Oracle® FMW

Deploying and Managing Oracle HTTP Server on Kubernetes

14.1.2
G22975-05
July 2025

Oracle FMW Deploying and Managing Oracle HTTP Server on Kubernetes, 14.1.2

G22975-05

Primary Authors: Russell Hodgson,

Contents

Preface

Deploying and Managing Oracle HTTP Server on Kubernetes describes how to deploy, configure, and administer Oracle HTTP Server on Kubernetes.

Audience

This guide is intended for:

- Administrators responsible for deploying and managing Oracle HTTP Server on Kubernetes

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit My Oracle Support or visit About Oracle Accessibility Oracle Accessibility Learning and Support if you are hearing impaired.

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.

Related Documents

For general information about Oracle HTTP Server, Oracle WebGate, and Oracle Access Management, see the following documentation:

- Administering Oracle HTTP Server

- Administering Oracle Access Management
- Deploying Oracle Access Management 12c on Kubernetes

List of Figures

1

What's New in This Guide

This preface shows current and past versions of Oracle HTTP Server (OHS) 14c container images and deployment scripts on Kubernetes. If any new functionality is added, details are outlined.

Date	Version	Change
July 2025	14.1.2.0.0 GitHub release version 25.3.1	Supports Oracle HTTP Server 14.1.2.0.0 deployment using the July 2025 OHS container image which contains the July Patch Set Update (PSU) and other fixes released with the Critical Patch Update (CPU) program. The GitHub release version is the latest version of the deployment scripts used in Setting Up the Code Repository for OHS.
April 2025	14.1.2.0.0 GitHub release version 25.2.1	Supports Oracle HTTP Server 14.1.2.0.0 deployment using the April 2025 OHS container image which contains the April Patch Set Update (PSU) and other fixes released with the Critical Patch Update (CPU) program. The GitHub release version is the latest version of the deployment scripts used in Setting Up the Code Repository for OHS.
January 2025	14.1.2.0.0 GitHub release version 24.4.3	Initial release of Oracle HTTP Server 14.1.2.0.0 on Kubernetes. Supports Oracle HTTP Server 14.1.2.0.0 deployment using the OHS container image. The GitHub release version is the latest version of the deployment scripts used in Setting Up the Code Repository for OHS.

Part I

Introduction to Oracle HTTP Server on Kubernetes

This section includes the following chapters:

2

Introducing Oracle HTTP Server on Kubernetes

Oracle HTTP Server (OHS) is supported for deployment on Kubernetes.

This documentation explains how to configure Oracle HTTP Server (OHS) on a shared Kubernetes cluster where other applications are deployed, or on it’s own independent Kubernetes cluster.

Please note that this documentation does not explain how to configure a Kubernetes cluster given the product can be deployed on any compliant Kubernetes vendor.

This documentation assumes you are familiar with OHS and it’s configuration files. It also assumes that if you are using this OHS with Oracle WebGate and Oracle Access Management, that you are familiar with these products.

3

About the Kubernetes Deployment

Containers offer an excellent mechanism to bundle and run applications. In a production environment, you have to manage the containers that run the applications and ensure there is no downtime. For example, if a container goes down, another container has to start immediately. Kubernetes simplifies container management.

This chapter includes the following topics:

3.1 What is Kubernetes?

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services that facilitates both declarative configuration and automation.

Kubernetes sits on top of a container platform such as CRI-O or Docker. Kubernetes provides a mechanism which enables container images to be deployed to a cluster of hosts. When you deploy a container through Kubernetes, Kubernetes deploys that container on one of its worker nodes. The placement mechanism is transparent to the user.

Kubernetes provides:

- **Service Discovery and Load Balancing**: Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes balances the load and distributes the network traffic so that the deployment remains stable.
- **Storage Orchestration**: Kubernetes enables you to automatically mount a storage system of your choice, such as local storages, NAS storages, public cloud providers, and more.
- **Automated Rollouts and Rollbacks**: You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers, and adopt all their resources to the new container.
- **Automatic Bin Packing**: If you provide Kubernetes with a cluster of nodes that it can use to run containerized tasks, and indicate the CPU and memory (RAM) each container needs, Kubernetes can fit containers onto the nodes to make the best use of the available resources.
- **Self-healing**: Kubernetes restarts containers that fail, replaces containers, kills containers that do not respond to your user-defined health check, and does not advertise them to clients until they are ready to serve.
- **Secret and Configuration Management**: Kubernetes lets you store and manage sensitive information such as passwords, OAuth tokens, and SSH keys. You can deploy and update

secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.

When deploying Kubernetes, Oracle highly recommends that you use the traditional recommendations of keeping different workloads in separate Kubernetes clusters. For example, it is not a good practice to mix development and production workloads in the same Kubernetes cluster.

3.2 About the Kubernetes Architecture

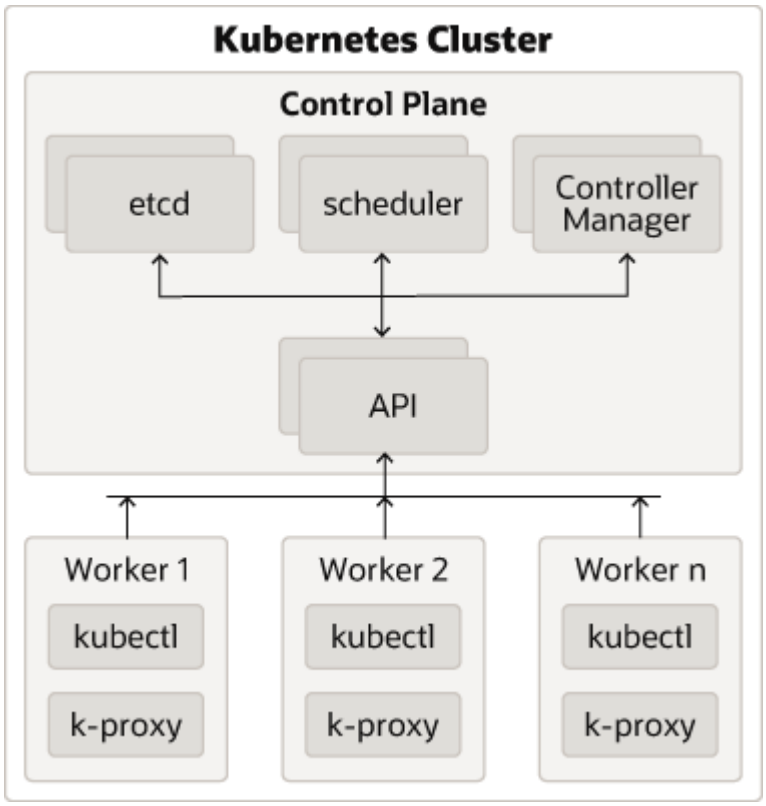
A Kubernetes host consists of a control plane and worker nodes.

Control Plane: A control plane is responsible for managing the Kubernetes components and deploying applications. In an enterprise deployment, you need to ensure that the Kubernetes control plane is highly available so that the failure of a control plane host does not fail the Kubernetes cluster.

Worker Nodes: Worker nodes which are where the containers are deployed.

> **Note**
>
> An individual host can be both a control plane host and a worker host.

Figure 3-1 An Illustration of the Kubernetes Cluster



Description of Components:

- **Control Plane**: The control plane comprises the following:
  - kube-api server: The API server is a component of the control plane that exposes the Kubernetes APIs.
  - etcd: It is used to store the Kubernetes backing store and all the cluster data.
  - Scheduler: The scheduler is responsible for the placement of containers on the worker nodes. It takes into account resource requirements, hardware and software policy constraints, affinity specifications, and data affinity.
  - Control Manager: It is responsible for running the controller processes. Controller processes consist of:
    * Node Controller
    * Route Controller
    * Service Controller

  The control plane consists of three nodes where the Kubernetes API server is deployed, front ended by an LBR.
- **Worker Node Components**: The worker nodes include the following components:
  - Kubelet: An Agent that runs on each worker node in the cluster. It ensures that the containers are running in a pod.
  - Kube Proxy: Kube proxy is a network proxy that runs on each node of the cluster. It maintains network rules, which enable inter pod communications as well as communications outside of the cluster.
  - Add-ons: Add-ons extend the cluster further, providing such services as:
    * DNS
    * Web UI Dashboard
    * Container Resource Monitoring
    * Logging

3.3 Key Components Used By an OHS Deployment

An Oracle HTTP Server (OHS) deployment uses the Kubernetes components such as pods and Kubernetes services.

Container Image

A container image is an unchangeable, static file that includes executable code. When deployed into Kubernetes, it is the container image that is used to create a pod. The image contains the system libraries, system tools, and Oracle binaries required to run in Kubernetes. The image shares the OS kernel of its host machine.

A container image is compiled from file system layers built onto a parent or base image. These layers encourage the reuse of various components. So, there is no need to create everything from scratch for every project.

A pod is based on a container image. This container image is read-only. Each pod has its own instance of a container image.

A container image contains all the software and libraries required to run the product. It does not require the entire operating system. Many container images do not include standard operating utilities such as the vi editor or ping.

When you upgrade a pod, you are actually instructing the pod to use a different container image. For example, if the container image for Oracle HTTP Server is based on the July 2025 bundle patch, then to upgrade the pod to use the July 2025 bundle patch, you have to tell the pod to use the July 2025 image and restart the pod. Further information on upgrading can be found in Patching and Upgrading.

Pods

A pod is a group of one or more containers, with shared storage/network resources, and a specification for how to run the containers. A pod's contents are always co-located and co-scheduled, and run in a shared context. A pod models an application-specific logical host that contains one or more application containers which are relatively tightly coupled.

In an Oracle HTTP Server (OHS) deployment, each OHS runs in a different pod.

If a node becomes unavailable, Kubernetes does not delete the pods automatically. Pods that run on an unreachable node attain the 'Terminating' or 'Unknown' state after a timeout. Pods may also attain these states when a user attempts to delete a pod on an unreachable node gracefully. You can remove a pod in such a state from the apiserver in one of the following ways:

- You or the Node Controller deletes the node object.
- The kubelet on the unresponsive node starts responding, terminates the pod, and removes the entry from the apiserver.
- You force delete the pod.

Oracle recommends the best practice of using the first or the second approach. If a node is confirmed to be dead (for example: permanently disconnected from the network, powered down, and so on), delete the node object. If the node suffers from a network partition, try to resolve the issue or wait for the partition to heal. When the partition heals, the kubelet completes the deletion of the pod and frees up its name in the apiserver.

Typically, the system completes the deletion if the pod is no longer running on a node or an administrator has deleted it. You may override this by force deleting the pod.

Pod Scheduling

By default, Kubernetes will schedule a pod to run on any worker node that has sufficient capacity to run that pod. In some situations, it is desirable that scheduling occurs on a subset of the worker nodes available. This type of scheduling can be achieved by using Kubernetes labels.

Kubernetes Services

Kubernetes services expose the processes running in the pods regardless of the number of pods that are running. For example, Oracle HTTP Servers each running in different pods will have a service associated with them. This service will redirect your request to the individual pods in the cluster.

Kubernetes services can be internal or external to the cluster. Internal services are of the type ClusterIP and external services are of the type NodePort.

Some deployments use a proxy in front of the service. This proxy is typically provided by an 'Ingress' load balancer such as **Ngnix**. Ingress allows a level of abstraction to the underlying Kubernetes services.

In this guide, Oracle HTTP Server (OHS) is exposed as type NodePort and Ingress is not used to access OHS.

The Kubernetes services use a small port range. Therefore, when a Kubernetes service is created, there will be a port mapping. For instance, if an OHS pod is using port 7777, then a Kubernetes nodeport service may use 30777 as its port, mapping port 30777 to 7001 internally. It is worth noting that using individual NodePort Services, the corresponding Kubernetes service port will be reserved on every worker node in the cluster.

Kubernetes/ingress services are known to each worker node, regardless of the worker node on which the containers are running. Therefore, a load balancer is often placed in front of the worker node to simplify routing and worker node scalability.

If OHS is communicating using mod_wl_ohs to a WebLogic Server, then it interacts with those services using the format: worker_node_hostname:Service port. This format is applicable whether you are using individual NodePort Services or a consolidated Ingress node port service.

If OHS communicates with multiple WebLogic worker nodes, then you should include multiple worker nodes in your calls to remove single points of failure. In this guide OHS makes direct proxy calls using WebLogicCluster directives. More information on this can be found in Supported Architectures for Oracle HTTP Server.

Ingress Controller

Whilst this guide uses NodePort for Oracle HTTP Server (OHS) access, if OHS communicates with WebLogic Server on an independent Kubernetes cluster, then the WebLogicCluster directive should point to the port of the Ingress controller used by WebLogic Server.

Ingress is a proxy server which sits inside the Kubernetes cluster, unlike the NodePort Services which reserve a port per service on every worker node in the cluster. With an ingress service, you can reserve single ports for all HTTP / HTTPS traffic.

An Ingress service works in the same way as the Oracle HTTP Server. It has the concept of virtual hosts and can terminate SSL, if required.

More information on this can be found in Supported Architectures for Oracle HTTP Server.

Domain Name System

Every service defined in the cluster (including the DNS server itself) is assigned a DNS name. By default, a client pod's DNS search list includes the pod's own namespace and the cluster's default domain.

The following types of DNS records are created for a Kubernetes cluster:

- **Services**
  Record Type: A or AAAA record

  Name format: my-svc.namespace.svc.cluster-example.com
- **Pods**
  Record Type: A or AAAA record

  Name format: podname.namespace.pod.cluster-example.com

Kubernetes uses a built-in DNS server called 'CoreDNS' which is used for the internal name resolution.

External name resolution (names used outside of the cluster, for example: loadbalancer.example.com) may not possible inside the Kubernetes cluster. If you encounter this issue, you can use one of the following options:

- **Option 1** - Add a secondary DNS server to CoreDNS for the company domain.

- **Option 2** - Add individual host entries to CoreDNS for the external hosts. For example: loadbalancer.example.com

Namespaces

Namespaces enable you to organize clusters into virtual sub-clusters which are helpful when different teams or projects share a Kubernetes cluster. You can add any number of namespaces within a cluster, each logically separated from others but with the ability to communicate with each other.

In this guide the OHS deployment uses the namespace ohsns.

3.4 Supported Architectures for Oracle HTTP Server

Oracle HTTP Server (OHS) can be deployed in the following Kubernetes scenarios:

- Oracle HTTP Server deployed on a shared Kubernetes cluster with other applications.
- Oracle HTTP Server deployed on an independent Kubernetes cluster.

Before deploying OHS you must consider what architecture to deploy and then plan accordingly.

Oracle HTTP Server on a Shared Kubernetes Cluster

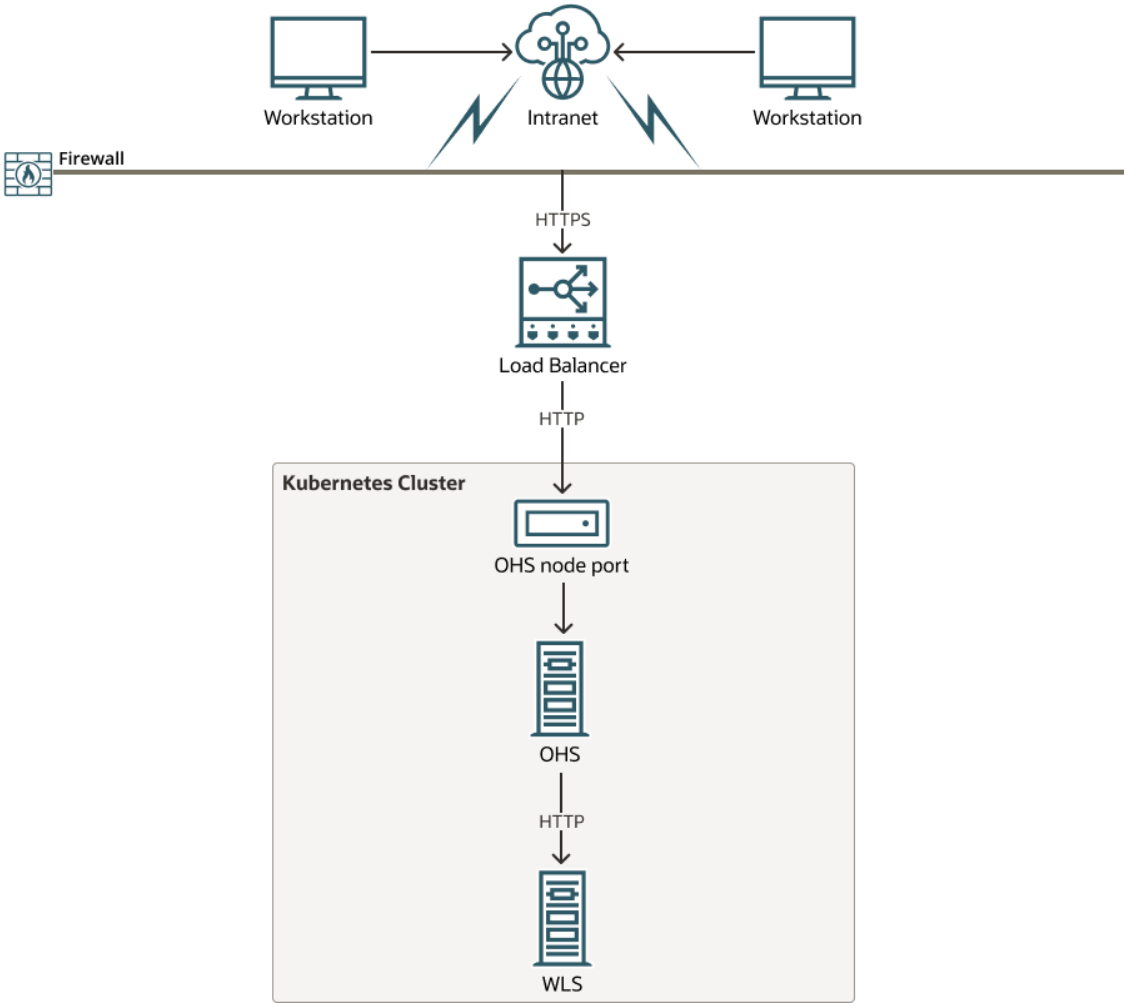
This deployment is recommended for sandbox and intranet environments only.

In this deployment OHS is installed on the same Kubernetes cluster as other Kubernetes deployed applications. For example, you may want to deploy OHS on the same Kubernetes cluster as other Oracle products such as Oracle WebLogic Server, or Oracle Access Management (OAM).

If OHS needs to communicate with other applications on the same Kubernetes cluster, for example using mod_wls_ohs to communicate with Oracle WebLogic Server, then the OHS communicates to the internal port of that Kubernetes service.

An example architecture is as follows:

Figure 3-2 OHS on a Shared Kubernetes Cluster



In this example:

- OHS is deployed in the same Kubernetes cluster as Oracle WebLogic Server.
- SSL is terminated at the load balancer.
- The load balancer communicates with OHS via the OHS nodeport using HTTP.
- OHS communicates with the WebLogic Administration and WebLogic Managed Servers via HTTP, using the internal port of the associated Kubernetes service: <service_name>.<namespace>.svc.cluster.local:<port>. The <service_name> and <port> can be found by running kubectl get svc -n <namespace> on your Kubernetes cluster.

> **Note**
>
> The load balancer is optional and you can connect direct to the OHS nodeport if required, either from workstations outside the firewall, or internally from other applications.

Oracle HTTP Server on an Independent Kubernetes Cluster

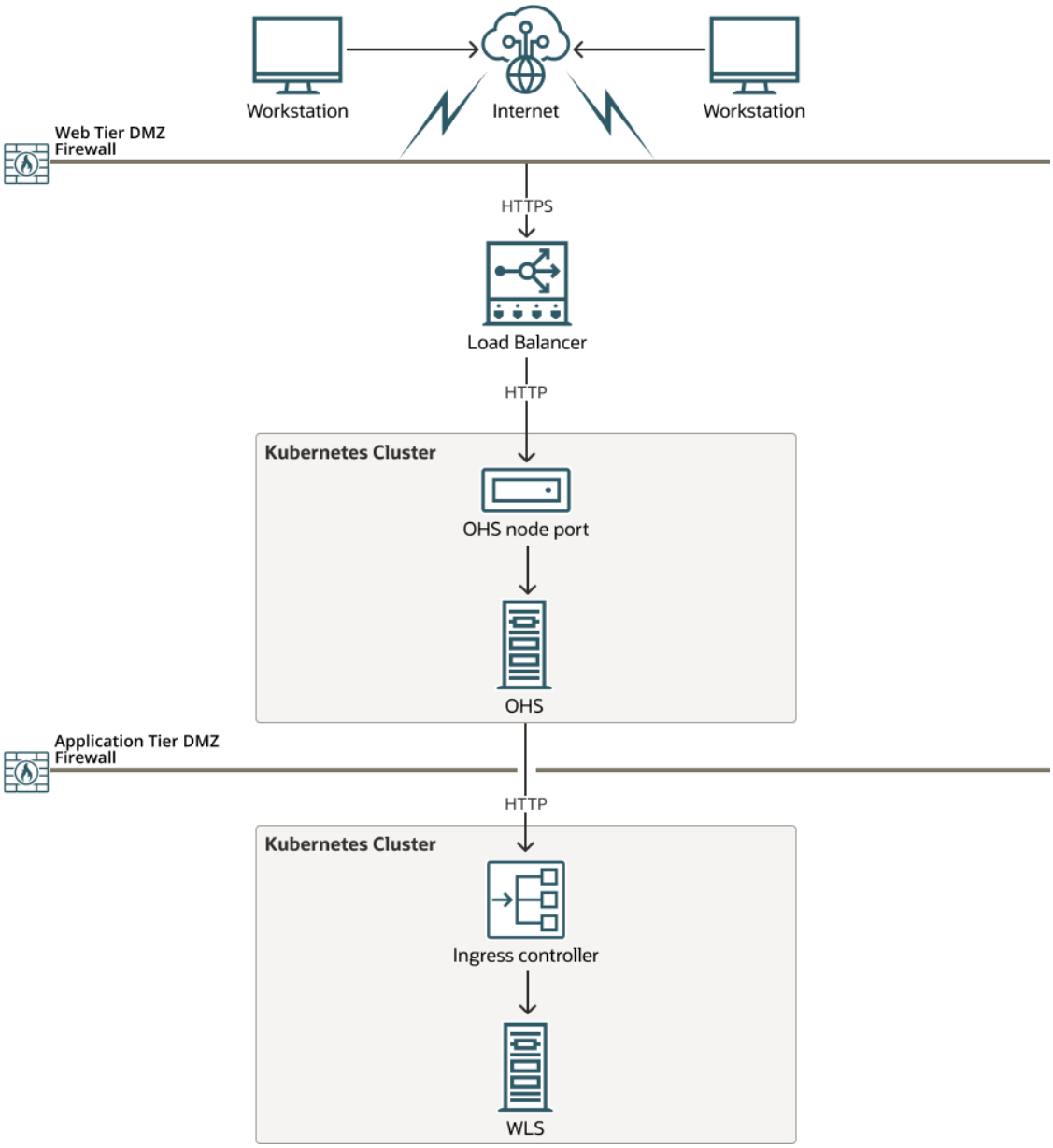
This deployment is recommended for internet facing, production environments.

In this deployment OHS is installed on it's own Kubernetes cluster inside a demilitarized zone.

If OHS needs to communicate with other applications deployed on a different Kubernetes cluster, for example using mod_wl_ohs to communicate with Oracle WebLogic Server, then the OHS communicates via HTTP to the Ingress controller port used by the application on that Kubernetes cluster, or to the appropriate NodePort Kubernetes service you wish to connect to.

An example architecture is as follows:

Figure 3-3 OHS on an Independent Kubernetes Cluster



In this example:

- SSL is terminated at the load balancer.
- The load balancer communicates with OHS via the OHS nodeport using HTTP.
- WebLogic Server is deployed on it's own Kubernetes cluster.
- OHS communicates with the WebLogic Administration and WebLogic Managed Servers using the HTTP port of the Ingress controller configured for WebLogic Server.

> **Note**
>
> The load balancer is optional and you can connect direct to the OHS nodeport if required, either from workstations outside the firewall, or internally from other applications.

3.5 Requirements for Oracle HTTP Server on Kubernetes

This section provides information about the system requirements and limitations for deploying and running Oracle HTTP Server (OHS) on Kubernetes.

System Requirements for OHS on Kubernetes

You must have a running Kubernetes cluster that meets the following requirements:

- The Kubernetes cluster and container engine must meet the minimum version requirements outlined in document ID 3058838.1 on My Oracle Support.
- An administrative host from which to deploy the products. This host could be a Kubernetes Control host, a Kubernetes Worker host, or an independent host. This host must have `kubectl` deployed using the same version as your cluster.
- The Kubernetes cluster must have sufficient nodes and resources.
- A supported container engine such as CRI-O or Docker must be installed and running on the Kubernetes cluster.
- The system clocks on node of the Kubernetes cluster must be synchronized. Run the date command simultaneously on all the nodes in each cluster and then synchronize accordingly.

> **Note**
>
> This documentation does not tell you how to install a Kubernetes cluster, the container engine, or how to push container images to a container registry. Please refer to your vendor specific documentation for this information.

Before deploying OHS you must consider what architecture to deploy and then plan accordingly.

Oracle Access Management Requirements

If you intend to use OHS with Oracle WebGate and Oracle Access Management (OAM), then OAM must have been deployed beforehand, either in an on-premises environment, or in a Kubernetes cluster. You must have an understanding of OAM and Oracle WebGate before proceeding.

Instructions for deploying OAM 14.1.2.1.0 in a Kubernetes cluster can be found in Deploying and Managing Oracle Access Management on Kubernetes. OAM in a Kubernetes cluster must be deployed as per one of the supported architectures defined. See, Supported Architectures for Oracle HTTP Server.

To use Oracle WebGate with OHS you must perform the following before deploying OHS:

- Update the Load Balancing and WebGate Traffic Load Balancer to the entry point for OAM. For example, if OAM is accessed via the load balancer (https://loadbalancer.example.com), then the **OAM Server Host**, **OAM Server Port**, and **OAM Server Protocol** should be updated to loadbalancer.example.com, 443, and HTTPS respectively. For more information, see Updating the OAM Hostname and Port for the Loadbalancer.
- Create an Agent in the Oracle Access Management console. After creating the agent, make sure the **User Defined Parameters** for OAMRestEndPointHostName, OAMRestEndPointPort, and OAMServerCommunicationMode are set to the same values as the load balancing settings above. See, Registering a WebGate Agent.
- In the Application Domain created for the WebGate, update the resources with any resources you wish to protect.
- Create any **Host Identifier(s)** for any URL's you require. For example if you access OAM via a load balancer, create a host identifier for both the load balancer hostname.domain and the OHS hostname.domain. If you access OAM directly via OHS, create a host identifier for the OHS hostname.domain. See, Creating Host Identifiers.
- Download the zip file for the Agent from the OAM Console. This zip file will later be copied and extracted to the $WORKDIR/ohsConfig/webgate/config directory. See, Preparing Your OHS Configuration Files.

Part II
Installing Oracle HTTP Server on Kubernetes

This section includes the following chapters:

- Preparing Your Environment
- Deploying Oracle HTTP Server on Kubernetes

4

Preparing Your Environment

Before starting an Oracle HTTP Server (OHS) deployment on Kubernetes, you must prepare your environment.

This chapter includes the following topics:

4.1 Confirm the Kubernetes Cluster is Ready

As per Requirements for Oracle HTTP Server on Kubernetes, a Kubernetes cluster should have already been configured.

1. Run the following command on the Kubernetes administrative node to check the cluster and worker nodes are running:

```
kubectl get nodes,pods -n kube-system
```

The output will look similar to the following:

```
NAME              STATUS   ROLES                  AGE   VERSION
node/worker-node1    Ready    <none>                 17h   v1.29.9+3.el8
node/worker-node2    Ready    <none>                 17h   v1.29.9+3.el8
node/master-node     Ready    control-plane,master   23h   v1.29.9+3.el8

NAME                                  READY   STATUS    RESTARTS   AGE
pod/coredns-66bff467f8-fnhbq          1/1     Running   0          23h
pod/coredns-66bff467f8-xtc8k          1/1     Running   0          23h
pod/etcd-master                       1/1     Running   0          21h
pod/kube-apiserver-master-node        1/1     Running   0          21h
pod/kube-controller-manager-master-node  1/1     Running   0          21h
pod/kube-flannel-ds-amd64-lxsfw       1/1     Running   0          17h
pod/kube-flannel-ds-amd64-pqrqr       1/1     Running   0          17h
pod/kube-flannel-ds-amd64-wj5nh       1/1     Running   0          17h
pod/kube-proxy-2kxv2                  1/1     Running   0          17h
```

```
pod/kube-proxy-82vvj              1/1     Running   0          17h
pod/kube-proxy-nrgw9              1/1     Running   0          23h
pod/kube-scheduler-master         1/1     Running   0          21h
```

4.2 Obtaining the OHS Container Image

The Oracle HTTP Server (OHS) Kubernetes deployment requires access to an OHS container image.

The OHS container image can be obtained in the following ways:

- Prebuilt OHS container image
- Build your own OHS container image using WebLogic Image Tool

Prebuilt OHS container image

The latest prebuilt OHS 14.1.2 container image can be downloaded from Oracle Container Registry. This image is prebuilt by Oracle and includes Oracle HTTP Server 14.1.2.0.0, the latest Patch Set Update (PSU) and other fixes released with the Critical Patch Update (CPU) program.

> **Note**
>
> Before using this image you must login to Oracle Container Registry, navigate to **Middleware** > **ohs** and accept the license agreement. For future releases (post January 25) that contain the latest Patch Set Update (PSU) and other fixes released with the Critical Patch Update (CPU) program, you should navigate to **Middleware** >**ohs_cpu**.

You can use this image in the following ways:

- Pull the container image from the Oracle Container Registry automatically during the OHS Kubernetes deployment.
- Manually pull the container image from the Oracle Container Registry and then upload it to your own container registry.
- Manually pull the container image from the Oracle Container Registry and manually stage it on each worker node.

Build your own OHS container image using WebLogic Image Tool

You can build your own OHS container image using the WebLogic Image Tool. This is recommended if you need to apply one off patches to a prebuilt OHS container image. For more information about building your own container image with WebLogic Image Tool, see Creating or Updating an OHS Image.

You can use an image built with WebLogic Image Tool in the following ways:

- Manually upload them to your own container registry.
- Manually stage them on each worker node.

> **Note**
>
> This documentation does not tell you how to pull or push the above images into a private container registry, or stage them on worker nodes. Details of this can be found in the Enterprise Deployment Guide. See, Procuring Software for an Enterprise Deployment.

4.3 Setting Up the Code Repository for OHS

To deploy Oracle HTTP Server (OHS) you need to set up the code repository which provides sample deployment yaml files.

Perform the following steps on a node that has access to the Kubernetes cluster.

1. Create a directory to setup the source code:

   ```
   mkdir <ohsscripts>
   ```

   For example:

   ```
   mkdir -p /OHSK8S/OHSscripts
   ```

2. Download the latest OHS deployment scripts from the OHS repository:

   ```
   cd <ohsscripts>
   git clone https://github.com/oracle/fmw-kubernetes.git
   ```

   For example:

   ```
   cd /OHSK8S/OHSscripts
   git clone https://github.com/oracle/fmw-kubernetes.git
   ```

3. Set the $SCRIPTDIR environment variable as follows:

   ```
   export SCRIPTDIR=<ohsscripts>/fmw-kubernetes/OracleHTTPServer/kubernetes
   ```

   For example:

   ```
   export SCRIPTDIR=/OHSK8S/OHSscripts/fmw-kubernetes/OracleHTTPServer/kubernetes
   ```

4.4 Preparing Your OHS Configuration Files

Before you deploy Oracle HTTP Server (OHS), you must prepare your OHS configuration files.

The steps below assume familiarity with on premises Oracle HTTP Server in terms of general configuration, and use of Oracle WebGate.

> **Note**
>
> Administrators should be aware of the following:
>
> - If you do not specify configuration files beforehand, then the OHS container is deployed with a default configuration of Oracle HTTP Server.
> - The directories listed below are optional. For example, if you do not want to deploy WebGate then you do not need to create the webgateConf and webgateWallet directories. Similarly, if you do not want to copy files to htdocs then you do not need to create the htdocs directory.

1. Make a directory to store your OHS configuration files:

   ```
   mkdir -p <myohsfiles>
   ```

   For example:

   ```
   mkdir -p /OHSK8S/myOHSfiles
   ```

2. Set the $MYOHSFILES environment variable as follows:

   ```
   export MYOHSFILES=<myohsfiles>
   ```

   For example:

   ```
   export MYOHSFILES=/OHSK8S/myOHSfiles
   ```

3. Create the following directories for your OHS configuration:

   ```
   mkdir -p $MYOHSFILES/ohsConfig/httpconf
   mkdir -p $MYOHSFILES/ohsConfig/moduleconf
   mkdir -p $MYOHSFILES/ohsConfig/htdocs
   mkdir -p $MYOHSFILES/ohsConfig/htdocs/myapp
   mkdir -p $MYOHSFILES/ohsConfig/webgate/config/wallet
   mkdir -p $MYOHSFILES/ohsConfig/wallet/mywallet
   ```

   Where:

   - httpconf - contains any configuration files you want to configure that are usually found in the $OHS_DOMAIN_HOME/config/fmwconfig/components/OHS/ohs1 directory. For example httpd.conf, ssl.conf and mod_wl_ohs.conf. The webgate.conf does not need to be copied as this will get generated automatically if deploying with Oracle WebGate.
   - moduleconf - contains any additional config files, for example virtual host configuration files that you want to copy to the $OHS_DOMAIN_HOME/config/fmwconfig/components/OHS/ohs1/moduleconf folder in the container.
   - htdocs - contains any html files, or similar, that you want to copy to the $OHS_DOMAIN_HOME/config/fmwconfig/components/OHS/ohs1/htdocs folder in the container.
   - htdocs/myapp - myapp is an example directory name that exists under htdocs. If you need to copy any directories under htdocs above, then create the directories you require.

- webgate/config - contains the extracted Oracle WebGate configuration. For example, when you download the <agent>.zip file from Oracle Access Management (OAM) Console, you extract the zip file into this directory. If you are accessing OAM URL's via SSL, this directory must also contain the Certificate Authority cacert.pem file(s) that signed the certificate of the OAM entry point. For example, if you will access OAM via a HTTPS Load Balancer URL, then cacert.pem is the CA certificate(s) that signed the load balancer certificate.
- webgate/config/wallet - contains the contents of the wallet directory extracted from the <agent>.zip file.
- wallet/mywallet - if OHS is to be configured to use SSL, this directory contains the preconfigured OHS Wallet file, cwallet.sso.

> **Note**
>
> Administrators should be aware of the following if configuring OHS for SSL:
>
> - The wallet must contain a valid certificate.
> - Only auto-login-only wallets (cwallet.sso only) are supported. For example, wallets created with orapki using the auto-login-only option. Password protected wallets (ewallet.p12) are not supported.
> - You must configure ssl.conf in $MYOHSFILES/ohsConfig/httpconf and set the directory for SSLWallet to:
>
>   SSLWallet "${ORACLE_INSTANCE}/config/fmwconfig/components/${COMPONENT_TYPE}/instances/${COMPONENT_NAME}/keystores/wallet/mywallet"

An example file system may contain the following:

```
ls -R $MYOHSFILES/ohsConfig
/OHSK8S/myOHSfiles/ohsConfig:
htdocs  httpconf  moduleconf  wallet  webgate

/OHSK8S/myOHSfiles/ohsConfig/htdocs:
myapp  mypage.html

/OHSK8S/myOHSfiles/ohsConfig/htdocs/myapp:
index.html

/OHSK8S/myOHSfiles/ohsConfig/httpconf:
httpd.conf  mod_wl_ohs.conf  ssl.conf

/OHSK8S/myOHSfiles/ohsConfig/moduleconf:
vh.conf

/OHSK8S/myOHSfiles/ohsConfig/wallet:
mywallet

/OHSK8S/myOHSfiles/ohsConfig/wallet/mywallet:
cwallet.sso

/OHSK8S/myOHSfiles/ohsConfig/webgate:
```

```
config

/OHSK8S/myOHSfiles/ohsConfig/webgate/config:
cacert.pem  cwallet.sso  cwallet.sso.lck  ObAccessClient.xml  wallet

/OHSK8S/myOHSfiles/ohsConfig/webgate/config/wallet:
cwallet.sso  cwallet.sso.lck
```

Set WLDNSRefreshInterval and WebLogicCluster Directives

If your OHS deployment is configured to communicate with Oracle WebLogic Server, then you must set the WLDNSRefreshInterval and WebLogicCluster directives in your OHS configuration files appropriately.

In the file where your WLS location directives reside, you must set the following:

```
<IfModule weblogic_module>
WLDNSRefreshInterval 10
</IfModule>
```

For WebLogicCluster, the values to set depend on whether the WLS is deployed on-premises, on the same Kubernetes cluster as OHS, or on a different Kubernetes cluster to OHS. The following sections explain how to set the values in each case.

On-premises Configuration
If OHS is connecting to a WebLogic Server deployed in an on-premises configuration (non-Kubernetes), then set:

```
WebLogicCluster <APPHOST1>:<PORT>,<APPHOST2>:<PORT>
```

For example, if you were connecting to the WebLogic Server Administration Server port:

```
<Location /console>
  WLSRequest ON
  DynamicServerList OFF
  WLProxySSL ON
  WLProxySSLPassThrough ON
  WLCookieName OAMJSESSIONID
  WebLogicCluster APPHOST1.example.com:7001,APPHOST2.example.com:7001
 </Location>
```

Oracle HTTP Server on a Shared Kubernetes Cluster
If OHS is connecting to a WebLogic Server deployed on the same Kubernetes cluster, then set the following depending on your environment:

```
WebLogicHost <service_name>.<namespace>.svc.cluster.local
WebLogicPort <port>
```

or:

```
WebLogicCluster <service_name>.<namespace>.svc.cluster.local:<port>
```

> **Note**
>
> You can get the <service_name> and <port> by running kubectl get svc -n <namespace> on your Kubernetes cluster.

The following shows an example when connecting to an Oracle Access Management (OAM) Managed Server cluster service and port:

```
<Location /oam>
WLSRequest ON
DynamicServerList OFF
WLProxySSL ON
WLProxySSLPassThrough ON
WLCookieName OAMJSESSIONID
WebLogicCluster accessdomain-cluster-oam-cluster.oamns.svc.cluster.local:14100
```

Oracle HTTP Server on an Independent Kubernetes Cluster
If OHS is connecting to a WebLogic Server deployed on a separate Kubernetes cluster, then set:

```
WebLogicCluster
<K8S_WORKER_HOST1>:30777,<K8S_WORKER_HOST2>:30777,<K8S_WORKER_HOST3>:30777
```

Where <K8S_WORKER_HOSTX> is your Kubernetes worker node hostname.domain, and 30777 is the HTTP port of the ingress controller.

For example:

```
<Location /console>
  WLSRequest ON
  DynamicServerList OFF
  WLProxySSL ON
  WLProxySSLPassThrough ON
  WLCookieName OAMJSESSIONID
  WebLogicCluster
K8_WORKER_HOST1.example.com:30777,K8_WORKER_HOST2.example.com:30777,K8_WORKER_HOST3.e
xample.com:30777
 </Location>
```

4.5 Creating the OHS Namespace

Create a Kubernetes namespace for Oracle HTTP Server (OHS).

Run the following command to create a namespace for OHS:

```
kubectl create namespace <namespace>
```

For example:

```
kubectl create namespace ohsns
```

The output will look similar to the following:

```
namespace/ohsns created
```

4.6 Creating ConfigMaps for the OHS Configuration Files

Create the required Kubernetes ConfigMaps for the Oracle HTTP Server (OHS) configuration files.

Before following this section, make sure you have created the directories and files as per Preparing Your OHS Configuration Files.

Run the following commands to create the required ConfigMaps for the OHS directories and files created in Preparing Your OHS Configuration Files:

```
cd $MYOHSFILES
kubectl create cm -n ohsns ohs-config --from-file=ohsConfig/moduleconf
kubectl create cm -n ohsns ohs-httpd --from-file=ohsConfig/httpconf
kubectl create cm -n ohsns ohs-htdocs --from-file=ohsConfig/htdocs
kubectl create cm -n ohsns ohs-myapp --from-file=ohsConfig/htdocs/myapp
kubectl create cm -n ohsns webgate-config --from-file=ohsConfig/webgate/config
kubectl create cm -n ohsns webgate-wallet --from-file=ohsConfig/webgate/config/wallet
kubectl create cm -n ohsns ohs-wallet --from-file=ohsConfig/wallet/mywallet
```

> **Note**
>
> Only create the ConfigMaps for directories that you want to copy to the OHS container.

4.7 Creating a Kubernetes Secret for the Container Registry

Create a Kubernetes secret that stores the credentials for the container registry where the Oracle HTTP Server (OHS) image is stored.

> **Note**
>
> If you are not using a container registry and have loaded the images on each of the worker nodes, then there is no need to create the registry secret.

Run the following command to create the secret:

```
kubectl create secret docker-registry "regcred" --docker-server=<CONTAINER_REGISTRY> \
--docker-username="<USER_NAME>" \
--docker-password=<PASSWORD> --docker-email=<EMAIL_ID> \
--namespace=<domain_namespace>
```

For example, if using Oracle Container Registry:

```
kubectl create secret docker-registry "regcred" --docker-server=container-registry.oracle.com \
--docker-username="user@example.com" \
```

```
--docker-password=password --docker-email=user@example.com \
--namespace=ohsns
```

Replace <USER_NAME> and <PASSWORD> with the credentials for the registry with the following caveats:

- If using Oracle Container Registry to pull the OHS container image, this is the username and password used to login to Oracle Container Registry. Before using this image you must login to Oracle Container Registry, navigate to **Middleware** > **ohs** and accept the license agreement. For future releases (post January 25) that contain the latest Patch Set Update (PSU) and other fixes released with the Critical Patch Update (CPU) program, you should navigate to **Middleware** >**ohs_cpu**.
- If using your own container registry to store the OHS container image, this is the username and password (or token) for your container registry.

The output will look similar to the following:

```
secret/regcred created
```

4.8 Creating a Kubernetes Secret for the OHS Domain Credentials

Create a Kubernetes secret that stores the credentials for the Oracle HTTP Server (OHS) domain.

Run the following command to create the secret:

```
kubectl create secret generic ohs-secret -n <namespace> --from-literal=username=weblogic --from-
literal=password='<password>'
```

For example:

```
kubectl create secret generic ohs-secret -n ohsns --from-literal=username=weblogic --from-
literal=password='<password>`
```

Replace <password> with a password of your choice.

The output will look similar to the following:

```
secret/ohs-secret created
```

4.9 Preparing the ohs.yaml File

Prepare the ohs.yaml file ready for Oracle HTTP Server (OHS) deployment.

Perform the following steps to prepare the ohs.yaml file:

1. Copy the sample yaml files to $MYOHSFILES:

   ```
   cd $MYOHSFILES
   cp $SCRIPTDIR/*.yaml .
   ```

2. Edit the $MYOHSFILES/ohs.yaml and change the following parameters to match your installation:
   - <NAMESPACE> to your namespace, for example ohsns.
   - <IMAGE_NAME> to the correct image tag on Oracle Container Registry. If you are using your own container registry for the image, you will need to change the image location appropriately. If your own container registry is open, you do not need the imagePullSecrets.
   - During the earlier creation of the ConfigMaps, and secret, if you changed the names from the given examples, then you will need to update the values accordingly.
   - All ConfigMaps are shown for completeness. Remove any ConfigMaps that you are not using, for example if you don't require htdocs then remove the ohs-htdocs ConfigMap. If you are not deploying Oracle WebGate then remove the webgate-config and webgate-wallet ConfigMaps, and so forth.
   - If you have created any additional directories under htdocs, then add the additional entries in that match the ConfigMap and directory names.
   - All ConfigMaps used must mount to the directories stated.
   - Ports can be changed if required.
   - Set DEPLOY_WG to true or false depending on whether Oracle WebGate is to be deployed.
   - If using SSL change <WALLET_NAME> to the wallet directory created under ohsConfig/webgate/config/wallet, for example mywallet.
   - initialDelaySeconds may need to be changed to 10 on slower systems. See, **Issues with Liveness Probe** in Troubleshooting and Common Problems.

An example ohs.yaml is shown below:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: ohs-script-configmap
  namespace: ohsns
data:
  ohs-script.sh: |
    #!/bin/bash
    mkdir -p /u01/oracle/bootdir /u01/oracle/config /u01/oracle/config/moduleconf /u01/oracle/config/webgate/config
    { echo -en "username=" && cat /ohs-config/username && echo -en "\npassword=" && cat /ohs-config/
password; } > /u01/oracle/bootdir/domain.properties
    /u01/oracle/provisionOHS.sh


---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: ohs-domain
  namespace: ohsns
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  selector:
```

```
    matchLabels:
      oracle: ohs
  template:
    metadata:
      labels:
        oracle: ohs
    spec:
      containers:
      - name: ohs
        image: container-registry.oracle.com/middleware/ohs_cpu:14.1.2.0-jdk17-ol8-<YYMMDD>
        env:
          - name: DEPLOY_WG
            value: "true"
        ports:
        - name: clear
          containerPort: 7777
        - name: https
          containerPort: 4443
        resources:
          requests:
            cpu: 1000m
            memory: 1Gi
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop:
            - ALL
          privileged: false
          runAsNonRoot: true
          runAsUser: 1000
        livenessProbe:
          exec:
            command:
            - /bin/bash
            - -c
            - pgrep httpd
          initialDelaySeconds: 10
          periodSeconds: 5
        readinessProbe:
          httpGet:
            port: 7777
            path: /helloWorld.html
        volumeMounts:
          - name: ohs-secret
            mountPath: /ohs-config
          - name: ohs-config
            mountPath: /u01/oracle/config/moduleconf
          - name: ohs-htdocs
            mountPath: /u01/oracle/config/htdocs
          - name: ohs-myapp
            mountPath: /u01/oracle/config/htdocs/myapp
          - name: ohs-httpd
            mountPath: /u01/oracle/config/httpd
          - name: webgate-config
            mountPath: /u01/oracle/config/webgate/config
          - name: webgate-wallet
```

```
      mountPath: /u01/oracle/config/webgate/config/wallet
    - name: ohs-wallet
      mountPath: /u01/oracle/config/wallet/mywallet
    - name: script-volume
      mountPath: /ohs-bin
      readOnly: true
  command: ["/ohs-bin/ohs-script.sh"]
imagePullSecrets:
- name: regcred
affinity:
  podAntiAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
    - weight: 100
      podAffinityTerm:
        labelSelector:
          matchExpressions:
          - key: oracle
            operator: In
            values:
            - ohs
        topologyKey: "kubernetes.io/hostname"
restartPolicy: Always
securityContext:
  seccompProfile:
    type: RuntimeDefault
terminationGracePeriodSeconds: 30
volumes:
- name: ohs-secret
  secret:
    defaultMode: 0444
    secretName: ohs-secret
- name: script-volume
  configMap:
     defaultMode: 0555
     name: ohs-script-configmap
- name: ohs-config
  configMap:
    defaultMode: 0555
    name: ohs-config
- name: ohs-httpd
     configMap:
    defaultMode: 0555
    name: ohs-httpd
- name: ohs-htdocs
  configMap:
    defaultMode: 0555
    name: ohs-htdocs
- name: ohs-myapp
  configMap:
    defaultMode: 0555
    name: ohs-myapp
- name: webgate-config
  configMap:
    defaultMode: 0555
    name: webgate-config
- name: webgate-wallet
```

```
      configMap:
        defaultMode: 0555
        name: webgate-wallet
    - name: ohs-wallet
      configMap:
        defaultMode: 0555
        name: ohs-wallet
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxUnavailable: 1
```

4.10 Preparing the ohs_service.yaml File

Prepare the ohs_service.yaml file for the Oracle HTTP Server (OHS) nodeport.

The OHS nodeport is the entry point for OHS. For example http://ohs.example.com:31777 or https://ohs.example.com:31443

Edit the $MYOHSFILES/ohs_service.yaml and make the following changes:

- <NAMESPACE> to your namespace, for example ohsns.
- If you want your OHS node port to listen on something other that 31777 and 31443, change accordingly.
- If you are using your own httpd.conf file and have changed the port to anything other than 7777, you must change the targetPort and port to match.
- If you are using your own ssl.conf file and have changed the port to anything other than 4443, you must change the targetPort and port to match.

An example ohs_service.yaml is shown below:

```
kind: Service
apiVersion: v1
metadata:
  name: ohs-domain-nodeport
  namespace: ohsns
spec:
  selector:
    oracle: ohs
  type: NodePort
  ports:
  - port: 7777
    name: http
    targetPort: 7777
    nodePort: 31777
    protocol: TCP
  - port: 4443
    name: https
    targetPort: 4443
    nodePort: 31443
    protocol: TCP
```

5

Deploying Oracle HTTP Server on Kubernetes

Deploying Oracle HTTP Server (OHS) on Kubernetes involves deploying the OHS Nodeport followed by the OHS Container.

Before following this section, make sure you have completed all the steps in Preparing Your Environment.

This chapter includes the following topics:

- Deploying the OHS Nodeport
- Deploying the OHS Container
- Validating the OHS Deployment

5.1 Deploying the OHS Nodeport

Create the Kubernetes service nodeport for Oracle HTTP Server (OHS).

To deploy the OHS nodeport:

1. Run the following command:

   ```
   kubectl create -f $MYOHSFILES/ohs_service.yaml
   ```

   The output will look similar to the following:

   ```
   service/ohs-domain-nodeport created
   ```

2. Validate the service has been created by running the following:

   ```
   kubectl get service -n <namespace>
   ```

   For example:

   ```
   kubectl get service -n ohsns
   ```

   The output will look similar to the following:

   ```
   NAME                TYPE      CLUSTER-IP     EXTERNAL-IP   PORT(S)                  AGE
   ohs-domain-nodeport   NodePort   10.98.163.75   <none>        7777:31777/TCP,4443:31443/TCP   22s
   ```

Administrators should be aware of the following:

- As this is a Kubernetes service, the port is accessible on all the worker nodes in the Kubernetes cluster.
- If in the future you create another OHS container on a different port, you will need to create another nodeport service for that OHS.

5.2 Deploying the OHS Container

Create the Oracle HTTP Server (OHS) container on Kubernetes.

1. To deploy the OHS container run the following command:

   ```
   kubectl create -f $MYOHSFILES/ohs.yaml
   ```

   The output will look similar to the following:

   ```
   configmap/ohs-script-configmap created
   deployment.apps/ohs-domain created
   ```

2. Run the following command to view the status of the pods:

   ```
   kubectl get pods -n <namespace> -w
   ```

   For example:

   ```
   kubectl get pods -n ohsns -w
   ```

   Whilst the OHS container is creating you, may see:

   ```
   NAME                    READY   STATUS              RESTARTS   AGE
   ohs-domain-d5b648bc5-vkp4s   0/1     ContainerCreating   0          2m13s
   ```

   To check what is happening while the pod is in `ContainerCreating` status, you can run:

   ```
   kubectl describe pod <podname> -n <namespace>
   ```

   For example:

   ```
   kubectl describe pod ohs-domain-d5b648bc5-vkp4s -n ohsns
   ```

   Once the container is created, it will go to a `READY` status of `0/1` with `STATUS` of `Running`. For example:

   ```
   NAME                    READY   STATUS    RESTARTS   AGE
   ohs-domain-d5b648bc5-vkp4s   1/1     Running   0          3m10s
   ```

   To check what is happening while the pod is in this status, you can run:

   ```
   kubectl logs -f <pod> -n <namespace>
   ```

   For example:

   ```
   kubectl logs -f <pod> -n ohsns
   ```

Once everything is started you should see the OHS is running (READY 1/1):

```
NAME                    READY   STATUS    RESTARTS   AGE
ohs-domain-d5b648bc5-vkp4s   1/1     Running   0          4m10s
```

If there are any failures, see Troubleshooting and Common Problems.

5.3 Validating the OHS Deployment

Validate the Oracle HTTP Server (OHS) container and check you can access OHS using the nodeport.

Validating the OHS Container File System

To validate the OHS container file system:

1. Run the following command to get the name of the OHS container:

   ```
   kubectl get pods -n <namespace>
   ```

   For example:

   ```
   kubectl get pods -n ohsns
   ```

   The output will look similar to the following:

   ```
   NAME                    READY   STATUS    RESTARTS   AGE
   ohs-domain-d5b648bc5-vkp4s   1/1     Running   0          5m34s
   ```

2. Run the following command to create a bash shell inside the container:

   ```
   kubectl exec -n <namespace> -ti <pod> -- /bin/bash
   ```

   For example:

   ```
   kubectl exec -n ohsns -ti ohs-domain-79f8f99575-8qwfh -- /bin/bash
   ```

   This will take you to a bash shell inside the container:

   ```
   [oracle@ohs-domain-75fbd9b597-z77d8 oracle]$
   ```

3. Inside the bash shell navigate to the /u01/oracle/user_projects/domains/ohsDomain/config/fmwconfig/components/OHS/ohs1/ directory:

   ```
   cd /u01/oracle/user_projects/domains/ohsDomain/config/fmwconfig/components/OHS/ohs1/
   ```

   From within this directory, you can navigate around and list (ls) or cat any files you configured using the ConfigMaps.

Validating the OHS Nodeport

Validate the OHS nodeport by accessing the OHS URL's.

In the examples below, ${OHS-HOSTNAME} refers to the hostname.domain of the server where the OHS nodeport was deployed. ${OHS-NODEPORT} refers to the nodePort specified in your ohs-service.yaml, for example 31777 for HTTP, or 31443 for HTTPS.

> **Note**
>
> If OHS is accessed via a loadbalancer, replace ${OHS-HOSTNAME} and ${OHS-NODEPORT} with the loadbalancer hostname.domain and port.

If you have any problems accessing the URL's, refer to Troubleshooting and Common Problems.

1. Launch a browser and access the following:

   > **Note**
   >
   > If you have deployed OHS with Oracle WebGate, then it will depend on your policy setup as to whether the URL's below are accessible or not.

   - The OHS homepage http(s)://${OHS-HOSTNAME}:${OHS-NODEPORT}.
   - Any other files copied in your ohs-htdocs ConfigMap, for example: http(s)://${OHS-HOSTNAME}:${OHS-NODEPORT}/mypage.html.
   - Any files from directories created under htdocs, for example the ohs-myapp ConfigMap: http(s)://${OHS-HOSTNAME}:${OHS-NODEPORT}/myapp.
   - Any URI's defined for mod_wl_ohs in your httpd.conf, ssl.conf or moduleconf/*.conf files, for example: http(s)://${OHS-HOSTNAME}:${OHS-NODEPORT}/console.
   - If Oracle WebGate is deployed, any protected applications, for example: http(s)://${OHS-HOSTNAME}:${OHS-NODEPORT}/myprotectedapp.

Part III

Administering Oracle HTTP Server on Kubernetes

This section includes the following chapters:

6
Scaling OHS Containers

Learn the basic operations to scale OHS containers in Kubernetes.

6.1 Viewing Existing OHS Servers

The default OHS deployment starts one OHS server, assuming `replicas: 1` in the `ohs.yaml`.

To view the running OHS servers, run the following command:

```
kubectl get pods -n <namespace>
```

For example:

```
kubectl get pods -n ohsns
```

The output should look similar to the following:

```
NAME                    READY   STATUS    RESTARTS   AGE
ohs-domain-d5b648bc5-vkp4s   1/1     Running   0          5h17m
```

6.2 Scaling Up OHS Servers

The number of Oracle HTTP Server (OHS) servers running is dependent on the `replicas` parameter configured for OHS. To add additional OHS servers:

1. Run the following kubectl command to start additional OHS servers:

   ```
   kubectl -n <namespace> patch deployment ohs-domain -p '{"spec": {"replicas": <replica count>}}'
   ```

   where `<replica count>` is the number of OHS servers to start.
   In the example below, two additional OHS servers are started, by increasing `replicas` to `3`:

   ```
   kubectl -n ohsns patch deployment  ohs-domain -p '{"spec": {"replicas": 3}}'
   ```

   The output will look similar to the following:

   ```
   deployment.apps/ohs-domain patched
   ```

2. While the new OHS containers are being started, you can run the following command to monitor the progress:

   ```
   kubectl get pods -n <namespace> -w
   ```

   For example:

   ```
   kubectl get pods -n ohsns -w
   ```

   The output will look similar to the following:

   ```
   NAME                        READY   STATUS              RESTARTS   AGE
   ohs-domain-d5b648bc5-2q8bw  0/1     ContainerCreating   0          26s
   ohs-domain-d5b648bc5-qvdjn  0/1     Running             0          26s
   ohs-domain-d5b648bc5-vkp4s  1/1     Running             0          5h21m
   ```

   Two new OHS pods have been created, in this example ohs-domain-d5b648bc5-2q8bw and ohs-domain-d5b648bc5-qvdjn.

3. To check what is happening while the pods are in ContainerCreating status, you can run:

   ```
   kubectl describe pod <podname> -n <namespace>
   ```

4. To check what is happening while the pods are in 0/1 Running status, you can run:

   ```
   kubectl logs -f <pod> -n <namespace>
   ```

5. Once everything is started you should see all the additional OHS containers are running (READY 1/1):

   ```
   NAME                        READY   STATUS    RESTARTS     AGE
   ohs-domain-d5b648bc5-2q8bw  1/1     Running   0            9m34s
   ohs-domain-d5b648bc5-qvdjn  1/1     Running   0            9m34s
   ohs-domain-d5b648bc5-vkp4s  1/1     Running   0            5h30m
   ```

6.3 Scaling Down OHS Servers

As previously referenced, the number of Oracle HTTP Server (OHS) servers running is dependent on the replicas parameter configured for OHS. To stop one or more OHS servers:

1. Run the following kubectl command to scale down OHS servers:

   ```
   kubectl -n <namespace> patch deployment ohs-domain -p '{"spec": {"replicas": <replica count>}}'
   ```

   where <replica count> is the number of OHS servers to run.
   In the example below, replicas is reduced to 1, so only one OHS is running:

   ```
   kubectl -n ohsns patch deployment  ohs-domain -p '{"spec": {"replicas": 1}}'
   ```

The output will look similar to the following:

```
deployment.apps/ohs-domain patched
```

2. Run the following kubectl command to view the pods:

```
kubectl get pods -n <namespace>
```

For example:

```
kubectl get pods -n ohsns
```

The output will look similar to the following:

```
NAME                    READY   STATUS        RESTARTS   AGE
ohs-domain-d5b648bc5-2q8bw   0/1     Terminating   0          12m
ohs-domain-d5b648bc5-qvdjn   0/1     Terminating   0          12m
ohs-domain-d5b648bc5-vkp4s   1/1     Running       0          5h31m
```

Two pods now have a STATUS of Terminating. Keep executing the command until the pods have disappeared and you are left with the one OHS pod:

```
NAME                    READY   STATUS    RESTARTS   AGE
ohs-domain-d5b648bc5-vkp4s   1/1     Running   0          5h32m
```

7

Modifying the OHS Container

Learn how to modify the Oracle HTTP Server (OHS) configuration after the OHS container is deployed.

Modifying the deployed OHS container configuration can be achieved in one of the following ways:

- Edting Files in $MYOHFILES/ohsconfig
- Editing the ConfigMap

7.1 Edting Files in $MYOHFILES/ohsconfig

To edit the configuration files in $MYOHFILES/ohsconfig:

1. Edit the required files in the $MYOHSFILES/ohsConfig subdirectories.
2. Delete the ConfigMaps for any files you have changed. For example if you have changed httpd.conf and files in moduleconf, run:

   ```
   kubectl delete cm ohs-httpd -n ohsns
   kubectl delete cm ohs-config -n ohsns
   ```

3. Recreate the required ConfigMaps:

   ```
   cd $MYOHSFILES
   kubectl create cm -n ohsns ohs-httpd --from-file=ohsConfig/httpconf
   kubectl create cm -n ohsns ohs-config --from-file=ohsConfig/moduleconf
   ```

4. Find the name of the existing OHS pod:

   ```
   kubectl get pods -n <namespace>
   ```

   For example:

   ```
   kubectl get pods -n ohsns
   ```

   The output will look similar to the following:

   ```
   NAME                    READY   STATUS    RESTARTS   AGE
   ohs-domain-d5b648bc5-vkp4s   1/1     Running   0          55s
   ```

5. Delete the pod using the following command:

   ```
   kubectl delete pod <pod> -n <namespace>
   ```

For example:

```
kubectl delete pod ohs-domain-d5b648bc5-vkp4s -n ohsns
```

The output will look similar to the following:

```
pod "ohs-domain-d5b648bc5-vkp4s" deleted
```

6. Run the following command to make sure the pod has restarted:

```
kubectl get pods -n ohsns
```

The output will look similar to the following:

```
NAME                    READY   STATUS    RESTARTS   AGE
ohs-domain-d5b648bc5-gdvnp   1/1     Running   0          39s
```

7.2 Editing the ConfigMap

To edit the ConfigMap:

1. Run the following command to edit the OHS configuration:

```
kubectl edit configmap <configmap> -n <namespace>
```

Where <configmap> is either ohs-httpd or ohs-config to modify the httpd.conf and moduleconf files respectively.
For example:

```
kubectl edit configmap ohs-httpd -n ohsns
```

> **Note**
>
> This opens an edit session for the ConfigMap where parameters can be changed using standard vi commands.

2. In the edit session, edit the required parameters accordingly. Save the file and exit (:wq!).
3. Find the name of the existing OHS pod:

```
kubectl get pods -n <namespace>
```

The output will look similar to the following:

```
NAME                    READY   STATUS    RESTARTS   AGE
ohs-domain-d5b648bc5-vkp4s   1/1     Running   0          2h33s
```

4. Delete the pod using the following command:

```
kubectl delete pod <pod> -n <namespace>
```

For example:

```
kubectl delete pod ohs-domain-d5b648bc5-vkp4s -n ohsns
```

The output will look similar to the following:

```
pod "ohs-domain-d5b648bc5-vkp4s" deleted
```

5. Run the following command to make sure the pod has restarted:

```
kubectl get pods -n ohsns -w
```

The output will look similar to the following:

```
NAME                    READY   STATUS    RESTARTS   AGE
ohs-domain-d5b648bc5-gdvnp   1/1     Running   0          39s
```

8

Deleting the OHS Container

Learn how to delete the Oracle HTTP Server (OHS) container.

The following commands show how to remove the OHS container, OHS nodeport service, ConfigMaps, secrets, and namespace:

1. Run the following command to delete the OHS nodeport service:

   ```
   kubectl delete -f $MYOHSFILES/ohs_service.yaml
   ```

2. Run the following command to delete the OHS container:

   ```
   kubectl delete -f $MYOHSFILES/ohs.yaml
   ```

3. Run the following commands to delete any configmaps you have created, for example:

   ```
   kubectl delete cm -n ohsns ohs-config
   kubectl delete cm -n ohsns ohs-httpd
   kubectl delete cm -n ohsns ohs-htdocs
   kubectl delete cm -n ohsns ohs-myapp
   kubectl delete cm -n ohsns webgate-config
   kubectl delete cm -n ohsns webgate-wallet
   kubectl delete cm -n ohsns ohs-wallet
   ```

4. Run the following command to delete the secrets:

   ```
   kubectl delete secret regcred -n ohsns
   kubectl delete secret ohs-secret -n ohsns
   ```

5. Run the following command to delete the namespace:

   ```
   kubectl delete namespace ohsns
   ```

9

Creating or Updating an OHS Image

As described in Obtaining the OHS Container Image you can create your own OHS container image if required. Similarly, if you have access to My Oracle Support (MOS), and there is a need to build a new image with an interim or one off patch, you can update an existing image accordingly.

Regardless of whether you are creating your own image, or updating an existing image, it is recommended to use the WebLogic Image Tool to build an OHS image for production deployments.

This chapter includes the following topics:

- Setting Up the WebLogic Image Tool
- Creating an Image
- Updating an Image

9.1 Setting Up the WebLogic Image Tool

Using the WebLogic Image Tool, you can create a new Oracle HTTP Server (OHS) image with Patch Set Updates (PSU's) and interim patches, or update an existing image with one or more interim patches.

Administrators should be aware of the following recommendations:

- Use Creating an Image for creating a new OHS image containing the OHS binaries, bundle patch and interim patches. This is the recommended approach if you have access to the OHS patches because it optimizes the size of the image.
- Use Updating an Image for patching an existing OHS image with a single interim patch. Please note that the patched image size may increase considerably due to additional image layers introduced by the patch application tool.

The following sections explain how to set up the WebLogic Image Tool:

- WebLogic Image Tool Prerequisites
- Configure the WebLogic Image Tool
- Validating the Setup
- Setting the Build and Cache Directories

9.1.1 WebLogic Image Tool Prerequisites

Verify that the environment where you will build the image meets the following prerequisites:

- Docker client and daemon is installed, with minimum Docker version 18.03.1.ce.
- Bash version 4.0 or later, to enable the command complete feature.
- The JAVA_HOME environment variable set to the supported JDK location.

> **Note**
>
> JDK 8 or higher is supported for WebLogic Image Tool.

For example:

```
export JAVA_HOME=/scratch/export/oracle/product/jdk
```

9.1.2 Configure the WebLogic Image Tool

To set up the WebLogic Image Tool:

1. Create a working directory and navigate to it:

   ```
   mkdir <workdir>
   cd <workdir>
   ```

   For example:

   ```
   mkdir /scratch/imagetool-setup
   cd /scratch/imagetool-setup
   ```

2. Download the latest version of the WebLogic Image Tool from the Releases page:

   ```
   wget https://github.com/oracle/weblogic-image-tool/releases/download/release-X.X.X/imagetool.zip
   ```

   where X.X.X is the latest release referenced on the Releases page.

   > **Note**
   >
   > You must use WebLogic Image Tool 1.14.3 or later.

3. Unzip the release ZIP file in the imagetool-setup directory:

   ```
   unzip imagetool.zip
   ```

4. Execute the following commands to set up the WebLogic Image Tool:

   ```
   cd <workdir>/imagetool-setup/imagetool/bin
   source setup.sh
   ```

   For example:

   ```
   cd /scratch/imagetool-setup/imagetool/bin
   source setup.sh
   ```

9.1.3 Validating the Setup

To validate the setup of the WebLogic Image Tool:

1. Enter the following command to retrieve the version of the WebLogic Image Tool:

   ```
   imagetool --version
   ```

2. Enter `imagetool` then press the Tab key to display the available `imagetool` commands:

   ```
   imagetool <TAB>
   ```

   The output will look similar to the following:

   ```
   cache   create  help    rebase  update
   ```

9.1.4 Setting the Build and Cache Directories

Optionally create the WebLogic Image Tool build and cache directories.

WebLogic Image Tool Build Directory

The WebLogic Image Tool creates a temporary Docker context directory, prefixed by `wlsimgbuilder_temp`, every time the tool runs. Under normal circumstances, this context directory will be deleted. However, if the process is aborted or the tool is unable to remove the directory, it is safe for you to delete it manually. By default, the WebLogic Image Tool creates the Docker context directory under the user's home directory. If you prefer to use a different directory for the temporary context, set the environment variable `WLSIMG_BLDDIR`:

```
export WLSIMG_BLDDIR="/path/to/build/dir"
```

WebLogic Image Tool Cache Directory

The WebLogic Image Tool maintains a local file cache store. This store is used to look up where the OHS and JDK installers, and OHS patches reside in the local file system. By default, the cache store is located in the user's `$HOME/cache` directory. Under this directory, the lookup information is stored in the `.metadata` file. All automatically downloaded patches also reside in this directory. You can change the default cache store location by setting the environment variable `WLSIMG_CACHEDIR`:

```
export WLSIMG_CACHEDIR="/path/to/cachedir"
```

9.1.5 Setting Up Additional Build Scripts

Creating an Oracle HTTP Server (OHS) container image using the WebLogic Image Tool requires additional container scripts for OHS domains.

1. Clone the docker-images repository to set up the required scripts:

   ```
   cd <workdir>/imagetool-setup
   git clone https://github.com/oracle/docker-images.git
   ```

   For example:

   ```
   cd /scratch/imagetool-setup
   $ git clone https://github.com/oracle/docker-images.git
   ```

9.2 Creating an Image

Before creating an Oracle HTTP Server (OHS) image, make sure you have followed Setting Up the WebLogic Image Tool.

This sections includes the following topics:

- Exporting the PWD Variable
- Downloading the OHS Installation Binaries and Patches
- Updating the Required Build Files
- Creating the Image

9.2.1 Exporting the PWD Variable

In order for the WebLogic Image Tool to build OHS with all the latest patches, the image creation downloads patches from My Oracle Support.

During the image build you are asked to enter your My Oracle Support credentials, however the password is passed as a variable. Set the variable as follows:

```
export MYPWD="MY_ORACLE_SUPPORT_PWD"
```

9.2.2 Downloading the OHS Installation Binaries and Patches

You must download the required Oracle HTTP Server (OHS) installation binaries and JDK as listed below from Oracle Software Delivery Cloud and My Oracle Support. Save them to a directory of your choice.

The installation binaries and JDK required are:

- Oracle Web Tier 14.1.2.0.0
  - V1045136-01.zip

> **Note**
>
> You will need to unzip the file after downloading to get the `fmw_14.1.2.0.0_ohs_linux64.bin` which is used with the imageTool.

- Oracle JDK v17 or v21
  - jdk-17.X.X_linux-x64.tar.gz or jdk-21.X.X_linux-x64.tar.gz

> **Note**
>
> 17.0.14 or higher, or 21.0.6 or higher are supported.

9.2.3 Updating the Required Build Files

The following files are used for creating the image. These files must be updated before creating the image:

- additionalBuildCmds.txt
- buildArgs

1. Create the <workdir>/imagetool-setup/docker-images/OracleHTTPServer/additionalBuildCmds.txt file and add the following:

```
[package-manager-packages]
binutils make glibc-devel procps
[final-build-commands]
ENV PATH=$PATH:/u01/oracle/ohssa/oracle_common/common/bin \
   NM_PORT=5556 \
   OHS_LISTEN_PORT=7777 \
   OHS_SSL_PORT=4443 \
   MW_HOME=/u01/oracle/ohssa \
   DOMAIN_NAME=ohsDomain \
   OHS_COMPONENT_NAME=ohs1 \
   PATH=$PATH:$ORACLE_HOME/oracle_common/common/bin:$ORACLE_HOME/user_projects/domains/
ohsDomain/bin:/u01/oracle/ \
   WLST_HOME=/u01/oracle/ohssa/oracle_common/common/bin
COPY --chown=oracle:root files/create-sa-ohs-domain.py files/configureWLSProxyPlugin.sh files/
mod_wl_ohs.conf.sample files/provisionOHS.sh files/start-ohs.py files/stop-ohs.py files/helloWorld.html /u01/
oracle/
WORKDIR ${ORACLE_HOME}
CMD ["/u01/oracle/provisionOHS.sh"]
```

> **Note**
>
> Administrators should be aware of the following:
>
> - oracle:root is used for OpenShift which has more stringent policies. Users who do not want those permissions can change to the permissions they require.
> - The packages listed in the [package-manager-packages] is for Oracle Linux 8 images. If you want to build Oracle Linux 9 images you need to add libxcrypt-compat to this list also.

2. Create the <workdir>/imagetool-setup/docker-images/OracleHTTPServer/buildArgs file as follows and change the following:
   - <workdir> to your working directory, for example /scratch/
   - %BUILDTAG% to the tag you want create for the image, for example oracle/ohs:14.1.2.0.0
   - %JDK_VERSION% to the version of your JDK, for example 21.0.6
   - <user> to your My Oracle Support username

```
create
--tag=%BUILDTAG%
--additionalBuildCommands /<workdir>/imagetool-setup/docker-images/OracleHTTPServer/
additionalBuildCmds.txt
--additionalBuildFiles <workdir>/imagetool-setup/docker-images/OracleHTTPServer/dockerfiles/14.1.2.0.0/
container-scripts/create-sa-ohs-domain.py,<workdir>/imagetool-setup/docker-images/OracleHTTPServer/
dockerfiles/14.1.2.0.0/container-scripts/provisionOHS.sh,<workdir>/imagetool-setup/docker-images/
OracleHTTPServer/dockerfiles/14.1.2.0.0/container-scripts/configureWLSProxyPlugin.sh,<workdir>/imagetool-
setup/docker-images/OracleHTTPServer/dockerfiles/14.1.2.0.0/container-scripts/
```

```
mod_wl_ohs.conf.sample,<workdir>/imagetool-setup/docker-images/OracleHTTPServer/dockerfiles/14.1.2.0.0/
container-scripts/start-ohs.py,<workdir>/imagetool-setup/docker-images/OracleHTTPServer/dockerfiles/
14.1.2.0.0/container-scripts/stop-ohs.py,<workdir>/imagetool-setup/docker-images/OracleHTTPServer/
dockerfiles/14.1.2.0.0/container-scripts/helloWorld.html
--type=OHS
--pull
--recommendedPatches
--chown=oracle:root
--user=<user>
--passwordEnv=MYPWD
--version=14.1.2.0.0
--jdkVersion=<version>
```

For example:

```
create
--tag=oracle/ohs:14.1.2.0.0
--additionalBuildCommands /scratch/imagetool-setup/docker-images/OracleHTTPServer/
additionalBuildCmds.txt
--additionalBuildFiles /scratch/imagetool-setup/docker-images/OracleHTTPServer/dockerfiles/14.1.2.0.0/
container-scripts/create-sa-ohs-domain.py,/scratch/imagetool-setup/docker-images/OracleHTTPServer/
dockerfiles/14.1.2.0.0/container-scripts/provisionOHS.sh,/scratch/imagetool-setup/docker-images/
OracleHTTPServer/dockerfiles/14.1.2.0.0/container-scripts/configureWLSProxyPlugin.sh,/scratch/imagetool-
setup/docker-images/OracleHTTPServer/dockerfiles/14.1.2.0.0/container-scripts/mod_wl_ohs.conf.sample,/
scratch/imagetool-setup/docker-images/OracleHTTPServer/dockerfiles/14.1.2.0.0/container-scripts/start-ohs.py,/
scratch/imagetool-setup/docker-images/OracleHTTPServer/dockerfiles/14.1.2.0.0/container-scripts/stop-ohs.py,/
scratch/imagetool-setup/docker-images/OracleHTTPServer/dockerfiles/14.1.2.0.0/container-scripts/
helloWorld.html
--type=OHS
--pull
--recommendedPatches
--chown=oracle:root
--user=user@example.com
--passwordEnv=MYPWD
--version=14.1.2.0.0
--jdkVersion=21.0.6
```

> **Note**
>
> jdkVersion can also be 17.0.14 or higher.

For more information on the complete list of options available with the WebLogic Image Tool create command. see Create Image.

9.2.4 Creating the Image

To create the OHS image, run the following commands:

1. Add the JDK package to the WebLogic Image Tool cache. For example:

```
imagetool cache addInstaller --type jdk --version 21.0.6 --path <download location>/jdk-21_linux-x64.tar.gz
```

2. Add the downloaded installation binaries to the WebLogic Image Tool cache. For example:

```
imagetool cache addInstaller --type ohs --version 14.1.2.0.0 --path <download location>/
fmw_14.1.2.0.0_ohs_linux64.bin
```

3. Create the Oracle HTTP Server image:

```
imagetool @<absolute path to buildargs file>
```

> **Note**
>
> By default ImageTool builds an image with Oracle Linux 8. If you want to build an image with Oracle Linux 9. you must append `--fromImage ghcr.io/oracle/oraclelinux:9-slim` to the image tool command.

For example for Oracle Linux 8:

```
imagetool @/scratch/imagetool-setup/docker-images/OracleHTTPServer/buildArgs
```

For Oracle Linux 9:

```
imagetool @/scratch/imagetool-setup/docker-images/OracleHTTPServer/buildArgs --fromImage ghcr.io/oracle/
oraclelinux:9-slim
```

4. After the image has been created, check the created image using the `docker images` command:

```
docker images | grep ohs
```

The output will look similar to the following:

```
oracle/ohs:14.1.2.0.0          14.1.2.0.0          ad732fc7c16b        About a minute ago   3.68GB
```

5. If you want to see what patches were installed, you can run:

```
imagetool inspect --image=<REPOSITORY>:<TAG> --patches
```

For example:

```
imagetool inspect --image=oracle/ohs:14.1.2.0.0 --patches
```

6. Run the following command to save the container image to a tar file:

```
docker save -o <path>/<file>.tar <image>
```

For example:

```
docker save -o $WORKDIR/ohs14.1.2.tar oracle/ohs:14.1.2.0.0
```

9.3 Updating an Image

Before updating an Oracle HTTP Server (OHS) image, make sure you have followed Setting Up the WebLogic Image Tool.

> **Note**
>
> The container image to be patched must be loaded in the local docker images repository before attempting these steps. In the examples below the image oracle/ohs:14.1.2.0.0 is updated with an interim patch:
>
> ```
> $ docker images
>
> REPOSITORY              TAG         IMAGE ID       CREATED         SIZE
> oracle/ohs:14.1.2.0.0   14.1.2.0.0  b051804ba15f   3 months ago    3.68GB
> ```

The steps below show how to update an existing Oracle HTTP Server image with an interim patch:

1. Download the required interim patch(es) and latest Opatch (28186730) from My Oracle Support. and save them in a directory of your choice.
2. Add the OPatch patch to the WebLogic Image Tool cache, for example:

   ```
   imagetool cache addEntry --key 28186730_13.9.4.2.18 --value <downloaded-patches-location>/
   p28186730_1394218_Generic.zip
   ```

3. Execute the imagetool cache addEntry command for each patch to add the required patch(es) to the WebLogic Image Tool cache. For example, to add patch p6666666_141200_Generic.zip:

   > **Note**
   >
   > This is not a real patch number, it is used purely for an example.

   ```
   imagetool cache addEntry --key=6666666_14.1.2.141200 --value <downloaded-patches-location>/
   p6666666_141200_Generic.zip
   ```

4. Provide the following arguments to the WebLogic Image Tool update command:
   - --fromImage - Identify the image that needs to be updated. In the example below, the image to be updated is oracle/ohs:14.1.2.0.0.
   - --patches - Multiple patches can be specified as a comma-separated list.
   - --tag - Specify the new tag to be applied for the image being built.

   For more information on the complete list of options available with the WebLogic Image Tool update command, see Update Image.

> **Note**
>
> The WebLogic Image Tool cache should have the latest OPatch zip. The WebLogic Image Tool will update the OPatch if it is not already updated in the image.

For example:

```
imagetool update --fromImage oracle/ohs:14.1.2.0.0  --tag=oracle/ohs-new:14.1.2.0.0 --
patches=6666666_14.1.2.141200 --opatchBugNumber=28186730_13.9.4.2.18
```

> **Note**
>
> If the command fails because the files in the image being upgraded are not owned by oracle:root, then add the parameter --chown <userid>:<groupid> to correspond with the values returned in the error.

5. Check the built image using the docker images command:

```
docker images | grep OHS
```

The output will look similar to the following:

```
REPOSITORY                    TAG         IMAGE ID       CREATED          SIZE
oracle/ohs-new:14.1.2.0.0     14.1.2.0.0  78ccd1ad67eb   5 minutes ago    4.5GB
oracle/ohs:14.1.2.0.0         14.1.2.0.0  b051804ba15f   3 months ago     3.68GB
```

6. Run the following command to save the patched container image to a tar file:

```
docker save -o <path>/<file>.tar <image>
```

For example:

```
docker save -o $WORKDIR/ohs-new14.1.2.tar oracle/ohs-new:14.1.2
```

10

Patching and Upgrading

This chapter includes the following topics:

10.1 Patching and Upgrading an Image in 14.1.2

Learn how to patch or upgrade the Oracle HTTP Server (OHS) image used by an OHS 14.1.2 container.

The instructions in this section relate to patching or upgrading an existing 14.1.2 OHS container with a new container image.

1. To show the version of the image the OHS container is currently running, run the following command:

   ```
   kubectl describe pod <pod> -n <namespace> | grep Image
   ```

   For example:

   ```
   kubectl describe pod ohs-domain-d5b648bc5-qsgts -n ohsns | grep Image
   ```

   The output will look similar to the following:

   ```
   Image:         container-registry.oracle.com/middleware/ohs_cpu:14.1.2.0-jdk17-ol8-<version>
   Image ID:      9a7199ac903114793d6ad1f320010c3dbd59a39ad9bc987d926d3422a68603e7
   ```

2. If using an image from Oracle Container Registry, you must login to Oracle Container Registry, navigate to **Middleware** > **ohs_cpu** and accept the license agreement.
3. Run the following command to update the container with the new image:

   ```
   kubectl set image deployment/ohs-domain -n <namespace> ohs=<new_image>
   ```

   For example:

   ```
   kubectl set image deployment/ohs-domain -n ohsns ohs=container-registry.oracle.com/middleware/
   ohs_cpu:14.1.2.0-jdk17-ol8-<new>
   ```

   The output will look similar to the following:

   ```
   deployment.apps/ohs-domain image updated
   ```

> **Note**
>
> This command will perform a rolling restart of the OHS container by shutting down the existing OHS container and starting a new one.

4. Run the following kubectl command to view the pods:

   ```
   kubectl get pods -n <domain_namespace>
   ```

   For example:

   ```
   kubectl get pods -n ohsns
   ```

   The output will look similar to the following:

   ```
   NAME                    READY   STATUS              RESTARTS   AGE
   ohs-domain-5c9c9879d-kpt9j   0/1     ContainerCreating   0          8s
   ohs-domain-d5b648bc5-qsgts   1/1     Terminating         0          17h
   ```

   The existing OHS pod will move to a STATUS of Terminating and a new OHS pod will be started.
   To check what is happening while the pods are in ContainerCreating status, you can run:

   ```
   kubectl describe pod <podname> -n <namespace>
   ```

   To check what is happening while the pods are in 0/1 Running status, you can run:

   ```
   kubectl logs -f <pod> -n <namespace>
   ```

   Keep running the kubectl get pods -n <namespace> command until the pod is Running and at READY 1\1:

   ```
   NAME                    READY   STATUS    RESTARTS   AGE
   ohs-domain-5c9c9879d-kpt9j   1/1     Running   0          6m40s
   ```

5. To show the OHS container is running the new image, run the following command:

   ```
   kubectl describe pod <pod> -n <namespace> | grep Image
   ```

   For example:

   ```
   kubectl describe pod ohs-domain-5c9c9879d-kpt9j -n ohsns | grep Image
   ```

   The output will look similar to the following:

   ```
   Image:         container-registry.oracle.com/middleware/ohs_cpu:14.1.2.0-jdk17-ol8-<new>
   Image ID:      118c5c3713ddd6804cb699ecd0c7bd4a26ebf7e1427c5351c63244b5eb74ca94
   ```

10.2 Upgrading from OHS 12.2.1.4 to OHS 14.1.2

Learn how to upgrade from Oracle HTTP Server (OHS) 12.2.1.4 to OHS 14.1.2.

The instructions in this section relate to upgrading an existing 12.2.1.4 OHS deployment to OHS 14.1.2.

1. Take a backup of your existing OHS configuration files in $MYOHSFILES.
2. Read What's New in this Release and look at the **New Features** and **Deprecated Features** in 14.1.2. Make any required changes to your OHS configuration files in $MYOHSFILES/ohsConfig.
3. Delete any ConfigMaps that relate to any files you have changed. For example if you have changed httpd.conf and files in moduleconf, run:

   ```
   kubectl delete cm ohs-httpd -n ohsns
   kubectl delete cm ohs-config -n ohsns
   ```

4. Recreate the required ConfigMaps:

   ```
   cd $MYOHSFILES
   kubectl create cm -n ohsns ohs-httpd --from-file=ohsConfig/httpconf
   kubectl create cm -n ohsns ohs-config --from-file=ohsConfig/moduleconf
   ```

5. Edit the $MYOHSFILES/ohs.yaml and change the <IMAGE_NAME> to the correct 14.1.2 image tag on Oracle Container Registry. If you are using your own container registry for the image, you will need to change the image location appropriately.

   > **Note**
   >
   > Before using this image you must login to Oracle Container Registry, navigate to **Middleware** > **ohs** and accept the license agreement. For future releases (post January 25) that contain the latest Patch Set Update (PSU) and other fixes released with the Critical Patch Update (CPU) program, you should navigate to **Middleware** >**ohs_cpu**.

6. Find the name of the existing OHS pod:

   ```
   kubectl get pods -n <namespace>
   ```

   For example:

   ```
   kubectl get pods -n ohsns
   ```

   The output will look similar to the following:

   ```
   NAME                    READY   STATUS    RESTARTS   AGE
   ohs-domain-d5b648bc5-vkp4s   1/1     Running   0          55s
   ```

7. Delete the pod using the following command:

```
kubectl delete pod <pod> -n <namespace>
```

For example:

```
kubectl delete pod ohs-domain-d5b648bc5-vkp4s -n ohsns
```

The output will look similar to the following:

```
pod "ohs-domain-d5b648bc5-vkp4s" deleted
```

8. Run the following command to make sure the pod has restarted:

```
kubectl get pods -n ohsns
```

The output will look similar to the following:

```
NAME                        READY   STATUS    RESTARTS   AGE
ohs-domain-d5b648bc5-gdvnp  1/1     Running   0          39s
```

11

Troubleshooting and Common Problems

The information in this section relates to problems creating Oracle HTTP Server (OHS) containers, and how to view log files.

OHS Container in CreatingContainer Status

During OHS container creation you may see:

```
NAME                    READY   STATUS              RESTARTS   AGE
ohs-domain-d5b648bc5-vkp4s   0/1     ContainerCreating   0          2m13s
```

To check what is happening while the pod is in ContainerCreating status, you can run:

```
kubectl describe pod <podname> -n <namespace>
```

For example:

```
kubectl describe pod ohs-domain-d5b648bc5-vkp4s -n ohsns
```

The details of the above command can help identify possible problems.

In the Events, if you see Pulling image <image>, this means that the container is pulling the image from the container-registry. Depending on the speed of the network this could take 5-10 minutes. Once the image is pulled you should see the pod go to RUNNING 0/1 status, before eventually going to RUNNING 1\1.

OHS Container in ImagePullBackOff

If you see the following:

```
kubectl get pods -n ohsns
NAME                    READY   STATUS              RESTARTS   AGE
ohs-domain-58b8dc4749-hzlc9   0/1     ImagePullBackOff   0          16s
```

This could be because you have put the wrong image location in the ohs.yaml, there is a problem with the image itself, or the secrets created are incorrect.

Once the problem is identified and resolved, you can delete the container and try again:

```
cd $MYOHSFILES
kubectl delete -f ohs.yaml
kubectl create -f ohs.yaml
```

OHS Container in 0/1 Running Status

During OHS container creation you may see:

```
NAME                    READY   STATUS          RESTARTS   AGE
ohs-domain-d5b648bc5-vkp4s   0/1     Running         0          2m13s
```

This is normal behavior during any startup, however the pod should eventually go to RUNNING 1/1.

Whilst the pod is in 0/1 status, you can check what is happening by running:

```
kubectl logs -f <pod> -n <namespace>
```

For example:

```
kubectl logs -f ohs-domain-d5b648bc5-vkp4s -n ohsns
```

If there are any problems or errors during startup, they will be logged here.

You can also describe the pod to determine potential problems:

```
kubectl describe pod <pod> -n <namespace>
```

For example:

```
kubectl describe pod ohs-domain-d5b648bc5-vkp4s -n ohsns
```

Additionally, you can view the OHS log files inside the container. See, Troubleshooting and Common Problems, **Viewing OHS log files** later in this chapter.

Depending on the error, you may need to fix the files in the $MYOHSFILES/ohsConfig directories.

Once you have fixed your configuration files, you will need to delete the appropriate configmap(s) and recreate. For example if the problem was in httpd.conf, ssl.conf, or mod_wl_ohs.conf:

```
cd $MYOHSFILES
kubectl delete -f ohs.yaml
kubectl delete cm ohs-httpd -n ohsns
kubectl create cm -n ohsns ohs-httpd --from-file=ohsConfig/httpconf
kubectl create -f ohs.yaml
```

Issues with LivenessProbe

If you see OHS Container in 0/1 Running Status and the container constantly restarts:

```
NAME                    READY   STATUS          RESTARTS   AGE
ohs-domain-d5b648bc5-vkp4s   0/1     Running         4          2m13s
```

then, if this occurs and kubectl logs -f <pod> -n <namespace> is showing no errors, then run:

```
kubectl describe pod <podname> -n <namespace>
```

If the output shows:

```
----    ------     ----            ----            -------
  Normal   Scheduled  63s              default-scheduler  Successfully assigned ohsns/ohs-domain-857c5d97d5-8nnx9
to doc-worker1
  Normal   Pulled     17s (x2 over 62s)  kubelet          Container image "<image>" already present on machine
  Normal   Created    17s (x2 over 62s)  kubelet          Created container ohs
  Normal   Started    17s (x2 over 62s)  kubelet          Started container ohs
  Warning  Unhealthy  2s (x9 over 61s)   kubelet          Readiness probe failed: Get "http://10.244.1.150:7777/
helloWorld.html": dial tcp 10.244.1.150:7777: connect: connection refused
  Warning  Unhealthy  2s (x6 over 57s)   kubelet          Liveness probe failed:
  Normal   Killing    2s (x2 over 47s)   kubelet          Container ohs failed liveness probe, will be restarted
```

then it's possible the liveness probe is killing and restarting the container because the httpd process has not started before the liveness probe checks. This can happen on slow systems.

If this occurs delete the container:

```
cd $MYOHSFILES
kubectl delete -f ohs.yaml
```

and edit the ohs.yaml file and increase the initialDelaySeconds from 10 to 30:

```
    livenessProbe:
      exec:
        command:
        - /bin/bash
        - -c
        - pgrep httpd
      initialDelaySeconds: 30
      periodSeconds: 5
```

Then try creating the container again:

```
cd $MYOHSFILES
kubectl create -f ohs.yaml
```

Viewing OHS Log Files

To view OHS log files inside the container, run the following commands:

```
kubectl exec -n <namespace> -ti <pod> -- /bin/bash
```

For example:

```
kubectl exec -n ohsns -ti ohs-domain-79f8f99575-8qwfh -- /bin/bash
```

This will take you to a bash shell inside the container:

```
[oracle@ohs-domain-75fbd9b597-z77d8 oracle]$
```

Inside the bash shell navigate to the /u01/oracle/user_projects/domains/ohsDomain/server/ohs1/logs directory:

```
cd  /u01/oracle/user_projects/domains/ohsDomain/server/ohs1/logs
```

From within this directory, you can cat the OHS log files to help diagnose problems.

Glossary

Index