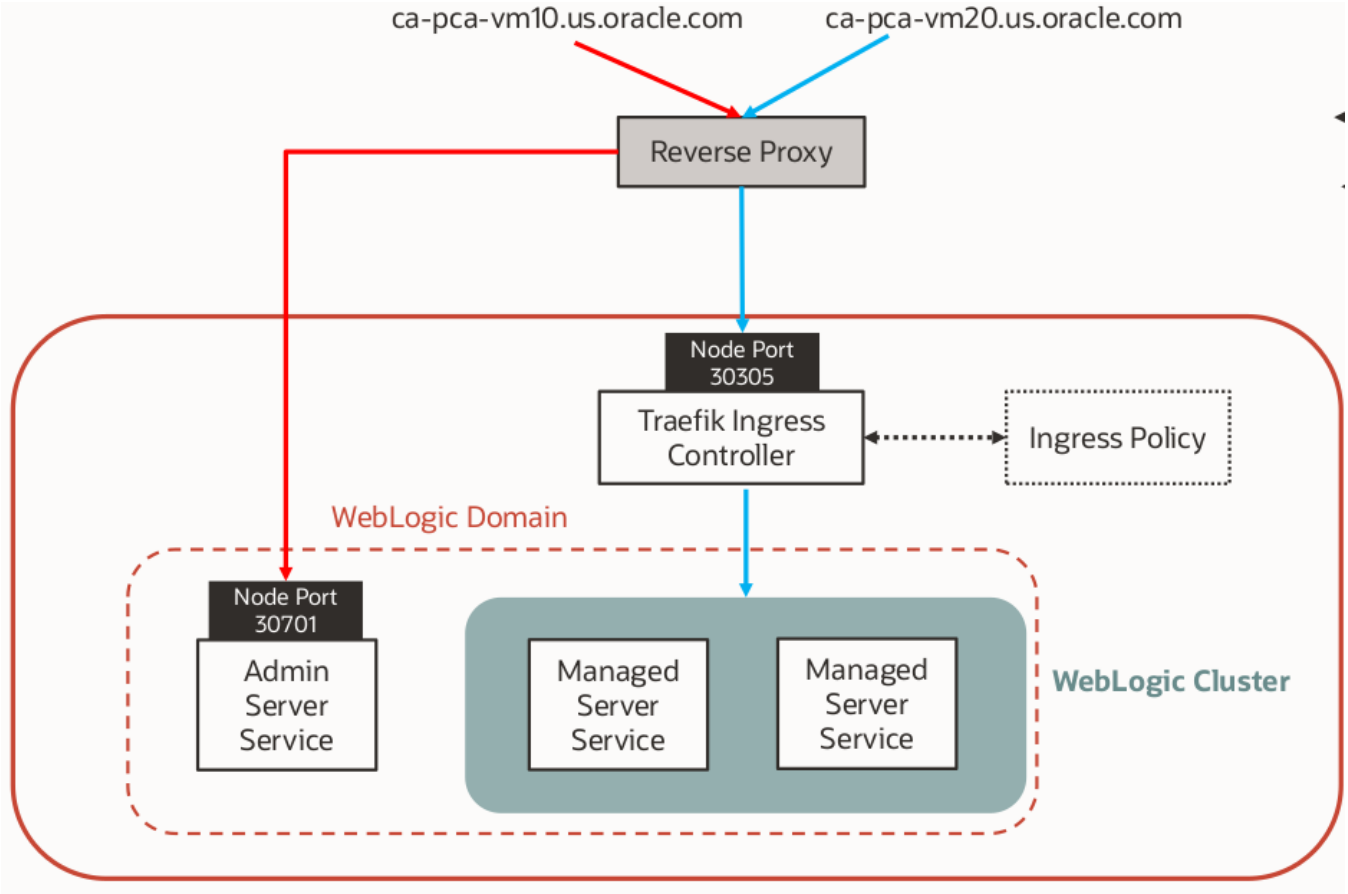


Image Caption 3. Demonstration Network Traffic Flow

Demonstration Network topology

For this demonstration we use the network architecture illustrated in the Level 3 diagram below.

There are three networks: a DMZ, the Load Balancer Backend, and the Kubernetes Pod network. The DMZ is the network where hosted applications are exposed to end users through the load balancer. The Load Balancer Backend network links the backend of the load balancer to the frontend of the Kubernetes cluster. And the Kubernetes pod network is usually a private, non-routed network for traffic that is internal to the Kubernetes cluster. As a PCA best practice, use a VLAN based on the PCA `default_internal` network for best latency, throughput, and network isolation.

One of the benefits of this network structure is that the cluster can be placed behind a firewall, with only the load balancer exposed. This achieves security, public IP address conservation, and operational flexibility. It enhances performance by minimizing network contention and allowing each subnet to be separately tuned. Finally, a private certificate authority can be used for the backend infrastructure, which permits cost savings for certificates.

For these reasons, most customer deployments should consider this topology as a minimal baseline for their on-premises Kubernetes network design, then make appropriate changes for specific requirements.

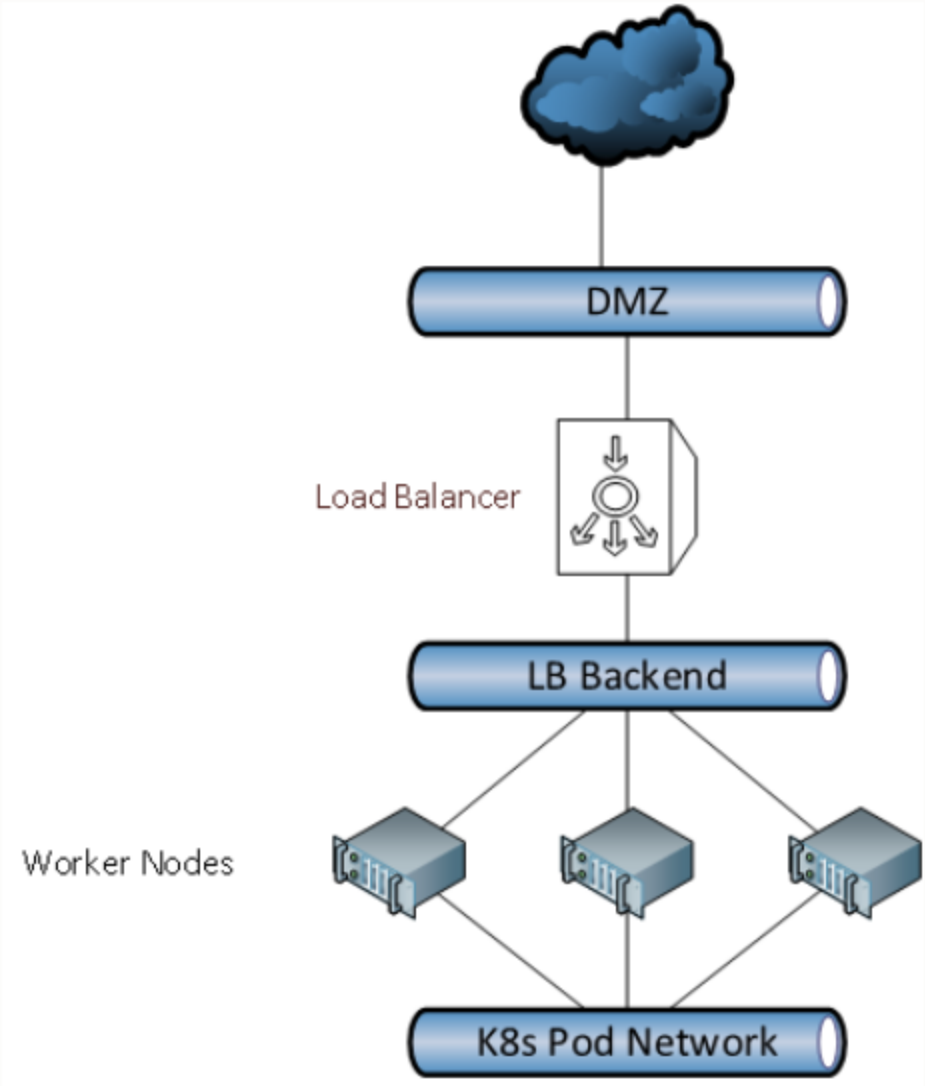


Image Caption 4. Demonstration Network Topology

DEMONSTRATION ENVIRONMENT

For this demonstration we used the configuration shown below and instructions in the "Oracle Cloud Native Environment Getting Started Guide" up to step 3.6.2.

Note that OCNE High Availability (HA) requires a minimum of 3 control plane nodes to form a quorum, and the number of control plane nodes must be an odd number so one partition can 'win' a vote. There must also be at least two worker nodes.

In this example we use the following configuration:

- Network definitions were set up for inter-instance communication between OCNE nodes, and for access to and from hosts external to the PCA. We used a VLAN based on the pre-configured PCA network `default_external` for access to and from non-PCA hosts. For production deployments a virtual network based on a VLAN on the pre-configured PCA network `default_internal` is recommended for low-latency, private networks between OCNE nodes. This configuration provides external access to hosts outside the PCA system while providing optimized private access for inter-node communication.
- Seven VMs running Oracle Linux 7.9 were created on the PCA:
    - 1 operator node, 3 control plane nodes, and 3 worker nodes.
    - The operator node was assigned 4 CPU threads (2 cores) and 8GB of RAM, control plane nodes had 4 CPU threads (2 cores) and 16GB of RAM, and worker nodes have 8 CPU threads (4 cores) and 16GB of RAM. In a non-demo environment, size the worker nodes based on the application requirements.
    - Oracle VM Manager anti-affinity rules were created to ensure that each control plane node was on a different physical compute node server from other control plane nodes, and to ensure that worker nodes are assigned to different compute nodes. This is a best practice for PCA to enhance availability.
- All nodes run Oracle Linux 7.9 with latest patches applied by running '`yum update`'.
- The OCNE 1.3 Getting Started Guide was followed, completing all steps through 3.6.2.
- Uses Private CA Certificates. You can also use vault certificates.
- Uses Oracle Container Registry instead of a private registry.
- Proxy settings are configured on all nodes.
- JAVA_HOME environment variable is set on the operator node to `/usr/lib/jvm/java-11-openjdk-11.0.12.0.7-0.0.1.el7_9.x86_64`
- `git` is installed on the operator node, which will run the `kubectl` command.

For commands requiring root privileges, either become root and issue the commands directly, or use a non-root user and preface the command with `sudo`, according to your preferences and standards. In this paper, we indicate whether root or non-root is being used by showing the shell prompt character **#** or `$`. Output from standard Linux commands (yum, systemctl, etc.) will not be shown.

OCNE SETUP

The following material shows procedures used in this demo. It is assumed that each of the Oracle Linux instances have already been installed and updated to current patch levels.

Enable access to OCNE packages

The first step is to add the OCNE repositories by issuing the following commands on each node.

On all nodes issue:

```
# yum install oracle-olcne-release-el7
# yum-config-manager --enable ol7_olcne13 ol7_kvm_utils ol7_addons ol7_latest ol7_UEKR6
# yum-config-manager --disable ol7_olcne ol7_olcne11 ol7_olcne12 ol7_developer
```

Install chrony and disable swap

Kubernetes is a clustered environment that requires synchronized time on control plane and worker nodes. It also requires disabling swap. On control plane and worker nodes issue:

```
# yum install chrony
# systemctl enable --now chronyd.service
# swapoff –
```

Set up firewall and netfilter

The following steps set up firewall settings to permit access to necessary ports.

On operator node:

```
# firewall-cmd --add-port=8091/tcp --permanent
# systemctl restart firewalld.service
```

On control plane nodes:

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

If installing OCNE Release 1.2 or later on Oracle Linux 7:

```
# firewall-cmd --add-masquerade --permanent
```

Then

```
# systemctl restart firewalld.service
```

On worker nodes:

```
# firewall-cmd --zone=trusted --add-interface=cni0 --permanent
# firewall-cmd --add-port=8090/tcp --permanent
# firewall-cmd --add-port=10250/tcp --permanent
```

```
# firewall-cmd --add-port=10255/tcp --permanent
# firewall-cmd --add-port=8472/udp --permanent
# firewall-cmd --add-masquerade --permanent
# systemctl restart firewalld.service
```

Set up HA cluster firewall roles on control plane nodes:

```
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
sudo systemctl restart firewalld.service
```

Check for netfilter and install if needed

On all nodes issue:

```
# lsmod | grep br_netfilter
```

If there is no output, issue:

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

Install OCNE software and configure proxy servers

The following steps install the OCNE software and configure the proxy servers.

On operator node - Install Platform CLI, Platform API Server and utilities

```
# yum install olcnectl olcne-api-server olcne-utils
# systemctl enable olcne-api-server.service
```

On Kubernetes nodes (control plane + worker nodes):

Install platform agent and utilities:

```
# yum install olcne-agent olcne-utils
# systemctl enable olcne-agent.service
```

Configure proxy server with CRI-O:

```
# mkdir /etc/systemd/system/crio.service.d
```

Edit `proxy.conf` and set the following lines, substituting in the hostname of your site's proxy server and port number, and your institution's domain name. The last part indicates which hosts can be accessed without going to your proxy server.

```
[Service]
Environment="HTTP_PROXY=http://my-proxy-server-FQDN:80"
Environment="HTTPS_PROXY=http://my-proxy-server-FQDN:80"
Environment="NO_PROXY=127.0.0.0/8,localhost,10.0.0.0/8,.my-domain-name”
```

Then issue:

```
# systemctl disable --now docker.service
# systemctl disable --now containerd.service
```

Prepare for built-in load balancer

On control plane nodes issue:

```
# firewall-cmd --add-port=6444/tcp
# firewall-cmd --add-port=6444/tcp --permanent
# firewall-cmd --add-protocol=vrrp
# firewall-cmd --add-protocol=vrrp --permanent
```

Set up Certificates

This example uses private CA certificates. You can also use HashiCorp Vault certificates or certificates validated by a trusted Certificate Authority. 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

On the operator node issue the following, substituting in your fully qualified domain names. The *–cert-request-** operands in italics are optional.

```
# cd /etc/olcne
# ./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 \
nodename0.myFQDN,nodename1.myFQDN,nodename2.myFQDN,nodename3.myFQDN,nodename104.myFQDN,nodena
me5.myFQDN,nodename106.myFQDN
```

The next line copies certificates to all nodes.

```
# bash -ex /etc/olcne/configs/certificates/olcne-tranfer-certs.sh
# ls /etc/olcne/configs/certificates/production
```

The `scp` command can also be used to copy certificates.

Set up X.509 Certificates for External IPs Kubernetes Service

On operator node issue the following. The *–cert-request-** operands in italics are optional.

```
# cd /etc/olcne
# ./gen-certs-helper.sh \
--cert-dir /etc/olcne/configs/certificates/restrict_external_ip/production/ \
--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 externalip-validation-webhook-service.externalip-validation-system.svc,\
externalip-validation-webhook-service.externalip-validation-system.svc.cluster.local \
--one-cert
```

Configure and Start Platform API Server and Platform Agent Services

On operator node:

```
# /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
# systemctl status olcne-api-server.service
```

On Kubernetes nodes:

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

Create OCNE Environment

Before you create an OCNE cluster, you have to create an environment. In this example, we create an environment directly using certificates, but you can also create an environment using certificates managed by HashiCorp Vault. So on the operator node, run the following command.

The `olcne-node-cert-path` points to where node certificates are in, `olcne-ca-path` option is where the CA certificate is in, and `node-key-path` is where the node key is in. These certificates were created in the prior configuration steps. Make sure these paths are consistent across all nodes.

On operator node:

```
# 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
```

Create, Validate, and Install Kubernetes Module

Once you have created an environment, you can create a Kubernetes module

On operator node:

```
$ olcnectl module create \
--environment-name myenvironment \
--module kubernetes \
--name mycluster \
--container-registry container-registry.oracle.com/olcne \
--virtual-ip 10.147.36.192\
--master-nodes nodename1.myFQDN:8090,nodename2.myFQDN:8090,nodename3.myFQDN:8090 \
--worker-nodes nodename4.myFQDN:8090,nodename5.myFQDN:8090,nodename6.myFQDN:8090 \
--selinux enforcing \
--restrict-service-externalip-ca-
cert=/etc/olcne/configs/certificates/restrict_external_ip/production/production/ca.cert\
--restrict-service-externalip-tls-
cert=/etc/olcne/configs/certificates/restrict_external_ip/production/production/node.cert\
--restrict-service-externalip-tls-
key=/etc/olcne/configs/certificates/restrict_external_ip/production/production/node.key\
--pod-network-iface eth1
$ olcnectl module validate \
--environment-name myenvironment \
--name mycluster
$ olcnectl module install \
--environment-name myenvironment \
 --name mycluster
```

SET UP KUBECTL

Preceding steps established the Kubernetes environment. The Kubernetes command-line tool, kubectl, allows you to run commands against Kubernetes clusters. You can use `kubectl` to deploy applications, inspect and manage cluster resources, and view logs.

Manage the Cluster from your local machine

You can manage your Kubernetes cluster deployed on Oracle Private Cloud Appliance/Private Cloud at Customer from your local desktop or laptop. This requires you to install `kubectl` on your local machine. Depending on the operating system on your local machine, follow the steps in Kubernetes documentation to install `kubectl`:
https://kubernetes.io/docs/tasks/tools/install-kubectl/

Installing kubectl locally allows you to manage all your Kubernetes clusters from a single central machine, thereby removing the need to be on the master management node of Oracle Private Cloud Appliance. This simplifies operations for Kubernetes cluster management.

Install kubectl

In this exercise, we show the steps to install `kubectl-1.17.4` on macOS. Similar command sequences are used for Linux. You can also use a package manager for your operating system, such as Homebrew, yum, apt-get as appropriate

```
$ curl -LO \
https://storage.googleapis.com/kubernetesrelease/release/v1.17.4/bin/darwin/amd64/kubectl
$ chmod +x ./kubectl;
$ sudo mv ./kubectl /usr/local/bin/kubectl
```

Configure kubectl

The next several steps describe how to configure kubectl for CLI and browser

If on a control plane node:

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

If on operator node:

```
$ olcnectl module property get \
  --environment-name myenvironment \
  --name mycluster \
  --property kubecfg | base64 -d > kubeconfig.yaml
$ mkdir -p $HOME/.kube
s
$ export KUBECONFIG=$HOME/.kube/config
$ echo 'export KUBECONFIG=$HOME/.kube/config' >> $HOME/.bashrc
```

Use kubectl CLI

Once the Kubernetes configuration is exported, you can run `kubectl` commands to display and manage the cluster:

```
$ kubectl get nodes
NAME                   STATUS    ROLES     AGE    VERSION
mydemo-1-cluster-m-1    Ready     master    1d    v1.17.4+1.0.1.el7
mydemo-1-cluster-m-2    Ready     master    1d    v1.17.4+1.0.1.el7
mydemo-1-cluster-m-3    Ready     master    1d    v1.17.4+1.0.1.el7
mydemo-1-cluster-w-0    Ready     <none>    1d    v1.17.4+1.0.1.el7
mydemo-1-cluster-w-1    Ready     <none>    1d    v1.17.4+1.0.1.el7
mydemo-1-cluster-w-2    Ready     <none>    1d    v1.17.4+1.0.1.el7
```

All the pods that have been deployed in the 'kube-system' namespace can be viewed as follows:

```
$ kubectl get pods -n kube-system
NAME                                          READY    STATUS    RESTARTS    AGE
coredns-d6c8c99d8-ft5hs                       1/1      Running   0           1d
coredns-d6c8c99d8-tknbm                       1/1      Running   0           1d
etcd-mydemo-1-cluster-m-1                     1/1      Running   0           1d
etcd-mydemo-1-cluster-m-2                     1/1      Running   0           1d
etcd-mydemo-1-cluster-m-3                     1/1      Running   0           1d
kube-apiserver-mydemo-1-cluster-m-1           1/1      Running   3           1d
kube-apiserver-mydemo-1-cluster-m-2           1/1      Running   0           1d
kube-apiserver-mydemo-1-cluster-m-3           1/1      Running   0           1d
kube-controller-manager-mydemo-1-cluster-m-1  1/1      Running   1           1d
kube-controller-manager-mydemo-1-cluster-m-2  1/1      Running   2           1d
kube-controller-manager-mydemo-1-cluster-m-3  1/1      Running   4           1d
kube-flannel-ds-amd64-5xffg                   1/1      Running   6           1d
kube-flannel-ds-amd64-grhtq                   1/1      Running   0           1d
kube-flannel-ds-amd64-hq7n5                   1/1      Running   0           1d
kube-flannel-ds-amd64-jqbpg                   1/1      Running   0           1d
kube-flannel-ds-amd64-lm7n4                   1/1      Running   4           1d
kube-flannel-ds-amd64-nlpxl                   1/1      Running   0           1d
kube-proxy-2p8cg                              1/1      Running   0           1d
kube-proxy-49k6m                              1/1      Running   0           1d
kube-proxy-4xsjc                              1/1      Running   0           1d
kube-proxy-74w9w                              1/1      Running   0           1d
kube-proxy-lt4pg                              1/1      Running   0           1d
kube-proxy-z2v4d                              1/1      Running   0           1d
kube-scheduler-mydemo-1-cluster-m-1           1/1      Running   2           1d
kube-scheduler-mydemo-1-cluster-m-2           1/1      Running   0           1d
kube-scheduler-mydemo-1-cluster-m-3           1/1      Running   1           1d
```

Access Kubernetes Dashboard

The Dashboard is a web-based Kubernetes user interface. You can use Dashboard to deploy containerized applications to a Kubernetes cluster, troubleshoot your containerized application, and manage the cluster resources. You can use Dashboard to get an overview of applications running on your cluster, as well as for creating or modifying individual Kubernetes resources (such as Deployments, Jobs, etc.).

The Kubernetes Dashboard container is created as part of the kube-system namespace. It provides an intuitive graphical user interface to a Kubernetes cluster that can be accessed using a standard web browser. The Kubernetes Dashboard is described in the upstream Kubernetes documentation at: https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/

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

Display the Kubernetes Dashboard for monitoring

This pod can be seen running in the '`kubernetes-dashboard`' namespace.

```
$ kubectl get pods -n kubernetes-dashboard
NAME                                    READY   STATUS    RESTARTS   AGE
kubernetes-dashboard-74f8fcbc74-88697   1/1     Running   0          28d
```

Get token needed to authenticate.

On the node with `kubectl`, obtain the secret token needed to access the dashboard and launch the proxy service.

This generates a long line of text. Copy the entire line for pasting into your browser in the step 'Access the dashboard' below

```
$ kubectl -n kube-system describe $(kubectl -n kube-system get secret -n kube-system -o name
| grep namespace) | grep token:
…eyJhbGciOiJSUzI1NiIsImtpZCI6IjItMDZtN3ZhMHFXemh2….much more text……
```

Optionally create new user for dashboard access

Optionally, you can create a new user using Service Account mechanism of Kubernetes to display and login to the web-based dashboard. You can create this user, grant the user cluster-admin permissions and login to Dashboard using bearer token tied to this user.

To do this, we used the following file 'dashboard.yaml'

```
$ cat dashboard.yml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
```

```
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: admin-user
    namespace: kube-system
```

The tasks in this 'dashboard.yaml' file can be executed with the kubectl apply command

```
$ kubectl apply -f dashboard.yml
```

Use a similar method as above to extract the authentication token, which we then copy, and later paste into the “Enter Token” field in the browser page.

```
$ kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep admin-user | grep token:
```

Start the proxy service

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

Leave that terminal window undisturbed to let the service operate. You can stop it by issuing CTRL-C, You can run it as a system service by issuing

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

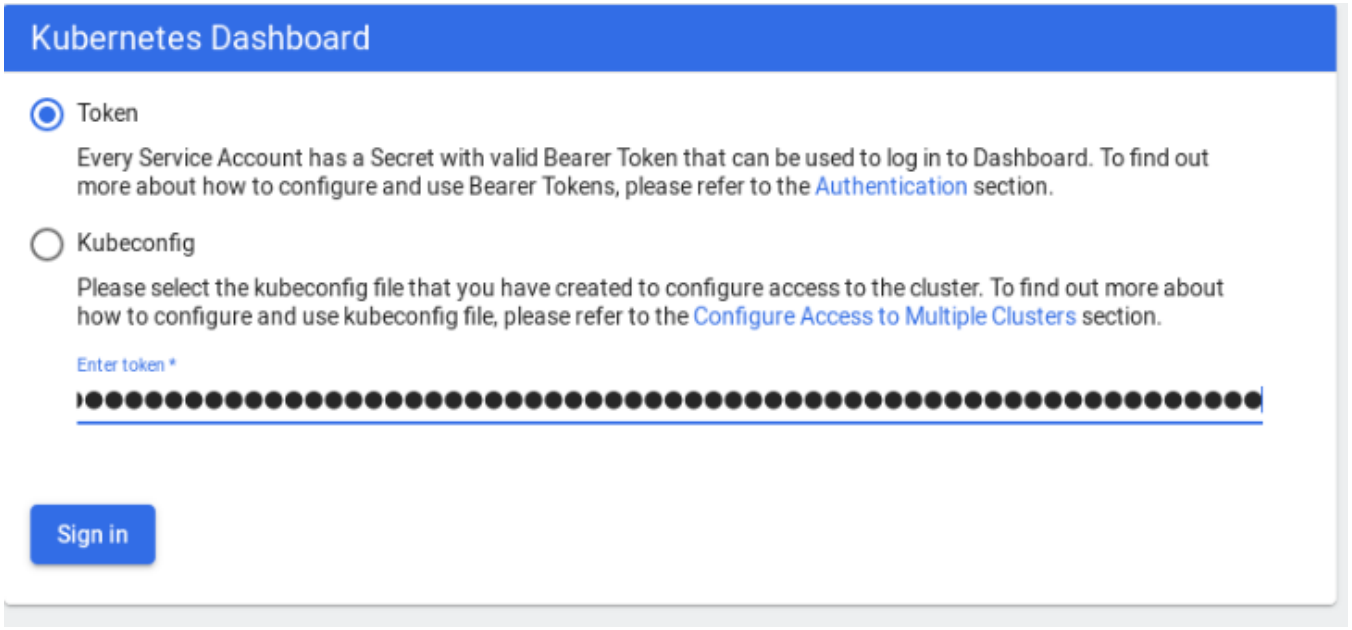
Access the dashboard

If `kubectl` is running remotely from your desktop or laptop, first `ssh` to that host with port forwarding. Substitute in the IP address of the host running the `kubectl` proxy:

```
$ ssh -L 8001:127.0.0.1:8001 root@my-IP-address:
```

Then, whether `kubectl` is running remotely or locally, point your web browser to:

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



Paste in the authentication token obtained previously. After authenticating you'll see the dashboard and can navigate through its screens. Several of the standard screens are illustrated below:

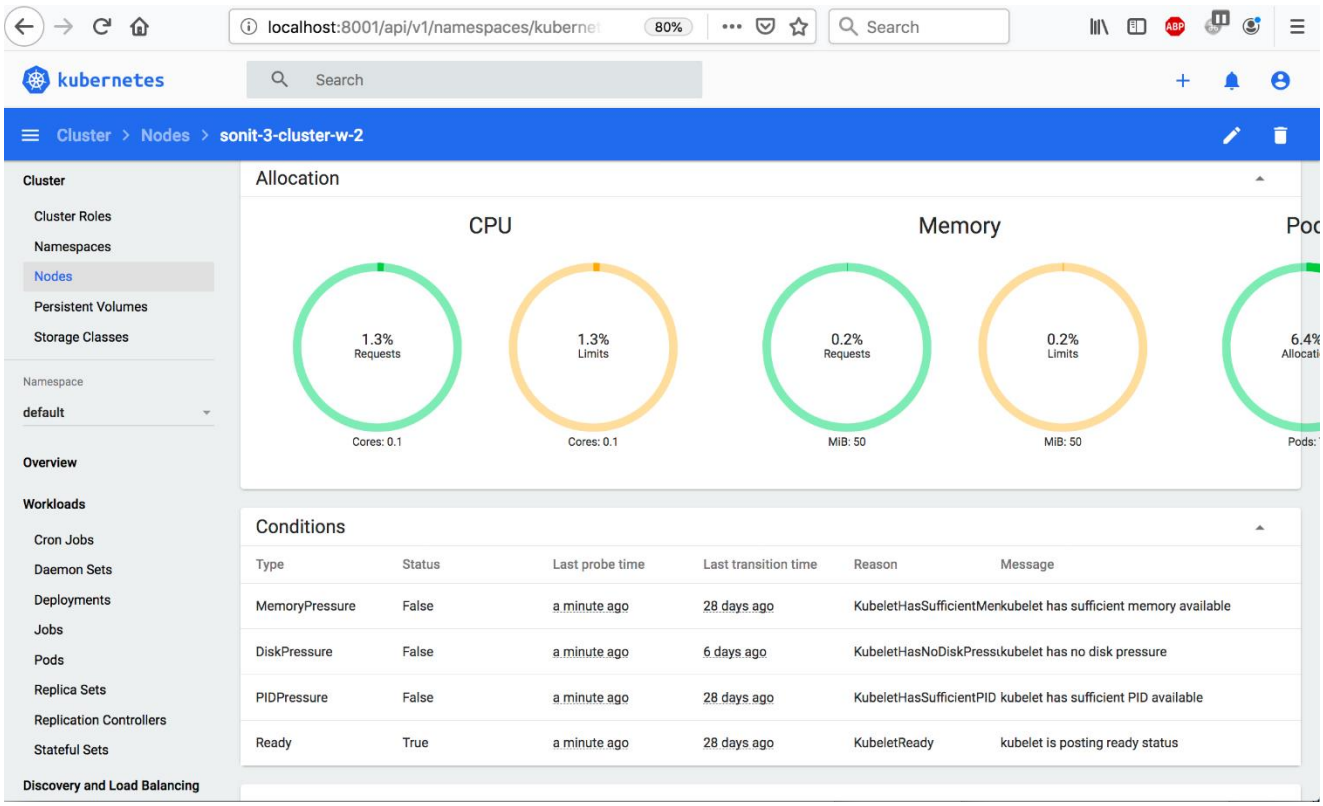
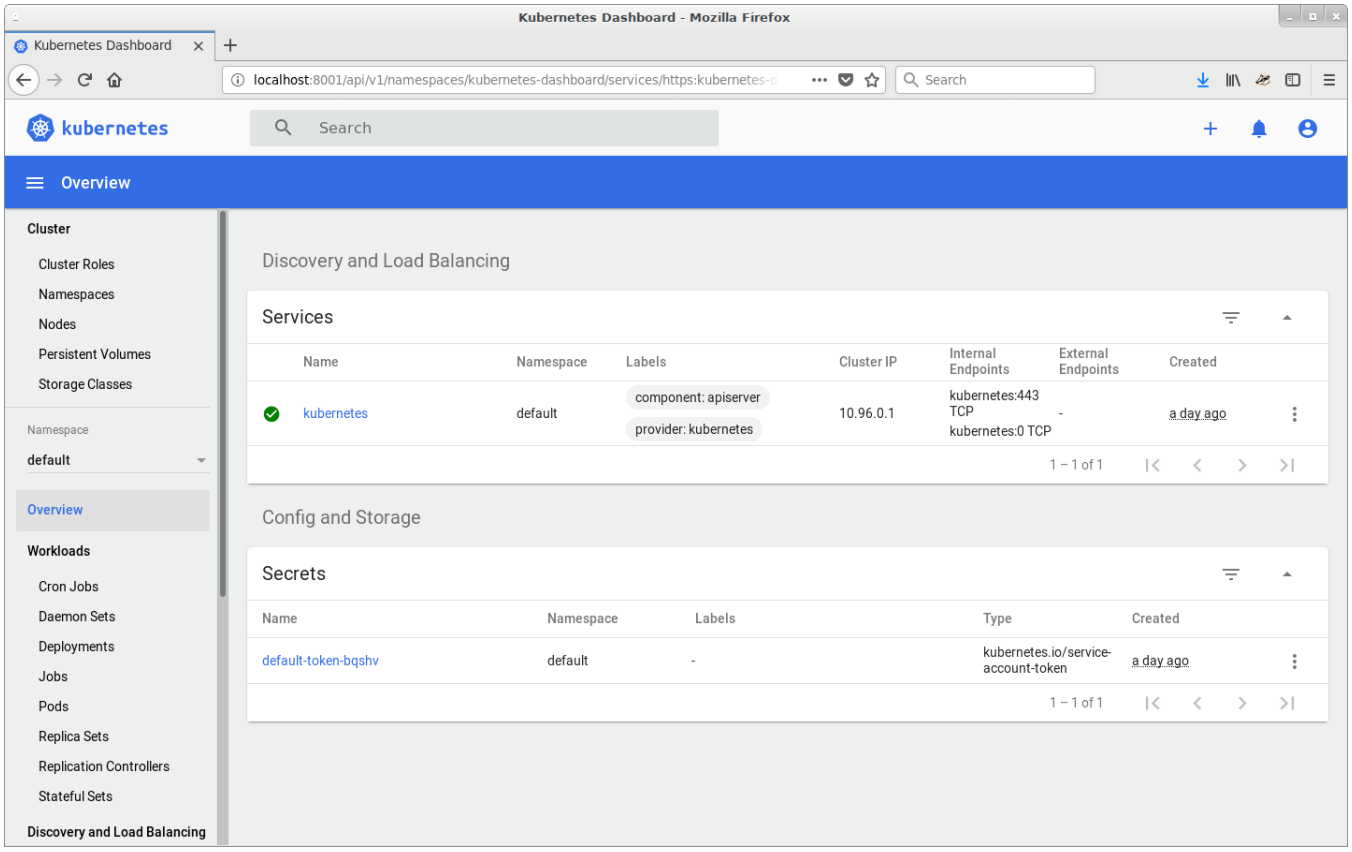
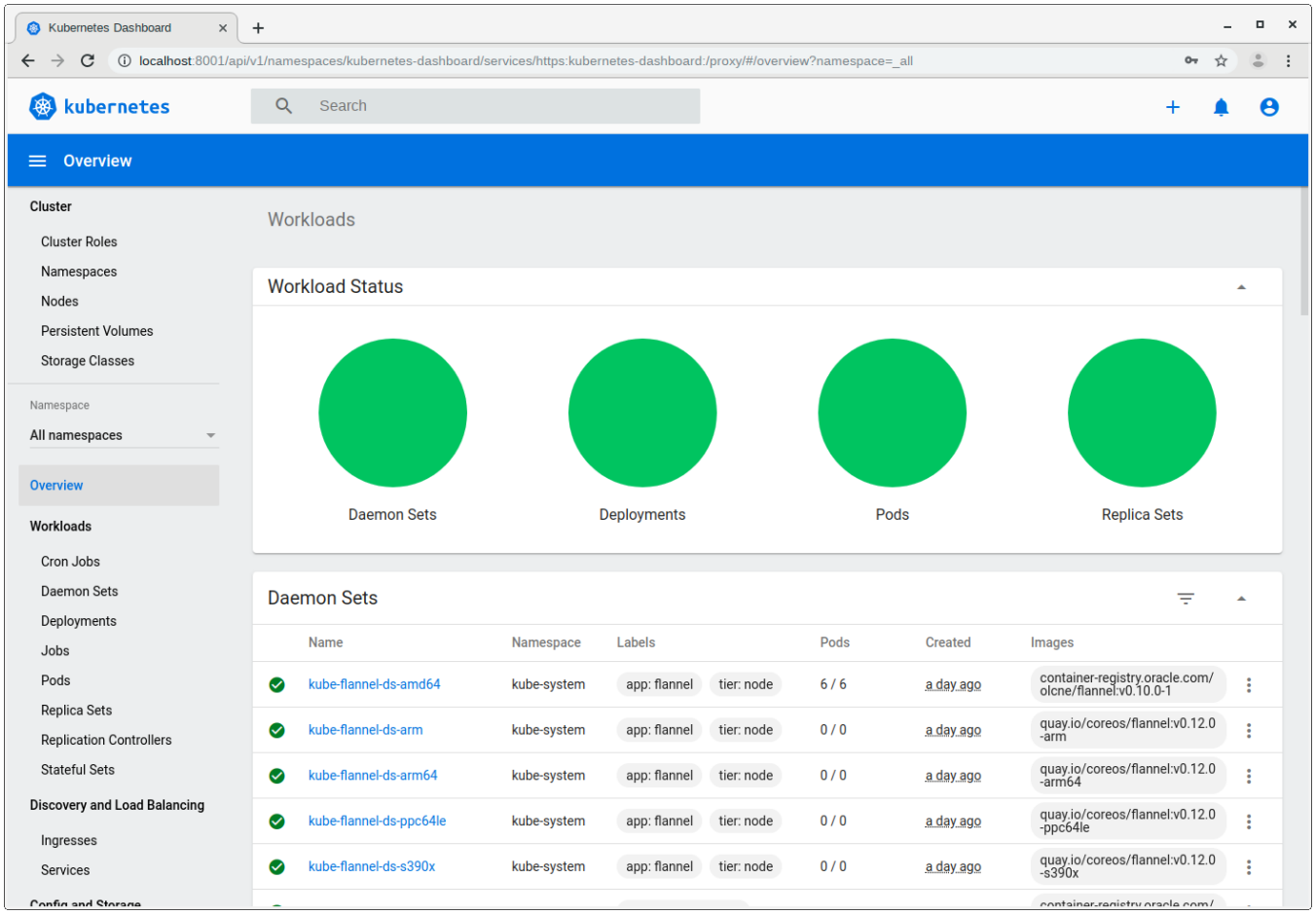


Image Caption 5. Kubernetes Dashboard for cluster 'sonit-3-cluster'





Removal and cleanup

Run these commands if you wish to remove the demo environment after completing the exercise. Alternatively, you can stop and delete the VMs used for this demonstration.

Uninstall Kubernetes module

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

Delete environment

```
$ olcnectl environment delete \
--environment-name myenvironment
```

WEBLOGIC SERVER IN OCNE ON PRIVATE CLOUD APPLIANCE

The OCNE environment established above is now ready to install Helm and WebLogic server. Helm is the Kubernetes package manager, used to locate and install software repositories for Kubernetes environments. Helm Charts are used to install, manage, and upgrade Kubernetes applications. See http://helm.sh for further information.

Install Helm and Clone WebLogic Repository

All WebLogic related commands are run on the host running `kubectl`. First we obtain the `get_help.sh` script from the Helm website, and then clone the WebLogic Kubenetes repository.

Install helm:

```
$ curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
$ chmod +x get_helm.sh
$ ./get_helm.sh
```

Clone Repository:

The next step clones the repository and populates a directory containing all the files needed to install WebLogic.

```
$ mkdir -p ~/WLS_K8S_OLCNE_demo
$ cd WLS_K8S_OLCNE_demo
$ git clone --branch v3.2.3 https://github.com/oracle/weblogic-kubernetes-operator
```

Install WebLogic Operator

Create namespace and service account for operator:

This step creates a namespace and service account for a Kubernetes operator:

```
$ kubectl create namespace sample-weblogic-operator- ns
$ kubectl create serviceaccount -n sample-weblogic-operator-ns sample-weblogic-operator-sa
```

Set up helm:

```
$ helm repo add traefik https://containous.github.io/traefik-helm-chart/ --force-update
```

Install operator using helm:

Specify the namespace, image, and the service account that was created earlier. The `enableClusterRoleBinding` option enables the operator to have privilege in all namespaces. The `labelselector` will allow the operator to manage any Kubernetes domains with the label `weblogic-operator` enabled. You can also add a debug option to see the process of installing the operator.

```
$ cd weblogic-kubernetes-operator
$ helm install sample-weblogic-operator kubernetes/charts/weblogic-operator \
  --namespace sample-weblogic-operator-ns \
  --set image=ghcr.io/oracle/weblogic-kubernetes-operator:3.2.5 \
  --set serviceAccount=sample-weblogic-operator-sa \
  --set "enableClusterRoleBinding=true" \
  --set "domainNamespaceSelectionStrategy=LabelSelector" \
  --set "domainNamespaceLabelSelector=weblogic-operator\=enabled" \
  --wait
```

Confirm WebLogic Operator

Run the following commands to confirm that the WebLogic operator is functional. The following commands check the pods, logs and Helm chart.

```
$ kubectl get pods -n sample-weblogic-operator-ns
$ kubectl logs -n sample-weblogic-operator -ns \
       -c weblogic-operator deployments/weblogic-operator
$ helm list -n sample-weblogic-operator-ns
```

Prepare for WebLogic Domain

The following steps create and label the domain namespace, create the Traefik namespace and install the Traefik ingress controller.

Create domain namespace:

```
$ kubectl create namespace sample-domain1-ns
```

Label domain namespace:

```
$ kubectl label ns sample-domain1-ns weblogic-operator=enabled
```

Create Traefik ingress controller's namespace:

```
$ kubectl create namespace traefik
```

Install Traefik ingress controller:

```
$ helm install traefik-operator traefik/traefik \
    --namespace traefik \
    --values kubernetes/samples/charts/traefik/values.yaml \
    --set "kubernetes.namespaces={traefik,sample-domain1-ns}"
```

Confirm Traefik install

Run the following commands to validate that Traefik is installed:

```
$ kubectl get pods -n traefik
$ kubectl get pods -n traefik -o wide # provides more detail
$ kubectl get svc -n traefik          # show all services running in the Traefik namespace
```

Note that the external IP field remains pending because for on-premises deployment there is no cloud provider that will assign an external IP.

Create Weblogic Domain

To create a domain, first create a Kubernetes secret for the WebLogic domain admin credentials using `the create-weblogic credentials` script that was downloaded.

Create domain credentials:

```
$ kubernetes/samples/scripts/create-weblogic-domain-credentials/create-weblogic-credentials.sh \
  -u weblogic -p welcome1 -n sample-domain1-ns -d sample-domain1
```

Upgrade WebLogic operator to manage domain:

This step adds the `domainNamespaces`

```
$ helm upgrade sample-weblogic-operator kubernetes/charts/weblogic-operator \
      --namespace sample-weblogic-operator-ns --reuse-values \
      --set "domainNamespaces={sample-domain1-ns}" --wait
```

Copy domain inputs yaml file to my version:

This step lets us create our own script settings by using a new yaml file instead of changing the distributed one.

```
$ cp create-domain-inputs.yaml my-inputs.yaml
```

Edit the following in `my-inputs.yaml`:

```
domainUID: sample-domain1
domainHome: /u01/oracle/user_projects/domains/sample-domain1
weblogicCredentialsSecretName: sample-domain1-weblogic-credentials
namespace: sample-domain1-ns
domainHomeImageBase: container-registry.oracle.com/middleware/weblogic:12.2.1.4
```

Finally – create the domain:

The next step runs the `create-domain` script with `my-input`, specifies the output directory that will contain the domain configuration yaml file, with the username of the secret created earlier, and the password.

```
$ ./create-domain.sh -i my-inputs.yaml -o ~/WLS_K8S_OLCNE_demo/weblogic-kubernetes-operator \
      -u weblogic -p welcome1 -e
```

Edit the domain.yaml file that was generated:

```
$ cd ~/WLS_K8S_OLCNE_demo/weblogic-kubernetes-operator/weblogic-domains/sample-domain1
$ vi domains.yaml
```

Edit the image name: `"iad.ocir.io/weblogick8s/weblogic-operator-tutorial-store:1.0"`

Uncomment the admin server part so the admin server can be exposed over a node port:

```
# adminServer is used to configure the desired behavior for starting the administration server.
  adminServer:
    # serverStartState legal values are "RUNNING" or "ADMIN"
```

```
    # "RUNNING" means the listed server will be started up to "RUNNING" mode
    # "ADMIN" means the listed server will be start up to "ADMIN" mode
    serverStartState: "RUNNING"
    adminService:
      channels:
    # The Admin Server's NodePort
       - channelName: default
         nodePort: 30701
    # Uncomment to export the T3Channel as a service
    #    - channelName: T3Channel
```

Then reapply the `domain.yaml` file to apply changes to the domain, then confirm the domain results.

```
$ kubectl apply -f domain.yaml
$ kubectl describe domain sample-domain1 -n sample-domain1-ns
$ kubectl get pods -n sample-domain1-ns
$ kubectl get pods -n sample-domain1-ns -o wide # for more details
$ kubectl get services -n sample-domain1-ns
```

Install the Ingress

```
$ cd ~/WLS_K8S_OLCNE_demo/weblogic-kubernetes-operator
$ helm install sample-domain1-ingress kubernetes/samples/charts/ingress-per-domain \
  --namespace sample-domain1-ns \
  --set wlsDomain.domainUID=sample-domain1 \
  --set traefik.hostname=my-traefik-host.FQDN
$ kubectl get ingress -n sample-domain1-ns
$ kubectl edit ingress -n sample-domain1-ns
```

Then add more traefik hostnames by issuing the above `kubectl edit ingress`

CONSIDERATIONS FOR PRODUCTION DEPLOYMENT

The preceding material describes a demonstration of Oracle Cloud Native Environment on Private Cloud Environment. For a production deployment, additional planning is needed. While this is outside the scope of this paper, issues to keep in mind are:

- Cluster sizing: sufficient capacity should be available to run peak loads both in terms of size of each worker node, and the number of nodes being used. OCNE provides the ability to scale up as needed.
- Upgrade planning: as new OS, Kubernetes and applications become available, there has to be planning to upgrade instances or to create new instances to test in parallel and then transition into production.
- Alignment of Kubernetes zones to hardware fault domains: workload should be distributed across separate physical hardware to prevent single point of failure (SPOF) taking the application down. OCNE and PCA support this by providing mechanisms to assign instances to different fault domains (hardware failure separation boundaries) or by setting 'anti-affinity' rules.
- PKI (Vault) management to store and control certificates, encryption keys and private name/value information needed by applications.
- Network planning
    - Segregate pod network from other traffic
    - Configure application load balancing
- Plan for persistent storage – see the OCNE documentation for further detail
    - Kubernetes storage classes
    - External databases

CONCLUSION

Oracle Private Cloud offers the most optimized infrastructure to host middleware and applications. They are ideal for consolidating enterprise mission-critical workloads along with modern cloud native containerized workloads and manage them from a single pane of glass. You can create fully HA Kubernetes clusters and scale them in minutes, simplifying your journey to digital transformation.

The integrated cloud native environment allows you to modernize your Weblogic server applications while providing you the best price/performance. Weblogic Kubernetes operator 3.0 is fully tested and supported on Oracle Private Cloud Appliance.

RESOURCES

- Oracle Private Cloud Appliance website
- Oracle Private Cloud Appliance documentation
- Oracle Cloud Native Environment documentation https://docs.oracle.com/en/operating-systems/olcne/start/
- *Introducing Oracle Verrazzano Enterprise Container Platform*: https://blogs.oracle.com/developers/post/introducing-oracle-verrazzano-enterprise-container-platform
- Weblogic Kubernetes Operator User Guide https://oracle.github.io/weblogic-kubernetes-operator/
- Helm description and documentation https://helm.sh

CONNECT WITH US

Call +1.800.ORACLE1 or visit oracle.com.
Outside North America, find your local office at oracle.com/contact.

blogs.oracle.com facebook.com/oracle twitter.com/oracle

Oracle Cloud Native Environment on Oracle Private Cloud Appliance
November 2121
Author: Sonit Tayal