Oracle® FMW

Deploying and Managing Oracle Access Management on Kubernetes

G21958-05
March 2026

Oracle FMW Deploying and Managing Oracle Access Management on Kubernetes,

G21958-05

Primary Author: Russell Hodgson

Contents

List of Figures

1

What's New in This Release?

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

Table 1-1 Release Notes for Oracle Access Management 14c on Kubernetes

Date	Version	Change
March 2026	14.1.2.1.0 GitHub release version 25.3.1	Supports Oracle Access Management 14.1.2.1.0 domain deployment using the April 2026 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 OAM.
July 2025	14.1.2.1.0 GitHub release version 25.3.1	Supports Oracle Access Management 14.1.2.1.0 domain deployment using the July 2025 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 OAM.
April 2025	14.1.2.1.0 GitHub release version 25.2.1	Supports Oracle Access Management 14.1.2.1.0 domain deployment using the April 2025 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 OAM.

Table 1-1 (Cont.) Release Notes for Oracle Access Management 14c on Kubernetes

Date	Version	Change
March 2025	14.1.2.1.0 GitHub release version 25.1.3	Initial release of Oracle Access Management 14.1.2.1.0 on Kubernetes. Supports Oracle Access Management 14.1.2.1.0 deployment using the OAM container image and WebLogic Kubernetes Operator 4.2.10. The GitHub release version is the latest version of the deployment scripts used in Setting Up the Code Repository for OAM.

Part I

Introduction to Oracle Access Management on Kubernetes

Oracle Access Management (OAM) can be deployed on Kubernetes.

This section includes the following chapters:

2

Introducing Oracle Access Management on Kubernetes

Oracle Access Management (OAM) is supported for deployment on Kubernetes.

This chapter includes the following topics:

2.1 Overview of Oracle Access Management on Kubernetes

Oracle Access Management provides an enterprise-level security platform, delivers risk-aware end-to-end user authentication, single sign-on, and authorization protection. Oracle Access Management enables enterprises to secure access and seamlessly integrate social identities with applications.

Oracle Access Management can be deployed using modern container orchestration with Kubernetes, bringing enhanced agility and scalability to IT environments.

2.2 Key Features of Oracle Access Management on Kubernetes

The key features of using Oracle Access Management (OAM) on Kubernetes are:

- **Simplified Deployment and DevOps**: Containers allow teams to automate deployments and streamline application lifecycle management, reducing manual effort, cost, and time to deploy.
- **Portability**: Containerized OAM can run seamlessly across different environments, including on-premises data centers, public clouds, and hybrid setups
- **Scalability**: Containers allow organizations to scale their security components dynamically, ensuring that they can handle fluctuating workloads
- **Improved Resource Efficiency**: Containers provide lightweight, efficient runtime environments that optimize resource utilization compared to traditional virtual machines.

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 resource.
- **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

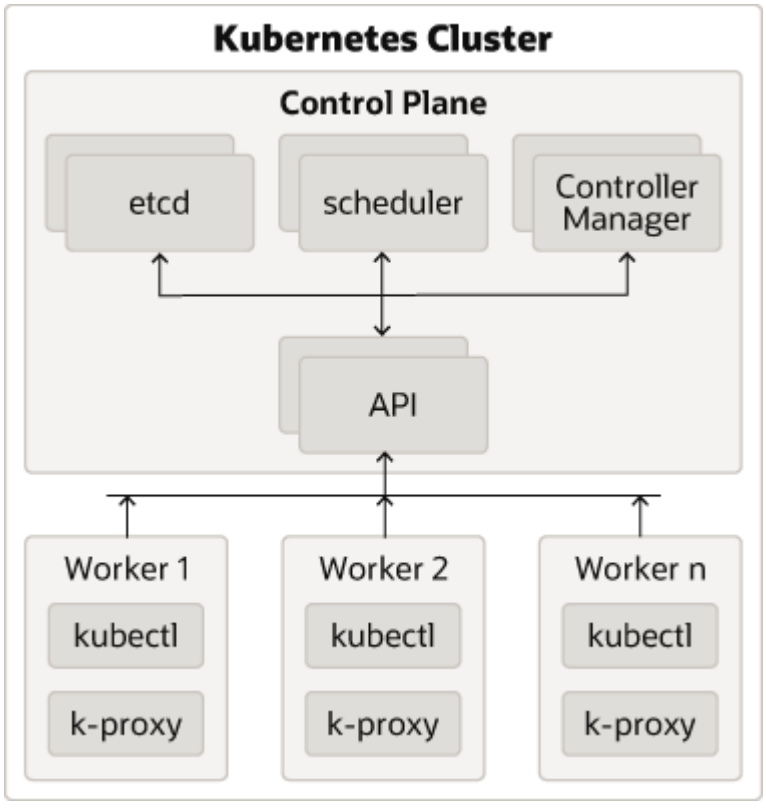
A Kubernetes host consists of a control plane and worker nodes.

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 OAM Deployment

An Oracle Access Management (OAM) deployment uses the Kubernetes components such as pods and Kubernetes services.

Container Image

A container image is an immutable, 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 promote 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 Access Management is based on the July Critical Patch Update (CPU), then to upgrade the pod to use the October CPU image, you have to tell the pod to use the October CPU image and restart the pod. Further information on upgrading can be found in Patching and Upgrading.

Oracle containers are built using a specific user and group ID. Oracle supplies its container images using the user ID 1000 and group ID 0. To enable writing to file systems or persistent volumes, you should grant the write access to this user ID. Oracle supplies all container images using this user and group ID.

If your organization already uses this user or group ID, you should reconfigure the image to use different IDs. This feature is outside the scope of this document.

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 Access Management (OAM) deployment, each OAM server 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 may be desirable that scheduling occurs on a subset of the worker nodes available. This type of scheduling can be achieved by using Kubernetes labels.

Persistent Volumes

When a pod is created, it is based on a container image. A container image is supplied by Oracle for the products you are deploying. When a pod gets created, a runtime environment is created based upon that image. That environment is refreshed with the container image every time the pod is restarted. This means that any changes you make inside a runtime environment are lost whenever the container gets restarted.

A persistent volume is an area of disk, usually provided by NFS that is available to the pod but not part of the image itself. This means that the data you want to keep, for example the OAM domain configuration, is still available after you restart a pod, that is to say, that the data is persistent.

There are two ways of mounting a persistent volume (PV) to a pod:

1. Mount the PV to the pod directly, so that wherever the pod starts in the cluster the PV is available to it. The upside to this approach is that a pod can be started anywhere without extra configuration. The downside to this approach is that there is one NFS volume which is mounted to the pod. If the NFS volume becomes corrupted, you will have to either revert to a backup or have to failover to a disaster recovery site.
2. Mount the PV to the worker node and have the pod interact with it as if it was a local file system. The advantages of this approach are that you can have different NFS volumes mounted to different worker nodes, providing built-in redundancy. The disadvantages of this approach are:
   - Increased management overhead.
   - Pods have to be restricted to nodes that use a specific version of the file system. For example, all odd numbered pods use odd numbered worker nodes mounted to file system 1, and all even numbered pods use even numbered worker nodes mounted to file system 2.
   - File systems have to be mounted to every worker node on which a pod may be started. This requirement is not an issue in a small cluster, unlike in a large cluster.
   - Worker nodes become linked to the application. When a worker node undergoes maintenance, you need to ensure that file systems and appropriate labels are restored.

   You will need to set up a process to ensure that the contents of the NFS volumes are kept in sync by using something such as the rsync cron job.
   If maximum redundancy and availability is your goal, then you should adopt this solution.

Kubernetes Services

Kubernetes services expose the processes running in the pods regardless of the number of pods that are running. For example, OAM 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.

When using Kubernetes, NodePort Services have a similar result as using Ingress. In the NodePort mode, Ingress allows for consolidated management of these services.

This guide describes how to use Ingress using the Nginx Ingress Controller.

The Kubernetes services use a small port range. Therefore, when a Kubernetes service is created, there will be a port mapping. For instance, if a pod is using port 7001, then a Kubernetes/Ingress service may use 30701 as its port, mapping port 30701 to 7001 internally. It is worth noting that if you are using individual NodePort Services, then 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.

To interact with a service, you have to refer to it using the format: `worker_node_hostname:Service` port.

If you have multiple worker nodes, then you should include multiple worker nodes in your calls to remove single points of failure. You can do this in a number of ways including:

- Load balancer
- Direct proxy calls
- DNS CNames

Ingress Controller

There are two ways of interacting with your Kubernetes services. You can create an externally facing service for each Kubernetes object you want to access. This type of service is known as the Kubernetes NodePort Service. Alternatively, you can use an ingress service inside the Kubernetes cluster to redirect requests internally.

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 has the concept of virtual hosts and can terminate SSL, if required. There are various implementations of Ingress. However, this guide describes the installation and configuration of NGNIX. The installation will be similar for other Ingress services but the command syntax may be different. Therefore, when you use a different Ingress, see the appropriate vendor documentation for the equivalent commands. Ingress can proxy HTTP, HTTPS, LDAP, and LDAPS protocols. Ingress is not mandatory

Ingress runs inside the Kubernetes cluster. You can configure it in different ways:

- **Load Balancer**: Load balancer provides an external IP address to which you can connect to interact with the Kubernetes services.
- **NodePort**: In this mode, Ingress acts as a simple load balancer between the Kubernetes services. The difference between using an Ingress NodePort Service as opposed to individual node port services is that the Ingress controller reserves one port for each service type it offers. For example, one for all HTTP communications, another for all LDAP communications, and so on. Individual node port services reserve one port for each service and type used in an application.

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.

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 OAM deployment uses the namespace `oamns`.

3.4 Overview of WebLogic Kubernetes Operator

The WebLogic Kubernetes Operator (the “operator”) supports running Oracle Access Management (OAM) domains on Kubernetes.

The operator takes advantage of the Kubernetes operator pattern, which means that it uses Kubernetes APIs to provide support for operations, such as: provisioning, lifecycle management, application versioning, product patching, scaling, and security. The operator also enables the use of tooling that is native to this infrastructure for monitoring, logging, tracing, and security.

OAM domains are supported using the “domain on a persistent volume” model only, where the domain home is located in a persistent volume (PV).

Domain on persistent volume (Domain on PV) is an operator domain home source type, which requires that the domain home exists on a persistent volume. The domain home can be created either manually using the WebLogic Scripting Tool (WLST) scripts or automatically with WebLogic Deployment Tool (WDT) models by specifying the section, `domain.spec.configuration.initializeDomainOnPV`, in the domain resource YAML file. The initial domain topology and resources are described using WebLogic Deploy Tooling (WDT) models.

> **Note**
>
> The initializeDomainOnPV section provides a one time only domain home initialization. The operator creates the domain when the domain resource is first deployed. After the domain is created, this section is ignored. Subsequent domain lifecycle updates must be controlled by the WebLogic Server Administration Console, WebLogic Scripting Tool (WLST), or other mechanisms.

The WebLogic Kubernetes Operator has several key features to assist you with deploying and managing Oracle Access Management domains in a Kubernetes environment. You can:

- Create OAM instances in a Kubernetes persistent volume. This persistent volume can reside in an NFS file system or other Kubernetes volume types.
- Start servers based on declarative startup parameters and desired states.

- Expose the OAM Services through external access.
- Scale OAM domains by starting and stopping Managed Servers on demand.
- Publish operator and WebLogic Server logs into Elasticsearch and interact with them in Kibana.
- Monitor the OAM instance using Prometheus and Grafana.

WebLogic Kubernetes Operator Limitations with OAM

Compared to running a WebLogic Server domain in Kubernetes using the operator, the following limitations currently exist for OAM domains:

- OAM domains are supported using the “domain on a persistent volume” model only, where the domain home is located in a persistent volume (PV).The “domain in image” model is not supported.
- Only configured clusters are supported. Dynamic clusters are not supported for OAM domains. Note that you can still use all of the scaling features, but you need to define the maximum size of your cluster at domain creation time, using the parameter `configuredManagedServerCount`. For more details on this parameter, see Preparing the Create Domain Script. It is recommended to pre-configure your cluster so it’s sized a little larger than the maximum size you plan to expand it to. You must rigorously test at this maximum size to make sure that your system can scale as expected.
- The WebLogic Monitoring Exporter currently supports the WebLogic MBean trees only. Support for JRF MBeans has not been added yet.
- We do not currently support running OAM in non-Linux containers.

3.5 OAM Deployment Methods

Oracle Access Management (OAM) can be deployed using one of the following methods:

- WebLogic Scripting Tool (WLST) configuration scripts
- WebLogic Deploy Tooling (WDT) models

WebLogic Scripting Tool Configuration Scripts

The OAM WebLogic Scripting Tool (WLST) deployment scripts require you to deploy a separate Kubernetes job that creates the OAM domain on an existing Kubernetes persistent volume (PV) and persistent volume claim (PVC). The Repository Creation Utility (RCU) schemas required for OAM must be created manually in the Oracle Database. The WLST deployment scripts also generate the domain YAML file, which can then be used to start the Kubernetes resources of the corresponding domain.

WebLogic Deploy Tooling Models

WebLogic Deploy Tooling (WDT) models are a convenient and simple alternative to WLST configuration scripts. They compactly define a WebLogic domain using model files, variable properties files, and application archive files.

Using WDT models, all the required information is specified in the domain custom resource YAML file, eliminating the requirement for a separate Kubernetes job. With WDT models, the WebLogic Kubernetes Operator will create the RCU schemas, create the persistent volume and claim, then create the WebLogic domain on the persistent volume, prior to starting the servers.

For more information about the model format and its integration, see Usage and Working With WDT Model Files. The WDT model format is fully described in the open source, WebLogic Deploy Tooling GitHub project.

The main benefits of WDT models are:

- A set of single-purpose tools supporting Weblogic domain configuration lifecycle operations.
- All tools work off of a shared, declarative model, eliminating the need to maintain specialized WLST scripts.
- WDT knowledge base understands the MBeans, attributes, and WLST capabilities/bugs across WLS versions.

Part II

Installing Oracle Access Management on Kubernetes

Install Oracle Access Management (OAM) on Kubernetes.

This section contains the following chapters:

4

Before You Begin

This documentation explains how to configure Oracle Access Management (OAM) on a Kubernetes cluster where no other Oracle Identity Management products will be deployed. For detailed information about this type of deployment, start at System Requirements for OAM on Kubernetes and follow the documentation sequentially.

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.

If you are deploying multiple Oracle Identity Management products on the same Kubernetes cluster, then you must follow Enterprise Deployment Guide for Oracle Identity and Access Management in a Kubernetes Cluster. Please note, you also have the option to follow the Enterprise Deployment Guide for Oracle Identity and Access Management in a Kubernetes Cluster even if you are only installing OAM and no other Oracle Identity Management products.

If you need to understand how to configure a Kubernetes cluster ready for an Oracle Access Management deployment, you should follow Enterprise Deployment Guide for Oracle Identity and Access Management in a Kubernetes Cluster. The Enterprise Deployment Automation section in that guide also contains details on automation scripts that can:

- Automate the creation of a Kubernetes cluster on Oracle Cloud Infrastructure (OCI), ready for the deployment of Oracle Identity Management products.
- Automate the deployment of Oracle Identity Management products on any compliant Kubernetes cluster.

Considerations for Deploying OAM and OHS on Kubernetes

If you intend to use Oracle HTTP Server (OHS) and Oracle WebGate, and want to deploy OHS on Kubernetes, you must read and understand the Supported Architectures in Deploying and Managing Oracle HTTP Server on Kubernetes before continuing.

5

System Requirements for OAM on Kubernetes

This section provides information about the system requirements and limitations for deploying and running Oracle Access Management (OAM) on Kubernetes with the WebLogic Kubernetes Operator 4.2.10.

Kubernetes Requirements

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 2723908.1 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.

> **Note**
>
> All the commands in this guide should be run from the Kubernetes administrative host unless otherwise stated.

- The Kubernetes cluster must have sufficient nodes and resources.
- You must have the `cluster-admin` role to install the WebLogic Kubernetes Operator.
- An installation of Helm is required on the Kubernetes cluster. Helm is used to create and deploy the necessary resources on the Kubernetes cluster.
- A supported container engine such as CRI-O or Docker must be installed and running on the Kubernetes cluster.
- The nodes in the Kubernetes cluster must have access to a persistent volume such as a Network File System (NFS) mount, or a shared file system.
- 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, Helm, or the container engine. Please refer to your vendor specific documentation for this information. Also see Before You Begin.

Database Requirements

You must have a running Oracle Database that meets the following requirements:

- Oracle Database 19.23 or later. The database must be a supported version for OAM as outlined in Oracle Fusion Middleware 14c Certifications.

- The database must meet the requirements as outlined in About Database Requirements for an Oracle Fusion Middleware Installation and in RCU Requirements for Oracle Databases.
- It is recommended that the database initialization parameters are set as per Minimum Initialization Parameters.

Container Registry Requirements

You must have your own container registry to store container and domain images in the following circumstances:

- If your Kubernetes cluster does not have network access to Oracle Container Registry, then you must have your own container registry to store the OAM container images.
- If you intend to deploy OAM with WDT models, you must have a container registry to store the domain image.

Your container registry must be accessible from all nodes in the Kubernetes cluster.

Alternatively if you don't have your own container registry, you can load the images on each worker node in the cluster. Loading the images on each worker node is not recommended as it incurs a large administrative overhead.

> **Note**
>
> This documentation does not tell you how to install a container registry. Please refer to your vendor specific documentation for this information.

6
Preparing Your Environment

Before embarking on Oracle Access Management (OAM) deployment on Kubernetes, you must prepare your environment.

This chapter contains the following topics:

6.1 Confirming the Kubernetes Cluster is Ready

As per System Requirements for OAM 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   1.30.3+1.el8
    node/worker-node2     Ready    <none>                 17h   1.30.3+1.el8
    node/master-node      Ready    control-plane,master   23h   1.30.3+1.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
   ```

6.2 Obtaining the OAM Container image

The Oracle Access Management (OAM) Kubernetes deployment requires access to an OAM container image.

Prebuilt OAM Container Image

The latest prebuilt OAM 14.1.2.1.0 container image can be downloaded from Oracle Container Registry. This image is prebuilt by Oracle and includes Oracle Access Management 14.1.2.1.0, the latest Patch Set Update (PSU) and other fixes released with the Critical Patch Update (CPU) program.

- The OAM container images available can be found on Oracle Container Registry, by navigating to **Middleware** > **oam** for the initial March 2025 release, and **Middleware** > **oam_cpu** for subsequent releases that contain the latest PSU and CPU fixes.
- Before using the image you must login and accept the license agreement.
- Throughout this documentation, the image repository and tag used is: `container-registry.oracle.com/middleware/oam_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD>` where `<YYMMDD>` is the date shown in the image tag. For the initial March 2025 release, replace with `container-registry.oracle.com/middleware/oam:14.1.2.1.0-jdk17-ol8-<YYMMDD>`.

You can use this image in the following ways:

- Pull the container image from the Oracle Container Registry automatically during the OAM 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.

6.3 Creating a Persistent Volume Directory

As referenced in System Requirements for OAM on Kubernetes, the nodes in the Kubernetes cluster must have access to a persistent volume such as a Network File System (NFS) mount or a shared file system.

In the examples below an NFS volume is mounted on all nodes in the Kubernetes cluster, and is accessible via the directory `/nfs_volumes/oam/accessdomainpv`.

Perform the following steps:

1. On the administrative host, run the following command to create an `accessdomainpv` directory:

```
cd <persistent_volume>
mkdir accessdomainpv
sudo chown -R 1000:0 accessdomainpv
```

For example:

```
cd /nfs_volumes/oam
mkdir accessdomainpv
sudo chown -R 1000:0 accessdomainpv
```

2. On the administrative host run the following to ensure it is possible to read and write to the persistent volume:

> **Note**
>
> The following assumes the user creating the file has userid `1000` or is part of group `0`.

```
cd <persistent_volume>/accessdomainpv
touch fileadmin.txt
ls fileadmin.txt
```

For example:

```
cd /nfs_volumes/oam/accessdomainpv
touch fileadmin.txt
ls fileadmin.txt
```

6.4 Setting Up the Code Repository for OAM

To deploy Oracle Access Management (OAM) you need to set up the code repository which provides sample deployment yaml files.

The OAM deployment on Kubernetes leverages the WebLogic Kubernetes Operator infrastructure, and deployment scripts provided by Oracle for creating OAM containers.

Perform the following steps to set up the OAM deployment scripts:

> **Note**
>
> The steps below should be performed on the administrative node that has access to the Kubernetes cluster.

1. Create a working directory to setup the source code:

```
mkdir <workdir>
```

For example:

```
mkdir /OAMK8S
```

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

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

For example:

```
cd /OAMK8S
git clone https://github.com/oracle/fmw-kubernetes.git
```

The output will look similar to the following:

```
Cloning into 'fmw-kubernetes'...
remote: Enumerating objects: 41547, done.
remote: Counting objects: 100% (6171/6171), done.
remote: Compressing objects: 100% (504/504), done.
remote: Total 41547 (delta 5638), reused 5919 (delta 5481), pack-reused
35376 (from 3)
Receiving objects: 100% (41547/41547), 70.32 MiB | 13.12 MiB/s, done.
Resolving deltas: 100% (22214/22214), done.
Checking connectivity... done.
Checking out files: 100% (19611/19611), done
```

3. Set the `$WORKDIR` environment variable as follows:

```
export WORKDIR=<workdir>/fmw-kubernetes/OracleAccessManagement
```

For example:

```
export WORKDIR=/OAMK8S/fmw-kubernetes/OracleAccessManagement
```

4. Run the following command and see if the WebLogic custom resource definition name already exists:

```
kubectl get crd
```

In the output you should see:

```
No resources found
```

If you see any of the following:

```
NAME                       AGE
clusters.weblogic.oracle 5d
domains.weblogic.oracle  5d
```

then run the following command to delete the existing crd's:

```
kubectl delete crd clusters.weblogic.oracle
```

```
kubectl delete crd domains.weblogic.oracle
```

6.5 Installing the WebLogic Kubernetes Operator

Oracle Access Management (OAM) on Kubernetes leverages the WebLogic Kubernetes Operator.

1. Create a Kubernetes namespace for the WebLogic Kubernetes Operator by running the following command:

   ```
   kubectl create namespace <sample-kubernetes-operator-ns>
   ```

   For example:

   ```
   kubectl create namespace opns
   ```

   The output will look similar to the following:

   ```
   namespace/opns created
   ```

2. Create a service account for the operator in the operator's namespace by running the following command:

   ```
   kubectl create serviceaccount -n <sample-kubernetes-operator-ns> <sample-
   kubernetes-operator-sa>
   ```

   For example:

   ```
   kubectl create serviceaccount -n opns op-sa
   ```

   The output will look similar to the following:

   ```
   serviceaccount/op-sa created
   ```

3. Navigate to the `$WORKDIR`:

   ```
    cd $WORKDIR
   ```

4. Run the following helm command to install and start the operator:

   ```
   helm install weblogic-kubernetes-operator kubernetes/charts/weblogic-
   operator \
   --namespace <sample-kubernetes-operator-ns> \
   --set image=ghcr.io/oracle/weblogic-kubernetes-operator:4.2.10 \
   --set serviceAccount=<sample-kubernetes-operator-sa> \
   --set "enableClusterRoleBinding=true" \
   --set "domainNamespaceSelectionStrategy=LabelSelector" \
   ```

```
--set "domainNamespaceLabelSelector=weblogic-operator\=enabled" \
--set "javaLoggingLevel=FINE" --wait
```

For example:

```
helm install weblogic-kubernetes-operator kubernetes/charts/weblogic-
operator \
--namespace opns \
--set image=ghcr.io/oracle/weblogic-kubernetes-operator:4.2.10 \
--set serviceAccount=op-sa \
--set "enableClusterRoleBinding=true" \
--set "domainNamespaceSelectionStrategy=LabelSelector" \
--set "domainNamespaceLabelSelector=weblogic-operator\=enabled" \
--set "javaLoggingLevel=FINE" --wait
```

The output will look similar to the following:

```
NAME: weblogic-kubernetes-operator
LAST DEPLOYED: <DATE>
NAMESPACE: opns
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

5. Verify that the operator's pod and services are running by executing the following command:

```
kubectl get all -n <sample-kubernetes-operator-ns>
```

For example:

```
kubectl get all -n opns
```

The output will look similar to the following:

```
NAME                                                READY   STATUS
RESTARTS   AGE
pod/weblogic-operator-676d5cc6f4-wct7b              1/1     Running
0          40s
pod/weblogic-operator-webhook-7996b8b58b-9sfhd      1/1     Running
0          40s

NAME                                        TYPE        CLUSTER-IP
EXTERNAL-IP   PORT(S)             AGE
service/weblogic-operator-webhook-svc       ClusterIP   10.100.91.237
<none>        8083/TCP,8084/TCP   47s

NAME                                           READY   UP-TO-DATE
AVAILABLE   AGE
deployment.apps/weblogic-operator              1/1     1
1           40s
deployment.apps/weblogic-operator-webhook      1/1     1
1           40s
```

```
NAME                                                  DESIRED   CURRENT
READY   AGE
replicaset.apps/weblogic-operator-676d5cc6f4          1         1
1       40s
replicaset.apps/weblogic-operator-webhook-7996b8b58b  1         1
1       46s
```

6. Verify the operator pod's log:

   ```
   kubectl logs -n <sample-kubernetes-operator-ns> -c weblogic-operator
   deployments/weblogic-operator
   ```

   For example:

   ```
   kubectl logs -n opns -c weblogic-operator deployments/weblogic-operator
   ```

   The output will look similar to the following:

   ```
   ...
   {"timestamp":"<DATE>","thread":21,"fiber":"","namespace":"","domainUID":"",
   "level":"FINE","class":"oracle.kubernetes.operator.DeploymentLiveness","met
   hod":"run","timeInMillis":1678183291191,"message":"Liveness file last
   modified time set","exception":"","code":"","headers":{},"body":""}
   {"timestamp":"<DATE>","thread":37,"fiber":"","namespace":"","domainUID":"",
   "level":"FINE","class":"oracle.kubernetes.operator.DeploymentLiveness","met
   hod":"run","timeInMillis":1678183296193,"message":"Liveness file last
   modified time set","exception":"","code":"","headers":{},"body":""}
   {"timestamp":"<DATE>","thread":31,"fiber":"","namespace":"","domainUID":"",
   "level":"FINE","class":"oracle.kubernetes.operator.DeploymentLiveness","met
   hod":"run","timeInMillis":1678183301194,"message":"Liveness file last
   modified time set","exception":"","code":"","headers":{},"body":""}
   {"timestamp":"<DATE>","thread":31,"fiber":"","namespace":"","domainUID":"",
   "level":"FINE","class":"oracle.kubernetes.operator.DeploymentLiveness","met
   hod":"run","timeInMillis":1678183306195,"message":"Liveness file last
   modified time set","exception":"","code":"","headers":{},"body":""}
   ```

6.6 Creating a Kubernetes Namespace

You must create a namespace to store the Kubernetes objects for Oracle Access Management (OAM).

1. Create a Kubernetes namespace for the OAM deployment by running the following command:

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

   For example:

   ```
   kubectl create namespace oamns
   ```

The output will look similar to the following:

```
namespace/oamns created
```

2. Run the following command to tag the namespace so the WebLogic Kubernetes Operator can manage it:

```
kubectl label namespaces <domain_namespace> weblogic-operator=enabled
```

For example:

```
kubectl label namespaces oamns weblogic-operator=enabled
```

The output will look similar to the following:

```
namespace/oamns labeled
```

3. Run the following command to check the label was created:

```
kubectl describe namespace <domain_namespace>
```

For example:

```
kubectl describe namespace oamns
```

The output will look similar to the following:

```
Name:         oamns
Labels:       kubernetes.io/metadata.name=oamns
              weblogic-operator=enabled
Annotations:  <none>
Status:       Active

No resource quota.

No LimitRange resource.
```

6.7 Creating a Kubernetes Secret for the Container Registry

Create a Kubernetes secret to stores the credentials for the container registry where the Oracle Access Management (OAM) image is stored. This step must be followed if using Oracle Container Registry or your own private container registry. If you are not using a container registry and have loaded the images on each of the worker nodes, you can skip this section.

1. Run the following command to create the secret:

```
kubectl create secret docker-registry "orclcred" --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 "orclcred" --docker-server=container-
registry.oracle.com \
--docker-username="user@example.com" \
--docker-password=password --docker-email=user@example.com \
--namespace=oamns
```

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

- If using Oracle Container Registry to pull the OAM container image, this is the username and password used to login to Oracle Container Registry. Before you can use this image you must login to Oracle Container Registry, navigate to **Middleware** > **oam** and accept the license agreement. For future releases (post March 2025) that contain the latest Patch Set Update (PSU) and other fixes released with the Critical Patch Update (CPU) program, you should navigate to **Middleware** > **oam_cpu**.
- If using your own container registry to store the OAM container image, this is the username and password (or token) for your container registry.

The output will look similar to the following:

```
secret/orclcred created
```

7

Creating Oracle Access Management Domains

Choose one of the following supported methods to create an Oracle Access Management (OAM) domain:

- Creating OAM Domains Using WLST Offline Scripts
- Creating OAM Domains Using WDT Models

7.1 Creating OAM Domains Using WLST Offline Scripts

The Oracle Access Management (OAM) deployment scripts demonstrate the creation of an OAM domain home on an existing Kubernetes persistent volume (PV) and persistent volume claim (PVC). The scripts also generate the domain YAML file, which can then be used to start the Kubernetes artifacts of the corresponding domain.

Before following this section, make sure you have followed Preparing Your Environment, and ensure your Oracle Database is running.

This section includes the following topics:

- Creating the RCU Schemas
- Creating a Kubernetes Secret for the WLST Domain
- Creating a Kubernetes Secret for RCU in WLST
- Creating a Kubernetes Persistent Volume and Persistent Volume Claim
- Preparing the Create Domain Script
- Creating the domain.yaml
- Setting the OAM Server Memory Parameters
- Deploying the WLST OAM Domain
- Verifying the WLST OAM Deployment

7.1.1 Creating the RCU Schemas

In this section you create the Repository Creation Utility (RCU) schemas in the Oracle Database.

> **Note**
>
> Before following the steps below, make sure that the Oracle Database and Listener are up and running, and you can connect to the database via SQL*Plus or other client tool.

1. Run the following command to create a helper pod to run RCU:

- If using Oracle Container Registry or your own container registry for the Oracle Access Management (OAM) container image:

  ```
  kubectl run --image=<image_name-from-registry>:<tag> \
  --image-pull-policy="IfNotPresent" \
  --overrides='{"apiVersion": "v1", "spec":{"imagePullSecrets": [{"name":
  "orclcred"}]}}' \
  helper -n <domain_namespace> \
  -- sleep infinity
  ```

  For example:

  ```
  kubectl run --image=container-registry.oracle.com/middleware/
  oam_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD> \
  --image-pull-policy="IfNotPresent" \
  --overrides='{"apiVersion": "v1","spec":{"imagePullSecrets": [{"name":
  "orclcred"}]}}' \
  helper -n oamns \
  -- sleep infinity
  ```

- If you are not using a container registry and have loaded the image on each of the worker nodes, run the following command:

  ```
  kubectl run helper --image <image>:<tag> -n oamns -- sleep infinity
  ```

  For example:

  ```
  kubectl run helper --image oracle/oam_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD>
  -n oamns -- sleep infinity
  ```

The output will look similar to the following:

```
pod/helper created
```

2. Run the following command to check the pod is running:

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

   For example:

   ```
   kubectl get pods -n oamns
   ```

   The output will look similar to the following:

   ```
   NAME     READY   STATUS    RESTARTS   AGE
   helper   1/1     Running   0          3m
   ```

> **Note**
>
> If you are pulling the image from a container registry it may take several minutes before the pod has a `READY` status of `1\1`. While the pod is starting you can check the status of the pod, by running the following command:
>
> ```
> kubectl describe pod helper -n oamns
> ```

3. Run the following command to start a bash shell in the helper pod:

   ```
   kubectl exec -it helper -n <domain_namespace> -- /bin/bash
   ```

   For example:

   ```
   kubectl exec -it helper -n oamns -- /bin/bash
   ```

   This will take you into a bash shell in the running helper pod:

   ```
   [oracle@helper ~]$
   ```

4. In the helper bash shell run the following commands to set the environment:

   ```
   export CONNECTION_STRING=<db_host.domain>:<db_port>/<service_name>


   export RCUPREFIX=<rcu_schema_prefix>


   echo -e <db_pwd>"\n"<rcu_schema_pwd> > /tmp/pwd.txt


   cat /tmp/pwd.txt
   ```

   Where:

   - `<db_host.domain>:<db_port>/<service_name>` is your database connect string.
   - `<rcu_schema_prefix>` is the RCU schema prefix you want to set.
   - `<db_pwd>` is the SYS password for the database.
   - `<rcu_schema_pwd>` is the password you want to set for the `<rcu_schema_prefix>`

For example:

```
export CONNECTION_STRING=mydatabasehost.example.com:1521/orcl.example.com


export RCUPREFIX=OAMK8S


echo -e <password>"\n"<password> > /tmp/pwd.txt


cat /tmp/pwd.txt
```

Ensure the `cat /tmp/pwd.txt` command shows the correct passwords.

5. In the helper bash shell, run the following command to create the RCU schemas in the database:

```
/u01/oracle/oracle_common/bin/rcu -silent -createRepository -databaseType
ORACLE -connectString \
$CONNECTION_STRING -dbUser sys -dbRole sysdba -
useSamePasswordForAllSchemaUsers true \
-selectDependentsForComponents true -schemaPrefix $RCUPREFIX -component
MDS -component IAU \
-component IAU_APPEND -component IAU_VIEWER -component OPSS -component WLS
-component STB -component OAM -f < /tmp/pwd.txt
```

The output will look similar to the following:

```
RCU Logfile: /tmp/RCU<DATE>/logs/rcu.log
Processing command line ....
Repository Creation Utility - Checking Prerequisites
Checking Global Prerequisites
Repository Creation Utility - Checking Prerequisites
Checking Component Prerequisites
Repository Creation Utility - Creating Tablespaces
Validating and Creating Tablespaces
Create tablespaces in the repository database
Repository Creation Utility - Create
Repository Create in progress.
Executing pre create operations
Percent Complete: 18
Percent Complete: 18
Percent Complete: 19
Percent Complete: 20
Percent Complete: 21
Percent Complete: 21
Percent Complete: 22
Percent Complete: 22
Creating Common Infrastructure Services(STB)
Percent Complete: 30
Percent Complete: 30
Percent Complete: 39
Percent Complete: 39
Percent Complete: 39
```

```
Creating Audit Services Append(IAU_APPEND)
Percent Complete: 46
Percent Complete: 46
Percent Complete: 55
Percent Complete: 55
Percent Complete: 55
Creating Audit Services Viewer(IAU_VIEWER)
Percent Complete: 62
Percent Complete: 62
Percent Complete: 63
Percent Complete: 63
Percent Complete: 64
Percent Complete: 64
Creating Metadata Services(MDS)
Percent Complete: 73
Percent Complete: 73
Percent Complete: 73
Percent Complete: 74
Percent Complete: 74
Percent Complete: 75
Percent Complete: 75
Percent Complete: 75
Creating Weblogic Services(WLS)
Percent Complete: 80
Percent Complete: 80
Percent Complete: 83
Percent Complete: 83
Percent Complete: 91
Percent Complete: 98
Percent Complete: 98
Creating Audit Services(IAU)
Percent Complete: 100
Creating Oracle Platform Security Services(OPSS)
Creating Oracle Access Manager(OAM)
Executing post create operations
Repository Creation Utility: Create - Completion Summary
Database details:
-----------------------------
Host Name : mydatabasehost.example.com
Port : 1521
Service Name : ORCL.EXAMPLE.COM
Connected As : sys
Prefix for (prefixable) Schema Owners : OAMK8S
RCU Logfile                                 : /tmp/RCU<DATE>/logs/rcu.log

Component schemas created:
-----------------------------
Component                                   Status         Logfile

Common Infrastructure Services              Success        /tmp/RCU<DATE>/
logs/stb.log
Oracle Platform Security Services           Success        /tmp/RCU<DATE>/
logs/opss.log
Oracle Access Manager                       Success        /tmp/RCU<DATE>/
logs/oam.log
Audit Services                              Success        /tmp/RCU<DATE>/
```

```
logs/iau.log
Audit Services Append                          Success        /tmp/RCU<DATE>/
logs/iau_append.log
Audit Services Viewer                          Success        /tmp/RCU<DATE>/
logs/iau_viewer.log
Metadata Services                              Success        /tmp/RCU<DATE>/
logs/mds.log
WebLogic Services                              Success        /tmp/RCU<DATE>/
logs/wls.log

Repository Creation Utility - Create : Operation Completed
[oracle@helper ~]$
```

6. Exit the helper bash shell by issuing the command `exit`.

7.1.2 Creating a Kubernetes Secret for the WLST Domain

Create a Kubernetes secret for the domain using the `create-weblogic-credentials` script.

1. Navigate to the `$WORKDIR/kubernetes/create-weblogic-domain-credentials` directory:

   ```
   cd $WORKDIR/kubernetes/create-weblogic-domain-credentials
   ```

2. Run the following command to create the secret:

   ```
   ./create-weblogic-credentials.sh -u weblogic -p <pwd> -n
   <domain_namespace> -d <domain_uid> -s <kubernetes_domain_secret>
   ```

   Where:

   - `-u weblogic` is the WebLogic username.
   - `-p <pwd>` is the password for the WebLogic user.
   - `-n <domain_namespace>` is the domain namespace.
   - `-d <domain_uid>` is the domain UID to be created.
   - `-s <kubernetes_domain_secret>` is the name you want to create for the secret for this namespace.

   For example:

   ```
   ./create-weblogic-credentials.sh -u weblogic -p <password> -n oamns -d
   accessdomain -s accessdomain-credentials
   ```

   The output will look similar to the following:

   ```
   secret/accessdomain-credentials created
   secret/accessdomain-credentials labeled
   The secret accessdomain-credentials has been successfully created in the
   oamns namespace.
   ```

3. Verify the secret is created using the following command:

   ```
   kubectl get secret <kubernetes_domain_secret> -o yaml -n <domain_namespace>
   ```

For example:

```
kubectl get secret accessdomain-credentials -o yaml -n oamns
```

The output will look similar to the following:

```
apiVersion: v1
data:
  password: V2VsY29tZTE=
  username: d2VibG9naWM=
kind: Secret
metadata:
  creationTimestamp: "<DATE>"
  labels:
    weblogic.domainName: accessdomain
    weblogic.domainUID: accessdomain
  name: accessdomain-credentials
  namespace: oamns
  resourceVersion: "29428101"
  uid: 6dac0561-d157-4144-9ed7-c475a080eb3a
type: Opaque
```

7.1.3 Creating a Kubernetes Secret for RCU in WLST

Create a Kubernetes secret for RCU using the `create-rcu-credentials` script.

1. Navigate to the following directory:

   ```
   cd $WORKDIR/kubernetes/create-rcu-credentials
   ```

2. Run the following command to create the secret:

   ```
   ./create-rcu-credentials.sh -u <rcu_prefix> -p <rcu_schema_pwd> -a sys -q
   <sys_db_pwd> -d <domain_uid> -n <domain_namespace> -s
   <kubernetes_rcu_secret>
   ```

   Where:

   - `-u <rcu_prefix>` is the name of the RCU schema prefix created in Creating the RCU Schemas.
   - `-p <rcu_schema_pwd>` is the password for the RCU schema prefix.
   - `-q <sys_db_pwd>` is the SYS database password.
   - `-d <domain_uid>` is the same domain UID that you specified in Creating a Kubernetes Secret for the WLST Domain.
   - `-n <domain_namespace>` is the domain namespace.
   - `-s <kubernetes_rcu_secret>` is the name of the RCU secret to create.

   For example:

   ```
   ./create-rcu-credentials.sh -u OAMK8S -p <password> -a sys -q <password> -
   d accessdomain -n oamns -s accessdomain-rcu-credentials
   ```

The output will look similar to the following:

```
secret/accessdomain-rcu-credentials created
secret/accessdomain-rcu-credentials labeled
The secret accessdomain-rcu-credentials has been successfully created in
the oamns namespace.
```

3. Verify the secret is created using the following command:

```
kubectl get secret <kubernetes_rcu_secret> -o yaml -n <domain_namespace>
```

For example:

```
kubectl get secret accessdomain-rcu-credentials -o yaml -n oamns
```

The output will look similar to the following:

```
apiVersion: v1
data:
  password: T3JhY2xlXzEyMw==
  sys_password: T3JhY2xlXzEyMw==
  sys_username: c3lz
  username: T0FNSzhT
kind: Secret
metadata:
  creationTimestamp: "<DATE>"
  labels:
    weblogic.domainName: accessdomain
    weblogic.domainUID: accessdomain
  name: accessdomain-rcu-credentials
  namespace: oamns
  resourceVersion: "29428242"
  uid: 1b81b6e0-fd7d-40b8-a060-454c8d23f4dc
type: Opaque
```

7.1.4 Creating a Kubernetes Persistent Volume and Persistent Volume Claim

As referenced in Creating a Persistent Volume Directory , the nodes in the Kubernetes cluster must have access to a persistent volume such as a Network File System (NFS) mount or a shared file system.

A persistent volume is the same as a disk mount but is inside a container. A Kubernetes persistent volume is an arbitrary name (determined in this case, by Oracle) that is mapped to a physical volume on a disk.

When a container is started, it needs to mount that volume. The physical volume should be on a shared disk accessible by all the Kubernetes worker nodes because it is not known on which worker node the container will be started. In the case of Oracle Access Management, the persistent volume does not get erased when a container stops. This enables persistent configurations.

The example below uses an NFS mounted volume (`<persistent_volume>/accessdomainpv`). Other volume types can also be used. See, Volumes for more information.

To create a Kubernetes persistent volume, perform the following steps:

1. Navigate to the `$WORKDIR/kubernetes/create-weblogic-domain-pv-pvc` directory:

   ```
   cd $WORKDIR/kubernetes/create-weblogic-domain-pv-pvc
   ```

2. Make a backup copy of the `create-pv-pvc-inputs.yaml` file and create an `output` directory:

   ```
   cp create-pv-pvc-inputs.yaml create-pv-pvc-inputs.yaml.orig
   ```

   ```
   mkdir output
   ```

3. Edit the `create-pv-pvc-inputs.yaml` file and update the following parameters to reflect your settings. Save the file when complete:

   ```
   baseName: <domain>
   domainUID: <domain_uid>
   namespace: <domain_namespace>
   weblogicDomainStorageType: NFS
   weblogicDomainStorageNFSServer: <nfs_server>
   weblogicDomainStoragePath: <physical_path_of_persistent_storage>
   weblogicDomainStorageSize: 10Gi
   ```

   For example:

   ```
   # The base name of the pv and pvc
   baseName: domain

   # Unique ID identifying a domain.
   # If left empty, the generated pv can be shared by multiple domains
   # This ID must not contain an underscope ("_"), and must be lowercase and
   unique across all domains in a Kubernetes cluster.
   domainUID: accessdomain

   # Name of the namespace for the persistent volume claim
   namespace: oamns
   ...
   # Persistent volume type for the persistent storage.
   # The value must be 'HOST_PATH' or 'NFS'.
   # If using 'NFS', weblogicDomainStorageNFSServer must be specified.
   weblogicDomainStorageType: NFS

   # The server name or ip address of the NFS server to use for the
   persistent storage.
   # The following line must be uncomment and customized if
   weblogicDomainStorateType is NFS:
   weblogicDomainStorageNFSServer: mynfsserver

   # Physical path of the persistent storage.
   # When weblogicDomainStorageType is set to HOST_PATH, this value should be
   set the to path to the
   ```

```
# domain storage on the Kubernetes host.
# When weblogicDomainStorageType is set to NFS, then
weblogicDomainStorageNFSServer should be set
# to the IP address or name of the DNS server, and this value should be
set to the exported path
# on that server.
# Note that the path where the domain is mounted in the WebLogic
containers is not affected by this
# setting, that is determined when you create your domain.
# The following line must be uncomment and customized:
weblogicDomainStoragePath: /nfs_volumes/oam/accessdomainpv

# Reclaim policy of the persistent storage
# The valid values are: 'Retain', 'Delete', and 'Recycle'
weblogicDomainStorageReclaimPolicy: Retain

# Total storage allocated to the persistent storage.
weblogicDomainStorageSize: 10Gi
```

4. Execute the `create-pv-pvc.sh` script to create the PV and PVC configuration files:

   ```
   ./create-pv-pvc.sh -i create-pv-pvc-inputs.yaml -o output
   ```

   The output will be similar to the following:

   ```
   Input parameters being used
   export version="create-weblogic-sample-domain-pv-pvc-inputs-v1"
   export baseName="domain"
   export domainUID="accessdomain"
   export namespace="oamns"
   export weblogicDomainStorageType="NFS"
   export weblogicDomainStorageNFSServer="mynfsserver"
   export weblogicDomainStoragePath="/nfs_volumes/oam/accessdomainpv"
   export weblogicDomainStorageReclaimPolicy="Retain"
   export weblogicDomainStorageSize="10Gi"

   Generating output/pv-pvcs/accessdomain-domain-pv.yaml
   Generating output/pv-pvcs/accessdomain-domain-pvc.yaml
   The following files were generated:
     output/pv-pvcs/accessdomain-domain-pv.yaml.yaml
     output/pv-pvcs/accessdomain-domain-pvc.yaml
   ```

5. Run the following command to show the files are created:

   ```
   ls output/pv-pvcs
   ```

   The output will look similar to the following:

   ```
   accessdomain-domain-pv.yaml  accessdomain-domain-pvc.yaml  create-pv-pvc-
   inputs.yaml
   ```

6. Run the following command to create the PV in the domain namespace:

```
kubectl create -f output/pv-pvcs/accessdomain-domain-pv.yaml -n
<domain_namespace>
```

For example:

```
kubectl create -f output/pv-pvcs/accessdomain-domain-pv.yaml -n oamns
```

The output will look similar to the following:

```
persistentvolume/accessdomain-domain-pv created
```

7. Run the following commands to verify the PV was created successfully:

```
kubectl describe pv accessdomain-domain-pv
```

The output will look similar to the following:

```
Name:            accessdomain-domain-pv
Labels:          weblogic.domainUID=accessdomain
Annotations:     pv.kubernetes.io/bound-by-controller: yes
Finalizers:      [kubernetes.io/pv-protection]
StorageClass:    accessdomain-domain-storage-class
Status:          Bound
Claim:           oamns/accessdomain-domain-pvc
Reclaim Policy: Retain
Access Modes:    RWX
VolumeMode:      Filesystem
Capacity:        10Gi
Node Affinity:  <none>
Message:
Source:
    Type:      NFS (an NFS mount that lasts the lifetime of a pod)
    Server:    mynfsserver
    Path:      /nfs_volumes/oam/accessdomainpv
    ReadOnly:  false
Events: <none>
```

8. Run the following command to create the PVC in the domain namespace:

```
kubectl create -f output/pv-pvcs/accessdomain-domain-pvc.yaml -n
<domain_namespace>
```

For example:

```
kubectl create -f output/pv-pvcs/accessdomain-domain-pvc.yaml -n oamns
```

The output will look similar to the following:

```
persistentvolume/accessdomain-domain-pvc created
```

9. Run the following commands to verify the PVC was created successfully:

```
kubectl describe pvc accessdomain-domain-pvc -n <namespace>
```

For example:

```
kubectl describe pvc accessdomain-domain-pvc -n oamns
```

The output will look similar to the following:

```
Name:            accessdomain-domain-pvc
Namespace:       oamns
StorageClass:    accessdomain-domain-storage-class
Status:          Bound
Volume:          accessdomain-domain-pv
Labels:          weblogic.domainUID=accessdomain
Annotations:     pv.kubernetes.io/bind-completed: yes
                 pv.kubernetes.io/bound-by-controller: yes
Finalizers:     [kubernetes.io/pvc-protection]
Capacity:       10Gi
Access Modes:   RWX
VolumeMode:     Filesystem
Events:         <none>
Mounted By:     <none>
```

7.1.5 Preparing the Create Domain Script

The sample scripts for Oracle Access Management (OAM) domain deployment are available in the `$WORKDIR/kubernetes/create-access-domain` directory. You must prepare the scripts before deploying OAM.

1. Navigate to the `$WORKDIR/kubernetes/create-access-domain/domain-home-on-pv` directory:

```
cd $WORKDIR/kubernetes/create-access-domain/domain-home-on-pv
```

2. Make a copy of the `create-domain-inputs.yaml` file:

```
cp create-domain-inputs.yaml create-domain-inputs.yaml.orig
```

3. Edit the `create-domain-inputs.yaml` and modify the following parameters. Save the file when complete:

```
domainUID: <domain_uid>
domainHome: /u01/oracle/user_projects/domains/<domain_uid>
image: <image_name>:<tag>
imagePullSecretName: <container_registry_secret>
weblogicCredentialsSecretName: <kubernetes_domain_secret>
logHome: /u01/oracle/user_projects/domains/logs/<domain_uid>
namespace: <domain_namespace>
persistentVolumeClaimName: <pvc_name>
rcuSchemaPrefix: <rcu_prefix>
```

```
rcuDatabaseURL: <rcu_db_host>:<rcu_db_port>/<rcu_db_service_name>
rcuCredentialsSecret: <kubernetes_rcu_secret>
```

For example:

```
domainUID: accessdomain
domainHome: /u01/oracle/user_projects/domains/accessdomain
image: container-registry.oracle.com/middleware/oam_cpu:14.1.2.1.0-jdk17-
ol8-<YYMMDD>
imagePullSecretName: orclcred
weblogicCredentialsSecretName: accessdomain-credentials
logHome: /u01/oracle/user_projects/domains/logs/accessdomain
namespace: oamns
persistentVolumeClaimName: accessdomain-domain-pvc
rcuSchemaPrefix: OAMK8S
rcuDatabaseURL: mydatabasehost.example.com:1521/orcl.example.com
rcuCredentialsSecret: accessdomain-rcu-credentials
```

A full list of parameters in the `create-domain-inputs.yaml` file are shown below:

Parameter	Definition	Default
adminPort	Port number for the Administration Server inside the Kubernetes cluster.	7001
adminNodePort	Port number of the Administration Server outside the Kubernetes cluster.	30701
adminServerName	Name of the Administration Server.	AdminServer
clusterName	Name of the WebLogic cluster instance to generate for the domain. By default the cluster name is oam_cluster for the OAM domain.	oam_cluster
configuredManagedServerCount	Number of Managed Server instances to generate for the domain.	5

Parameter	Definition	Default
createDomainFilesDir	Directory on the host machine to locate all the files to create a WebLogic domain, including the script that is specified in the `createDomainScriptName` property. By default, this directory is set to the relative path `wlst`, and the create script will use the built-in WLST offline scripts in the `wlst` directory to create the WebLogic domain. It can also be set to the relative path `wdt`, and then the built-in WDT scripts will be used instead. An absolute path is also supported to point to an arbitrary directory in the file system. The built-in scripts can be replaced by the user-provided scripts or model files as long as those files are in the specified directory. Files in this directory are put into a Kubernetes config map, which in turn is mounted to the `createDomainScriptsMountPath`, so that the Kubernetes pod can use the scripts and supporting files to create a domain home.	wlst
createDomainScriptsMountPath	Mount path where the create domain scripts are located inside a pod. The `create-domain.sh` script creates a Kubernetes job to run the script (specified in the `createDomainScriptName` property) in a Kubernetes pod to create a domain home. Files in the `createDomainFilesDir` directory are mounted to this location in the pod, so that the Kubernetes pod can use the scripts and supporting files to create a domain home.	/u01/weblogic

Parameter	Definition	Default
createDomainScriptName	Script that the create domain script uses to create a WebLogic domain. The create-domain.sh script creates a Kubernetes job to run this script to create a domain home. The script is located in the in-pod directory that is specified in the createDomainScriptsMountPath property. If you need to provide your own scripts to create the domain home, instead of using the built-it scripts, you must use this property to set the name of the script that you want the create domain job to run.	create-domain-job.sh
domainHome	Home directory of the OAM domain. If not specified, the value is derived from the `domainUID` as `/shared/domains/<domainUID>`.	/u01/oracle/user_projects/domains/accessdomain
domainPVMountPath	Mount path of the domain persistent volume.	/u01/oracle/user_projects/domains
domainUID	Unique ID that will be used to identify this particular domain. Used as the name of the generated WebLogic domain as well as the name of the Kubernetes domain resource. This ID must be unique across all domains in a Kubernetes cluster. This ID cannot contain any character that is not valid in a Kubernetes service name.	accessdomain
domainType	Type of the domain. Mandatory input for OAM domains. You must provide one of the supported domain type value: `oam` (deploys an OAM domain)	oam
exposeAdminNodePort	Boolean indicating if the Administration Server is exposed outside of the Kubernetes cluster.	false
exposeAdminT3Channel	Boolean indicating if the T3 administrative channel is exposed outside the Kubernetes cluster.	false
image	OAM container image. The operator requires OAM 14.1.2. Refer to Obtaining the OAM Container image for details on how to obtain or create the image.	oracle/oam:14.1.2.1.0
imagePullPolicy	WebLogic container image pull policy. Legal values are `IfNotPresent`, `Always`, or `Never`	IfNotPresent

Parameter	Definition	Default
imagePullSecretName	Name of the Kubernetes secret to access the container registry to pull the OAM container image. The presence of the secret will be validated when this parameter is specified.	orclcred
includeServerOutInPodLog	Boolean indicating whether to include the server .out to the pod's stdout.	true
initialManagedServerReplicas	Number of Managed Servers to initially start for the domain.	1
javaOptions	Java options for starting the Administration Server and Managed Servers. A Java option can have references to one or more of the following pre-defined variables to obtain WebLogic domain information: `$(DOMAIN_NAME)`, `$(DOMAIN_HOME)`, `$(ADMIN_NAME)`, `$(ADMIN_PORT)`, and `$(SERVER_NAME)`.	-Dweblogic.StdoutDebugEnabled=false
logHome	The in-pod location for the domain log, server logs, server out, and Node Manager log files. If not specified, the value is derived from the `domainUID` as `/shared/logs/<domainUID>`.	/u01/oracle/user_projects/domains/logs/accessdomain
managedServerNameBase	Base string used to generate Managed Server names.	oam_server
managedServerPort	Port number for each Managed Server.	14100
namespace	Kubernetes namespace in which to create the domain.	oamns
persistentVolumeClaimName	Name of the persistent volume claim created to host the domain home. If not specified, the value is derived from the `domainUID` as `<domainUID>-weblogic-sample-pvc`.	accessdomain-domain-pvc
productionModeEnabled	Boolean indicating if production mode is enabled for the domain.	true
serverStartPolicy	Determines which WebLogic Server instances will be started. Legal values are `Never`, `IfNeeded`, `AdminOnly`.	IfNeeded
t3ChannelPort	Port for the T3 channel of the NetworkAccessPoint.	30012

Parameter	Definition	Default
t3PublicAddress	Public address for the T3 channel. This should be set to the public address of the Kubernetes cluster. This would typically be a load balancer address. For development environments only: In a single server (all-in-one) Kubernetes deployment, this may be set to the address of the master, or at the very least, it must be set to the address of one of the worker nodes.	If not provided, the script will attempt to set it to the IP address of the Kubernetes cluster
weblogicCredentialsSecretName	Name of the Kubernetes secret for the Administration Server's user name and password. If not specified, then the value is derived from the `domainUID` as `<domainUID>-weblogic-credentials`.	accessdomain-domain-credentials
serverPodCpuRequest, serverPodMemoryRequest, serverPodCpuCLimit, serverPodMemoryLimit	The maximum amount of compute resources allowed, and minimum amount of compute resources required, for each server pod. Please refer to the Kubernetes documentation on `Managing Compute Resources for Containers` for details.	Resource requests and resource limits are not specified.
rcuSchemaPrefix	The schema prefix to use in the database, for example OAM1. You may wish to make this the same as the domainUID in order to simplify matching domains to their RCU schemas.	OAM1
rcuDatabaseURL	The database URL.	xxxxx.example.com:1521/ oampdb1.example.com
rcuCredentialsSecret	The Kubernetes secret containing the database credentials.	accessdomain-rcu-credentials
datasourceType	Type of JDBC datasource applicable for the OAM domain. Legal values are agl and generic. Choose agl for Active GridLink datasource and generic for Generic datasource. For enterprise deployments, Oracle recommends that you use GridLink data sources to connect to Oracle RAC databases. See, Enterprise Deployment Guide for Oracle Identity and Access Management in a Kubernetes Cluster for further details.	generic

> **Note**
>
> The names of the Kubernetes resources in the generated YAML files may be formed with the value of some of the properties specified in the `create-inputs.yaml` file. Those properties include the `adminServerName`, `clusterName` and `managedServerNameBase`. If those values contain any characters that are invalid in a Kubernetes service name, those characters are converted to valid values in the generated YAML files. For example, an uppercase letter is converted to a lowercase letter and an underscore ("_") is converted to a hyphen ("-").
> The sample demonstrates how to create an OAM domain home and associated Kubernetes resources for a domain that has one cluster only. In addition, the sample provides the capability for users to supply their own scripts to create the domain home for other use cases. The generated domain YAML file could also be modified to cover more use cases.

7.1.6 Creating the domain.yaml

To create the `domain.yaml` file used in the Oracle Access Management (OAM) deployment:

1. Navigate to the `$WORKDIR/kubernetes/create-access-domain/domain-home-on-pv` directory:

   ```
   cd $WORKDIR/kubernetes/create-access-domain/domain-home-on-pv
   ```

2. Run the create domain script, specifying your inputs file and an output directory to store the generated artifacts. This command creates a `domain.yaml` file required for domain creation:

   ```
   ./create-domain.sh -i create-domain-inputs.yaml -o output
   ```

   The output will look similar to the following:

   ```
   export version="create-weblogic-sample-domain-inputs-v1"
   export adminPort="7001"
   export adminServerSSLPort="7002"
   export adminServerName="AdminServer"
   export domainUID="accessdomain"
   export domainType="oam"
   export domainHome="/u01/oracle/user_projects/domains/accessdomain"
   export serverStartPolicy="IfNeeded"
   export clusterName="oam_cluster"
   export configuredManagedServerCount="5"
   export initialManagedServerReplicas="1"
   export managedServerNameBase="oam_server"
   export managedServerPort="14100"
   export image="container-registry.oracle.com/middleware/oam_cpu:14.1.2.1.0-
   jdk17-ol8-<YYMMDD>"
   export imagePullPolicy="IfNotPresent"
   export imagePullSecretName="orclcred"
   export productionModeEnabled="true"
   export weblogicCredentialsSecretName="accessdomain-credentials"
   export includeServerOutInPodLog="true"
   export logHome="/u01/oracle/user_projects/domains/logs/accessdomain"
   ```

```
export httpAccessLogInLogHome="true"
export t3ChannelPort="30012"
export exposeAdminT3Channel="false"
export adminNodePort="30701"
export exposeAdminNodePort="false"
export namespace="oamns"
javaOptions=-Dweblogic.StdoutDebugEnabled=false
export persistentVolumeClaimName="accessdomain-domain-pvc"
export domainPVMountPath="/u01/oracle/user_projects/domains"
export createDomainScriptsMountPath="/u01/weblogic"
export createDomainScriptName="create-domain-job.sh"
export createDomainFilesDir="wlst"
export rcuSchemaPrefix="OAMK8S"
export rcuDatabaseURL="mydatabasehost.example.com:1521/orcl.example.com"
export rcuCredentialsSecret="accessdomain-rcu-credentials"
export datasourceType="generic"

validateWlsDomainName called with accessdomain
createFiles - valuesInputFile is create-domain-inputs.yaml
createDomainScriptName is create-domain-job.sh
Generating output/weblogic-domains/accessdomain/create-domain-job.yaml
Generating output/weblogic-domains/accessdomain/delete-domain-job.yaml
Generating output/weblogic-domains/accessdomain/domain.yaml
Checking to see if the secret accessdomain-credentials exists in namespace
oamns
configmap/accessdomain-create-oam-infra-domain-job-cm created
Checking the configmap accessdomain-create-oam-infra-domain-job-cm was
created
configmap/accessdomain-create-oam-infra-domain-job-cm labeled
Checking if object type job with name accessdomain-create-oam-infra-domain-
job exists
No resources found in oamns namespace.
Creating the domain by creating the job output/weblogic-domains/
accessdomain/create-domain-job.yaml
job.batch/accessdomain-create-oam-infra-domain-job created
Waiting for the job to complete...
status on iteration 1 of 20
pod accessdomain-create-oam-infra-domain-job-6tgw4 status is Running
status on iteration 2 of 20
pod accessdomain-create-oam-infra-domain-job-6tgw4 status is Running
status on iteration 3 of 20
pod accessdomain-create-oam-infra-domain-job-6tgw4 status is Running
status on iteration 4 of 20
pod accessdomain-create-oam-infra-domain-job-6tgw4 status is Running
status on iteration 5 of 20
pod accessdomain-create-oam-infra-domain-job-6tgw4 status is Running
status on iteration 6 of 20
pod accessdomain-create-oam-infra-domain-job-6tgw4 status is Completed

Domain accessdomain was created and will be started by the WebLogic
Kubernetes Operator

The following files were generated:
  output/weblogic-domains/accessdomain/create-domain-inputs.yaml
  output/weblogic-domains/accessdomain/create-domain-job.yaml
  output/weblogic-domains/accessdomain/domain.yaml
```

> **Note**
>
> If the domain creation fails, refer to **Domain Creation Failure With WLST** in Known Issues.

7.1.7 Setting the OAM Server Memory Parameters

By default, the java memory parameters assigned to the `oam-cluster` are very small. The minimum recommended values are `-Xms4096m -Xmx8192m`. However, Oracle recommends you to set these to `-Xms8192m -Xmx8192m` in a production environment.

1. Navigate to the `/output/weblogic-domains/<domain_uid>` directory:

   ```
   cd $WORKDIR/kubernetes/create-access-domain/domain-home-on-pv/output/
   weblogic-domains/<domain_uid>
   ```

   For example:

   ```
   cd $WORKDIR/kubernetes/create-access-domain/domain-home-on-pv/output/
   weblogic-domains/accessdomain
   ```

2. Take a backup of the `domain.yaml`:

   ```
   cp domain.yaml domain.yaml.orig
   ```

3. Edit the `domain.yaml` file and inside name: `accessdomain-oam-cluster`, add the memory setting as below:

   ```
     serverPod:
       env:
       - name: USER_MEM_ARGS
         value: "-XX:+UseContainerSupport -Djava.security.egd=file:/dev/./
   urandom -Xms8192m -Xmx8192m"
       resources:
        limits:
         cpu: "2"
         memory: "8Gi"
        requests:
         cpu: "1000m"
         memory: "4Gi"
   ```

   For example:

   ```
   apiVersion: weblogic.oracle/v1
   kind: Cluster
   metadata:
     name: accessdomain-oam-cluster
     namespace: oamns
   spec:
     clusterName: oam_cluster
     serverService:
       precreateService: true
     serverPod:
   ```

```
    env:
    - name: USER_MEM_ARGS
      value: "-XX:+UseContainerSupport -Djava.security.egd=file:/dev/./
urandom -Xms8192m -Xmx8192m"
    resources:
     limits:
      cpu: "2"
      memory: "8Gi"
     requests:
      cpu: "1000m"
      memory: "4Gi"
  replicas: 1
```

> **Note**
>
> Administrators should be aware of the following:
>
> - The above CPU and memory values are for examples only. For Enterprise Deployments, please review the performance recommendations and sizing requirements in Enterprise Deployment Guide for Oracle Identity and Access Management in a Kubernetes Cluster .
> - Limits and requests for CPU resources are measured in CPU units. One CPU in Kubernetes is equivalent to 1 vCPU/Core for cloud providers, and 1 hyperthread on bare-metal Intel processors. An “m” suffix in a CPU attribute indicates ‘milli-CPU’, so 500m is 50% of a CPU. Memory can be expressed in various units, where one Mi is one IEC unit mega-byte (1024^2), and one Gi is one IEC unit giga-byte (1024^3). For more information, see Resource Management for Pods and Containers, Assign Memory Resources to Containers and Pods, and Assign CPU Resources to Containers and Pods
> - The parameters above are also utilized by the Kubernetes Horizontal Pod Autoscaler (HPA). For more details on HPA, see Kubernetes Horizontal Pod Autoscaler.
> - If required you can also set the same resources and limits for the `accessdomain-policy-cluster`.

4. In the `domain.yaml` locate the section of the file starting with `adminServer:`. Under the `env:` tag add the following `CLASSPATH` entries. This is required for running the `idmconfigtool` from the Administration Server:

```
- name: CLASSPATH
  value: "/u01/oracle/wlserver/server/lib/weblogic.jar"
```

For example:

```
  # adminServer is used to configure the desired behavior for starting the
administration server.
  adminServer:
    # adminService:
    #   channels:
    # The Admin Server's NodePort
    #    - channelName: default
    #      nodePort: 30701
```

```
    # Uncomment to export the T3Channel as a service
    #    - channelName: T3Channel
    serverPod:
      # an (optional) list of environment variable to be set on the admin
servers
      env:
      - name: USER_MEM_ARGS
        value: "-Djava.security.egd=file:/dev/./urandom -Xms512m -Xmx1024m
"
      - name: CLASSPATH
        value: "/u01/oracle/wlserver/server/lib/weblogic.jar"
```

5. If required, you can add the optional parameter `maxClusterConcurrentStartup` to the spec section of the `domain.yaml`. This parameter specifies the number of managed servers to be started in sequence per cluster.

   For example,if you updated the `initialManagedServerReplicas` to `4` in `create-domain-inputs.yaml` and only had 2 nodes, then setting `maxClusterConcurrentStartup: 1` will start one managed server at a time on each node, rather than starting them all at once. This can be useful to take the strain off individual nodes at startup.

   Below is an example with the parameter added:

   ```
   apiVersion: "weblogic.oracle/v9"
   kind: Domain
   metadata:
     name: accessdomain
     namespace: oamns
     labels:
       weblogic.domainUID: accessdomain
   spec:
     # The WebLogic Domain Home
     domainHome: /u01/oracle/user_projects/domains/accessdomain
     maxClusterConcurrentStartup: 1

     # The domain home source type
     # Set to PersistentVolume for domain-in-pv, Image for domain-in-image,
   or FromModel for model-in-image
     domainHomeSourceType: PersistentVolume
     ....
   ```

6. Save the changes to `domain.yaml`

7.1.8 Deploying the WLST OAM Domain

Deploy the Oracle Access Management (OAM) domain using the `domain.yaml`.

1. Run the following command to deploy the OAM domain:

   ```
   kubectl apply -f $WORKDIR/kubernetes/create-access-domain/domain-home-on-
   pv/output/weblogic-domains/<domain_uid>/domain.yaml
   ```

For example:

```
kubectl apply -f $WORKDIR/kubernetes/create-access-domain/domain-home-on-
pv/output/weblogic-domains/accessdomain/domain.yaml
```

The output will look similar to the following:

```
domain.weblogic.oracle/accessdomain created
cluster.weblogic.oracle/accessdomain-oam-cluster created
cluster.weblogic.oracle/accessdomain-policy-cluster created
```

7.1.9 Verifying the WLST OAM Deployment

Verifying the domain, pods and services

Verify the domain, servers pods and services are created and in the `READY` state with a status of `1/1`, by running the following command:

```
kubectl get all,domains -n <domain_namespace>
```

For example:

```
kubectl get all,domains -n oamns
```

The output will look similar to the following:

```
NAME                                                READY   STATUS
RESTARTS   AGE
pod/accessdomain-adminserver                        1/1     Running
0          11m
pod/accessdomain-create-oam-infra-domain-job-7c9r9   0/1     Completed
0          18m
pod/accessdomain-oam-policy-mgr1                    1/1     Running
0          3m31s
pod/accessdomain-oam-server1                        1/1     Running
0          3m31s

NAME                                          TYPE        CLUSTER-IP
EXTERNAL-IP   PORT(S)     AGE
service/accessdomain-adminserver              ClusterIP   None
<none>        7001/TCP    11m
service/accessdomain-cluster-oam-cluster      ClusterIP   10.101.59.154
<none>        14100/TCP   3m31s
service/accessdomain-cluster-policy-cluster   ClusterIP   10.98.236.51
<none>        15100/TCP   3m31s
service/accessdomain-oam-policy-mgr1          ClusterIP   None
<none>        15100/TCP   3m31s
service/accessdomain-oam-policy-mgr2          ClusterIP   10.104.92.12
<none>        15100/TCP   3m31s
service/accessdomain-oam-policy-mgr3          ClusterIP   10.96.244.37
<none>        15100/TCP   3m31s
service/accessdomain-oam-policy-mgr4          ClusterIP   10.105.201.23
<none>        15100/TCP   3m31s
```

```
service/accessdomain-oam-policy-mgr5          ClusterIP   10.110.12.227
<none>        15100/TCP   3m31s
service/accessdomain-oam-server1              ClusterIP   None
<none>        14100/TCP   3m31s
service/accessdomain-oam-server2              ClusterIP   10.96.137.33
<none>        14100/TCP   3m31s
service/accessdomain-oam-server3              ClusterIP   10.103.178.35
<none>        14100/TCP   3m31s
service/accessdomain-oam-server4              ClusterIP   10.97.254.78
<none>        14100/TCP   3m31s
service/accessdomain-oam-server5              ClusterIP   10.105.65.104
<none>        14100/TCP   3m31s

NAME                                                    COMPLETIONS   DURATION
AGE
job.batch/accessdomain-create-oam-infra-domain-job   1/1           2m6s
18m

NAME                                    AGE
domain.weblogic.oracle/accessdomain   12m

NAME                                                     AGE
cluster.weblogic.oracle/accessdomain-oam-cluster      11m
cluster.weblogic.oracle/accessdomain-policy-cluster   11m
```

> **Note**
>
> It will take several minutes before all the services listed above show. When a pod has a STATUS of 0/1 the pod is started but the OAM server associated with it is currently starting. While the pods are starting you can check the startup status in the pod logs, by running the following command:
>
> ```
> kubectl logs <pod> -n <namespace>
> ```
>
> For example:
>
> ```
> kubectl logs accessdomain-adminserver -n accessdomain-adminserver
> ```

The default domain created by the script has the following characteristics:

- An Administration Server named `AdminServer` listening on port `7001`.
- A configured OAM cluster named `oam_cluster` of size `5`.
- A configured Policy Manager cluster named `policy_cluster` of size `5`.
- One started OAM Managed Server, named `oam_server1`, listening on port `14100`.
- One started Policy Manager Managed Servers named `oam-policy-mgr1`, listening on port `15100`.
- Log files that are located in `<persistent_volume>/logs/<domainUID>`.

Verifying the Domain

Run the following command to describe the domain:

```
kubectl describe domain <domain_uid> -n <domain_namespace>
```

For example:

```
kubectl describe domain accessdomain -n oamns
```

The output will look similar to the following:

```
Name:         accessdomain
Namespace:    oamns
Labels:       weblogic.domainUID=accessdomain
Annotations:  <none>
API Version:  weblogic.oracle/v9
Kind:         Domain
Metadata:
  Creation Timestamp:  <DATE>
  Generation:          1
  Managed Fields:
    API Version:  weblogic.oracle/v9
    Fields Type:  FieldsV1
    fieldsV1:
      f:metadata:
        f:annotations:
          .:
          f:kubectl.kubernetes.io/last-applied-configuration:
        f:labels:
          .:
          f:weblogic.domainUID:
      f:spec:
        .:
        f:adminServer:
          .:
          f:adminChannelPortForwardingEnabled:
          f:serverPod:
            .:
            f:env:
          f:serverStartPolicy:
        f:clusters:
        f:dataHome:
        f:domainHome:
        f:domainHomeSourceType:
        f:failureRetryIntervalSeconds:
        f:failureRetryLimitMinutes:
        f:httpAccessLogInLogHome:
        f:image:
        f:imagePullPolicy:
        f:imagePullSecrets:
        f:includeServerOutInPodLog:
        f:logHome:
        f:logHomeEnabled:
```

```
          f:logHomeLayout:
          f:maxClusterConcurrentShutdown:
          f:maxClusterConcurrentStartup:
          f:maxClusterUnavailable:
          f:replicas:
          f:serverPod:
            .:
            f:env:
            f:volumeMounts:
            f:volumes:
          f:serverStartPolicy:
          f:webLogicCredentialsSecret:
            .:
            f:name:
    Manager:      kubectl-client-side-apply
    Operation:    Update
    Time:         <DATE>
    API Version:  weblogic.oracle/v9
    Fields Type:  FieldsV1
    fieldsV1:
      f:status:
        .:
        f:clusters:
        f:conditions:
        f:observedGeneration:
        f:servers:
        f:startTime:
    Manager:         Kubernetes Java Client
    Operation:       Update
    Subresource:     status
    Time:            <DATE>
  Resource Version:  2074089
  UID:               e194d483-7383-4359-adb9-bf97de36518b
Spec:
  Admin Server:
    Admin Channel Port Forwarding Enabled:  true
    Server Pod:
      Env:
        Name:             USER_MEM_ARGS
        Value:            -Djava.security.egd=file:/dev/./urandom -Xms512m -
Xmx1024m
        Name:             CLASSPATH
        Value:            /u01/oracle/wlserver/server/lib/weblogic.jar
    Server Start Policy:  IfNeeded
  Clusters:
    Name:                          accessdomain-oam-cluster
    Name:                          accessdomain-policy-cluster
  Data Home:
  Domain Home:                     /u01/oracle/user_projects/domains/
accessdomain
  Domain Home Source Type:         PersistentVolume
  Failure Retry Interval Seconds:  120
  Failure Retry Limit Minutes:     1440
  Http Access Log In Log Home:     true
  Image:                           container-registry.oracle.com/middleware/
oam_cpu:12.2.1.4-jdk8-ol8-<January'25>
```

```
  Image Pull Policy:              IfNotPresent
  Image Pull Secrets:
    Name:                          orclcred
  Include Server Out In Pod Log:   true
  Log Home:                        /u01/oracle/user_projects/domains/logs/
accessdomain
  Log Home Enabled:                true
  Log Home Layout:                 ByServers
  Max Cluster Concurrent Shutdown: 1
  Max Cluster Concurrent Startup:  0
  Max Cluster Unavailable:         1
  Replicas:                        1
  Server Pod:
    Env:
      Name:   JAVA_OPTIONS
      Value:  -Dweblogic.StdoutDebugEnabled=false
      Name:   USER_MEM_ARGS
      Value:  -Djava.security.egd=file:/dev/./urandom -Xms256m -Xmx1024m
    Volume Mounts:
      Mount Path:  /u01/oracle/user_projects/domains
      Name:        weblogic-domain-storage-volume
    Volumes:
      Name:  weblogic-domain-storage-volume
      Persistent Volume Claim:
        Claim Name:     accessdomain-domain-pvc
  Server Start Policy:  IfNeeded
  Web Logic Credentials Secret:
    Name:  accessdomain-credentials
Status:
  Clusters:
    Cluster Name:  oam_cluster
    Conditions:
      Last Transition Time:  <DATE>
      Status:                True
      Type:                  Available
      Last Transition Time:  <DATE>
      Status:                True
      Type:                  Completed
    Label Selector:
weblogic.domainUID=accessdomain,weblogic.clusterName=oam_cluster
    Maximum Replicas:        5
    Minimum Replicas:        0
    Observed Generation:     1
    Ready Replicas:          1
    Replicas:                1
    Replicas Goal:           1
    Cluster Name:            policy_cluster
    Conditions:
      Last Transition Time:  <DATE>
      Status:                True
      Type:                  Available
      Last Transition Time:  <DATE>
      Status:                True
      Type:                  Completed
    Label Selector:
weblogic.domainUID=accessdomain,weblogic.clusterName=policy_cluster
```

```
  Maximum Replicas:        5
  Minimum Replicas:        0
  Observed Generation:     1
  Ready Replicas:          1
  Replicas:                1
  Replicas Goal:           1
Conditions:
  Last Transition Time:  <DATE>
  Status:                True
  Type:                  Available
  Last Transition Time:  <DATE>
  Status:                True
  Type:                  Completed
Observed Generation:     1
Servers:
  Health:
    Activation Time:  <DATE>
    Overall Health:   ok
    Subsystems:
      Subsystem Name:  ServerRuntime
      Symptoms:
  Node Name:     worker-node2
  Pod Phase:     Running
  Pod Ready:     True
  Server Name:   AdminServer
  State:         RUNNING
  State Goal:    RUNNING
  Cluster Name:  oam_cluster
  Health:
    Activation Time:  <DATE>
    Overall Health:   ok
    Subsystems:
      Subsystem Name:  ServerRuntime
     Symptoms:
  Node Name:     worker-node1
  Pod Phase:     Running
  Pod Ready:     True
  Server Name:   oam_server1
  State:         RUNNING
  State Goal:    RUNNING
  Cluster Name:  oam_cluster
  Server Name:   oam_server2
  State:         SHUTDOWN
  State Goal:    SHUTDOWN
  Cluster Name:  oam_cluster
  Server Name:   oam_server3
  State:         SHUTDOWN
  State Goal:    SHUTDOWN
  Cluster Name:  oam_cluster
  Server Name:   oam_server4
  State:         SHUTDOWN
  State Goal:    SHUTDOWN
  Cluster Name:  oam_cluster
  Server Name:   oam_server5
  State:         SHUTDOWN
  State Goal:    SHUTDOWN
```

```
    Cluster Name:  policy_cluster
    Health:
      Activation Time:  <DATE>
      Overall Health:   ok
      Subsystems:
        Subsystem Name:  ServerRuntime
        Symptoms:
    Node Name:     worker-node1
    Pod Phase:     Running
    Pod Ready:     True
    Server Name:   oam_policy_mgr1
    State:         RUNNING
    State Goal:    RUNNING
    Cluster Name:  policy_cluster
    Server Name:   oam_policy_mgr2
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
    Cluster Name:  policy_cluster
    Server Name:   oam_policy_mgr3
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
    Cluster Name:  policy_cluster
    Server Name:   oam_policy_mgr4
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
    Cluster Name:  policy_cluster
    Server Name:   oam_policy_mgr5
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
  Start Time:      <DATE>
Events:
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Created    15m    weblogic.operator  Domain accessdomain was
created.
  Normal  Available  2m56s  weblogic.operator  Domain accessdomain is
available: a sufficient number of its servers have reached the ready state.
  Normal  Completed  2m56s  weblogic.operator  Domain accessdomain is
complete because all of the following are true: there is no failure detected,
there are no pending server shutdowns, and all servers expected to be running
are ready and at their target image, auxiliary images, restart version, and
introspect version.
```

In the `Status` section of the output, the available servers and clusters are listed.

Verifying the Pods

Run the following command to view the pods and the nodes they are running on:

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

For example:

```
kubectl get pods -n oamns -o wide
```

The output will look similar to the following:

```
NAME                                                READY   STATUS
RESTARTS   AGE     IP            NODE             NOMINATED NODE   READINESS
GATES
accessdomain-adminserver                              1/1     Running
0          18m   10.244.6.63   10.250.42.252   <none>           <none>
accessdomain-create-oam-infra-domain-job-7c9r9   0/1     Completed
0          25m   10.244.6.61   10.250.42.252   <none>           <none>
accessdomain-oam-policy-mgr1                          1/1     Running
0          10m   10.244.5.13   10.250.42.255   <none>           <none>
accessdomain-oam-server1                              1/1     Running
0          10m   10.244.5.12   10.250.42.255   <none>           <none>
```

Configuring the Ingress

If the domain deploys successfully, and all the above checks are verified, you are ready to configure the Ingress. See, Configuring NGINX.

7.2 Creating OAM Domains Using WDT Models

Using WDT models, all the required information is specified in the domain custom resource YAML file. With WDT models, the WebLogic Kubernetes Operator will create the RCU schemas, create the persistent volume and claim, then create the WebLogic domain on the persistent volume, prior to starting the servers.

In this section a domain creation image is built using the supplied model files and that image is used for domain creation. You will need your own container registry to upload the domain image to. Having your own container repository is a prerequisite before creating an OAM domain with WDT models. If you don't have your own container registry, you can load the image on each node in the cluster instead. This documentation does not explain how to create your own container registry, or how to load the image onto each node. Consult your vendor specific documentation for more information.

Building a domain creation image is a one time activity. The domain creation image can be used to create an OAM domain in multiple environments. You do not need to rebuild the domain creation image every time you create a domain.

Before following this section, make sure you have followed Preparing Your Environment, and ensure your Oracle Database is running.

This section includes the following topics:

- Creating a Kubernetes Secret for the WDT Domain
- Creating a Kubernetes Secret for RCU in WDT
- Preparing the WDT Create Domain YAML File
- Creating the WDT YAML files
- Building the Domain Creation Image
- Deploying the WDT OAM Domain
- Verifying the WDT OAM Deployment

7.2.1 Creating a Kubernetes Secret for the WDT Domain

Create a Kubernetes secret for the Oracle Access Management (OAM) domain using the `create-secret.sh` script.

1. Navigate to the `wdt-utils` directory:

   ```
   cd $WORKDIR/kubernetes/create-access-domain/domain-home-on-pv/wdt-utils
   ```

2. Run the following command to create the secret:

   ```
   ./create-secret.sh -l \
   "username=weblogic" \
   -l "password=<password>" \
   -n <domain_namespace> \
   -d <domain_uid> \
   -s <domain-uid>-weblogic-credentials
   ```

   Where:

   - `<password>` is the password for the WebLogic user.
   - `<domain_namespace>` is the domain namespace for OAM.
   - `<domain_uid>` is the domain UID to be created.
   - `<domain-uid>-weblogic-credentials` is the name you want to create for the secret for this namespace.

   > **Note**
   >
   > The secret name must follow the format `<domain-uid>-weblogic-credentials` or domain creation will fail.

   For example:

   ```
   ./create-secret.sh -l \
   "username=weblogic" \
   -l "password=<password>" \
   -n oamns \
   -d accessdomain \
   -s accessdomain-weblogic-credentials
   ```

   The output will look similar to the following:

   ```
   @@ Info: Setting up secret 'accessdomain-weblogic-credentials'.
   secret/accessdomain-weblogic-credentials created
   secret/accessdomain-weblogic-credentials labeled
   ```

3. Verify the secret is created using the following command:

   ```
   kubectl get secret <kubernetes_domain_secret> -o yaml -n <domain_namespace>
   ```

For example:

```
kubectl get secret accessdomain-weblogic-credentials -o yaml -n oamns
```

The output will look similar to the following:

```
apiVersion: v1
data:
  password: <password>
  username: d2VibG9naWM=
kind: Secret
metadata:
  creationTimestamp: "<DATE>"
  labels:
    weblogic.domainUID: accessdomain
  name: accessdomain-weblogic-credentials
  namespace: oamns
  resourceVersion: "44175245"
  uid: a135780e-6f3b-4be1-8643-f81bfb9ba399
type: Opaque
```

7.2.2 Creating a Kubernetes Secret for RCU in WDT

Create a Kubernetes secret for RCU using the `create-secret.sh` script.

1. Navigate to the `wdt-utils` directory:

   ```
   cd $WORKDIR/kubernetes/create-access-domain/domain-home-on-pv/wdt-utils
   ```

2. Run the following command to create the secret:

   ```
   ./create-secret.sh -l "rcu_prefix=<rcu_prefix>" \
   -l "rcu_schema_password=<rcu_schema_pwd>" \
   -l "db_host=<db_host.domain>" \
   -l "db_port=<db_port>" \
   -l "db_service=<service_name>" \
   -l "dba_user=<sys_db_user>" \
   -l "dba_password=<sys_db_pwd>" \
   -n <domain_namespace> \
   -d <domain_uid> \
   -s <domain_uid>-rcu-credentials
   ```

   Where:

   - `<rcu_prefix>` is the name of the RCU schema prefix to be created.
   - `<rcu_schema_pwd>` is the password you want to create for the RCU schema prefix.
   - `<db_host.domain>` is the hostname.domain of the database.
   - `<db_port>` is the database listener port.
   - `<service_name>` is the service name of the database.
   - `<sys_db_user>` is the database user with SYSDBA privilege.
   - `<sys_db_pwd>` is the SYS database password.

- <domain_uid>is the domain_uid that you want to create. This must be the same domain_uid used in Creating a Kubernetes Secret for the WDT Domain.
- <domain_namespace> is the OAM domain namespace.
- <domain_uid>-rcu-credentials is the name you want to create for the RCU secret for this namespace.

> **Note**
>
> The secret name must follow the format <domain_uid>-rcu-credentials or domain creation will fail.

For example:

```
./create-secret.sh -l "rcu_prefix=OAMK8S" \
-l "rcu_schema_password=<password>" \
-l "db_host=mydatabasehost.example.com" \
-l "db_port=1521" \
-l "db_service=orcl.example.com" \
-l "dba_user=sys" \
-l "dba_password=<password>" \
-n oamns \
-d accessdomain \
-s accessdomain-rcu-credentials
```

The output will look similar to the following:

```
@@ Info: Setting up secret 'accessdomain-rcu-credentials'.
secret/accessdomain-rcu-credentials created
secret/accessdomain-rcu-credentials labeled
```

3. Verify the secret is created using the following command:

```
kubectl get secret <kubernetes_rcu_secret> -o yaml -n <domain_namespace>
```

For example:

```
kubectl get secrets -n oamns accessdomain-rcu-credentials -o yaml
```

The output will look similar to the following:

```
apiVersion: v1
data:
  db_host: <DB_HOST>
  db_port: MTUyMQ==
  db_service: <SERVICE_NAME>
  dba_password: <PASSWORD>
  dba_user: c3lz
  rcu_prefix: <RCU_PREFIX>
  rcu_schema_password: <RCU_PWD>
kind: Secret
metadata:
  creationTimestamp: "<DATE>"
```

```
  labels:
    weblogic.domainUID: accessdomain
  name: accessdomain-rcu-credentials
  namespace: oamns
  resourceVersion: "866948"
  uid: b5e3b4e0-9458-4413-a6ff-874e9af7511b
type: Opaque
```

7.2.3 Preparing the WDT Create Domain YAML File

Prepare the `create-domain-wdt.yaml` file by running the following commands:

1. Navigate to the `$WORKDIR/kubernetes/create-access-domain/domain-home-on-pv/wdt-utils/generate_models_utils` directory:

   ```
   cd $WORKDIR/kubernetes/create-access-domain/domain-home-on-pv/wdt-utils/
   generate_models_utils
   ```

2. Make a copy of the `create-domain-wdt.yaml` file:

   ```
   cp create-domain-wdt.yaml create-domain-wdt.yaml.orig
   ```

3. Edit the `create-domain-wdt.yaml` and modify the following parameters. Save the file when complete:

   ```
   appVersion: 14c
   domainUID: <domain_uid>
   domainHome: /u01/oracle/user_projects/domains/<domain_uid>
   image: <image_name>:<tag>
   imagePullSecretName: <container_registry_secret>
   logHome: /u01/oracle/user_projects/domains/logs/<domain_uid>
   namespace: <domain_namespace>
   weblogicDomainStorageType: NFS
   weblogicDomainStorageNFSServer: <nfs_server>
   weblogicDomainStoragePath: <physical_path_of_persistent_storage>
   weblogicDomainStorageSize: 10Gi
   ```

   For example:

   ```
   appVersion: 14c
   domainUID: accessdomain
   domainHome: /u01/oracle/user_projects/domains/accessdomain
   image: container-registry.oracle.com/middleware/oam_cpu:14.1.2.1.0-jdk17-
   ol8-<YYMMDD>
   imagePullSecretName: orclcred
   logHome: /u01/oracle/user_projects/domains/logs/accessdomain
   namespace: oamns
   weblogicDomainStorageType: NFS
   weblogicDomainStorageNFSServer: mynfsserver
   weblogicDomainStoragePath: /nfs_volumes/oam/accessdomainpv
   weblogicDomainStorageSize: 10Gi
   ```

> **Note**
>
> If using a shared file system instead of NFS, set `weblogicDomainStorageType: HOST_PATH` and remove `weblogicDomainStorageNFSServer`.

A full list of parameters in the `create-domain-wdt.yaml` file are shown below:

Parameter	Definition	Default
adminPort	Port number for the Administration Server inside the Kubernetes cluster.	7001
adminNodePort	Port number of the Administration Server outside the Kubernetes cluster.	30701
configuredManagedServerCount	Number of Managed Server instances to generate for the domain.	5
datasourceType	Type of JDBC datasource applicable for the OAM domain. Legal values are `agl` and `generic`. Choose `agl` for Active GridLink datasource and `generic` for Generic datasource. For enterprise deployments, Oracle recommends that you use GridLink data sources to connect to Oracle RAC databases. See the Preparing an Existing Database for an Enterprise Deployment for further details.	generic
domainHome	Home directory of the OAM domain. If not specified, the value is derived from the `domainUID` as `/shared/domains/<domainUID>`.	/u01/oracle/user_projects/domains/accessdomain
domainPVMountPath	Mount path of the domain persistent volume.	/u01/oracle/user_projects/domains
domainUID	Unique ID that will be used to identify this particular domain. Used as the name of the generated WebLogic domain as well as the name of the Kubernetes domain resource. This ID must be unique across all domains in a Kubernetes cluster. This ID cannot contain any character that is not valid in a Kubernetes service name.	accessdomain
edgInstall	Used only if performing an install using the Enterprise Deployment Guide. See, Enterprise Deployment Guide for Oracle Identity and Access Management in a Kubernetes Cluster	false

Parameter	Definition	Default
exposeAdminNodePort	Boolean indicating if the Administration Server is exposed outside of the Kubernetes cluster.	false
exposeAdminT3Channel	Boolean indicating if the T3 administrative channel is exposed outside the Kubernetes cluster.	true
image	OAM container image. The operator requires OAM 14.1.2. Refer to Obtaining the OAM Container image for details on how to obtain or create the image.	oracle/oam:14.1.2.1.0
imagePullSecretName	Name of the Kubernetes secret to access the container registry to pull the OAM container image. The presence of the secret will be validated when this parameter is specified.	orclcred
initialManagedServerReplicas	Number of Managed Servers to initially start for the domain.	2
javaOptions	Java options for starting the Administration Server and Managed Servers. A Java option can have references to one or more of the following pre-defined variables to obtain WebLogic domain information: `$(DOMAIN_NAME)`, `$(DOMAIN_HOME)`, `$(ADMIN_NAME)`, `$(ADMIN_PORT)`, and `$(SERVER_NAME)`.	-Dweblogic.StdoutDebugEnabled=false
logHome	The in-pod location for the domain log, server logs, server out, and Node Manager log files. If not specified, the value is derived from the `domainUID` as `/shared/logs/<domainUID>`.	/u01/oracle/user_projects/domains/logs/accessdomain
namespace	Kubernetes namespace in which to create the domain.	oamns
oamCPU	Initial CPU Units, 1000m = 1 CPU core.	1000m
oamMaxCPU	Max CPU a pod is allowed to consume.	2
oamMemory	Initial memory allocated to a pod.	4Gi
oamMaxMemory	Max memory a pod is allowed to consume.	8Gi
oamServerJavaParams	The memory parameters to use for the OAM managed servers.	"-Xms8192m -Xmx8192m"
productionModeEnabled	Boolean indicating if production mode is enabled for the domain.	true

Parameter	Definition	Default
t3PublicAddress	Public address for the T3 channel. This should be set to the public address of the Kubernetes cluster. This would typically be a load balancer address. For development environments only: In a single server (all-in-one) Kubernetes deployment, this may be set to the address of the master, or at the very least, it must be set to the address of one of the worker nodes.	If not provided, the script will attempt to set it to the IP address of the Kubernetes cluster
weblogicDomainStorageType	Persistent volume storage type. Options are `NFS` for `NFS` volumes or `HOST_PATH` for shared file system.	NFS
weblogicDomainStorageNFSServer	Hostname or IP address of the NFS Server.	nfsServer
weblogicDomainStoragePath	Physical path to the persistent volume.	/scratch/accessdomainpv
weblogicDomainStorageSize	Total storage allocated to the persistent storage.	10Gi

> **Note**
>
> The above CPU and memory values are for examples only. For Enterprise Deployments, please review the performance recommendations and sizing requirements in Enterprise Deployment Guide for Oracle Identity and Access Management in a Kubernetes Cluster.

7.2.4 Creating the WDT YAML files

Generate the required WDT models for the OAM domain, along with the domain resource yaml files.

1. Navigate to the `$WORKDIR/kubernetes/create-access-domain/domain-home-on-pv/wdt-utils/generate_models_utils`

   ```
   cd $WORKDIR/kubernetes/create-access-domain/domain-home-on-pv/wdt-utils/generate_models_utils
   ```

2. Run the `generate_wdt_models.sh`, specifying your input file and an output directory to store the generated artifacts:

   ```
   ./generate_wdt_models.sh -i create-domain-wdt.yaml -o
   <path_to_output_directory>
   ```

   For example:

   ```
   ./generate_wdt_models.sh -i create-domain-wdt.yaml -o output
   ```

The output will look similar to the following:

```
input parameters being used
export version="create-weblogic-sample-domain-inputs-v1"
export appVersion="14c"
export adminPort="7001"
export domainUID="accessdomain"
export configuredManagedServerCount="5"
export initialManagedServerReplicas="1"
export productionModeEnabled="true"
export t3ChannelPort="30012"
export datasourceType="generic"
export edgInstall="false"
export domainHome="/u01/oracle/user_projects/domains/accessdomain"
export image="container-registry.oracle.com/middleware/oam_cpu:14.1.2.1.0-
jdk17-ol8-<YYMMDD>"
export imagePullSecretName="orclcred"
export logHome="/u01/oracle/user_projects/domains/logs/accessdomain"
export exposeAdminT3Channel="false"
export adminNodePort="30701"
export exposeAdminNodePort="false"
export namespace="oamns"
javaOptions=-Dweblogic.StdoutDebugEnabled=false
export domainPVMountPath="/u01/oracle/user_projects"
export weblogicDomainStorageType="NFS"
export weblogicDomainStorageNFSServer="mynfsServer"
export weblogicDomainStoragePath="/nfs_volumes/oam/accessdomainpv"
export weblogicDomainStorageReclaimPolicy="Retain"
export weblogicDomainStorageSize="10Gi"
export oamServerJavaParams="-Xms8192m -Xmx8192m"
export oamMaxCPU="2"
export oamCPU="1000m"
export oamMaxMemory="8Gi"
export oamMemory="4Gi"

validateWlsDomainName called with accessdomain
WDT model file, property file and sample domain.yaml are genereted
successfully at output/weblogic-domains/accessdomain
```

> **Note**
>
> This will generate the `domain.yaml`, `oam.yaml` and `oam.properties` in `output/weblogic-domains/accessdomain`.

3. Copy the generated files to a `$WORKDIR/yaml` directory:

```
mkdir $WORKDIR/yaml


cp output/weblogic-domains/accessdomain/*.* $WORKDIR/yaml
```

7.2.5 Building the Domain Creation Image

You must build a domain creation image to host the WebLogic Deploy Tooling (WDT) model files and (WDT) installer.

Domain creation images are used for deploying a domain using a Domain on PV model. The domain creation image contains:

- WDT model files
- WDT variables files
- WDT application archive files (collectively known as WDT model files)
- The directory where the WebLogic Deploy Tooling software is installed (known as the WDT Home),

You distribute WDT model files and the WDT executable using these images, and the WebLogic Kubernetes Operator uses them to manage the domain.

> **Note**
>
> These images are only used for creating the domain and will not be used to update the domain. The domain creation image is used for domain creation only, it is not the product container image used for Oracle Access Management (OAM).

The steps to build the domain creation image are shown in the sections below.

Prerequisites

Verify that your environment meets the following prerequisites:

- You have created the domain YAML files are per Creating the WDT YAML files.
- A container image client on the build machine, such as Docker or Podman:
  - For Docker, a minimum version of 18.03.1.ce is required.
  - For Podman, a minimum version of 3.0.1 is required.
- An installed version of JDK to run Image Tool, version 8+.
- Proxies are set accordingly at the OS level if required.

Preparing the Build Domain Image Script

1. Navigate to the `$WORKDIR/kubernetes/create-access-domain/domain-home-on-pv/wdt-utils/build-domain-creation-image/properties` directory:

   ```
   cd $WORKDIR/kubernetes/create-access-domain/domain-home-on-pv/wdt-utils/
   build-domain-creation-image/properties
   ```

2. Make a copy of the `build-domain-creation-image.properties`:

   ```
   cp build-domain-creation-image.properties build-domain-creation-
   image.properties.orig
   ```

3. Edit the `build-domain-creation-image.properties` and modify the following parameters. Save the file when complete:

```
JAVA_HOME=<Java home location>
IMAGE_TAG=<Image tag name>
REPOSITORY= <Container image repository to push the image>
REG_USER= <Container registry username>
IMAGE_PUSH_REQUIRES_AUTH=<Whether image push requires authentication to
the registry>
WDT_MODEL_FILE=<Full Path to WDT Model file oam.yaml>
WDT_VARIABLE_FILE=<Full path to WDT variable file oam.properties>
WDT_ARCHIVE_FILE=<Full Path to WDT Archive file>
WDT_VERSION="Version of WebLogic Deploy Tool version to use"
WIT_VERSION="Version of WebLogic Image Tool to use"
```

For example:

```
JAVA_HOME=/scratch/jdk
IMAGE_TAG=oam-aux-generic-v1
BASE_IMAGE=ghcr.io/oracle/oraclelinux:8-slim
REPOSITORY=container-registry.example.com/mytenancy/idm
REG_USER=mytenancy/myemail@example.com
IMAGE_PUSH_REQUIRES_AUTH=true
WDT_MODEL_FILE="/OAMK8S/fmw-kubernetes/OracleAccessManagement/yaml/
oam.yaml"
WDT_VARIABLE_FILE="/OAMK8S/fmw-kubernetes/OracleAccessManagement/yaml/
oam.properties"
WDT_ARCHIVE_FILE=""
WDT_VERSION="4.2.0"
WIT_VERSION="1.14.3"
```

A full list of parameters and descriptions in the `build-domain-creation-image.properties` file are shown below:

Parameter	Definition	Default
JAVA_HOME	Path to the JAVA_HOME for the JDK8+.	/scratch/jdk/jdk1.8.0_351
IMAGE_TAG	Image tag for the final domain creation image.	oam-aux-generic-v1
BASE_IMAGE	The Oracle Linux product container image to use as a base image.	ghcr.io/oracle/oraclelinux:8-slim
REPOSITORY	Container image repository that will host the domain creation image.	iad.ocir.io/mytenancy/idm
REG_USER	Username to authenticate to the <REGISTRY> and push the domain creation image.	mytenancy/ oracleidentitycloudservice/ myemail@example.com
IMAGE_PUSH_REQUIRES_AUTH	If authentication to <REGISTRY> is required then set to true, else set to false. If set to false, <REG_USER> is not required.	true

Parameter	Definition	Default
WDT_MODEL_FILE	Absolute path to WDT model file oam.yaml. For example `$WORKDIR/yaml/oam.yaml`.	/scratch/model/oam.yaml
WDT_MODEL_FILE	Absolute path to WDT variable file `oam.properties`. For example `$WORKDIR/yaml/oam.properties`.	/scratch/model/oam.properties
WDT_ARCHIVE_FILE	Absolute path to WDT archive file.	
WDT_VERSION	WebLogic Deploy Tool version. If not specified the latest available version will be downloaded. It is recommended to use the default value.	4.2.0
WIT_VERSION	WebLogic Image Tool Version. If not specified the latest available version will be downloaded. It is recommended to use the default value.	1.14.3
TARGET	Select the target environment in which the created image will be used. Supported values: Default or OpenShift. See Additional Information.	Default
CHOWN	userid:groupid to be used for creating files within the image, such as the WDT installer, WDT model, and WDT archive. If the user or group does not exist in the image, they will be added with useradd/groupadd.	oracle:oracle

Note

If `IMAGE_PUSH_REQUIRES_AUTH=true`, you must edit the `$WORKDIR/kubernetes/create-access-domain/domain-home-on-pv/wdt-utils/build-domain-creation-image/properties/.regpassword` and change `<REGISTRY_PASSWORD>` to your registry password:

```
REG_PASSWORD="<REGISTRY_PASSWORD>"
```

Running the build-domain-creation-image Script

1. Navigate to the `$WORKDIR/kubernetes/create-access-domain/domain-home-on-pv/wdt-utils/build-domain-creation-image` directory:

   ```
   cd $WORKDIR/kubernetes/create-access-domain/domain-home-on-pv/wdt-utils/
   build-domain-creation-image
   ```

2. Execute the `build-domain-creation-image.sh` by specifying the input properties parameter files. Executing this command will build the image and push it to the container registry :

```
./build-domain-creation-image.sh -i properties/build-domain-creation-
image.properties
```

> **Note**
>
> Administrators should be aware of the following:
>
> - If using a password file, you must add the following to the end of the command:
>
>   ```
>   -p properties/.regpassword
>   ```
>
> - You can use the same domain creation image to create a domain in multiple environments, based on your need. You do not need to rebuild it every time during domain creation. This is a one time activity.

The output will look similar to the following:

```
using WDT_DIR: /OAMK8S/fmw-kubernetes/OracleAccessManagement/kubernetes/
create-access-domain/domain-home-on-pv/wdt-utils/build-domain-creation-
image/workdir
Using WDT_VERSION 4.2.0
Using WIT_DIR /OAMK8S/fmw-kubernetes/OracleAccessManagement/kubernetes/
create-access-domain/domain-home-on-pv/wdt-utils/build-domain-creation-
image/workdir
Using WIT_VERSION 1.14.3
Using Image tag: oam-aux-generic-v1
using Base Image: ghcr.io/oracle/oraclelinux:8-slim
using IMAGE_BUILDER_EXE /usr/bin/podman
JAVA_HOME is set to /scratch/jdk
@@  Info: WIT_INSTALL_ZIP_URL is ''
@@ WIT_INSTALL_ZIP_URL is not set
@@ imagetool.sh not found in /OAMK8S/fmw-kubernetes/OracleAccessManagement/
kubernetes/create-access-domain/domain-home-on-pv/wdt-utils/build-domain-
creation-image/workdir/imagetool/bin. Installing imagetool...
@@ Info:  Downloading https://github.com/oracle/weblogic-image-tool/
releases/download/release-1.14.3/imagetool.zip
@@ Info:  Downloading https://github.com/oracle/weblogic-image-tool/
releases/download/release-1.14.3/imagetool.zip with https_proxy="http://
proxy.example.com:80"
@@ Info: Archive downloaded to /OAMK8S/fmw-kubernetes/
OracleAccessManagement/kubernetes/create-access-domain/domain-home-on-pv/
wdt-utils/build-domain-creation-image/workdir/imagetool.zip, about to
unzip via '/home/opc/jdk/bin/jar xf'.
@@ Info: imageTool cache does not contain a valid entry for wdt_4.2.0.
Installing WDT
@@  Info: WDT_INSTALL_ZIP_URL is ''
@@ WDT_INSTALL_ZIP_URL is not set
@@ Info:  Downloading https://github.com/oracle/weblogic-deploy-tooling/
```

```
releases/download/release-4.2.0/weblogic-deploy.zip
@@ Info:  Downloading https://github.com/oracle/weblogic-deploy-tooling/
releases/download/release-4.2.0/weblogic-deploy.zip with
https_proxy="http://proxy.example.com:80"
@@ Info: Archive downloaded to /OAMK8S/fmw-kubernetes/
OracleAccessManagement/kubernetes/create-access-domain/domain-home-on-pv/
wdt-utils/build-domain-creation-image/workdir/weblogic-deploy.zip
[INFO   ] Successfully added to cache. wdt_4.2.0=/OAMK8S/fmw-kubernetes/
OracleAccessManagement/kubernetes/create-access-domain/domain-home-on-pv/
wdt-utils/build-domain-creation-image/workdir/weblogic-deploy.zip
@@ Info: Install succeeded, imagetool install is in the /OAMK8S/fmw-
kubernetes/OracleAccessManagement/kubernetes/create-access-domain/domain-
home-on-pv/wdt-utils/build-domain-creation-image/workdir/imagetool
directory.
Starting Building Image registry.example.com/mytenancy/idm:oam-aux-generic-
v1
 Login Succeeded!
WDT_MODEL_FILE is set to /OAMK8S/fmw-kubernetes/OracleAccessManagement/
yaml/oam.yaml
WDT_VARIABLE_FILE is set to /OAMK8S/fmw-kubernetes/OracleAccessManagement/
yaml/oam.properties
Additional Build Commands file is set to /OAMK8S/fmw-kubernetes/
OracleAccessManagement/kubernetes/create-access-domain/domain-home-on-pv/
wdt-utils/build-domain-creation-image/additonal-build-files/build-files.txt
Additonal Build file is set to /OAMK8S/fmw-kubernetes/
OracleAccessManagement/kubernetes/create-access-domain/domain-home-on-pv/
wdt-utils/build-domain-creation-image/additonal-build-files/OAM.json
[INFO   ] WebLogic Image Tool version 1.14.3
[INFO   ] Image Tool build ID: 0c9aa58f-808b-4707-a11a-7766fb301cbb
[INFO   ] Temporary directory used for image build context: /home/oracle/
wlsimgbuilder_temp1198331326550546381
[INFO   ] Copying /OAMK8S/fmw-kubernetes/OracleAccessManagement/kubernetes/
create-access-domain/domain-home-on-pv/wdt-utils/build-domain-creation-
image/additonal-build-files/OAM.json to build context folder.
[INFO   ] User specified fromImage ghcr.io/oracle/oraclelinux:8-slim
[INFO   ] Inspecting ghcr.io/oracle/oraclelinux:8-slim, this may take a
few minutes if the image is not available locally.
[INFO   ] Copying /OAMK8S/fmw-kubernetes/OracleAccessManagement/yaml/
oam.yaml to build context folder.
[INFO   ] Copying /OAMK8S/fmw-kubernetes/OracleAccessManagement/yaml/
oam.properties to build context folder.
[INFO   ] Copying /OAMK8S/fmw-kubernetes/OracleAccessManagement/kubernetes/
create-access-domain/domain-home-on-pv/wdt-utils/build-domain-creation-
image/workdir/weblogic-deploy.zip to build context folder.
[INFO   ] Starting build: /usr/bin/podman build --no-cache --force-rm --
tag registry.example.com/mytenancy/idm:oam-aux-generic-v1 --pull --build-
arg http_proxy=http://proxy.example.com:80 --build-arg https_proxy=http://
proxy.example.com:80 --build-arg
no_proxy=localhost,127.0.0.0/8,.example.com,,/var/run/crio/
crio.sock,X.X.X.X /home/oracle/wlsimgbuilder_temp1198331326550546381
[1/3] STEP 1/5: FROM ghcr.io/oracle/oraclelinux:8-slim AS os_update
[1/3] STEP 2/5: LABEL
com.oracle.weblogic.imagetool.buildid="0c9aa58f-808b-4707-
a11a-7766fb301cbb"
--> ba91c351bf94
[1/3] STEP 3/5: USER root
```

```
--> d8f89c65892a
[1/3] STEP 4/5: RUN microdnf update     && microdnf install gzip tar unzip
libaio libnsl jq findutils diffutils shadow-utils     && microdnf clean all
Downloading metadata...
Downloading metadata...
Package                                          Repository            Size
Upgrading:
 libgcc-8.5.0-20.0.3.el8.x86_64                  ol8_baseos_latest  93.4 kB
  replacing libgcc-8.5.0-20.0.2.el8.x86_64
 libstdc++-8.5.0-20.0.3.el8.x86_64               ol8_baseos_latest 474.6 kB
  replacing libstdc++-8.5.0-20.0.2.el8.x86_64
 systemd-libs-239-78.0.4.el8.x86_64              ol8_baseos_latest   1.2 MB
   replacing systemd-libs-239-78.0.3.el8.x86_64
Transaction Summary:
 Installing:        0 packages
 Reinstalling:      0 packages
 Upgrading:         3 packages
 Obsoleting:        0 packages
 Removing:          0 packages
 Downgrading:       0 packages
Downloading packages...
Running transaction test...
Updating: libgcc;8.5.0-20.0.3.el8;x86_64;ol8_baseos_latest
Updating: libstdc++;8.5.0-20.0.3.el8;x86_64;ol8_baseos_latest
Updating: systemd-libs;239-78.0.4.el8;x86_64;ol8_baseos_latest
Cleanup: libstdc++;8.5.0-20.0.2.el8;x86_64;installed
Cleanup: systemd-libs;239-78.0.3.el8;x86_64;installed
Cleanup: libgcc;8.5.0-20.0.2.el8;x86_64;installed
Complete.
Package                            Repository            Size
Installing:
 diffutils-3.6-6.el8.x86_64        ol8_baseos_latest 369.3 kB
 findutils-1:4.6.0-21.el8.x86_64   ol8_baseos_latest 539.8 kB
 gzip-1.9-13.el8_5.x86_64          ol8_baseos_latest 170.7 kB
 jq-1.6-7.0.3.el8.x86_64           ol8_appstream     206.5 kB
 libaio-0.3.112-1.el8.x86_64       ol8_baseos_latest  33.4 kB
 libnsl-2.28-236.0.1.el8.7.x86_64  ol8_baseos_latest 111.4 kB
 oniguruma-6.8.2-2.1.el8_9.x86_64  ol8_appstream     191.5 kB
 unzip-6.0-46.0.1.el8.x86_64       ol8_baseos_latest 201.0 kB
Transaction Summary:
Installing:        8 packages
 Reinstalling:      0 packages
 Upgrading:         0 packages
 Obsoleting:        0 packages
 Removing:          0 packages
 Downgrading:       0 packages
Downloading packages...
Running transaction test...
Installing: oniguruma;6.8.2-2.1.el8_9;x86_64;ol8_appstream
Installing: jq;1.6-7.0.3.el8;x86_64;ol8_appstream
Installing: unzip;6.0-46.0.1.el8;x86_64;ol8_baseos_latest
Installing: libnsl;2.28-236.0.1.el8.7;x86_64;ol8_baseos_latest
Installing: libaio;0.3.112-1.el8;x86_64;ol8_baseos_latest
Installing: gzip;1.9-13.el8_5;x86_64;ol8_baseos_latest
Installing: findutils;1:4.6.0-21.el8;x86_64;ol8_baseos_latest
Installing: diffutils;3.6-6.el8;x86_64;ol8_baseos_latest
```

```
Complete.
Complete.
--> 73fb79fa40b2
[1/3] STEP 5/5: RUN if [ -z "$(getent group oracle)" ]; then groupadd
oracle || exit 1 ; fi  && if [ -z "$(getent group oracle)" ]; then
groupadd oracle || exit 1 ; fi  && if [ -z "$(getent passwd oracle)" ];
then useradd -g oracle oracle || exit 1; fi  && mkdir -p /u01  && chown
oracle:oracle /u01  && chmod 775 /u01
--> ff6cf74351d1
[2/3] STEP 1/4: FROM
ff6cf74351d1e0124121321174eaa64ebefa0bc3eef80ec88caec12feb9e8fb3 AS
wdt_build
[2/3] STEP 2/4: RUN mkdir -p /auxiliary  && mkdir -p /auxiliary/models  &&
chown oracle:oracle /auxiliary
--> a061b678fa0a
[2/3] STEP 3/4: COPY --chown=oracle:oracle ["weblogic-deploy.zip", "/tmp/
imagetool/"]
--> 3daccfef2f06
[2/3] STEP 4/4: RUN test -d /auxiliary/weblogic-deploy && rm -rf /
auxiliary/weblogic-deploy || echo Initial WDT install         && unzip -q
"/tmp/imagetool/weblogic-deploy.zip" -d /auxiliary
Initial WDT install
--> b77b02f66a83
[3/3] STEP 1/12: FROM
ff6cf74351d1e0124121321174eaa64ebefa0bc3eef80ec88caec12feb9e8fb3 AS final
[3/3] STEP 2/12: ENV AUXILIARY_IMAGE_PATH=/auxiliary     WDT_HOME=/
auxiliary     WDT_MODEL_HOME=/auxiliary/models
--> 10dc1832266f
[3/3] STEP 3/12: RUN mkdir -p /auxiliary && chown oracle:oracle /auxiliary
--> 0b85f8e7399a
[3/3] STEP 4/12: COPY --from=wdt_build --chown=oracle:oracle /auxiliary /
auxiliary/
--> c64bf2bef430
[3/3] STEP 5/12: RUN mkdir -p /auxiliary/models && chown oracle:oracle /
auxiliary/models
--> d8817f84ab58
[3/3] STEP 6/12: COPY --chown=oracle:oracle ["oam.yaml", "/auxiliary/
models/"]
--> 45b1d25264b9
[3/3] STEP 7/12: COPY --chown=oracle:oracle ["oam.properties", "/auxiliary/
models/"]
--> 2ceba77ee226
[3/3] STEP 8/12: RUN chmod -R 640 /auxiliary/models/*
--> 34385bac7974
[3/3] STEP 9/12: USER oracle
--> 409f6e3ccce4
[3/3] STEP 10/12: WORKDIR /auxiliary
--> aaa2f154f512
[3/3] STEP 11/12: COPY --chown=oracle:oracle files/OAM.json /auxiliary/
weblogic-deploy/lib/typedefs
--> c8a9d29106d3
[3/3] STEP 12/12: RUN chmod -R 755 /auxiliary
[3/3] COMMIT registry.example.com/mytenancy/idm:oam-aux-generic-v1
--> 0797418499a1
Successfully tagged registry.example.com/mytenancy/idm:oam-aux-generic-v1
0797418499a1dfd6d2a28672948c17ed747291ad069cebca5fac1b0410978d75
```

```
[INFO   ] Build successful. Build time=72s. Image tag=registry.example.com/
mytenancy/idm:oam-aux-generic-v1
Getting image source signatures
 Copying blob 462ffb36555c done
Copying blob 3db4d3748983 done
Copying blob 7e9f3f6c7a0a done
Copying blob 32aa5f13e19b done
Copying blob d979da323f64 done
Copying blob f18b9e5f415f done
Copying blob aaaea7c1392f done
Copying blob 5504fa641a87 done
Copying blob 5aa81493c602 done
Copying blob f56f992ba90d done
Copying blob 2b1e0644fbd3 done
Copying config a39dc6ae7f done
Writing manifest to image destination
Pushed image registry.example.com/mytenancy/idm/oam-aux-generic-v1 to
image registry Docker Hub
```

7.2.6 Deploying the WDT OAM Domain

You must modify the Oracle Access Management (OAM) `domain.yaml` and deploy the OAM domain using the build image created.

Modify the OAM domain.yaml

1. Edit the `$WORKDIR/yaml/domain.yaml` and update the `%DOMAIN_CREATION_IMAGE%` with the previously generated image name:

> **Note**
>
> `%DOMAIN_CREATION_IMAGE%` takes the format of `<REPOSITORY>:<TAG>`.

```
domain:
         # Domain | DomainAndRCU
         createIfNotExists: DomainAndRCU
         # Image containing WDT installer and Model files.
         domainCreationImages:
             - image: '%DOMAIN_CREATION_IMAGE%'
         domainType: OAM
```

For example:

```
domain:
        # Domain | DomainAndRCU
        createIfNotExists: DomainAndRCU
        # Image containing WDT installer and Model files.
        domainCreationImages:
            - image: 'container-registry.example.com/mytenancy/idm:oam-aux-
generic-v1'
        domainType: OAM
```

2. In circumstances where you may be pulling the OAM product container image from Oracle Container Registry, and then the domain image from a private registry, you must first create a secret (`privatecred`) for the private registry. For example:

```
kubectl create secret docker-registry "privatecred" --docker-
server=container-registry.example.com \
--docker-username="user@example.com" \
--docker-password=password --docker-email=user@example.com \
--namespace=oamns
```

Then specify both secrets for `imagePullSecrets` in the `domain.yaml`. For example:

```
 ...
spec:
  # The WebLogic Domain Home
  domainHome: /u01/oracle/user_projects/domains/accessdomain

  # The domain home source type
  # Set to PersistentVolume for domain-in-pv, Image for domain-in-image,
or FromModel for model-in-image
  domainHomeSourceType: PersistentVolume

  # The WebLogic Server image that the Operator uses to start the domain
  image: "container-registry.oracle.com/middleware/oam_cpu:14.1.2.1.0-
jdk17-ol8-<YYMMDD>"

  # imagePullPolicy defaults to "Always" if image version is :latest
  imagePullPolicy: IfNotPresent

  imagePullSecrets:
  - name: orclcred
  - name: privatecred
  # Identify which Secret contains the WebLogic Admin credentials
...
```

For more information about the configuration parameters in `domain.yaml`, see Domain Resources.

A sample `domain.yaml` is shown below:

```
# Copyright (c) 2024, Oracle and/or its affiliates.
# Licensed under the Universal Permissive License v 1.0 as shown at https://
oss.oracle.com/licenses/upl.
#
# This is an example of how to define an OAM Domain. For details about the
fields in domain specification, refer https://oracle.github.io/weblogic-
kubernetes-operator/managing-domains/domain-resource/
#
apiVersion: "weblogic.oracle/v9"
kind: Domain
metadata:
  name: accessdomain
  namespace: oamns
  labels:
    weblogic.domainUID: accessdomain
```

```
spec:
  # The WebLogic Domain Home
  domainHome: /u01/oracle/user_projects/domains/accessdomain

  # The domain home source type
  # Set to PersistentVolume for domain-in-pv, Image for domain-in-image, or
FromModel for model-in-image
  domainHomeSourceType: PersistentVolume

  # The WebLogic Server image that the Operator uses to start the domain
  image: "container-registry.oracle.com/middleware/oam_cpu:14.1.2.1.0-jdk17-
ol8-<YYMMDD>"

  # imagePullPolicy defaults to "Always" if image version is :latest
  imagePullPolicy: IfNotPresent

  # Add additional secret name if you are using a different registry for
domain creation image.
  # Identify which Secret contains the credentials for pulling an image
  imagePullSecrets:
  - name: orclcred
  - name: privatecred
  # Identify which Secret contains the WebLogic Admin credentials
  webLogicCredentialsSecret:
    name: accessdomain-weblogic-credentials

  # Whether to include the server out file into the pod's stdout, default is
true
  includeServerOutInPodLog: true

  # Whether to enable log home
  logHomeEnabled: true

  # Whether to write HTTP access log file to log home
  httpAccessLogInLogHome: true

  # The in-pod location for domain log, server logs, server out, introspector
out, and Node Manager log files
  logHome: /u01/oracle/user_projects/domains/logs/accessdomain
  # An (optional) in-pod location for data storage of default and custom file
stores.
  # If not specified or the value is either not set or empty (e.g. dataHome:
"") then the
  # data storage directories are determined from the WebLogic domain home
configuration.
  dataHome: ""

  # serverStartPolicy legal values are "Never, "IfNeeded", or "AdminOnly"
  # This determines which WebLogic Servers the Operator will start up when it
discovers this Domain
  # - "Never" will not start any server in the domain
  # - "AdminOnly" will start up only the administration server (no managed
servers will be started)
  # - "IfNeeded" will start all non-clustered servers, including the
administration server and clustered servers up to the replica count
  serverStartPolicy: IfNeeded
```

```
  serverPod:
    initContainers:
      #DO NOT CHANGE THE NAME OF THIS INIT CONTAINER
      - name: compat-connector-init
        # OAM Product image, same as spec.image mentioned above
        image: "container-registry.oracle.com/middleware/oam_cpu:14.1.2.1.0-
jdk17-ol8-<YYMMDD>"
        imagePullPolicy: IfNotPresent
        command: [ "/bin/bash", "-c", "mkdir -p  /u01/oracle/user_projects/
domains/wdt-logs"]
        volumeMounts:
          - mountPath: /u01/oracle/user_projects
            name: weblogic-domain-storage-volume
    # a mandatory list of environment variable to be set on the servers
    env:
    - name: JAVA_OPTIONS
      value: -Dweblogic.StdoutDebugEnabled=false
    - name: USER_MEM_ARGS
      value: "-Djava.security.egd=file:/dev/./urandom -Xms256m -Xmx1024m "
    - name: WLSDEPLOY_LOG_DIRECTORY
      value: "/u01/oracle/user_projects/domains/wdt-logs"
    - name: WLSDEPLOY_PROPERTIES
      value: "-Dwdt.config.disable.rcu.drop.schema=true"
    volumes:
    - name: weblogic-domain-storage-volume
      persistentVolumeClaim:
        claimName: accessdomain-domain-pvc
    volumeMounts:
    - mountPath: /u01/oracle/user_projects
      name: weblogic-domain-storage-volume

  # adminServer is used to configure the desired behavior for starting the
administration server.
  adminServer:
    # adminService:
    #   channels:
    # The Admin Server's NodePort
    #    - channelName: default
    #      nodePort: 30701
    # Uncomment to export the T3Channel as a service
    #    - channelName: T3Channel
    serverPod:
      # an (optional) list of environment variable to be set on the admin
servers
      env:
      - name: USER_MEM_ARGS
        value: "-Djava.security.egd=file:/dev/./urandom -Xms512m -Xmx1024m "
      - name: CLASSPATH
        value: "/u01/oracle/wlserver/server/lib/weblogic.jar"

  configuration:
      secrets: [ accessdomain-rcu-credentials ]
      initializeDomainOnPV:
        persistentVolume:
          metadata:
```

```
            name: accessdomain-domain-pv
          spec:
            storageClassName: accessdomain-domain-storage-class
            capacity:
          # Total storage allocated to the persistent storage.
              storage: 10Gi
          # Reclaim policy of the persistent storage
          # # The valid values are: 'Retain', 'Delete', and 'Recycle'
            persistentVolumeReclaimPolicy: Retain
          # Persistent volume type for the persistent storage.
          # # The value must be 'hostPath' or 'nfs'.
          # # If using 'nfs', server must be specified.
           nfs:
              server: mynfserver
            # hostPath:
              path: "/nfs_volumes/oam/accessdomainpv"
        persistentVolumeClaim:
          metadata:
              name: accessdomain-domain-pvc
          spec:
            storageClassName: accessdomain-domain-storage-class
            resources:
                requests:
                    storage: 10Gi
            volumeName: accessdomain-domain-pv
        domain:
            # Domain | DomainAndRCU
            createIfNotExists: DomainAndRCU
            # Image containing WDT installer and Model files.
            domainCreationImages:
                - image: 'container-registry.example.com/mytenancy/idm:oam-
aux-generic-v1'
            domainType: OAM
  # References to Cluster resources that describe the lifecycle options for
all
  # the Managed Server members of a WebLogic cluster, including Java
  # options, environment variables, additional Pod content, and the ability to
  # explicitly start, stop, or restart cluster members. The Cluster resource
  # must describe a cluster that already exists in the WebLogic domain
  # configuration.
  clusters:
  - name: accessdomain-oam-cluster
  - name: accessdomain-policy-cluster

  # The number of managed servers to start for unlisted clusters
  # replicas: 1

---
# This is an example of how to define a Cluster resource.
apiVersion: weblogic.oracle/v1
kind: Cluster
metadata:
  name: accessdomain-oam-cluster
  namespace: oamns
spec:
  clusterName: oam_cluster
```

```
  serverService:
    precreateService: true
  replicas: 1
  serverPod:
  serverPod:
    env:
    - name: USER_MEM_ARGS
      value: "-XX:+UseContainerSupport -Djava.security.egd=file:/dev/./
urandom -Xms8192m -Xmx8192m "
    resources:
      limits:
        cpu: "2"
        memory: "8Gi"
      requests:
        cpu: "1000m"
        memory: "4Gi"


---
# This is an example of how to define a Cluster resource.
apiVersion: weblogic.oracle/v1
kind: Cluster
metadata:
  name: accessdomain-policy-cluster
  namespace: oamns
spec:
  clusterName: policy_cluster
  serverService:
    precreateService: true
  replicas: 1
```

Optional WDT Models ConfigMap

If required, you can provide a Kubernetes ConfigMap with additional WDT models and WDT variables files as supplements, or overrides, to those in `domainCreationImages`.

For example in the `output/weblogic-domains/accessdomain/domain.yaml`:

```
      domain:
          ...
          domainCreationImages:
              ...
          domainCreationConfigMap: mymodel-domain-configmap
```

The files inside `domainCreationConfigMap` must have file extensions, `.yaml`, `.properties`, or `.zip`.

To create a configmap run the following commands:

```
cd $WORKDIR/kubernetes/create-access-domain/domain-home-on-pv/wdt-utils


./create-configmap.sh -n oamns -d accessdomain -c mymodel-domain-configmap -f
wdt_models/mymodel.yaml
```

For more information on the usage of additional configuration, see Optional WDT models ConfigMap.

Deploying the OAM Domain

Deploy the OAM domain using the `domain.yaml`:

1. Run the following command to create OAM domain resources:

   ```
   kubectl create -f $WORKDIR/yaml/domain.yaml
   ```

   The following steps will be performed by WebLogic Kubernetes Operator:

   - Run the introspector job.
   - The introspection job will create the RCU Schemas.
   - The introspector job pod will create the domain on PV using the model provided in the domain creation image.
   - The introspector job pod will execute OAM offline configuration actions post successful creation of domain via WDT.
   - Brings up the Administration Server, OAM Managed Server (`oam_server1`), and the OAM Policy Managed Server (`oam_policy_mgr1`).

   The output will look similar to the following:

   ```
   domain.weblogic.oracle/accessdomain created
   cluster.weblogic.oracle/accessdomain-oam-cluster created
   cluster.weblogic.oracle/accessdomain-policy-cluster created
   ```

   Whilst the domain creation is running, you can run the following command to monitor the progress:

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

   > **Note**
   >
   > The `-w` flag allows you watch the status of the pods as they change.

   For example:

   ```
   kubectl get pods -n oamns -w
   ```

   You can also tail the logs for the pods by running:

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

   > **Note**
   >
   > WDT specific logs can be found in `<persistent_volume>/domains/wdt-logs`.

2. Once everything is started, you should see the Administration Server and OAM servers are running:

```
NAME                             READY   STATUS    RESTARTS        AGE
accessdomain-adminserver         1/1     Running   0               11m
accessdomain-oam-policy-mgr1     1/1     Running   0               3m53s
accessdomain-oam-server1         1/1     Running   0               3m53s
```

If there are any failures, follow **Domain Creation Failure with WDT Models** in Known Issues.

7.2.7 Verifying the WDT OAM Deployment

Verifying the Domain, Pods and Services

Verify the domain, servers pods and services are created and in the `READY` state with a status of `1/1`, by running the following command:

```
kubectl get all,domains -n <domain_namespace>
```

For example:

```
kubectl get all,domains -n oamns
```

The output will look similar to the following:

```
NAME                                 READY   STATUS    RESTARTS        AGE
pod/accessdomain-adminserver         1/1     Running   0               12m
pod/accessdomain-oam-policy-mgr1     1/1     Running   0               4m19s
pod/accessdomain-oam-server1         1/1     Running   0               4m19s

NAME                                          TYPE        CLUSTER-IP
EXTERNAL-IP   PORT(S)     AGE
service/accessdomain-adminserver              ClusterIP   None
<none>        7001/TCP    12m
service/accessdomain-cluster-oam-cluster      ClusterIP   10.104.17.83
<none>        14100/TCP   4m19s
service/accessdomain-cluster-policy-cluster   ClusterIP   10.98.157.157
<none>        15100/TCP   4m19s
service/accessdomain-oam-policy-mgr1          ClusterIP   None
<none>        15100/TCP   4m19s
service/accessdomain-oam-policy-mgr2          ClusterIP   10.101.141.238
<none>        15100/TCP   4m19s
service/accessdomain-oam-policy-mgr3          ClusterIP   10.107.167.143
<none>        15100/TCP   4m19s
service/accessdomain-oam-policy-mgr4          ClusterIP   10.106.100.191
<none>        15100/TCP   4m19s
service/accessdomain-oam-policy-mgr5          ClusterIP   10.105.5.126
<none>        15100/TCP   4m19s
service/accessdomain-oam-server1              ClusterIP   None
<none>        14100/TCP   4m19s
service/accessdomain-oam-server2              ClusterIP   10.98.248.74
<none>        14100/TCP   4m19s
```

```
service/accessdomain-oam-server3              ClusterIP   10.106.224.54
<none>        14100/TCP   4m19s
service/accessdomain-oam-server4              ClusterIP   10.104.241.109
<none>        14100/TCP   4m19s
service/accessdomain-oam-server5              ClusterIP   10.96.189.205
<none>        14100/TCP   4m19s

NAME                                    AGE
domain.weblogic.oracle/accessdomain   18m

NAME                                                    AGE
cluster.weblogic.oracle/accessdomain-oam-cluster      18m
cluster.weblogic.oracle/accessdomain-policy-cluster   18m
```

The default domain created by the script has the following characteristics:

- An Administration Server named `AdminServer` listening on port `7001`.
- A configured OAM cluster named `oam_cluster` of size `5`.
- A configured Policy Manager cluster named `policy_cluster` of size `5`.
- One started OAM Managed Server, named `oam_server1`, listening on port `14100`.
- One started Policy Manager Managed Servers named `oam-policy-mgr1`, listening on port `15100`.
- Log files that are located in `<persistent_volume>/logs/<domainUID>`.

If the OAM deployment fails refer to **Domain Creation Failure with WDT Models** in Known Issues.

Verifying the Domain

Run the following command to describe the domain:

```
kubectl describe domain <domain_uid> -n <domain_namespace>
```

For example:

```
kubectl describe domain accessdomain -n oamns
```

The output will look similar to the following:

```
Name:         accessdomain
Namespace:    oamns
Labels:       weblogic.domainUID=accessdomain
Annotations:  <none>
API Version:  weblogic.oracle/v9
Kind:         Domain
Metadata:
  Creation Timestamp:  <DATE>
  Generation:          1
  Resource Version:    2930591
  UID:                 7eafcfd3-f0f3-436d-a1f0-87c31f042d62
Spec:
  Admin Server:
    Admin Channel Port Forwarding Enabled:  true
```

```
    Server Pod:
      Env:
        Name:             USER_MEM_ARGS
        Value:            -Djava.security.egd=file:/dev/./urandom -Xms512m -
Xmx1024m
        Name:             CLASSPATH
        Value:            /u01/oracle/wlserver/server/lib/weblogic.jar
    Server Start Policy:  IfNeeded
  Clusters:
    Name:  accessdomain-oam-cluster
    Name:  accessdomain-policy-cluster
  Configuration:
    Initialize Domain On PV:
      Domain:
        Create If Not Exists:  DomainAndRCU
        Domain Creation Images:
          Image:      container-registry.example.com/mytenancy/idm:oam-aux-
generic-v1
        Domain Type:  OAM
      Persistent Volume:
        Metadata:
          Name:  accessdomain-domain-pv
        Spec:
          Capacity:
            Storage:  10Gi
          Nfs:
            Path:                              /<NFS_PATH>/accessdomainpv
            Server:                            <NFS_SERVER>
          Persistent Volume Reclaim Policy:  Retain
          Storage Class Name:                accessdomain-domain-storage-class
      Persistent Volume Claim:
        Metadata:
          Name:  accessdomain-domain-pvc
        Spec:
          Resources:
            Requests:
              Storage:               10Gi
          Storage Class Name:        accessdomain-domain-storage-class
          Volume Name:               accessdomain-domain-pv
    Override Distribution Strategy:  Dynamic
    Secrets:
      accessdomain-rcu-credentials
  Data Home:
  Domain Home:                    /u01/oracle/user_projects/domains/
accessdomain
  Domain Home Source Type:        PersistentVolume
  Failure Retry Interval Seconds:  120
  Failure Retry Limit Minutes:    1440
  Http Access Log In Log Home:    true
  Image:                          container-registry.oracle.com/middleware/
oam_cpu:14.1.2.1.0-jdk17-ol8-<DDMMYY>
  Image Pull Policy:              IfNotPresent
  Image Pull Secrets:
    Name:                           orclcred
  Include Server Out In Pod Log:    true
  Log Home:                         /u01/oracle/user_projects/domains/logs/
```

```
accessdomain
  Log Home Enabled:                   true
  Max Cluster Concurrent Shutdown:    1
  Max Cluster Concurrent Startup:     0
  Max Cluster Unavailable:            1
  Replace Variables In Java Options:  false
  Replicas:                           1
  Server Pod:
    Env:
      Name:   JAVA_OPTIONS
      Value:  -Dweblogic.StdoutDebugEnabled=false
      Name:   USER_MEM_ARGS
      Value:  -Djava.security.egd=file:/dev/./urandom -Xms256m -Xmx1024m
      Name:   WLSDEPLOY_LOG_DIRECTORY
      Value:  /u01/oracle/user_projects/domains/wdt-logs
      Name:   WLSDEPLOY_PROPERTIES
      Value:  -Dwdt.config.disable.rcu.drop.schema=true
    Init Containers:
      Command:
        /bin/bash
        -c
        mkdir -p  /u01/oracle/user_projects/domains/wdt-logs
      Image:              container-registry.oracle.com/middleware/
oam_cpu:14.1.2.1.0-jdk17-ol8-<DDMMYY>
      Image Pull Policy:  IfNotPresent
      Name:               compat-connector-init
      Volume Mounts:
        Mount Path:  /u01/oracle/user_projects
        Name:        weblogic-domain-storage-volume
    Volume Mounts:
      Mount Path:  /u01/oracle/user_projects
      Name:        weblogic-domain-storage-volume
    Volumes:
      Name:  weblogic-domain-storage-volume
      Persistent Volume Claim:
        Claim Name:     accessdomain-domain-pvc
  Server Start Policy:  IfNeeded
  Web Logic Credentials Secret:
    Name:  accessdomain-weblogic-credentials
Status:
  Clusters:
    Cluster Name:  oam_cluster
    Conditions:
      Last Transition Time:  <DATE>
      Status:                False
      Type:                  Available
      Last Transition Time:  <DATE>
      Status:                False
      Type:                  Completed
    Label Selector:
weblogic.domainUID=accessdomain,weblogic.clusterName=oam_cluster
    Maximum Replicas:        5
    Minimum Replicas:        0
    Observed Generation:     1
    Replicas:                1
    Replicas Goal:           1
```

```
    Cluster Name:            policy_cluster
    Conditions:
      Last Transition Time:  <DATE>
      Status:                False
      Type:                  Available
      Last Transition Time:  <DATE>
      Status:                False
      Type:                  Completed
    Label Selector:
weblogic.domainUID=accessdomain,weblogic.clusterName=policy_cluster
    Maximum Replicas:        5
    Minimum Replicas:        0
    Observed Generation:     1
    Replicas:                1
    Replicas Goal:           1
  Conditions:
    Last Transition Time:  <DATE>
    Message:               No application servers are ready.
    Status:                False
    Type:                  Available
    Last Transition Time:  <DATE>
    Status:                False
    Type:                  Completed
  Observed Generation:     1
  Servers:
    Health:
      Activation Time:  <DATE>
      Overall Health:   ok
      Subsystems:
        Subsystem Name:  ServerRuntime
        Symptoms:
    Node Name:      worker-node1
    Pod Phase:     Running
    Pod Ready:     True
    Server Name:   AdminServer
    State:         RUNNING
    State Goal:    RUNNING
    Cluster Name:  oam_cluster
    Node Name:      worker-node2
    Pod Phase:     Running
    Pod Ready:     False
    Server Name:   oam_server1
    State:         STARTING
    State Goal:    SHUTDOWN
    Cluster Name:  oam_cluster
    Server Name:   oam_server2
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
    Cluster Name:  oam_cluster
    Server Name:   oam_server3
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
    Cluster Name:  oam_cluster
    Server Name:   oam_server4
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
```

```
    Cluster Name:  oam_cluster
    Server Name:   oam_server5
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
    Cluster Name:  policy_cluster
    Node Name:      worker-node1
    Pod Phase:     Running
    Pod Ready:     False
    Server Name:   oam_policy_mgr1
    State:         STARTING
    State Goal:    SHUTDOWN
    Cluster Name:  policy_cluster
    Server Name:   oam_policy_mgr2
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
    Cluster Name:  policy_cluster
    Server Name:   oam_policy_mgr3
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
    Cluster Name:  policy_cluster
    Server Name:   oam_policy_mgr4
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
    Cluster Name:  policy_cluster
    Server Name:   oam_policy_mgr5
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
  Start Time:      <DATE>

 Events:
   Type     Reason                     Age                From
Message
   ----     ------                     ----               ----
-------
   Normal   Created                    19m                weblogic.operator
Domain accessdomain was created.
   Warning  Failed                     19m                weblogic.operator
Domain accessdomain failed due to 'Persistent volume claim unbound':
PersistentVolumeClaim 'accessdomain-domain-pvc' is not bound; the status
phase is 'Pending'.. Operator is waiting for the persistent volume claim to
be bound, it may be a temporary condition. If this condition persists, then
ensure that the PVC has a correct volume name or storage class name and is in
bound status..
   Normal   PersistentVolumeClaimBound  19m                weblogic.operator
The persistent volume claim is bound and ready.
   Normal   Available                  3m19s              weblogic.operator
Domain accessdomain is available: a sufficient number of its servers have
reached the ready state.
   Normal   Completed                  3m19s              weblogic.operator
Domain accessdomain is complete because all of the following are true: there
is no failure detected, there are no pending server shutdowns, and all
servers expected to be running are ready and at their target image, auxiliary
images, restart version, and introspect version.
```

In the `Status` section of the output, the available servers and clusters are listed.

Verifying the Pods

Run the following command to view the pods and the nodes they are running on:

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

For example:

```
kubectl get pods -n oamns -o wide
```

The output will look similar to the following:

```
NAME                                          READY   STATUS
RESTARTS   AGE     IP           NODE             NOMINATED NODE   READINESS
GATES
accessdomain-adminserver                       1/1     Running
0          18m   10.244.6.63   10.250.42.252   <none>           <none>
accessdomain-oam-policy-mgr1                   1/1     Running
0          10m   10.244.5.13   10.250.42.255   <none>           <none>
accessdomain-oam-server1                       1/1     Running
0          10m   10.244.5.12   10.250.42.255   <none>           <none>
```

Configuring the Ingress

If the domain deploys successfully, and all the above checks are verified, you are ready to configure the Ingress. See, Configuring NGINX.

8

Setting Up a Load Balancer

The WebLogic Kubernetes Operator supports ingress-based load balancers such as Traefik and NGINX (kubernetes/ingress-nginx).

- Configuring Traefik
  Configure the ingress-based Traefik load balancer for Oracle Access Management domains
- Configuring NGINX
  Configure the ingress-based NGINX load balancer for Oracle Access Management domains.

8.1 Configuring Traefik

You must configure an ingress controller to allow access to Oracle Access Management (OAM).

The ingress can be configured in the following ways:

- Without SSL
- With SSL
- OAM URI's are accessible from all hosts
- OAM URI's are accessible using virtual hostnames only

The option you choose will depend on the architecture you are configuring. For example, if you have an architecture such as Oracle HTTP Server on an Independent Kubernetes cluster, where SSL is terminated at the load balancer, then you would configure the ingress without SSL.

In almost all circumstances, the ingress should be configured to be accessible from all hosts (using `host.enabled: false` in the `values.yaml`). You can only configure ingress to use virtual hostnames (using `host.enabled: true` in the `values.yaml`), if all of the following criteria are met:

- SSL is terminated at the load balancer
- The SSL port is 443
- You have separate hostnames for OAM administration URL's (for example `https://admin.example.com/em`), and OAM runtime URL's (for example `https://runtime.example.com/oam/server`).

This chapter includes the following topics:

- Installing Traefik
- Creating a Kubernetes Namespace for Traefik
- Generating SSL Certificates for Traefik
- Installing the Traefik Controller
- Preparing the Traefik values.yaml

- Creating the Traefik Ingress
- Verifying Access to the Domain URLs

8.1.1 Installing Traefik

To install the Traefik ingress controller:

1. Add the helm chart repository for Traefik using the following command:

   ```
   $ helm repo add traefik https://helm.traefik.io/traefik --force-
   updateforce-update
   ```

   The output will look similar to the following:

   ```
   "traefik" has been added to your repositories
   ```

2. Update the repository using the following command:

   ```
   $ helm repo update
   ```

   The output will look similar to the following:

   ```
   Hang tight while we grab the latest from your chart repositories...
   ...Successfully got an update from the "stable" chart repository
   Update Complete. Happy Helming!
   ```

8.1.2 Creating a Kubernetes Namespace for Traefik

Create a Kubernetes namespace for the Traefik deployment by running the following command:

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

For example:

```
kubectl create namespace traefik
```

The output will look similar to the following:

```
namespace/traefik created
```

8.1.3 Generating SSL Certificates for Traefik

This section should only be followed if you want to configure your ingress controller to use SSL.

For production environments it is recommended to use a commercially available certificate, traceable to a trusted Certificate Authority.

For sandbox environments, you can generate your own self-signed certificates.

> **Note**
>
> Using self-signed certificates you will get certificate errors when accessing the ingress controller via a browser.

Using a Third Party CA for Generating Certificates

If you are configuring the ingress controller to use SSL, you must use a wildcard certificate to prevent issues with the Common Name (CN) in the certificate. A wildcard certificate is a certificate that protects the primary domain and it's sub-domains. It uses a wildcard character (*) in the CN, for example `*.yourdomain.com`.

How you generate the key and certificate signing request for a wildcard certificate will depend on your Certificate Authority. Contact your Certificate Authority vendor for details.

In order to configure the ingress controller for SSL you require the following files:

- The private key for your certificate, for example `oam.key`.
- The certificate, for example `oam.crt` in PEM format.
- The trusted certificate authority (CA) certificate, for example `rootca.crt` in PEM format.
- If there are multiple trusted CA certificates in the chain, you need all the certificates in the chain, for example `rootca1.crt`, `rootca2.crt` etc.

Once you have received the files, perform the following steps:

1. On the administrative host, create a `$WORKDIR>/ssl` directory and navigate to the folder:

   ```
   mkdir $WORKDIR>/ssl


   cd $WORKDIR>/ssl
   ```

2. Copy the files listed above to the `$WORKDIR>/ssl` directory.
3. If your CA has multiple certificates in a chain, create a `bundle.pem` that contains all the CA certificates:

   ```
   cat rootca.pem rootca1.pem rootca2.pem >>bundle.pem
   ```

Using Self-Signed Certificates

1. On the administrative host, create a `$WORKDIR>/ssl` directory and navigate to the folder:

   ```
   mkdir $WORKDIR>/ssl


   cd $WORKDIR>/ssl
   ```

2. Run the following command to create the self-signed certificate:

   ```
   openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout oam.key -out
   oam.crt -subj "/CN=<hostname>"
   ```

For example:

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout oam.key -out
oam.crt -subj "/CN=oam.example.com"
```

The output will look similar to the following:

```
Generating a 2048 bit RSA private key
.....................................+++
.....................................................................
.........................+++
writing new private key to 'oam.key'
-----
```

Creating a Kubernetes Secret for SSL

Run the following command to create a Kubernetes secret for SSL:

```
kubectl -n mytraefikns create secret tls <domain_uid>-tls-cert --
key $WORKDIR>/ssl/oam.key --cert $WORKDIR>/ssl/oam.crt
```

> **Note**
>
> If you have multiple CA certificates in the chain use `--cert <workdir>/bundle.crt..`

For example:

```
kubectl -n mytraefikns create secret tls accessdomain-tls-cert --key /
OAMK8S/ssl/oam.key --cert /OAMK8S/ssl/oam.crt
```

The output will look similar to the following:

```
secret/accessdomain-tls-cert created
```

8.1.4 Installing the Traefik Controller

Follow these steps to set up Traefik as a load balancer for Oracle Access Management domain in a Kubernetes cluster:

Configuring an Ingress Controller with SSL

Use Helm to install Traefik.

1. Create a $WORKDIR/kubernetes/helm/traefik-ingress-values-override.yaml that contains the following:

> **Note**
>
> The configuration below deploys an ingress using LoadBalancer. If you prefer to use NodePort, change the configuration accordingly.
>
> For more details about Traefik configuration see: Traefik Ingress Controller.

```
ingressRoute:
  dashboard:
    enabled: true
providers:
  kubernetesCRD:
    enabled: true
  kubernetesIngress:
    enabled: true
ports:
  traefik:
    port: 9000
    exposedPort: 9000
    protocol: TCP
  web:
    port: 8000
    exposedPort: 30080
    nodePort: 30080
    protocol: TCP
  websecure:
    port: 8443
    exposedPort: 30443
    nodePort: 30443
    protocol: TCP
service:
  spec:
    type: LoadBalancer
```

2. To install and configure Traefik ingress issue the following command:

   ```
   $ helm install traefik --namespace <namespace> \
   --values traefik-ingress-values-override.yaml \
   traefik/traefik
   ```

   Where:

   - `traefik` is your deployment name
   - `traefik/traefik` is the chart reference

For example:

```
$ helm install --namespace traefik \
--values traefik-ingress-values-override.yaml \
traefik traefik/traefik
```

Configure an Ingress Controller Without SSL

Use Helm to install Traefik.

1. Create a $WORKDIR/kubernetes/kubernetes/charts/traefik/traefik-ingress-values-override.yaml that contains the following:

```
ingressRoute:
  dashboard:
    enabled: true
providers:
  kubernetesCRD:
    enabled: true
  kubernetesIngress:
    enabled: true
ports:
  traefik:
    port: 9000
    exposedPort: 9000
    protocol: TCP
  web:
    port: 8000
    exposedPort: 30305
    nodePort: 30305
    protocol: TCP
  websecure:
    expose:
      default: false
service:
  spec:
    type: LoadBalancer
```

2. To install and configure Traefik ingress, run the following command:

```
helm install traefik --namespace <namespace> \
--values traefik-ingress-values-override.yaml \
traefik/traefik
```

Where:

- `<namespace>` is your namespace, for example `traefik`.
- `ports.web.exposedPort` is the HTTP port that you want the controller to listen on, for example `30305`.
- 'service.spec.type` is the controller type. If using NodePort set to `NodePort`.
- `traefik` is your deployment name
- `traefik/traefik` is the chart reference

For example:

```
$ helm install --namespace traefik \
--values traefik-ingress-values-override.yaml \
traefik traefik/traefik
```

8.1.5 Preparing the Traefik values.yaml

To prepare the `values.yaml` for the ingress:

1. Navigate to the following directory:

   ```
   cd $WORKDIR/kubernetes/charts/ingress-per-domain
   ```

2. Make a copy of the `values.yaml`:

   ```
   cp values.yaml $WORKDIR/
   ```

3. Edit the `$WORKDIR/kubernetes/charts/ingress-per-domain/values.yaml` and modify the following parameters if required:

   - `domainUID:` - If you created your OAM domain with anything other than the default `accessdomain`, change accordingly.
   - `sslType:` - Values supported are `SSL` and `NONSSL`. If you created your ingress controller to use SSL then set to `SSL`, otherwise set to `NONSSL`.
   - `hostName.enabled: false` - This should be set to `false` in almost all circumstances. Setting to `false` allows OAM URI's to be accessible from all hosts. Setting to `true` configures ingress for virtual hostnames only. See Configuring NGINX for full details of the criteria that must be met set to this value to `true`.
   - `hostName.admin: <hostname>` - Should only be set if `hostName.enabled: true` and `sslType: NONSSL`. This should be set to the hostname.domain of the URL you access OAM administration URL's from, for example if you access the OAM Administration Console via `https://admin.example.com/oamconsole`, then set to `admin.example.com`.
   - `hostName.runtime: <hostname>` - Should only be set if `hostName.enabled: true`. This should be set to the hostname.domain of the URL you access OAM runtime URL's from, for example if the oam/server URI is accessed via `https://runtime.example.com/oam/server`, then set to `runtime.example.com`.

The following show example files based on different configuration types:

SSL values.yaml

```
# Load balancer type.  Supported values are: Traefik
type: Traefik

# Type of Configuration Supported Values are : SSL and NONSSL
sslType: SSL

# domainType. Supported values are: oam
domainType: oam


#WLS domain as backend to the load balancer
wlsDomain:
  domainUID: accessdomain
  adminServerName: AdminServer
  adminServerPort: 7001
  adminServerSSLPort:
```

```
  oamClusterName: oam_cluster
  oamManagedServerPort: 14100
  oamManagedServerSSLPort:
  policyClusterName: policy_cluster
  policyManagedServerPort: 14150
  policyManagedServerSSLPort:

# Host  specific values
hostName:
  enabled: false
  admin:
  runtime:
```

NONSSL values.yaml Using All Hostnames

```
# Load balancer type.  Supported values are: Traefik
type: Traefik

# Type of Configuration Supported Values are : SSL and NONSSL
sslType: NONSSL

# domainType. Supported values are: oam
domainType: oam


#WLS domain as backend to the load balancer
wlsDomain:
  domainUID: accessdomain
  adminServerName: AdminServer
  adminServerPort: 7001
  adminServerSSLPort:
  oamClusterName: oam_cluster
  oamManagedServerPort: 14100
  oamManagedServerSSLPort:
  policyClusterName: policy_cluster
  policyManagedServerPort: 14150
  policyManagedServerSSLPort:

# Host  specific values
hostName:
  enabled: false
  admin:
  runtime:
```

NONSSL values.yaml Using Virtual Hostnames

```
# Load balancer type.  Supported values are: Traefik
type: Traefik

# Type of Configuration Supported Values are : SSL and NONSSL
sslType: NONSSL

# domainType. Supported values are: oam
domainType: oam
```

```
#WLS domain as backend to the load balancer
wlsDomain:
  domainUID: accessdomain
  adminServerName: AdminServer
  adminServerPort: 7001
  adminServerSSLPort:
  oamClusterName: oam_cluster
  oamManagedServerPort: 14100
  oamManagedServerSSLPort:
  policyClusterName: policy_cluster
  policyManagedServerPort: 14150
  policyManagedServerSSLPort:

# Host  specific values
hostName:
  enabled: true
  admin: admin.example.com
  runtime: runtime.example.com
```

8.1.6 Creating the Traefik Ingress

Run the following commands to create the ingress:

1. Navigate to the `$WORKDIR`:

   ```
   cd $WORKDIR
   ```

2. Run the following helm command to create the ingress:

   ```
   helm install oam-traefik kubernetes/charts/ingress-per-domain \
   --namespace <domain_namespace> \
   --values kubernetes/charts/ingress-per-domain/values.yaml
   ```

   For example:

   ```
   helm install oam-traefik kubernetes/charts/ingress-per-domain \
   --namespace oamns \
   --values kubernetes/charts/ingress-per-domain/values.yaml
   ```

   The output will look similar to the following:

   ```
   NAME: oam-traefik
   LAST DEPLOYED: <DATE>
   NAMESPACE: oamns
   STATUS: deployed
   REVISION: 1
   TEST SUITE: None
   ```

3. Run the following command to show the ingress is created successfully:

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

For example

```
kubectl get ing -n oamns
```

If `hostname.enabled: false`, the output will look similar to the following:

```
NAME                 CLASS    HOSTS   ADDRESS   PORTS   AGE
accessdomain-traefik   traefik    *                 80      5s
```

If `hostname.enabled: true`, the output will look similar to the following:

```
NAME                 CLASS   HOSTS                   ADDRESS   PORTS   AGE
oamadmin-ingress     traefik   admin.example.com                 80
14s
oamruntime-ingress   traefik   runtime.example.com               80
14s
```

4. Run the following command to check the ingress:

```
kubectl describe ing <ingress> -n <domain_namespace>
```

For example:

```
kubectl describe ing accessdomain-traefik -n oamns
```

The output will look similar to the following for `accessdomain-traefik`:

```
Name:             accessdomain-traefik
Labels:           app.kubernetes.io/managed-by=Helm
Namespace:        oamns
Address:          10.109.22.22
Ingress Class:    traefik
Default backend:  <default>

Rules:
  Host        Path  Backends
  ----        ----  --------
  *
              /console                      accessdomain-
adminserver:7001 (10.244.1.200:7001)
              /consolehelp                  accessdomain-
adminserver:7001 (10.244.1.200:7001)
              /rreg/rreg                    accessdomain-
adminserver:7001 (10.244.1.200:7001)
              /em                           accessdomain-
adminserver:7001 (10.244.1.200:7001)
              /management                   accessdomain-
adminserver:7001 (10.244.1.200:7001)
              /oamconsole                   accessdomain-
adminserver:7001 (10.244.1.200:7001)
              /dms                          accessdomain-
adminserver:7001 (10.244.1.200:7001)
              /oam/services/rest            accessdomain-
```

```
adminserver:7001 (10.244.1.200:7001)
              /iam/admin/config              accessdomain-
adminserver:7001 (10.244.1.200:7001)
              /iam/admin/diag                accessdomain-
adminserver:7001 (10.244.1.200:7001)
              /iam/access                    accessdomain-cluster-oam-
cluster:14100 (10.244.2.127:14100)
              /oam/admin/api                 accessdomain-
adminserver:7001 (10.244.1.200:7001)
              /oam/services/rest/access/api  accessdomain-cluster-oam-
cluster:14100 (10.244.2.127:14100)
              /access                        accessdomain-cluster-policy-
cluster:15100 (10.244.2.126:14150)
              /oam                           accessdomain-cluster-oam-
cluster:14100 (10.244.2.127:14100)
              /                              accessdomain-cluster-oam-
cluster:14100 (10.244.2.127:14100)
Annotations:  meta.helm.sh/release-name: oam-traefik
              meta.helm.sh/release-namespace: oamns
              traefik.ingress.kubernetes.io/enable-access-log: false
              traefik.ingress.kubernetes.io/proxy-buffer-size: 2000k
Events:       <none>
   Type    Reason  Age   From                      Message
  ----    ------  ----  ----                     -------
  Normal  Sync    33s   traefik-ingress-controller  Scheduled for sync
```

The output will look similar to the following for `oamadmin-ingress`:

```
Name:             oamadmin-ingress
Labels:           app.kubernetes.io/managed-by=Helm
Namespace:        oamns
Address:          10.109.22.22
Ingress Class:    traefik
Default backend:  <default>
Rules:
  Host                                    Path  Backends
  ----                                    ----  --------
  admin.example.com
                                          /console
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                          /consolehelp
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                          /rreg/rreg
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                          /em
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                          /oamconsole
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                          /dms
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                          /oam/services/rest
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                          /iam/admin/config
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                          /oam/admin/api
```

```
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                        /iam/admin/diag
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                        /oam/services
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                        /iam/admin
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                        /oam/services/rest/11.1.2.0.0
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                        /oam/services/rest/ssa
accessdomain-cluster-oam-cluster:14100 (10.244.2.127:14100)
                                        /access
accessdomain-cluster-policy-cluster:14150 (10.244.2.126:15100)
Annotations:                            meta.helm.sh/release-name: oam-
traefik
                                        meta.helm.sh/release-namespace:
oamns
                                        traefik.ingress.kubernetes.io/
affinity: cookie
                                        traefik.ingress.kubernetes.io/
enable-access-log: false
                                        traefik.ingress.kubernetes.io/
ingress.allow-http: true
                                        traefik.ingress.kubernetes.io/
proxy-buffer-size: 2000k
                                        traefik.ingress.kubernetes.io/
ssl-redirect: false
Events:
  Type    Reason  Age   From                      Message
  ----    ------  ----  ----                      -------
  Normal  Sync    32s   traefik-ingress-controller  Scheduled for sync
```

The output will look similar to the following for `oamruntime-ingress`:

```
Name:             oamruntime-ingress
Labels:           app.kubernetes.io/managed-by=Helm
Namespace:        oamns
Address:          10.109.22.22
Ingress Class:    traefik
Default backend:  <default>
Rules:
  Host                                  Path  Backends
  ----                                  ----  --------
  runtime.example.com
                                        /
ms_oauth                            accessdomain-cluster-oam-cluster:14100
(10.244.2.127:14100)
                                        /oam/services/rest/
auth            accessdomain-cluster-oam-cluster:14100
(10.244.2.127:14100)
                                        /oam/services/rest/
access          accessdomain-cluster-oam-cluster:14100
(10.244.2.127:14100)
                                        /
oamfed                              accessdomain-cluster-oam-cluster:14100
```

```
(10.244.2.127:14100)
                                        /
otpfp/                              accessdomain-cluster-oam-cluster:14100
(10.244.2.127:14100)
                                        /
oauth2                              accessdomain-cluster-oam-cluster:14100
(10.244.2.127:14100)
                                        /oam
   accessdomain-cluster-oam-cluster:14100 (10.244.2.127:14100)
                                        /.well-known/openid-
configuration   accessdomain-cluster-oam-cluster:14100 (10.244.2.127:14100)
                                        /.well-known/oidc-
configuration     accessdomain-cluster-oam-cluster:14100
(10.244.2.127:14100)
                                        /
CustomConsent                       accessdomain-cluster-oam-cluster:14100
(10.244.2.127:14100)
                                        /iam/
access                          accessdomain-cluster-oam-cluster:14100
(10.244.2.127:14100)
Annotations:                            meta.helm.sh/release-name: oam-
traefik
                                        meta.helm.sh/release-namespace:
oamns
                                        traefik.ingress.kubernetes.io/
affinity: cookie
                                        traefik.ingress.kubernetes.io/
enable-access-log: false
                                        traefik.ingress.kubernetes.io/
proxy-buffer-size: 2000k
Events:
  Type    Reason  Age                   From                      Message
  ----    ------  ----                  ----                      -------
  Normal  Sync    3m34s (x2 over 4m10s)  traefik-ingress-controller
Scheduled for sync
```

5. To confirm that the new ingress is successfully routing to the domain's server pods, run the following command to send a request to the OAM Administration Console:

   - For SSL:

     ```
     curl -v -k https://${HOSTNAME}:${PORT}/oamconsole
     ```

   - For NONSSL:

     ```
     curl -v http://${HOSTNAME}:${PORT}/oamconsole
     ```

   The `${HOSTNAME}:${PORT}` to use depends on the value set for `hostName.enabled`.

   If `hostName.enabled: false` use the hostname and port where the ingress controller is installed, for example `http://oam.example.com:30777`.

   If using `hostName.enabled: true` then you can only access via the admin hostname, for example `https://admin.example.com/oamconsole`.

> **Note**
>
> You can only access via the admin URL if it is currently accessible and routing correctly to the ingress host and port.

For example:

```
curl -v http://oam.example.com:30777/oamconsole
```

The output will look similar to the following. You should receive a `302 Moved Temporarily` message:

```
> GET /oamconsole HTTP/1.1
> Host: oam.example.com:30777
> User-Agent: curl/7.61.1
> Accept: */*
>
< HTTP/1.1 302 Moved Temporarily
< Date: <DATE>
< Content-Type: text/html
< Content-Length: 333
< Connection: keep-alive
< Location: http://oam.example.com:30777/oamconsole/
< X-Content-Type-Options: nosniff
< X-Frame-Options: DENY
<
<html><head><title>302 Moved Temporarily</title></head>
<body bgcolor="#FFFFFF">
<p>This document you requested has moved
temporarily.</p>
<p>It's now at <a href="http://oam.example.com:30777/oamconsole/">http://
oam.example.com:30777/oamconsole/</a>.</p>
</body></html>
* Connection #0 to host oam.example.com left intact
```

After confirming the above, verify that the domain applications are accessible. See, Validating the Domain URLs.

8.1.7 Verifying Access to the Domain URLs

After setting up the Traefik ingress, verify that the domain applications are accessible as described in Validating the Domain URLs.

8.2 Configuring NGINX

You must configure an NGINX ingress controller to allow access to Oracle Access Management (OAM).

The ingress can be configured in the following ways:

- Without SSL
- With SSL

- OAM URI's are accessible from all hosts
- OAM URI's are accessible using virtual hostnames only

The option you choose will depend on the architecture you are configuring. For example, if you have an architecture such as Oracle HTTP Server on an Independent Kubernetes cluster, where SSL is terminated at the load balancer, then you would configure the ingress without SSL.

In almost all circumstances, the ingress should be configured to be accessible from all hosts (using `host.enabled: false` in the `values.yaml`). You can only configure ingress to use virtual hostnames (using `host.enabled: true` in the `values.yaml`), if all of the following criteria are met:

- SSL is terminated at the load balancer
- The SSL port is 443
- You have separate hostnames for OAM administration URL's (for example `https://admin.example.com/em`), and OAM runtime URL's (for example `https://runtime.example.com/oam/server`).

This chapter includes the following topics:

8.2.1 Installing the NGINX Repository

To install the NGINX ingress controller:

1. Add the Helm chart repository for NGINX using the following command:

   ```
   helm repo add stable https://kubernetes.github.io/ingress-nginx
   ```

   The output will look similar to the following:

   ```
   "stable" has been added to your repositories
   ```

2. Update the repository using the following command:

   ```
   helm repo update
   ```

   The output will look similar to the following:

   ```
   Hang tight while we grab the latest from your chart repositories...
   ...Successfully got an update from the "stable" chart repository
   Update Complete. Happy Helming!
   ```

8.2.2 Creating a Kubernetes Namespace for NGINX

Create a Kubernetes namespace for the NGINX deployment by running the following command:

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

For example:

```
kubectl create namespace mynginxns
```

The output will look similar to the following:

```
namespace/mynginxns created
```

8.2.3 Generating SSL Certificates

This section should only be followed if you want to configure your ingress controller to use SSL.

For production environments it is recommended to use a commercially available certificate, traceable to a trusted Certificate Authority.

For sandbox environments, you can generate your own self-signed certificates.

> **Note**
>
> Using self-signed certificates you will get certificate errors when accessing the ingress controller via a browser.

Using a Third Party CA for Generating Certificates

If you are configuring the ingress controller to use SSL, you must use a wildcard certificate to prevent issues with the Common Name (CN) in the certificate. A wildcard certificate is a certificate that protects the primary domain and it's sub-domains. It uses a wildcard character (*) in the CN, for example `*.yourdomain.com`.

How you generate the key and certificate signing request for a wildcard certificate will depend on your Certificate Authority. Contact your Certificate Authority vendor for details.

In order to configure the ingress controller for SSL you require the following files:

- The private key for your certificate, for example `oam.key`.
- The certificate, for example `oam.crt` in PEM format.
- The trusted certificate authority (CA) certificate, for example `rootca.crt` in PEM format.
- If there are multiple trusted CA certificates in the chain, you need all the certificates in the chain, for example `rootca1.crt`, `rootca2.crt` etc.

Once you have received the files, perform the following steps:

1. On the administrative host, create a `$WORKDIR>/ssl` directory and navigate to the folder:

   ```
   mkdir $WORKDIR>/ssl
   ```

   ```
   cd $WORKDIR>/ssl
   ```

2. Copy the files listed above to the `$WORKDIR>/ssl` directory.
3. If your CA has multiple certificates in a chain, create a `bundle.pem` that contains all the CA certificates:

   ```
   cat rootca.pem rootca1.pem rootca2.pem >>bundle.pem
   ```

Using Self-Signed Certificates

1. On the administrative host, create a `$WORKDIR>/ssl` directory and navigate to the folder:

   ```
   mkdir $WORKDIR>/ssl
   ```

   ```
   cd $WORKDIR>/ssl
   ```

2. Run the following command to create the self-signed certificate:

   ```
   openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout oam.key -out
   oam.crt -subj "/CN=<hostname>"
   ```

   For example:

   ```
   openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout oam.key -out
   oam.crt -subj "/CN=oam.example.com"
   ```

   The output will look similar to the following:

   ```
   Generating a 2048 bit RSA private key
   ..........................................+++
   ..........................................................................
   ..........................+++
   writing new private key to 'oam.key'
   -----
   ```

Creating a Kubernetes Secret for SSL

Run the following command to create a Kubernetes secret for SSL:

```
kubectl -n mynginxns create secret tls <domain_uid>-tls-cert --
key $WORKDIR>/ssl/oam.key --cert $WORKDIR>/ssl/oam.crt
```

> **Note**
>
> If you have multiple CA certificates in the chain use `--cert <workdir>/bundle.crt..`

For example:

```
kubectl -n mynginxns create secret tls accessdomain-tls-cert --key /
OAMK8S/ssl/oam.key --cert /OAMK8S/ssl/oam.crt
```

The output will look similar to the following:

```
secret/accessdomain-tls-cert created
```

8.2.4 Installing the NGINX Controller

In this section you install the NGINX controller.

If you can connect directly to a worker node hostname or IP address from a browser, then install NGINX with the `--set controller.service.type=NodePort` parameter.

If you are using a Managed Service for your Kubernetes cluster, for example Oracle Kubernetes Engine (OKE) on Oracle Cloud Infrastructure (OCI), and connect from a browser to the Load Balancer IP address, then use the `--set controller.service.type=LoadBalancer` parameter. This instructs the Managed Service to setup a Load Balancer to direct traffic to the NGINX ingress.

The instructions below use `--set controller.service.type=NodePort`. If using a managed service, change to `--set controller.service.type=LoadBalancer`.

Configuring an Ingress Controller with SSL

To configure the ingress controller to use SSL, run the following command:

```
helm install nginx-ingress \
-n <domain_namespace> \
--set controller.service.nodePorts.http=<http_port> \
--set controller.service.nodePorts.https=<https_port> \
--set controller.extraArgs.default-ssl-certificate=<domain_namespace>/
<ssl_secret> \
--set controller.service.type=<type> \
--set controller.config.use-forwarded-headers=true \
--set controller.config.enable-underscores-in-headers=true \
--set controller.admissionWebhooks.enabled=false \
stable/ingress-nginx \
--version 4.7.2
```

Where:

- `<domain_namespace>` is your namespace, for example `mynginxns`.
- `<http_port>` is the HTTP port that you want the controller to listen on, for example `30777`.
- `<https_port>` is the HTTPS port that you want the controller to listen on, for example `30443`.
- `<type>` is the controller type. If using NodePort set to NodePort. If using a managed service set to LoadBalancer. If using LoadBalancer remove `--set controller.service.nodePorts.http=<http_port>` and `--set controller.service.nodePorts.https=<https_port>`.
- `<ssl_secret>` is the secret you created in Generating SSL Certificates.

For example:

```
helm install nginx-ingress -n mynginxns \
--set controller.service.nodePorts.http=30777 \
--set controller.service.nodePorts.https=30443 \
--set controller.extraArgs.default-ssl-certificate=mynginxns/accessdomain-tls-
cert \
--set controller.service.type=NodePort \
--set controller.config.use-forwarded-headers=true \
--set controller.config.enable-underscores-in-headers=true \
--set controller.admissionWebhooks.enabled=false \
stable/ingress-nginx \
--version 4.7.2
```

The output will look similar to the following:

```
NAME: nginx-ingress
LAST DEPLOYED: <DATE>

NAMESPACE: mynginxns
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The nginx-ingress controller has been installed.
Get the application URL by running these commands:
  export HTTP_NODE_PORT=30777
  export HTTPS_NODE_PORT=30443
  export NODE_IP=$(kubectl --namespace mynginxns get nodes -o
jsonpath="{.items[0].status.addresses[1].address}")

  echo "Visit http://$NODE_IP:$HTTP_NODE_PORT to access your application via
HTTP."
  echo "Visit https://$NODE_IP:$HTTPS_NODE_PORT to access your application
via HTTPS."

An example Ingress that makes use of the controller:

  apiVersion: networking.k8s.io/v1
  kind: Ingress
  metadata:
    annotations:
      kubernetes.io/ingress.class: nginx
    name: example
    namespace: foo
  spec:
    ingressClassName: example-class
    rules:
       - host: www.example.com
         http:
           paths:
             - path: /
               pathType: Prefix
               backend:
                 service:
```

```
                name: exampleService
                port: 80
    # This section is only required if TLS is to be enabled for the Ingress
    tls:
      - hosts:
        - www.example.com
        secretName: example-tls
```

If TLS is enabled for the Ingress, a Secret containing the certificate and key must also be provided:

```
  apiVersion: v1
  kind: Secret
  metadata:
    name: example-tls
    namespace: foo
  data:
    tls.crt: <base64 encoded cert>
    tls.key: <base64 encoded key>
  type: kubernetes.io/tls
```

Configure an Ingress Controller Without SSL

To configure the ingress controller without SSL, run the following command:

```
helm install nginx-ingress \
-n <domain_namespace> \
--set controller.service.nodePorts.http=<http_port> \
--set controller.service.type=NodePort \
--set controller.config.use-forwarded-headers=true \
--set controller.config.enable-underscores-in-headers=true \
--set controller.admissionWebhooks.enabled=false \
stable/ingress-nginx
--version 4.7.2
```

Where:

- `<domain_namespace>` is your namespace, for example `mynginxns`.
- `<http_port>` is the HTTP port that you want the controller to listen on, for example `30777`.
- `<type>` is the controller type. If using NodePort set to NodePort. If using a managed service set to LoadBalancer. If using LoadBalancer remove `--set controller.service.nodePorts.http=<http_port>`.

For example:

```
helm install nginx-ingress \
-n mynginxns \
--set controller.service.nodePorts.http=30777 \
--set controller.service.type=NodePort \
--set controller.config.use-forwarded-headers=true \
--set controller.config.enable-underscores-in-headers=true \
--set controller.admissionWebhooks.enabled=false \
```

```
stable/ingress-nginx \
--version 4.7.2
```

The output will look similar to the following:

```
NAME: nginx-ingress
LAST DEPLOYED: <DATE>

NAMESPACE: mynginxns
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The nginx-ingress controller has been installed.
Get the application URL by running these commands:
  export HTTP_NODE_PORT=30777
  export HTTPS_NODE_PORT=$(kubectl --namespace mynginxns get services -o
jsonpath="{.spec.ports[1].nodePort}" nginx-ingress-ingress-nginx-controller)
  export NODE_IP=$(kubectl --namespace mygninx get nodes -o
jsonpath="{.items[0].status.addresses[1].address}")

  echo "Visit http://$NODE_IP:$HTTP_NODE_PORT to access your application via
HTTP."
  echo "Visit https://$NODE_IP:$HTTPS_NODE_PORT to access your application
via HTTPS."

An example Ingress that makes use of the controller:

  apiVersion: networking.k8s.io/v1
  kind: Ingress
  metadata:
    annotations:
      kubernetes.io/ingress.class: nginx
    name: example
    namespace: foo
  spec:
    ingressClassName: example-class
    rules:
       - host: www.example.com
         http:
           paths:
             - path: /
               pathType: Prefix
               backend:
                 service:
                   name: exampleService
                   port: 80
    # This section is only required if TLS is to be enabled for the Ingress
    tls:
      - hosts:
        - www.example.com
        secretName: example-tls


If TLS is enabled for the Ingress, a Secret containing the certificate and
```

```
key must also provided:

  apiVersion: v1
  kind: Secret
  metadata:
    name: example-tls
    namespace: foo
  data:
    tls.crt: <base64 encoded cert>
    tls.key: <base64 encoded key>
  type: kubernetes.io/tls
```

8.2.5 Preparing the Ingress values.yaml

To prepare the `values.yaml` for the ingress:

1. Navigate to the following directory:

   ```
   cd $WORKDIR/kubernetes/charts/ingress-per-domain
   ```

2. Make a copy of the `values.yaml`:

   ```
   cp values.yaml $WORKDIR/
   ```

3. Edit the `$WORKDIR/kubernetes/charts/ingress-per-domain/values.yaml` and modify the following parameters if required:

   - `domainUID:` - If you created your OAM domain with anything other than the default `accessdomain`, change accordingly.
   - `sslType:` - Values supported are `SSL` and `NONSSL`. If you created your ingress controller to use SSL then set to `SSL`, otherwise set to `NONSSL`.
   - `hostName.enabled: false` - This should be set to `false` in almost all circumstances. Setting to `false` allows OAM URI's to be accessible from all hosts. Setting to `true` configures ingress for virtual hostnames only. See Configuring NGINX for full details of the criteria that must be met set to this value to `true`.
   - `hostName.admin: <hostname>` - Should only be set if `hostName.enabled: true` and `sslType: NONSSL`. This should be set to the hostname.domain of the URL you access OAM administration URL's from, for example if you access the OAM Administration Console via `https://admin.example.com/oamconsole`, then set to `admin.example.com`.
   - `hostName.runtime: <hostname>` - Should only be set if `hostName.enabled: true`. This should be set to the hostname.domain of the URL you access OAM runtime URL's from, for example if the oam/server URI is accessed via `https://runtime.example.com/oam/server`, then set to `runtime.example.com`.

The following show example files based on different configuration types:

SSL values.yaml

```
# Load balancer type.  Supported values are: NGINX
type: NGINX

# Type of Configuration Supported Values are : SSL and NONSSL
```

```
sslType: SSL

# domainType. Supported values are: oam
domainType: oam


#WLS domain as backend to the load balancer
wlsDomain:
  domainUID: accessdomain
  adminServerName: AdminServer
  adminServerPort: 7001
  adminServerSSLPort:
  oamClusterName: oam_cluster
  oamManagedServerPort: 14100
  oamManagedServerSSLPort:
  policyClusterName: policy_cluster
  policyManagedServerPort: 14150
  policyManagedServerSSLPort:

# Host  specific values
hostName:
  enabled: false
  admin:
  runtime:
```

NONSSL values.yaml Using All Hostnames

```
# Load balancer type.  Supported values are: NGINX
type: NGINX

# Type of Configuration Supported Values are : SSL and NONSSL
sslType: NONSSL

# domainType. Supported values are: oam
domainType: oam


#WLS domain as backend to the load balancer
wlsDomain:
  domainUID: accessdomain
  adminServerName: AdminServer
  adminServerPort: 7001
  adminServerSSLPort:
  oamClusterName: oam_cluster
  oamManagedServerPort: 14100
  oamManagedServerSSLPort:
  policyClusterName: policy_cluster
  policyManagedServerPort: 14150
  policyManagedServerSSLPort:

# Host  specific values
hostName:
  enabled: false
  admin:
  runtime:
```

NONSSL values.yaml Using Virtual Hostnames

```
# Load balancer type.  Supported values are: NGINX
type: NGINX

# Type of Configuration Supported Values are : SSL and NONSSL
sslType: NONSSL

# domainType. Supported values are: oam
domainType: oam


#WLS domain as backend to the load balancer
wlsDomain:
  domainUID: accessdomain
  adminServerName: AdminServer
  adminServerPort: 7001
  adminServerSSLPort:
  oamClusterName: oam_cluster
  oamManagedServerPort: 14100
  oamManagedServerSSLPort:
  policyClusterName: policy_cluster
  policyManagedServerPort: 14150
  policyManagedServerSSLPort:

# Host  specific values
hostName:
  enabled: true
  admin: admin.example.com
  runtime: runtime.example.com
```

8.2.6 Creating the Ingress

Run the following commands to create the ingress:

1. Navigate to the `$WORKDIR`:

   ```
   cd $WORKDIR
   ```

2. Run the following helm command to create the ingress:

   ```
   helm install oam-nginx kubernetes/charts/ingress-per-domain \
   --namespace <domain_namespace> \
   --values kubernetes/charts/ingress-per-domain/values.yaml
   ```

   For example:

   ```
   helm install oam-nginx kubernetes/charts/ingress-per-domain \
   --namespace oamns \
   --values kubernetes/charts/ingress-per-domain/values.yaml
   ```

The output will look similar to the following:

```
NAME: oam-nginx
LAST DEPLOYED: <DATE>
NAMESPACE: oamns
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

3. Run the following command to show the ingress is created successfully:

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

For example

```
kubectl get ing -n oamns
```

If `hostname.enabled: false`, the output will look similar to the following:

```
NAME                 CLASS    HOSTS   ADDRESS   PORTS   AGE
accessdomain-nginx   nginx    *                 80      5s
```

If `hostname.enabled: true`, the output will look similar to the following:

```
NAME                 CLASS   HOSTS                    ADDRESS   PORTS   AGE
oamadmin-ingress     nginx   admin.example.com                  80      14s
oamruntime-ingress   nginx   runtime.example.com                80      14s
```

4. Run the following command to check the ingress:

```
kubectl describe ing <ingress> -n <domain_namespace>
```

For example:

```
kubectl describe ing accessdomain-nginx -n oamns
```

The output will look similar to the following for `accessdomain-nginx`:

```
Name:             accessdomain-nginx
Labels:           app.kubernetes.io/managed-by=Helm
Namespace:        oamns
Address:          10.109.22.22
Ingress Class:    nginx
Default backend:  <default>

Rules:
  Host        Path  Backends
  ----        ----  --------
  *
              /console                        accessdomain-
adminserver:7001 (10.244.1.200:7001)
              /consolehelp                    accessdomain-
```

```
adminserver:7001 (10.244.1.200:7001)
              /rreg/rreg                      accessdomain-
adminserver:7001 (10.244.1.200:7001)
              /em                             accessdomain-
adminserver:7001 (10.244.1.200:7001)
              /management                     accessdomain-
adminserver:7001 (10.244.1.200:7001)
              /oamconsole                     accessdomain-
adminserver:7001 (10.244.1.200:7001)
              /dms                            accessdomain-
adminserver:7001 (10.244.1.200:7001)
              /oam/services/rest              accessdomain-
adminserver:7001 (10.244.1.200:7001)
              /iam/admin/config               accessdomain-
adminserver:7001 (10.244.1.200:7001)
              /iam/admin/diag                 accessdomain-
adminserver:7001 (10.244.1.200:7001)
              /iam/access                     accessdomain-cluster-oam-
cluster:14100 (10.244.2.127:14100)
              /oam/admin/api                  accessdomain-
adminserver:7001 (10.244.1.200:7001)
              /oam/services/rest/access/api   accessdomain-cluster-oam-
cluster:14100 (10.244.2.127:14100)
              /access                         accessdomain-cluster-policy-
cluster:15100 (10.244.2.126:14150)
              /oam                            accessdomain-cluster-oam-
cluster:14100 (10.244.2.127:14100)
              /                               accessdomain-cluster-oam-
cluster:14100 (10.244.2.127:14100)
Annotations:  meta.helm.sh/release-name: oam-nginx
              meta.helm.sh/release-namespace: oamns
              nginx.ingress.kubernetes.io/enable-access-log: false
              nginx.ingress.kubernetes.io/proxy-buffer-size: 2000k
Events:       <none>
   Type    Reason  Age   From                      Message
  ----    ------  ----  ----                      -------
  Normal  Sync    33s   nginx-ingress-controller  Scheduled for sync
```

The output will look similar to the following for `oamadmin-ingress`:

```
Name:             oamadmin-ingress
Labels:           app.kubernetes.io/managed-by=Helm
Namespace:        oamns
Address:          10.109.22.22
Ingress Class:    nginx
Default backend:  <default>
Rules:
  Host                                  Path  Backends
  ----                                  ----  --------
  admin.example.com
                                        /console
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                        /consolehelp
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                        /rreg/rreg
```

```
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                          /em
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                          /oamconsole
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                          /dms
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                          /oam/services/rest
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                          /iam/admin/config
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                          /oam/admin/api
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                          /iam/admin/diag
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                          /oam/services
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                          /iam/admin
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                          /oam/services/rest/11.1.2.0.0
accessdomain-adminserver:7001 (10.244.1.200:7001)
                                          /oam/services/rest/ssa
accessdomain-cluster-oam-cluster:14100 (10.244.2.127:14100)
                                          /access
accessdomain-cluster-policy-cluster:14150 (10.244.2.126:15100)
Annotations:                              meta.helm.sh/release-name: oam-
nginx
                                          meta.helm.sh/release-namespace:
oamns
                                          nginx.ingress.kubernetes.io/
affinity: cookie
                                          nginx.ingress.kubernetes.io/
enable-access-log: false
                                          nginx.ingress.kubernetes.io/
ingress.allow-http: true
                                          nginx.ingress.kubernetes.io/
proxy-buffer-size: 2000k
                                          nginx.ingress.kubernetes.io/ssl-
redirect: false
Events:
  Type    Reason  Age   From                      Message
  ----    ------  ----  ----                      -------
  Normal  Sync    32s   nginx-ingress-controller  Scheduled for sync
```

The output will look similar to the following for `oamruntime-ingress`:

```
Name:             oamruntime-ingress
Labels:           app.kubernetes.io/managed-by=Helm
Namespace:        oamns
Address:          10.109.22.22
Ingress Class:    nginx
Default backend:  <default>
Rules:
  Host                                    Path  Backends
  ----                                    ----  --------
```

```
  runtime.example.com
                                                /
ms_oauth                              accessdomain-cluster-oam-cluster:14100
(10.244.2.127:14100)
                                                /oam/services/rest/
auth             accessdomain-cluster-oam-cluster:14100
(10.244.2.127:14100)
                                                /oam/services/rest/
access           accessdomain-cluster-oam-cluster:14100
(10.244.2.127:14100)
                                                /
oamfed                                accessdomain-cluster-oam-cluster:14100
(10.244.2.127:14100)
                                                /
otpfp/                                accessdomain-cluster-oam-cluster:14100
(10.244.2.127:14100)
                                                /
oauth2                                accessdomain-cluster-oam-cluster:14100
(10.244.2.127:14100)
                                                /oam
   accessdomain-cluster-oam-cluster:14100 (10.244.2.127:14100)
                                                /.well-known/openid-
configuration   accessdomain-cluster-oam-cluster:14100 (10.244.2.127:14100)
                                                /.well-known/oidc-
configuration     accessdomain-cluster-oam-cluster:14100
(10.244.2.127:14100)
                                                /
CustomConsent                         accessdomain-cluster-oam-cluster:14100
(10.244.2.127:14100)
                                                /iam/
access                           accessdomain-cluster-oam-cluster:14100
(10.244.2.127:14100)
Annotations:                                    meta.helm.sh/release-name: oam-
nginx
                                                meta.helm.sh/release-namespace:
oamns
                                                nginx.ingress.kubernetes.io/
affinity: cookie
                                                nginx.ingress.kubernetes.io/
enable-access-log: false
                                                nginx.ingress.kubernetes.io/
proxy-buffer-size: 2000k
Events:
  Type    Reason  Age                    From                      Message
  ----    ------  ----                   ----                      -------
  Normal  Sync    3m34s (x2 over 4m10s)  nginx-ingress-controller
Scheduled for sync
```

5. To confirm that the new ingress is successfully routing to the domain's server pods, run the following command to send a request to the OAM Administration Console:

   - For SSL:

     ```
     curl -v -k https://${HOSTNAME}:${PORT}/oamconsole
     ```

- For NONSSL:

```
curl -v http://${HOSTNAME}:${PORT}/oamconsole
```

The `${HOSTNAME}:${PORT}` to use depends on the value set for `hostName.enabled`.

If `hostName.enabled: false` use the hostname and port where the ingress controller is installed, for example `http://oam.example.com:30777`.

If using `hostName.enabled: true` then you can only access via the admin hostname, for example `https://admin.example.com/oamconsole`.

> **Note**
>
> You can only access via the admin URL if it is currently accessible and routing correctly to the ingress host and port.

For example:

```
curl -v http://oam.example.com:30777/oamconsole
```

The output will look similar to the following. You should receive a `302 Moved Temporarily` message:

```
> GET /oamconsole HTTP/1.1
> Host: oam.example.com:30777
> User-Agent: curl/7.61.1
> Accept: */*
>
< HTTP/1.1 302 Moved Temporarily
< Date: <DATE>
< Content-Type: text/html
< Content-Length: 333
< Connection: keep-alive
< Location: http://oam.example.com:30777/oamconsole/
< X-Content-Type-Options: nosniff
< X-Frame-Options: DENY
<
<html><head><title>302 Moved Temporarily</title></head>
<body bgcolor="#FFFFFF">
<p>This document you requested has moved
temporarily.</p>
<p>It's now at <a href="http://oam.example.com:30777/oamconsole/">http://
oam.example.com:30777/oamconsole/</a>.</p>
</body></html>
* Connection #0 to host oam.example.com left intact
```

After confirming the above, verify that the domain applications are accessible. See, Validating the Domain URLs.

9 Validating the Domain URLs

Launch a browser and access the following URL's. Login with the weblogic username and password (`weblogic/<password>`).

> **Note**
>
> The `${HOSTNAME}:${PORT}` depends on the architecture configured, and your ingress setup as per Configuring NGINX.

Console or Page	URL
Oracle Enterprise Manager Console	`http(s)://${HOSTNAME}:${PORT}/em`
Oracle Access Management Console	`http(s)://${HOSTNAME}:${PORT}/oamconsole`
Oracle Access Management Console	`http(s)://${HOSTNAME}:${PORT}/access`
Logout URL	`http(s)://${HOSTNAME}:${PORT}/oam/server/logout`

> **Note**
>
> Administrators should be aware of the following:
>
> - To monitor the Oracle Access Management (OAM) WebLogic Server domain in 14.1.2.1.0 you must use the Oracle WebLogic Remote Console. For more information about installing and configuring the console, see Getting Started Using Administration Console.
> - The Oracle WebLogic Remote Console and Oracle Enterprise Manager Console should only be used to monitor the servers in the OAM domain. To control the Administration Server and OAM Managed Servers (start/stop) you must use Kubernetes. See Scaling OAM Pods for more information.

The browser will give certificate errors if you used a self signed certificate and have not imported it into the browsers Certificate Authority store. If this occurs you can proceed with the connection and ignore the errors.

After validating the URL's proceed to Post Installation Configuration.

10

Post Installation Configuration

After the OAM domain is successfully deployed, you must perform some post configuration steps.

This chapter includes the following topics:

10.1 Creating a Server Overrides File

Perform the following steps to create a server overrides file for Oracle Access Management (OAM):

1. Navigate to the following directory:
   - For OAM domains created with WLST:

     ```
     cd $WORKDIR/kubernetes/create-access-domain/domain-home-on-pv/output/
     weblogic-domains/accessdomain
     ```

   - For OAM domains created with WDT:

     ```
     cd $WORKDIR/kubernetes/create-access-domain/domain-home-on-pv/
     ```

2. Create a `setUserOverrides.sh` with the following contents:

   ```
   DERBY_FLAG=false
   JAVA_OPTIONS="${JAVA_OPTIONS} -Djava.net.preferIPv4Stack=true"
   MEM_ARGS="-Xms8192m -Xmx8192m"
   ```

3. Copy the `setUserOverrides.sh` file to the Administration Server pod:

   ```
   chmod 755 setUserOverrides.sh


    kubectl cp setUserOverrides.sh <domain_namespace>/<domain-uid>-
   adminserver:/u01/oracle/user_projects/domains/<domain-uid>/bin/
   setUserOverrides.sh
   ```

For example:

```
 kubectl cp setUserOverrides.sh oamns/accessdomain-adminserver:/u01/oracle/
user_projects/domains/accessdomain/bin/setUserOverrides.sh
```

4. Stop the OAM domain using the following command:

```
kubectl -n <domain_namespace> patch domains <domain_uid> --type='json' -
p='[{"op": "replace", "path": "/spec/serverStartPolicy", "value":
"Never" }]'
```

For example:

```
kubectl -n oamns patch domains accessdomain --type='json' -p='[{"op":
"replace", "path": "/spec/serverStartPolicy", "value": "Never" }]'
```

The output will look similar to the following:

```
domain.weblogic.oracle/accessdomain patched
```

5. Check that all the pods are stopped:

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

For example:

```
kubectl get pods -n oamns
```

The output will look similar to the following:

```
NAME                                                    READY
STATUS        RESTARTS   AGE
accessdomain-adminserver                                1/1
Terminating   0          27m
accessdomain-oam-policy-mgr1                            1/1
Terminating   0          24m
accessdomain-oam-server1                                1/1
Terminating   0          24m
```

The Administration Server pod and Managed Server pods will move to a `STATUS` of `Terminating`. After a few minutes, run the command again and the pods should have disappeared.

6. Start the domain using the following command:

```
kubectl -n <domain_namespace> patch domains <domain_uid> --type='json' -
p='[{"op": "replace", "path": "/spec/serverStartPolicy", "value":
"IfNeeded" }]'
```

For example:

```
kubectl -n oamns patch domains accessdomain --type='json' -p='[{"op":
"replace", "path": "/spec/serverStartPolicy", "value": "IfNeeded" }]'
```

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

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

For example:

```
kubectl get pods -n oamns
```

The output will look similar to the following:

```
NAME                                                        READY
STATUS      RESTARTS   AGE
accessdomain-introspector-mckp2                             1/1
Running     0          8s
```

The Administration Server pod will start followed by the OAM Managed Servers pods. This process will take several minutes, so keep executing the command until all the pods are running with `READY` status `1/1`:

> **Note**
>
> You can watch the status of the pods by adding the watch flag, for example:
>
> ```
> kubectl get pods -n oamns -w
> ```

```
NAME                                                        READY
STATUS      RESTARTS   AGE
accessdomain-adminserver                                    1/1
Running     0          5m38s
accessdomain-oam-policy-mgr1                                1/1
Running     0          2m51s
accessdomain-oam-server1                                    1/1
Running     0          2m50s
```

10.2 Removing OAM Server from WebLogic Server 14c Default Coherence Cluster

Exclude all Oracle Access Management (OAM) clusters (including Policy Manager and OAM runtime server) from the default WebLogic Server 14c coherence cluster by using the WebLogic Server Administration Console.

In OAM 14.1.2.1.0, server-side session management uses the database and does not require coherence cluster to be established. In some environments, warnings and errors are observed

due to default coherence cluster initialized by WebLogic. To avoid or fix these errors, exclude all of the OAM clusters from default WebLogic Server coherence cluster using the following steps:

1. Connect to the OAM Administration Server in the WebLogic Remote Console.
2. In the left pane of the console, expand **Environment** and select **Coherence Clusters System Resources**.
3. Click **defaultCoherenceCluster** and select the **Members** tab.
4. From **Servers and Clusters**, deselect all OAM clusters (`oam_cluster` and `policy_cluster`).
5. Click **Save**.
6. Click the **Shopping Cart** in the top right of the console, and click **Commit Changes**.

10.3 WebLogic Server Tuning

For production environments, WebLogic Server tuning parameters must be set.

Installing the WebLogic Remote Console Extension

In order to perform the tuning tasks, you must install the latest WebLogic Remote Console extension `console-rest-ext-2.4.15.war` in the OAM *%DOMAIN_HOME%*.

To install the extension, perform the following steps:

1. On the administrative host, download the latest WebLogic Remote Console extension, `console-rest-ext-2.4.15.war`, from the WebLogic Remote Console GitHub Repository:

   ```
   cd $WORKDIR


   wget https://github.com/oracle/weblogic-remote-console/releases/download/
   v2.4.15/console-rest-ext-2.4.15.war
   ```

2. Run the following command to find the `Domain Home`:

   ```
   kubectl describe domains <domainUID> -n <namespace> | grep "Domain Home:"
   ```

   For example:

   ```
   kubectl describe domains accessdomain -n oamns | grep "Domain Home:"
   ```

   The output will look similar to the following:

   ```
     Domain Home:                    /u01/oracle/user_projects/domains/
   accessdomain
   ```

3. Execute the following command to enter a bash shell in the `<domainUID>-adminserver` pod:

   ```
    kubectl exec -it <domainUID>-adminserver -n <domain_namespace> -- /bin/
   bash
   ```

For example:

```
kubectl exec -it accessdomain-adminserver -n oamns -- /bin/bash
```

This will take you inside a bash shell of the pod:

```
[oracle@accessdomain-adminserver oracle]$
```

4. Inside the bash shell run the following commands to create a `management-services-ext` directory:

```
mkdir %DOMAIN_HOME%/management-services-ext
```

Where *%DOMAIN_HOME%* is the path returned earlier.
For example:

```
mkdir /u01/oracle/user_projects/domains/accessdomain/management-services-ext
```

5. Outside of the bash shell, run the following command to copy the `console-rest-ext-2.4.15.war` into the container:

```
kubectl -n <namespace> cp $WORKDIR/console-rest-ext-2.4.15.war <domainUID>-adminserver:%DOMAIN_HOME%/management-services-ext/
```

For example:

```
kubectl -n oamns cp $WORKDIR/console-rest-ext-2.4.15.war accessdomain-adminserver:/u01/oracle/user_projects/domains/accessdomain/management-services-ext/
```

6. Inside the bash shell, run the following to make sure the `console-rest-ext-2.4.15.war` was copied:

```
ls -l %DOMAIN_HOME%/management-services-ext
```

For example:

```
ls /u01/oracle/user_projects/domains/accessdomain/management-services-ext
```

The output will look similar to the following:

```
console-rest-ext-2.4.15.war
```

Add Minimum Thread Constraint and MaxThreadsCount

1. Connect to the OAM Administration Server in the WebLogic Remote Console.
2. In the Home page, select **Monitoring Tree**
3. In the left hand navigation menu, select **Deployments** > **Application Management**.

4. Click **oam_server**.
5. Click **Create Plan** and provide the **Plan Path** as: `/u01/oracle/user_projects/domains/accessdomain/Plan.xml` and click **Done**.
6. In the left hand navigation menu, select **Deployments** -> **Deployment Tasks** and check the plan was successfully deployed (**Progress: Success**).
7. In the left hand navigation menu, select **Deployments** > **Application Management** > **oam_server** > **Deployment Plan (Advanced)**.
8. Select the **Variable Assignments** tab.
9. Type CTRL + F (menu Edit > Find) and Search for: `/weblogic-web-app/work-manager/[name="wm/OAPOverRestWM"]/min-threads-constraint/[name="MinThreadsCount"]/count`.
10. Click the checkbox for that row, and click the **Edit** button (above the table). Change the following and click **Done**:
    - **Value**: `400`
    - **Operation**: `replace`
11. Type CTRL + F (menu Edit > Find) and Search for: `/weblogic-web-app/work-manager/[name="wm/OAPOverRestWM"]/max-threads-constraint/[name="MaxThreadsCount"]/count`.
12. Check the checkbox for that row, and click the **Edit** button (above the table). Change the following and click **Done**:
    - Value: 1000
    - Operation: Replace
13. In the left hand navigation menu, select **Deployments** > **Application Management**.
14. Click the checkbox in the **oam_server** row, and click **Update/Redeploy** and select **Redeploy - Deployment Source and Plan on Server**. Click **Done**.
15. In the left hand navigation menu, select **Deployments** -> **Deployment Tasks** and check the plan was successfully deployed (**Progress: Success**).
16. To check that the values have been updated, access the Oracle Enterprise Manager Fusion Middleware Control console, and perform the following steps:
    - From the WebLogic Domain drop-down menu, select **System MBean Browser**.
    - Click the Search icon, and search for the MBean `wm/OAPOverRestWM`.
    - In the left hand menu, expand `MinThreadsConstraintRuntime` > `MinThreadsCount`, and check the `Count` value is 400.
    - In the left hand menu, expand `MaxThreadsConstraintRuntime` > `MaxThreadsCount` and check the `Count` value is `1000`.

oamDS DataSource Tuning

1. Connect to the OAM Administration Server in the WebLogic Remote Console.
2. Click **Edit Tree** and in the left pane of the console, expand **Services** > **Data Sources** > **oamDS**. In the right hand pane click the **Connection Pool** tab.
3. Change **Initial Capacity**, **Maximum Capacity**, and **Minimum Capacity** to `800`.
4. Click **Save**.
5. Click the **Shopping Cart** in the top right of the console, and click **Commit Changes**.

10.4 Enabling Virtualization

To enable virtualization, perform the following steps:

Enable Virtualization

1. Login to Oracle Enterprise Manager Fusion Middleware Control.
2. Click **WebLogic Domain** > **Security** > **Security Provider Configuration**.
3. Expand **Security Store Provider**.
4. Expand **Identity Store Provider**.
5. Click **Configure**.
6. Add a custom property.
7. Select `virtualize` property with value `true` and click **OK**.
8. Click OK again to persist the change.

10.5 Restarting the Domain

You must restart the Oracle Access Management (OAM) domain for the post configuration changes to take effect.

1. Stop the OAM domain using the following command:

   ```
   kubectl -n <domain_namespace> patch domains <domain_uid> --type='json' -
   p='[{"op": "replace", "path": "/spec/serverStartPolicy", "value":
   "Never" }]'
   ```

   For example:

   ```
   kubectl -n oamns patch domains accessdomain --type='json' -p='[{"op":
   "replace", "path": "/spec/serverStartPolicy", "value": "Never" }]'
   ```

   The output will look similar to the following:

   ```
   domain.weblogic.oracle/accessdomain patched
   ```

2. Check that all the pods are stopped:

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

   For example:

   ```
   kubectl get pods -n oamns
   ```

   The output will look similar to the following:

   ```
   NAME                                                        READY
   STATUS        RESTARTS   AGE
   ```

```
accessdomain-adminserver                                  1/1
Terminating   0          27m
accessdomain-oam-policy-mgr1                              1/1
Terminating   0          24m
accessdomain-oam-server1                                  1/1
Terminating   0          24m
```

The Administration Server pod and Managed Server pods will move to a `STATUS` of `Terminating`. After a few minutes, run the command again and the pods should have disappeared.

3. Start the domain using the following command:

```
kubectl -n <domain_namespace> patch domains <domain_uid> --type='json' -
p='[{"op": "replace", "path": "/spec/serverStartPolicy", "value":
"IfNeeded" }]'
```

For example:

```
kubectl -n oamns patch domains accessdomain --type='json' -p='[{"op":
"replace", "path": "/spec/serverStartPolicy", "value": "IfNeeded" }]'
```

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

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

> **Note**
>
> The `-w` flag allows you watch the status of the pods as they change.

For example:

```
kubectl get pods -n oamns -w
```

The output will look similar to the following:

```
NAME                                                      READY
STATUS      RESTARTS   AGE
accessdomain-introspector-mckp2                           1/1
Running     0          8s
```

After the introspect job has run, the Administration Server pod will start followed by the OAM Managed Servers pods. This process will take several minutes, so keep executing the command until all the pods are running with `READY` status `1/1`:

```
NAME                                                      READY
STATUS      RESTARTS   AGE
accessdomain-adminserver                                  1/1
Running     0          5m38s
accessdomain-oam-policy-mgr1                              1/1
Running     0          2m51s
```

```
accessdomain-oam-server1                                  1/1
Running     0          2m50s
```

11

Validating Basic SSO Flow With Oracle WebGate

In this chapter you validate single-sign on (SSO) works with the Oracle Access Management (OAM) Kubernetes deployment using Oracle WebGate.

The instructions assume you have a running Oracle HTTP Server (OHS), for example `ohs1`, and Oracle WebGate installed either in an on-premises setup, or in a Kubernetes cluster.

If you are deploying OHS on a Kubernetes cluster, see Supported Architectures for Oracle HTTP Server.

The instructions also assume you have a working knowledge of OHS and Oracle WebGate.

The topics in this chapter include:

11.1 Updating the OAM Hostname and Port for the Load Balancer

You must update OAM with the protocol, hostname.domain, and port for your OAM entry point.

For example:

- `https://loadbalancer.example.com` - if OAM URL's are accessed directly via a load balancer URL, with hostname `loadbalancer.example.com` and port `443`.
- `https://ohs.example.com:4443` - if OAM URL's are accessed directly via an OHS URL, with hostname `ohs.example.com` and port `4443`
- `https://oam.example.com:31501` - if OAM URL's are accessed directly via the ingress controller, with hostname `oam.example.com` and port `31501`.

In the following examples change `{HOSTNAME}:${PORT}` accordingly.

1. Launch a browser and access the OAM console (`https://${HOSTNAME}:${PORT}/oamconsole`). Login with the weblogic username and password (`weblogic/<password>`).
2. Navigate to **Configuration** > **Settings (View)** > **Access Manager**.
3. Under Load Balancing modify the **OAM Server Host** and **OAM Server Port**, to point to the hostname.domain of your OAM entry point, for example **loadbalancer.example.com** and **443** respectively. In the **OAM Server Protocol** drop down list select https.
4. Under **WebGate Traffic Load Balancer** modify the **OAM Server Host** and **OAM Server Port**, to point to the hostname.domain of your OAM entry point, for example

`loadbalancer.example.com` and `443` respectively. In the **OAM Server Protocol** drop down list select **https**.

5. Click **Apply**.

11.2 Registering an Oracle WebGate Agent

To register an Oracle WebGate perform the following steps:

1. Launch a browser, and access the OAM console.
2. Navigate to **Application Security** > **Quick Start Wizards** > **SSO Agent Registration**. Register the agent in the usual way.
3. After creating the agent, make sure the **User Defined Parameters** for **OAMRestEndPointHostName**, **OAMRestEndPointPort**, and **OAMServerCommunicationMode** are set to the same values as per Updating the OAM Hostname and Port for the Load Balancer. Click **Apply**.
4. Click **Download** to download the agent zip file and keep in a safe place. This file this will be required in Configuring OHS to Use the Oracle WebGate.

11.3 Configuring the Application Domain

To configure the application domain, perform the following steps:

1. In the OAM console, navigate to **Application Security** > **Application Domains**. Click Search, and click the domain for the agent just created
2. In the Application Domain page, under **Resources**, click **Create** and protect a simple resource, for example `/myapp/**`. Change the following:
   - **Type**: `HTTP`
   - **Host Identifier**: `<your_application_domain>`
   - **Protection Level: Protected**: `Protected`
   - **Authentication Policy**: `Protected Resource Policy`
   - **Authorization Policy**: `Protected Resource Policy`

   > **Note**
   >
   > The purpose of the above is to test a simple page protection works. Once everything is confirmed as working, you can configure your desired resources and policies.

3. Click **Apply**.

11.4 Creating Host Identifiers

To create host identifiers, perform the following steps:

1. In the OAM console, navigate to **Application Security → Access Manager → Host Identifiers**. Click **Search**, and click the **Name** for the agent just created.
2. In the **Host Name Variations**, click **Add**.

3. In the new line that appears, add the details for any URL that will be used for this WebGate. For example if you access a protected URL via `https://loadbalancer.example.com`, then under **Host Name** enter `loadbalancer.example.com` and under **Port** enter `443`.
4. Click **Apply**.
5. Repeat for any other required URL's.

11.5 Configuring OHS to Use the Oracle WebGate

Follow the relevant section depending on whether your are using on-premises Oracle HTTP Server (OHS), or OHS deployed in Kubernetes.

On-premises OHS Installation

In all the examples below, change to the directory path for your installation.

1. Run the following command on the server with OHS and Oracle WebGate installed:

```
cd <OHS_ORACLE_HOME>/webgate/ohs/tools/deployWebGate
```

```
./deployWebGateInstance.sh -w <OHS_DOMAIN_HOME>/config/fmwconfig/
components/OHS/ohs1 -oh <OHS_ORACLE_HOME> -ws ohs
```

The output will look similar to the following:

```
Copying files from WebGate Oracle Home to WebGate Instancedir
```

2. Run the following command to update the OHS configuration files appropriately:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<OHS_ORACLE_HOME>/lib
```

```
cd <OHS_ORACLE_HOME>/webgate/ohs/tools/setup/InstallTools/
```

```
./EditHttpConf -w <OHS_DOMAIN_HOME>/config/fmwconfig/components/OHS/ohs1 -
oh <OHS_ORACLE_HOME>
```

The output will look similar to the following:

```
The web server configuration file was successfully updated
<OHS_DOMAIN_HOME>/config/fmwconfig/components/OHS/ohs1/httpd.conf has been
backed up as <OHS_DOMAIN_HOME>/config/fmwconfig/components/OHS/ohs1/
httpd.conf.ORIG
```

3. Copy the agent zip file downloaded earlier and copy to the OHS server directory , for example: `<OHS_DOMAIN_HOME>/config/fmwconfig/components/OHS/ohs1/webgate/config`. Extract the zip file.
4. Obtain the Certificate Authority (CA) certificate (`cacert.pem`) that signed the certificate for your OAM entry point. Copy to the to the same directory, for example: `<OHS_DOMAIN_HOME>/config/fmwconfig/components/OHS/ohs1/webgate/config`.

> **Note**
>
> Administrators should be aware of the following:
>
> - The CA certificate is the certificate that signed the certificate for your OAM entry point. For example if you access OAM directly via a load balancer, then this is the CA of the load balancer certificate.
> - The file must be renamed to `cacert.pem`.

5. Restart OHS.
6. Access the protected resource, for example `https://ohs.example.com/myapp`, and check you are redirected to the SSO login page. Login and make sure you are redirected successfully to the application.

OHS Deployed on Kubernetes

If deploying OHS on Kubernetes you must copy the agent zip file downloaded earlier to the `$WORKDIR/ohsConfig/webgate/config` directory on your Kubernetes administrative node, and extract it.

For detailed instructions, see Preparing Your OHS Configuration Files

Changing WebGate Agent to use OAP

> **Note**
>
> This section should only be followed if you need to change the OAM/WebGate Agent communication from HTTPS to OAP.

To change the WebGate agent to use OAP:

1. In the OAM Console click Application Security and then Agents.
2. Search for the agent you want modify and select it.
3. In the User Defined Parameters change:
   - `OAMServerCommunicationMode` from `HTTPS` to `OAP`. For example, `OAMServerCommunicationMode=OAP`
   - `OAMRestEndPointHostName=<hostname>` to the hostname the ingress controller is deployed. For example `OAMRestEndPointHostName=oam.example.com`.
4. In the **Server Lists** section click Add to add a new server with the following values:
   - `Access Server`: `Other`
   - `Host Name`: to the hostname the ingress controller is deployed. For example `oam.example.com`
   - `Host Port`: `<oamoap-service NodePort>`

> **Note**
>
> To find the value for `Host Port` run the following:
>
> ```
> kubectl describe svc accessdomain-oamoap-service -n oamns
> ```
>
> The output will look similar to the following:
>
> ```
> Name:                     accessdomain-oamoap-service
> Namespace:                oamns
> Labels:                   <none>
> Annotations:              <none>
> Selector:                 weblogic.clusterName=oam_cluster
> Type:                     NodePort
> IP Families:              <none>
> IP:                       10.100.202.44
> IPs:                      10.100.202.44
> Port:                     <unset>  5575/TCP
> TargetPort:               5575/TCP
> NodePort:                 <unset>  30540/TCP
> Endpoints:                10.244.5.21:5575,10.244.6.76:5575
> Session Affinity:         None
> External Traffic Policy:  Cluster
> Events:                   <none>
> ```
>
> In the example above the `NodePort` is `30540`.

5. Delete all servers in **Server Lists** except for the one just created, and click **Apply**.
6. Click **Download** to download the webgate zip file. Copy the zip file to the desired WebGate.
7. Delete the cache from `<OHS_DOMAIN_HOME>/servers/ohs1/cache` and restart **Oracle HTTP Server**.

Part III

Administering Oracle Access Management on Kubernetes

Administer Oracle Access Management on Kubernetes

This section contains the following chapters:

12

Scaling OAM Pods

As Oracle Access Management (OAM) domains use the WebLogic Kubernetes Operator, domain life cycle operations are managed using the WebLogic Kubernetes Operator itself.

> **Note**
>
> The instructions below are for starting, stopping, and scaling servers up or down manually. If you wish to use autoscaling, see Kubernetes Horizontal Pod Autoscaler. Please note, if you have enabled autoscaling, and then decide to run the commands manually, it is recommended to delete the autoscaler before running the commands in the topics below.

For more detailed information refer to Domain Life Cycle in the WebLogic Kubernetes Operator documentation.

This chapter includes the following topics:

12.1 Viewing Existing OAM Instances

The default Oracle Access Management (OAM) deployment starts the Administration Server (`AdminServer`), one OAM Managed Server (`oam_server1`) and one OAM Policy Manager server (`oam_policy_mgr1`).

The deployment also creates, but doesn't start, four extra OAM Managed Servers (`oam-server2` to `oam-server5`) and four more OAM Policy Manager servers (`oam_policy_mgr2` to `oam_policy_mgr5`).

All these servers are visible in the WebLogic Remote Console by navigating to **Environment** > **Servers**.

Run the following command to view the pods in the OAM deployment:

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

For example:

```
kubectl get pods -n oamns
```

The output should look similar to the following:

```
NAME                                                  READY   STATUS
RESTARTS   AGE
accessdomain-adminserver                              1/1     Running
0          3h29m
accessdomain-oam-policy-mgr1                          1/1     Running
0          3h21m
accessdomain-oam-server1                              1/1     Running
0          3h21m
```

12.2 Scaling Up OAM Servers

The number of Oracle Access Management (OAM) managed servers running, or policy managed servers running, is dependent on the `replicas` parameter configured for the `oam-cluster` and `policy_cluster` respectively.

To start more OAM servers perform the following steps:

1. Run the following command to edit the cluster
   - For OAM managed servers:

     ```
     kubectl edit cluster accessdomain-oam-cluster -n <domain_namespace>
     ```

     For example:

     ```
     kubectl edit cluster accessdomain-oam-cluster -n oamns
     ```

   - For OAM policy manager servers:

     ```
     kubectl edit cluster accessdomain-policy-cluster -n <domain_namespace>
     ```

     For example:

     ```
     kubectl edit cluster accessdomain-policy-cluster -n oamns
     ```

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

2. In the edit session, search for `spec:`, and then look for the `replicas` parameter under `clusterName: <cluster>`.
   By default the replicas parameter, for both OAM managed servers and policy manager servers, is set to "1" hence one OAM managed server and one policy manager server is started (`oam_server1` and `oam-policy-mgr1` respectively):
   - For `oam_cluster`:

     ```
     ...
     spec:
       clusterName: oam_cluster
     ```

```
    replicas: 1
    serverPod:
      env:
      - name: USER_MEM_ARGS
        value: -XX:+UseContainerSupport -Djava.security.egd=file:/dev/./
urandom -Xms8192m
        -Xmx8192m
...
```

- For `policy_cluster`:

```
...
spec:
  clusterName: policy_cluster
  replicas: 1
  serverService:
    precreateService: true
...
```

3. To start more OAM managed servers or policy manager servers, increase the `replicas` value as desired.
   In the example below, two more OAM managed servers (`oam-server2` and `oam-server3`) will be started by setting `replicas` to "3" for the `oam_cluster`:

```
...
spec:
  clusterName: oam_cluster
  replicas: 3
  serverPod:
    env:
    - name: USER_MEM_ARGS
      value: -XX:+UseContainerSupport -Djava.security.egd=file:/dev/./
urandom -Xms8192m
        -Xmx8192m
...
```

4. Save the file and exit (`:wq!`).
   The output will look similar to the following:

```
cluster.weblogic.oracle/accessdomain-oam-cluster edited
```

5. Run the following command to view the pods:

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

   For example:

```
kubectl get pods -n oamns
```

   The output will look similar to the following:

```
NAME                                                      READY
STATUS      RESTARTS   AGE
accessdomain-adminserver                                  1/1
```

```
Running     0          3h33m
accessdomain-oam-policy-mgr1                          1/1
Running     0          3h25m
accessdomain-oam-server1                              1/1
Running     0          3h25m
accessdomain-oam-server2                              0/1
Running     0          9s
accessdomain-oam-server3                              0/1
Pending     0          9s
```

Two new pods (`accessdomain-oam-server2` and `accessdomain-oam-server3`) are started, but currently have a `READY` status of `0/1`. This means `oam_server2` and `oam_server3` are not currently running but are in the process of starting.
The servers will take several minutes to start so keep executing the command until `READY` shows `1/1`:

> **Note**
>
> Alternatively, you can add the watch flag, `-w`, which allows you watch the status of the pods as they change.

```
NAME                                                  READY
STATUS      RESTARTS   AGE
accessdomain-adminserver                              1/1
Running     0          3h37m
accessdomain-oam-policy-mgr1                          1/1
Running     0          3h29m
accessdomain-oam-server1                              1/1
Running     0          3h29m
accessdomain-oam-server2                              1/1
Running     0          3m45s
accessdomain-oam-server3                              1/1
Running     0          3m45s
```

To check what is happening during server startup when `READY` is `0/1`, run the following command to view the log of the pod that is starting:

```
kubectl logs <pod> -n <domain_namespace>
```

For example:

```
kubectl logs accessdomain-oam-server3 -n oamns
```

12.3 Scaling Down OAM Servers

Scaling down Oracle Access Management (OAM) servers is performed in exactly the same way as in Scaling Up OAM Servers except the `replicaCount` is reduced to the required number of servers.

To stop one or more OAM servers, perform the following steps:

1. Run the following command to edit the cluster:
   - For OAM managed servers:

     ```
     kubectl edit cluster accessdomain-oam-cluster -n <domain_namespace>
     ```

     For example:

     ```
     kubectl edit cluster accessdomain-oam-cluster -n oamns
     ```

   - For OAM policy manager servers:

     ```
     kubectl edit cluster accessdomain-policy-cluster -n <domain_namespace>
     ```

     For example:

     ```
     kubectl edit cluster accessdomain-policy-cluster -n oamns
     ```

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

2. In the edit session, search for `spec:`, and then look for the `replicas` parameter under `clusterName: <cluster>`.
   In the example below `replicas` is set to "3", hence three OAM managed servers are started (`access-domain-oam_server1` - `access-domain-oam_server3`):

   ```
   ...
   spec:
     clusterName: oam_cluster
     replicas: 3
     serverPod:
       env:
       - name: USER_MEM_ARGS
         value: -XX:+UseContainerSupport -Djava.security.egd=file:/dev/./
   urandom -Xms8192m
           -Xmx8192m
   ...
   ```

3. To stop OAM servers, decrease the replicas value as desired. In the example below, two managed servers will be stopped by setting `replicas` to "1":

   ```
   ...
   spec:
     clusterName: oam_cluster
     replicas: 1
     serverPod:
       env:
       - name: USER_MEM_ARGS
         value: -XX:+UseContainerSupport -Djava.security.egd=file:/dev/./
   urandom -Xms8192m
   ```

```
        -Xmx8192m
...
```

4. Save the file and exit (`:wq!`).
   The output will look similar to the following:

   ```
   cluster.weblogic.oracle/accessdomain-oam-cluster edited
   ```

5. Run the following command to view the pods:

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

   For example:

   ```
   kubectl get pods -n oamns
   ```

   The output will look similar to the following:

   ```
   NAME                                                  READY
   STATUS        RESTARTS   AGE
   accessdomain-adminserver                              1/1
   Running       0          3h45m
   accessdomain-oam-policy-mgr1                          1/1
   Running       0          3h37m
   accessdomain-oam-server1                              1/1
   Running       0          3h37m
   accessdomain-oam-server2                              1/1
   Running       0          6m14s
   accessdomain-oam-server3                              1/1
   Terminating   0          6m14s
   ```

   One pod now has a `STATUS` of `Terminating` (`accessdomain-oam-server3`).

   The server will take a minute or two to stop. Once terminated the other pod (`accessdomain-oam-server2`) will move to `Terminating` and then stop.

   The servers will take several minutes to stop so keep executing the command until the pods have disappeared:

   ```
   NAME                                          READY   STATUS
   RESTARTS   AGE
   accessdomain-adminserver                              1/1
   Running     0          3h48m
   accessdomain-oam-policy-mgr1                          1/1
   Running     0          3h40m
   accessdomain-oam-server1                              1/1
   Running     0          3h40m
   ```

12.4 Stopping the OAM Domain

Stopping the Oracle Access Management (OAM) domain shuts down all the OAM servers and the Administration Server in one operation.

To stop the OAM domain:

1. Run the following kubectl command to edit the domain:

   ```
   kubectl edit domain <domain_uid> -n <domain_namespace>
   ```

   For example:

   ```
   kubectl edit domain accessdomain -n oamns
   ```

2. In the edit session, search for `serverStartPolicy: IfNeeded` under the domain spec:

   ```
   ...
      volumeMounts:
      - mountPath: /u01/oracle/user_projects/domains
        name: weblogic-domain-storage-volume
      volumes:
      - name: weblogic-domain-storage-volume
        persistentVolumeClaim:
          claimName: accessdomain-domain-pvc
    serverStartPolicy: IfNeeded
   ...
   ```

3. Change `serverStartPolicy: IfNeeded` to `Never` as follows:

   ```
   ...
      volumeMounts:
      - mountPath: /u01/oracle/user_projects/domains
        name: weblogic-domain-storage-volume
      volumes:
      - name: weblogic-domain-storage-volume
        persistentVolumeClaim:
          claimName: accessdomain-domain-pvc
    serverStartPolicy: Never
   ...
   ```

4. Save the file and exit (`:wq!`).
5. Run the following command to view the pods:

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

   For example:

   ```
   kubectl get pods -n oamns
   ```

   The output will look similar to the following:

   ```
   NAME                                                    READY
   STATUS        RESTARTS   AGE
   accessdomain-adminserver                                1/1
   Terminating   0          3h52m
   accessdomain-oam-policy-mgr1                            1/1
   Terminating   0          3h44m
   ```

```
accessdomain-oam-server1                                          1/1
Terminating   0          3h44m
```

The Administration Server pods and OAM server pods will move to a `STATUS` of `Terminating`. After a few minutes, run the command again and the pods should have disappeared.

6. To start the Administration Server and Managed Servers up again, repeat the previous steps but change `serverStartPolicy: Never` to `IfNeeded` as follows:

```
...
   volumeMounts:
   - mountPath: /u01/oracle/user_projects/domains
     name: weblogic-domain-storage-volume
   volumes:
   - name: weblogic-domain-storage-volume
     persistentVolumeClaim:
       claimName: accessdomain-domain-pvc
 serverStartPolicy: IfNeeded
...
```

7. Run the following command to view the pods:

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

For example:

```
kubectl get pods -n oamns
```

The output will look similar to the following:

```
NAME                                                              READY
STATUS        RESTARTS   AGE
accessdomain-introspector-jwqxw                                   1/1
Running     0          10s
```

The introspect job will start, followed by the Administration Server pod, and then the OAM server pods. This process will take several minutes, so keep executing the command until all the pods are running with `READY` status `1/1`:

> **Note**
>
> Alternatively, you can add the watch flag, `-w`, which allows you watch the status of the pods as they change.

```
NAME                                                              READY
STATUS      RESTARTS   AGE
accessdomain-adminserver                                          1/1
Running     0          10m
accessdomain-oam-policy-mgr1                                      1/1
Running     0          7m35s
```

```
accessdomain-oam-server1                                          1/1
Running     0          7m35s
```

12.5 Domain Life Cycle Scripts

The WebLogic Kubernetes Operator provides sample scripts to start up or shut down a specific Managed Server or cluster in a deployed domain, or the entire deployed domain.

> **Note**
>
> Prior to running these scripts, you must have previously created and deployed the domain.

The scripts are located in the `$WORKDIR/kubernetes/domain-lifecycle` directory.

For more information, see Sample Lifecycle Management Scripts.

13

WLST Administration Operations

This chapter contains the following topics:

13.1 Connecting to OAM via WLST

In order to use WLST to administer the Oracle Access Management (OAM) domain, use must use a helper pod.

1. Check to see if the helper pod exists by running:

   ```
   kubectl get pods -n <domain_namespace> | grep helper
   ```

   For example:

   ```
   kubectl get pods -n oamns | grep helper
   ```

   The output should look similar to the following:

   ```
   helper                                   1/1     Running     0          26h
   ```

   If the helper pod doesn't exist, run the following:

   - If using Oracle Container Registry or your own container registry for the OAM container image:

     ```
     kubectl run --image=<image_name-from-registry>:<tag> \
     --image-pull-policy="IfNotPresent" \
     --overrides='{"apiVersion": "v1", "spec":{"imagePullSecrets": [{"name":
     "orclcred"}]}}' \
     helper -n <domain_namespace> \
     -- sleep infinity
     ```

     For example:

     ```
     kubectl run --image=container-registry.oracle.com/middleware/
     oam_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD> \
     --image-pull-policy="IfNotPresent" \
     --overrides='{"apiVersion": "v1","spec":{"imagePullSecrets": [{"name":
     "orclcred"}]}}' \
     ```

```
helper -n oamns \
-- sleep infinity
```

- If you are not using a container registry and have loaded the image on each of the worker nodes, run the following command:

  ```
  kubectl run helper --image <image>:<tag> -n oamns -- sleep infinity
  ```

  For example:

  ```
  kubectl run helper --image oracle/oam_cpu:14.1.2.1-jdk17-ol8-<YYDDMM> -
  n oamns -- sleep infinity
  ```

  The output will look similar to the following:

  ```
  pod/helper created
  ```

2. Run the following command to start a bash shell in the helper pod:

   ```
   kubectl exec -it helper -n <domain_namespace> -- /bin/bash
   ```

   For example:

   ```
   kubectl exec -it helper -n oamns -- /bin/bash
   ```

   This will take you into a bash shell in the running helper pod:

   ```
   [oracle@helper ~]$
   ```

3. Inside the helper pod, connect to WLST using the following command:

   ```
   cd $ORACLE_HOME/oracle_common/common/bin
   ```

   ```
   ./wlst.sh
   ```

   The output will look similar to the following:

   ```
   Initializing WebLogic Scripting Tool (WLST) ...

   Jython scans all the jar files it can find at first startup. Depending on
   the system, this process may take a few minutes to complete, and WLST may
   not return a prompt right away.

   Welcome to WebLogic Server Administration Scripting Shell

   Type help() for help on available commands

   wls:/offline>
   ```

4. To access t3 for the Administration Server connect as follows:

```
connect('weblogic','<password>','t3://accessdomain-adminserver:7001')
```

The output will look similar to the following:

```
Connecting to t3://accessdomain-adminserver:7001 with userid weblogic ...
Successfully connected to Admin Server "AdminServer" that belongs to
domain "accessdomain".

Warning: An insecure protocol was used to connect to the server.
To ensure on-the-wire security, the SSL port or Admin port should be used
instead.

wls:/accessdomain/serverConfig/>
```

Or to access t3 for the OAM Cluster service, connect as follows:

```
connect('weblogic','<password>','t3://accessdomain-cluster-oam-
cluster:14100')
```

The output will look similar to the following:

```
Connecting to t3://accessdomain-cluster-oam-cluster:14100 with userid
weblogic ...
Successfully connected to managed Server "oam_server1" that belongs to
domain "accessdomain".

Warning: An insecure protocol was used to connect to the server.
To ensure on-the-wire security, the SSL port or Admin port should be used
instead.

wls:/accessdomain/serverConfig/>
```

13.2 Sample WLST Operations

The following are some sample WLST operations that can be performed against the Oracle Access Management (OAM) domain.

For a full list of WLST operations, see WebLogic Server WLST Online and Offline Command Reference.

Display Servers

1. Run the following commands to display the server:

```
wls:/accessdomain/serverConfig/> cd('/Servers')


wls:/accessdomain/serverConfig/Servers> ls()
```

The output will look similar to the following:

```
dr--   AdminServer
dr--   oam_policy_mgr1
dr--   oam_policy_mgr2
dr--   oam_policy_mgr3
dr--   oam_policy_mgr4
dr--   oam_policy_mgr5
dr--   oam_server1
dr--   oam_server2
dr--   oam_server3
dr--   oam_server4
dr--   oam_server5

wls:/accessdomain/serverConfig/Servers>
```

Configure Logging for Managed Servers

1. Run the following command to change to the `domainRuntime` tree:

```
wls:/accessdomain/serverConfig/> domainRuntime()
```

The output will look similar to the following:

```
Location changed to domainRuntime tree. This is a read-only tree
with DomainMBean as the root MBean.
For more help, use help('domainRuntime')

wls:/accessdomain/domainRuntime/>

wls:/accessdomain/domainRuntime/>
listLoggers(pattern="oracle.oam.*",target="oam_server1")
-----------------------------------------+-----------------
Logger                                   | Level
-----------------------------------------+-----------------
oracle.oam                               | <Inherited>
oracle.oam.admin.foundation.configuration | <Inherited>
oracle.oam.admin.service.config          | <Inherited>
oracle.oam.agent                         | <Inherited>
oracle.oam.agent-default                 | <Inherited>
oracle.oam.audit                         | <Inherited>
oracle.oam.binding                       | <Inherited>
oracle.oam.certvalidation                | <Inherited>
oracle.oam.certvalidation.mbeans         | <Inherited>
oracle.oam.common.healthcheck            | <Inherited>
oracle.oam.common.runtimeent             | <Inherited>
oracle.oam.commonutil                    | <Inherited>
oracle.oam.config                        | <Inherited>
oracle.oam.controller                    | <Inherited>
oracle.oam.default                       | <Inherited>
oracle.oam.diagnostic                    | <Inherited>
oracle.oam.engine.authn                  | <Inherited>
oracle.oam.engine.authz                  | <Inherited>
oracle.oam.engine.policy                 | <Inherited>
oracle.oam.engine.ptmetadata             | <Inherited>
```

```
oracle.oam.engine.session                | <Inherited>
oracle.oam.engine.sso                    | <Inherited>
oracle.oam.esso                          | <Inherited>
oracle.oam.extensibility.lifecycle       | <Inherited>
oracle.oam.foundation.access             | <Inherited>
oracle.oam.idm                           | <Inherited>
oracle.oam.install                       | <Inherited>
oracle.oam.install.bootstrap             | <Inherited>
oracle.oam.install.mbeans                | <Inherited>
oracle.oam.ipf.rest.api                  | <Inherited>
oracle.oam.oauth                         | <Inherited>
oracle.oam.plugin                        | <Inherited>
oracle.oam.proxy.oam                     | <Inherited>
oracle.oam.proxy.oam.workmanager         | <Inherited>
oracle.oam.proxy.opensso                 | <Inherited>
oracle.oam.pswd.service.provider         | <Inherited>
oracle.oam.replication                   | <Inherited>
oracle.oam.user.identity.provider        | <Inherited>
wls:/accessdomain/domainRuntime/>
```

2. Set the log level to `TRACE:32`:

```
wls:/accessdomain/domainRuntime/>
setLogLevel(target='oam_server1',logger='oracle.oam',level='TRACE:32',persi
st="1",addLogger=1)
```

3. Run the following command to view the log level was updated:

```
wls:/accessdomain/domainRuntime/>
listLoggers(pattern="oracle.oam.*",target="oam_server1")
```

The output will look similar to the following:

```
-----------------------------------------+-----------------
Logger                                   | Level
-----------------------------------------+-----------------
oracle.oam                               | TRACE:32
oracle.oam.admin.foundation.configuration | <Inherited>
oracle.oam.admin.service.config          | <Inherited>
oracle.oam.agent                         | <Inherited>
oracle.oam.agent-default                 | <Inherited>
oracle.oam.audit                         | <Inherited>
oracle.oam.binding                       | <Inherited>
oracle.oam.certvalidation                | <Inherited>
oracle.oam.certvalidation.mbeans         | <Inherited>
oracle.oam.common.healthcheck            | <Inherited>
oracle.oam.common.runtimeent             | <Inherited>
oracle.oam.commonutil                    | <Inherited>
oracle.oam.config                        | <Inherited>
oracle.oam.controller                    | <Inherited>
oracle.oam.default                       | <Inherited>
oracle.oam.diagnostic                    | <Inherited>
oracle.oam.engine.authn                  | <Inherited>
oracle.oam.engine.authz                  | <Inherited>
```

```
oracle.oam.engine.policy                | <Inherited>
oracle.oam.engine.ptmetadata            | <Inherited>
oracle.oam.engine.session               | <Inherited>
oracle.oam.engine.sso                   | <Inherited>
oracle.oam.esso                         | <Inherited>
oracle.oam.extensibility.lifecycle      | <Inherited>
oracle.oam.foundation.access            | <Inherited>
oracle.oam.idm                          | <Inherited>
oracle.oam.install                      | <Inherited>
oracle.oam.install.bootstrap            | <Inherited>
oracle.oam.install.mbeans               | <Inherited>
oracle.oam.ipf.rest.api                 | <Inherited>
oracle.oam.oauth                        | <Inherited>
oracle.oam.plugin                       | <Inherited>
oracle.oam.proxy.oam                    | <Inherited>
oracle.oam.proxy.oam.workmanager        | <Inherited>
oracle.oam.proxy.opensso                | <Inherited>
oracle.oam.pswd.service.provider        | <Inherited>
oracle.oam.replication                  | <Inherited>
oracle.oam.user.identity.provider       | <Inherited>
wls:/accessdomain/domainRuntime/>
```

4. Verify that `TRACE:32` log level is set by connecting to the Administration Server and viewing the logs:

   ```
   kubectl exec -it accessdomain-adminserver -n <domain_namespace> -- /bin/
   bash
   ```

   For example:

   ```
   kubectl exec -it accessdomain-adminserver -n oamns -- /bin/bash
   ```

   This will take you into a bash shell in the Administration Server pod:

   ```
   [oracle@accessdomain-adminserver oracle]$
   ```

   a. Navigate to the log directory:

      ```
      cd /u01/oracle/user_projects/domains/accessdomain/servers/oam_server1/
      logs
      ```

   b. View the `oam_server1-diagnostic.log`:

      ```
      tail oam_server1-diagnostic.log
      ```

      The output will look similar to the following:

      ```
      [<DATE>] [oam_server1] [TRACE:32] [] [oracle.oam.config] [tid:
      Configuration Store Observer] [userId: <anonymous>] [ecid: 8b3ac37b-
      c7cf-46dd-aeee-5ed67886be21-0000000b,0:1795] [APP: oam_server]
      [partition-name: DOMAIN] [tenant-name: GLOBAL] [SRC_CLASS:
      oracle.security.am.admin.config.util.observable.ObservableConfigStore$St
      oreWatcher] [SRC_METHOD: run] Start of run before start of detection at
      ```

```
1,635,848,774,793. Detector:
oracle.security.am.admin.config.util.observable.DbStoreChangeDetector:Da
tabase configuration store:DSN:jdbc/oamds. Monitor: { StoreMonitor:
{ disabled: 'false' } }
[<DATE>] [oam_server1] [TRACE] [] [oracle.oam.config] [tid:
Configuration Store Observer] [userId: <anonymous>] [ecid: 8b3ac37b-
c7cf-46dd-aeee-5ed67886be21-0000000b,0:1795] [APP: oam_server]
[partition-name: DOMAIN] [tenant-name: GLOBAL] [SRC_CLASS:
oracle.security.am.admin.config.util.store.StoreUtil] [SRC_METHOD:
getContainerProperty] Configuration property CONFIG_HISTORY not
specified
[<DATE>] [oam_server1] [TRACE] [] [oracle.oam.config] [tid:
Configuration Store Observer] [userId: <anonymous>] [ecid: 8b3ac37b-
c7cf-46dd-aeee-5ed67886be21-0000000b,0:1795] [APP: oam_server]
[partition-name: DOMAIN] [tenant-name: GLOBAL] [SRC_CLASS:
oracle.security.am.admin.config.util.store.StoreUtil] [SRC_METHOD:
getContainerProperty] Configuration property CONFIG not specified
[<DATE>] [oam_server1] [TRACE:32] [] [oracle.oam.config] [tid:
Configuration Store Observer] [userId: <anonymous>] [ecid: 8b3ac37b-
c7cf-46dd-aeee-5ed67886be21-0000000b,0:1795] [APP: oam_server]
[partition-name: DOMAIN] [tenant-name: GLOBAL] [SRC_CLASS:
oracle.security.am.admin.config.util.store.DbStore] [SRC_METHOD:
getSelectSQL] SELECT SQL:SELECT  version  from  IDM_OBJECT_STORE  where
id = ? and version = (select max(version) from  IDM_OBJECT_STORE  where
id = ?)
[<DATE>] [oam_server1] [TRACE] [] [oracle.oam.config] [tid:
Configuration Store Observer] [userId: <anonymous>] [ecid: 8b3ac37b-
c7cf-46dd-aeee-5ed67886be21-0000000b,0:1795] [APP: oam_server]
[partition-name: DOMAIN] [tenant-name: GLOBAL] [SRC_CLASS:
oracle.security.am.admin.config.util.store.DbStore] [SRC_METHOD: load]
Time (ms) to load key CONFIG:-1{FIELD_TYPES=INT, SELECT_FIELDS=SELECT
version  from  IDM_OBJECT_STORE }:4
```

13.3 Performing WLST Administration via SSL

The following steps show how to perform WLST administration via SSL:

1. By default the SSL port is not enabled for the Administration Server or Oracle Access Management (OAM) managed servers. To configure the SSL port for the Administration Server and managed servers:
   a. Login to WebLogic Remote Console.
   b. Click **Edit Tree** and in the left-hand navigation menu navigate to **Environment** > **Servers** > `<server_name>` and click on the **General** tab.
   c. Check the SSL Listen Port Enabled button and provide the SSL Port (For `AdminServer`: `7002` and for `oam_server1`): `14101`
   d. Click **Save**.
   e. Click the **Shopping Cart** and select **Commit Changes**.

> **Note**
>
> If configuring the OAM managed servers for SSL you must enable SSL on the same port for all servers (`oam_server1` through `oam_server5`).

2. Create a `myscripts` directory as follows:

```
cd $WORKDIR/kubernetes


mkdir myscripts


cd myscripts
```

3. Create a `<domain_uid>-adminserver-ssl.yaml` file in the `myscripts` directory for the OAM administration server:

> **Note**
>
> Update the `domainName`, `domainUID` and `namespace` based on your environment. For example:

```
apiVersion: v1
kind: Service
metadata:
  labels:
    serviceType: SERVER
    weblogic.domainName: accessdomain
    weblogic.domainUID: accessdomain
    weblogic.resourceVersion: domain-v2
    weblogic.serverName: AdminServer
  name: accessdomain-adminserverssl
  namespace: oamns
spec:
  clusterIP: None
  ports:
  - name: default
    port: 7002
    protocol: TCP
    targetPort: 7002
  selector:
    weblogic.createdByOperator: "true"
    weblogic.domainUID: accessdomain
    weblogic.serverName: AdminServer
  type: ClusterIP
```

4. Create a `<domain_uid>-oamcluster-ssl.yaml` for the OAM managed server:

```
apiVersion: v1
kind: Service
metadata:
  labels:
```

```
    serviceType: SERVER
    weblogic.domainName: accessdomain
    weblogic.domainUID: accessdomain
    weblogic.resourceVersion: domain-v2
  name: accessdomain-oamcluster-ssl
  namespace: oamns
spec:
  clusterIP: None
  ports:
  - name: default
    port: 14101
    protocol: TCP
    targetPort: 14101
  selector:
    weblogic.clusterName: oam_cluster
    weblogic.createdByOperator: "true"
    weblogic.domainUID: accessdomain
  type: ClusterIP
```

5. Apply the template using the following command for the administration server:

   ```
   kubectl apply -f <domain_uid>-adminserver-ssl.yaml
   ```

   For example:

   ```
   kubectl apply -f accessdomain-adminserver-ssl.yaml
   ```

   The output will look similar to the following:

   ```
   service/accessdomain-adminserverssl created
   ```

6. Apply the template using the following command for the OAM managed server:

   ```
   kubectl apply -f <domain_uid>-oamcluster-ssl.yaml
   ```

   For example:

   ```
   kubectl apply -f accessdomain-oamcluster-ssl.yaml
   ```

   The output will look similar to the following:

   ```
   service/accessdomain-oamcluster-ssl created
   ```

7. Validate that the Kubernetes services to access SSL ports are created successfully:

   ```
   kubectl get svc -n <domain_namespace> |grep ssl
   ```

   For example:

   ```
   kubectl get svc -n oamns |grep ssl
   ```

The output will look similar to the following:

```
accessdomain-adminserverssl           ClusterIP   None
<none>        7002/TCP                     102s
accessdomain-oamcluster-ssl           ClusterIP   None
<none>        14101/TCP                    35s
```

8. Inside the bash shell of the running helper pod, run the following:

```
export WLST_PROPERTIES="-
Dweblogic.security.SSL.ignoreHostnameVerification=true -
Dweblogic.security.TrustKeyStore=DemoTrust"


cd /u01/oracle/oracle_common/common/bin


./wlst.sh
```

The output will look similar to the following:

```
Initializing WebLogic Scripting Tool (WLST) ...

Welcome to WebLogic Server Administration Scripting Shell

Type help() for help on available commands
wls:/offline>
```

To connect to the Administration Server t3s service:

```
connect('weblogic','<password>','t3s://accessdomain-adminserverssl:7002')
```

The output will look similar to the following:

```
Connecting to t3s://accessdomain-adminserverssl:7002 with userid
weblogic ...
<<DATE>> <Info> <Security> <BEA-090905> <Disabling the CryptoJ JCE
Provider self-integrity check for better startup performance. To enable
this check, specify -
Dweblogic.security.allowCryptoJDefaultJCEVerification=true.>
<<DATE>> <Info> <Security> <BEA-090906> <Changing the default Random
Number Generator in RSA CryptoJ from ECDRBG128 to HMACDRBG. To disable
this change, specify -Dweblogic.security.allowCryptoJDefaultPRNG=true.>
<<DATE>> <Info> <Security> <BEA-090909> <Using the configured custom SSL
Hostname Verifier implementation:
weblogic.security.utils.SSLWLSHostnameVerifier$NullHostnameVerifier.>
Successfully connected to Admin Server "AdminServer" that belongs to
domain "accessdomain".

wls:/accessdomain/serverConfig/>
```

To connect to the OAM Managed Server t3s service:

```
connect('weblogic','<password>','t3s://accessdomain-oamcluster-ssl:14101')
```

The output will look similar to the following:

```
Connecting to t3s://accessdomain-oamcluster-ssl:14101 with userid
weblogic ...
<<DATE>> <Info> <Security> <BEA-090905> <Disabling the CryptoJ JCE
Provider self-integrity check for better startup performance. To enable
this check, specify -
Dweblogic.security.allowCryptoJDefaultJCEVerification=true.>
<<DATE>> <Info> <Security> <BEA-090906> <Changing the default Random
Number Generator in RSA CryptoJ from ECDRBG128 to HMACDRBG. To disable
this change, specify -Dweblogic.security.allowCryptoJDefaultPRNG=true.>
<<DATE>> <Info> <Security> <BEA-090909> <Using the configured custom SSL
Hostname Verifier implementation:
weblogic.security.utils.SSLWLSHostnameVerifier$NullHostnameVerifier.>
Successfully connected to managed Server "oam_server1" that belongs to
domain "accessdomain".
```

14

Logging and Visualization

This chapter describes how to publish WebLogic Kubernetes Operator and WebLogic server logs into Elasticsearch, and interact with them in Kibana.

The ELK stack consists of Elasticsearch, Logstash, and Kibana. Using ELK you can gain insights in real-time from the log data from your applications.

Elasticsearch is a distributed, RESTful search and analytics engine capable of solving a growing number of use cases. As the heart of the Elastic Stack, it centrally stores your data so you can discover the expected and uncover the unexpected.

Logstash is an open source, server-side data processing pipeline that ingests data from a multitude of sources simultaneously, transforms it, and then sends it to your favorite “stash.”

Kibana lets you visualize your Elasticsearch data and navigate the Elastic Stack. It gives you the freedom to select the way you give shape to your data, and you don’t always have to know what you’re looking for.

This chapter includes the following topics:

14.1 Installing Elasticsearch and Kibana

If you do not already have a centralized Elasticsearch (ELK) stack then you must configure this first.

For details on how to configure the ELK stack, see Installing the Monitoring and Visualization Software.

14.2 Creating the Logstash Pod

Topics in the section include:

14.2.1 Variables Used in This Section

In order to create the logstash pod, you must create several yaml files. These files contains variables which you must substitute with variables applicable to your ELK environment.

Most of the values for the variables will be based on your ELK deployment as per Installing the Monitoring and Visualization Software.

The table below outlines the variables and values you must set:

Variable	Sample Value	Description
<ELK_VER>	`8.3.1`	The version of logstash you want to install.
<ELK_SSL>	true	If SSL is enabled for ELK set the value to `true`, or if `NON-SSL` set to `false`. This value must be lowercase.
<ELK_HOSTS>	`https://elasticsearch.example.com:9200`	The URL for sending logs to Elasticsearch. HTTP if `NON-SSL` is used.
<ELK_USER>	`logstash_internal`	The name of the user for logstash to access Elasticsearch.
<ELK_PASSWORD>	`password`	The password for *<ELK_USER>*.
<ELK_APIKEY>	`apikey`	The API key details.

You will also need the BASE64 version of the Certificate Authority (CA) certificate(s) that signed the certificate of the Elasticsearch server. If using a self-signed certificate, this is the self signed certificate of the Elasticsearch server. See Copying the Elasticsearch Certificate, for details on how to get the correct certificate. In the example below the certificate is called `elk.crt`.

14.2.2 Creating a Kubernetes Secret for ELK

1. Create a Kubernetes secret for Elasticsearch using the API Key or Password:
   a. If ELK uses an API Key for authentication:

      ```
      kubectl create secret generic elasticsearch-pw-elastic -n
      <domain_namespace> --from-literal password=<ELK_APIKEY>
      ```

      For example:

      ```
      kubectl create secret generic elasticsearch-pw-elastic -n oamns --from-
      literal password=<ELK_APIKEY>
      ```

      The output will look similar to the following:

      ```
      secret/elasticsearch-pw-elastic created
      ```

   b. If ELK uses a password for authentication:

      ```
      kubectl create secret generic elasticsearch-pw-elastic -n
      <domain_namespace> --from-literal password=<ELK_PASSWORD>
      ```

For example:

```
kubectl create secret generic elasticsearch-pw-elastic -n oudns --from-
literal password=<ELK_PASSWORD>
```

The output will look similar to the following:

```
secret/elasticsearch-pw-elastic created
```

> **Note**
>
> It is recommended that the ELK Stack is created with authentication enabled. If no authentication is enabled you may create a secret using the values above.

2. Create a Kubernetes secret to access the required images on hub.docker.com:

> **Note**
>
> Before executing the command below, you must first have a user account on hub.docker.com.

```
kubectl create secret docker-registry "dockercred" --docker-
server="https://index.docker.io/v1/" \
--docker-username="<DOCKER_USER_NAME>" \
--docker-password=<DOCKER_PASSWORD> --docker-email=<DOCKER_EMAIL_ID> \
--namespace=<domain_namespace>
```

For example:

```
kubectl create secret docker-registry "dockercred" --docker-
server="https://index.docker.io/v1/" \
--docker-username="user@example.com" \
--docker-password=password --docker-email=user@example.com \
--namespace=oamns
```

The output will look similar to the following:

```
secret/dockercred created
```

14.2.3 Finding Required Domain Details

The YAML files for ELK require certain domain values to be added.

1. Run the following command to get the `mountPath` of your domain:

```
kubectl describe domains <domain_uid> -n <domain_namespace> | grep "Mount
Path"
```

For example:

```
kubectl describe domains accessdomain -n oamns | grep "Mount Path"
```

If you deployed OAM using WLST, the output will look similar to the following:

```
Mount Path: /u01/oracle/user_projects/domains
```

If you deployed OAM using WDT, the output will look similar to the following:

```
Mount Path: /u01/oracle/user_projects
```

2. Run the following command to get the `Domain Home` and `Log Home` of your domain:

```
kubectl describe domains <domain_uid> -n <domain_namespace> | egrep
"Domain Home: | Log Home:"
```

For example:

```
kubectl describe domains accessdomain -n oamns | egrep "Domain Home: | Log
Home:"
```

The output will look similar to the following:

```
Domain Home:                    /u01/oracle/user_projects/domains/
accessdomain
Http Access Log In Log Home:    true
Log Home:                          /u01/oracle/user_projects/domains/logs/
accessdomain
```

3. Run the following command to get the OAM domain persistence volume details:

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

For example:

```
kubectl get pv -n oamns
```

The output will look similar to the following:

```
NAME                    CAPACITY   ACCESS MODES   RECLAIM POLICY
STATUS  CLAIM
STORAGECLASS                        REASON   AGE
accessdomain-domain-pv   10Gi       RWX            Retain
Bound   oamns/accessdomain-domain-pvc   accessdomain-domain-storage-
class           23h
```

Make note of the `CLAIM` value. In the example above the value is `accessdomain-domain-pvc`.

14.2.4 Creating the ConfigMap

Perform the following steps to create the Kubernetes ConfigMap for ELK:

1. Copy the `elk.crt` file to the `$WORKDIR/kubernetes/elasticsearch-and-kibana` directory.
2. Navigate to the `$WORKDIR/kubernetes/elasticsearch-and-kibana` directory and run the following:

   ```
   kubectl create configmap elk-cert --from-file=elk.crt -n <namespace>
   ```

   For example:

   ```
   kubectl create configmap elk-cert --from-file=elk.crt -n oamns
   ```

   The output will look similar to the following:

   ```
   configmap/elk-cert created
   ```

3. Create a `logstash_cm.yaml` file in the `$WORKDIR/kubernetes/elasticsearch-and-kibana` directory as follows:

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: oam-logstash-configmap
     namespace: <ELKNS>
   data:
     logstash.yml: |
     #http.host: "0.0.0.0"
     logstash-config.conf: |
       input {
        file {
           path => "<Log Home>/**/logs/AdminServer*.log"
           tags => "Adminserver_log"
           start_position => beginning
         }
         file {
           path => "<Log Home>/**/logs/oam_policy_mgr*.log"
           tags => "Policymanager_log"
           start_position => beginning
         }
         file {
           path => "<Log Home>/**/logs/oam_server*.log"
           tags => "Oamserver_log"
           start_position => beginning
         }
         file {
           path => "<Domain Home>/servers/AdminServer/logs/AdminServer-
   diagnostic.log"
           tags => "Adminserver_diagnostic"
           start_position => beginning
         }
         file {
   ```

```
        path => "<Domain Home>/servers/**/logs/oam_policy_mgr*-
diagnostic.log"
        tags => "Policy_diagnostic"
        start_position => beginning
      }
      file {
      path => "<Domain Home>/servers/AdminServer/logs/auditlogs/OAM/
audit.log"
      tags => "Audit_logs"
      start_position => beginning
      }
    }
    filter {
      grok {
        match => [ "message", "<%{DATA:log_timestamp}> <%{WORD:log_level}>
<%{WORD:thread}> <%{HOSTNAME:hostname}> <%{HOSTNAME:servername}> <%
{DATA:timer}> <<%{DATA:kernel}>> <> <%{DATA:uuid}> <%{NUMBER:timestamp}> <%
{DATA:misc}> <%{DATA:log_number}> <%{DATA:log_message}>" ]
      }
    if "_grokparsefailure" in [tags] {
        mutate {
            remove_tag => [ "_grokparsefailure" ]
        }
    }
    }
    output {
      elasticsearch {
    hosts => ["<ELK_HOSTS>"]
    cacert => '/usr/share/logstash/config/certs/elk.crt'
    index => "oamlogs-000001"
    ssl => true
    ssl_certificate_verification => false
    user => "<ELK_USER>"
    password => "${ELASTICSEARCH_PASSWORD}"
    api_key => "${ELASTICSEARCH_PASSWORD}"
      }
    }
```

Change the values in the above file as follows:

- Change the `<ELKNS>`, `<ELK_HOSTS>`, `<ELK_SSL>`, and `<ELK_USER>` to match the values in Variables Used in This Section.
- Change `<Log Home>` and `<Domain Home>` to match the `Log Home` and `Domain Home` returned in Finding Required Domain Details.
- If using API KEY for your ELK authentication, delete the `user` and `password` lines.
- If using a password for ELK authentication, delete the `api_key` line.
- If no authentication is used for ELK, delete the `user`, `password`, and `api_key` lines.

For example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: oam-logstash-configmap
```

```
  namespace: oamns
data:
  logstash.yml: |
  #http.host: "0.0.0.0"
  logstash-config.conf: |
    input {
     file {
        path => "/u01/oracle/user_projects/domains/logs/accessdomain/**/
logs/AdminServer*.log"
        tags => "Adminserver_log"
        start_position => beginning
      }
      file {
        path => "/u01/oracle/user_projects/domains/logs/accessdomain/**/
logs/oam_policy_mgr*.log"
        tags => "Policymanager_log"
        start_position => beginning
      }
      file {
        path => "/u01/oracle/user_projects/domains/logs/accessdomain/**/
logs/oam_server*.log"
        tags => "Oamserver_log"
        start_position => beginning
      }
      file {
        path => "/u01/oracle/user_projects/domains/accessdomain/servers/
AdminServer/logs/AdminServer-diagnostic.log"
        tags => "Adminserver_diagnostic"
        start_position => beginning
      }
      file {
        path => "/u01/oracle/user_projects/domains/accessdomain/servers/**/
logs/oam_policy_mgr*-diagnostic.log"
        tags => "Policy_diagnostic"
        start_position => beginning
      }
      file {
      path => "/u01/oracle/user_projects/domains/accessdomain/servers/
AdminServer/logs/auditlogs/OAM/audit.log"
      tags => "Audit_logs"
      start_position => beginning
      }
    }
    filter {
      grok {
        match => [ "message", "<%{DATA:log_timestamp}> <%{WORD:log_level}>
<%{WORD:thread}> <%{HOSTNAME:hostname}> <%{HOSTNAME:servername}> <%
{DATA:timer}> <<%{DATA:kernel}>> <> <%{DATA:uuid}> <%{NUMBER:timestamp}> <%
{DATA:misc}> <%{DATA:log_number}> <%{DATA:log_message}>" ]
      }
    if "_grokparsefailure" in [tags] {
        mutate {
            remove_tag => [ "_grokparsefailure" ]
        }
    }
    }
```

```
    output {
      elasticsearch {
    hosts => ["https://elasticsearch.example.com:9200"]
    cacert => '/usr/share/logstash/config/certs/elk.crt'
    index => "oamlogs-000001"
    ssl => true
    ssl_certificate_verification => false
    user => "logstash_internal"
    password => "${ELASTICSEARCH_PASSWORD}"
      }
    }
```

4. Run the following command to create the ConfigMap:

```
kubectl apply -f logstash_cm.yaml
```

The output will look similar to the following:

```
configmap/oam-logstash-configmap created
```

14.2.5 Enabling Logstash

Perform the following steps to enable logstash:

1. Navigate to the `$WORKDIR/kubernetes/elasticsearch-and-kibana` directory and create a `logstash.yaml` file as follows:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: oam-logstash
  namespace: <ELKNS>
spec:
  selector:
    matchLabels:
      k8s-app: logstash
  template: # create pods using pod definition in this template
    metadata:
     labels:
        k8s-app: logstash
    spec:
      imagePullSecrets:
      - name: dockercred
      containers:
      - command:
        - logstash
        image: logstash:<ELK_VER>
        imagePullPolicy: IfNotPresent
        name: oam-logstash
        env:
        - name: ELASTICSEARCH_PASSWORD
          valueFrom:
            secretKeyRef:
              name: elasticsearch-pw-elastic
              key: password
```

```
    resources:
    ports:
    - containerPort: 5044
      name: logstash
    volumeMounts:
    - mountPath: <mountPath>
      name: weblogic-domain-storage-volume
    - name: shared-logs
      mountPath: /shared-logs
    - mountPath: /usr/share/logstash/pipeline/
      name: oam-logstash-pipeline
    - mountPath: /usr/share/logstash/config/logstash.yml
      subPath: logstash.yml
      name: config-volume
    - mountPath: /usr/share/logstash/config/certs
      name: elk-cert
  volumes:
  - configMap:
      defaultMode: 420
      items:
      - key: elk.crt
        path: elk.crt
      name: elk-cert
    name: elk-cert
  - configMap:
      defaultMode: 420
      items:
      - key: logstash-config.conf
        path: logstash-config.conf
      name: oam-logstash-configmap
    name: oam-logstash-pipeline
  - configMap:
      defaultMode: 420
      items:
      - key: logstash.yml
        path: logstash.yml
      name: oam-logstash-configmap
    name: config-volume
  - name: weblogic-domain-storage-volume
    persistentVolumeClaim:
      claimName: accessdomain-domain-pvc
  - name: shared-logs
    emptyDir: {}
```

- Change the *<ELK_VER>*, *<ELK_SSL>* to match the values for your environment.
- Change `<mountPath>` to match the `mountPath` returned in Finding Required Domain Details.
- Change the `claimName` value to match the `claimName` returned earlier
- If your Kubernetes environment does not allow access to the internet to pull the logstash image, you must load the logstash image in your own container registry and change image: `logstash:<ELK_VER>` to the location of the image in your container registry, for example `container-registry.example.com/logstash:8.3.1`

For example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: oam-logstash
  namespace: oamns
spec:
  selector:
    matchLabels:
      k8s-app: logstash
  template: # create pods using pod definition in this template
    metadata:
     labels:
        k8s-app: logstash
    spec:
      imagePullSecrets:
      - name: dockercred
      containers:
      - command:
        - logstash
        image: logstash:8.3.1
        imagePullPolicy: IfNotPresent
        name: oam-logstash
        env:
        - name: ELASTICSEARCH_PASSWORD
          valueFrom:
            secretKeyRef:
              name: elasticsearch-pw-elastic
              key: password
        resources:
        ports:
        - containerPort: 5044
          name: logstash
        volumeMounts:
        - mountPath: /u01/oracle/user_projects
          name: weblogic-domain-storage-volume
        - name: shared-logs
          mountPath: /shared-logs
        - mountPath: /usr/share/logstash/pipeline/
          name: oam-logstash-pipeline
        - mountPath: /usr/share/logstash/config/logstash.yml
          subPath: logstash.yml
          name: config-volume
        - mountPath: /usr/share/logstash/config/certs
          name: elk-cert
      volumes:
      - configMap:
          defaultMode: 420
          items:
          - key: elk.crt
            path: elk.crt
          name: elk-cert
        name: elk-cert
      - configMap:
          defaultMode: 420
```

```
      items:
      - key: logstash-config.conf
        path: logstash-config.conf
      name: oam-logstash-configmap
    name: oam-logstash-pipeline
  - configMap:
      defaultMode: 420
      items:
      - key: logstash.yml
        path: logstash.yml
      name: oam-logstash-configmap
    name: config-volume
  - name: weblogic-domain-storage-volume
    persistentVolumeClaim:
      claimName: accessdomain-domain-pvc
  - name: shared-logs
    emptyDir: {}
```

2. Deploy the logstash pod by executing the following command:

   ```
   kubectl create -f $WORKDIR/kubernetes/elasticsearch-and-kibana/
   logstash.yaml
   ```

   The output will look similar to the following:

   ```
   deployment.apps/oam-logstash created
   ```

14.3 Verifying the Pods

1. Run the following command to check the logstash pod is created correctly:

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

   For example:

   ```
   kubectl get pods -n oamns
   ```

   The output should look similar to the following:

   ```
   NAME                                                READY   STATUS
   RESTARTS   AGE
   accessdomain-adminserver                             1/1
   Running     0          18h
   accessdomain-oam-policy-mgr1                         1/1
   Running     0          18h
   accessdomain-oam-server1                             1/1
   Running     1          18h
   oam-logstash-bbbdf5876-85nkd                         1/1
   Running     0          4m23s
   ```

Wait a couple of minutes to make sure the logstash pod has not had any failures or restarts. If the pod fails you can view the pod log using:

```
kubectl logs -f oam-logstash-<pod> -n oamns
```

Most errors occur due to misconfiguration of the `logstash_cm.yaml` or `logstash.yaml`. This is usually because of an incorrect value set, or the certificate was not pasted with the correct indentation.
If the pod has errors, delete the pod and ConfigMap as follows:

```
kubectl delete -f $WORKDIR/kubernetes/elasticsearch-and-kibana/
logstash.yaml
```

```
kubectl delete -f $WORKDIR/kubernetes/elasticsearch-and-kibana/
logstash_cm.yaml
```

Once you have resolved the issue in the yaml files, run the commands outlined earlier to recreate the ConfigMap and logstash pod.

14.4 Verifying and Accessing the Kibana Console

To access the Kibana console you will need the Kibana URL as per Installing the Monitoring and Visualization Software.

Kibana Version 7.8.X or Higher

1. Access the Kibana console with `http://<hostname>:<port>/app/kibana` and login with your username and password.
2. From the Navigation menu, navigate to **Management** > **Kibana** > **Index Patterns**.
3. In the **Create Index Pattern** page enter `oamlogs*` for the **Index pattern** and click **Next Step**.
4. In the **Configure settings** page, from the **Time Filter field name** drop down menu select `@timestamp` and click **Create index pattern**.
5. Once the index pattern is created click on **Discover** in the navigation menu to view the OAM logs.

Kibana 7.7.x or Lower

1. Access the Kibana console with `http://<hostname>:<port>/app/kibana` and login with your username and password.
2. From the Navigation menu, navigate to **Management** > **Stack Management**.
3. Click **Data Views** in the **Kibana** section.
4. Click **Create Data View** and enter the following information:
   - Name: `oamlogs*`
   - Timestamp: `@timestamp`
5. Click **Create Data View**.
6. From the Navigation menu, click **Discover** to view the log file entries.

7. From the drop down menu, select `oamlogs*` to view the log file entries.

15

Monitoring an Oracle Access Management Domain

Using the WebLogic Monitoring Exporter you can scrape runtime information from a running Oracle Access Management (OAM) domain and monitor using Prometheus and Grafana.

To set up monitoring, see Monitor the Oracle Access Management instance using Prometheus and Grafana.

For more information on WebLogic Monitoring Exporter, see WebLogic Monitoring Exporter.

16

Kubernetes Horizontal Pod Autoscaler

Kubernetes Horizontal Pod Autoscaler (HPA) allows automatic scaling (up and down) of the Oracle Access Management (OAM) servers. If load increases then extra OAM servers will be started as required, up to the value `configuredManagedServerCount` defined when the domain was created. Similarly, if load decreases, OAM servers will be automatically shutdown.

For more information on HPA, see Horizontal Pod Autoscaling.

The instructions below show you how to configure and run an HPA to scale an OAM cluster (`accessdomain-oam-cluster`), based on CPU utilization or memory resource metrics. If required, you can also perform the following for the `accessdomain-policy-cluster`.

> **Note**
>
> If you enable HPA and then decide you want to start,stop, or scale OAM servers manually as per Scaling OAM Pods, it is recommended to delete HPA beforehand as per Deleting HPA.

This chapter includes the following topics:

16.1 Prerequisite Configurations

In order to use HPA, Oracle Access Management (OAM) must have been created with the required `resources` parameter. For OAM domains created with WLST scripts, this is as per Setting the OAM Server Memory Parameters. For OAM domains created with WDT models, the values should be set by default. For example:

For example:

```
serverPod:
  env:
  - name: USER_MEM_ARGS
    value: "-XX:+UseContainerSupport -Djava.security.egd=file:/dev/./urandom -
Xms8192m -Xmx8192m"
  resources:
    limits:
      cpu: "2"
```

```
    memory: "8Gi"
  requests:
    cpu: "1000m"
    memory: "4Gi"
```

If you created the OAM domain without setting these parameters, then you can update the domain using the following steps:

1. Run the following command to edit the cluster:

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

   ```
   kubectl edit cluster accessdomain-oam-cluster -n oamns
   ```

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

2. In the edit session, search for `spec:`, and then look for the replicas parameter under `clusterName: oam_cluster`. Change the entry so it looks as follows:

   ```
   spec:
     clusterName: oam_cluster
     replicas: 1
     serverPod:
       env:
       - name: USER_MEM_ARGS
         value: -XX:+UseContainerSupport -Djava.security.egd=file:/dev/./
   urandom -Xms8192m -Xmx8192m
       resources:
         limits:
           cpu: "2"
           memory: 8Gi
         requests:
           cpu: 1000m
           memory: 4Gi
     serverService:
       precreateService: true
       ...
   ```

3. Save the file and exit (`:wq!`).
   The output will look similar to the following:

   ```
   cluster.weblogic.oracle/accessdomain-oam-cluster edited
   ```

   The OAM managed server pods will then automatically be restarted.

16.2 Deploying the Kubernetes Metrics Server

Before deploying Horizontal Pod Autoscaler (HPA) you must deploy the Kubernetes Metrics Server.

1. Check to see if the Kubernetes Metrics Server is already deployed:

   ```
   kubectl get pods -n kube-system | grep metric
   ```

   If a row is returned as follows, then Kubernetes Metric Server is deployed and you can move to Deploying HPA:

   ```
   metrics-server-d9694457-mf69d           1/1     Running   0
   5m13s
   ```

2. If no rows are returned by the previous command, then the Kubernetes Metric Server needs to be deployed. Run the following commands to get the `components.yaml`:

   ```
   mkdir $WORKDIR/kubernetes/hpa
   ```

   ```
   cd $WORKDIR/kubernetes/hpa
   ```

   ```
   wget https://github.com/kubernetes-sigs/metrics-server/releases/latest/
   download/components.yaml
   ```

3. Deploy the Kubernetes Metrics Server by running the following command:

   ```
   kubectl apply -f components.yaml
   ```

   The output will look similar to the following:

   ```
   serviceaccount/metrics-server created
   clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader
   created
   clusterrole.rbac.authorization.k8s.io/system:metrics-server created
   rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created
   clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-
   delegator created
   clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created
   service/metrics-server created
   deployment.apps/metrics-server created
   apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io created
   ```

4. Run the following command to check Kubernetes Metric Server is running:

   ```
   kubectl get pods -n kube-system | grep metric
   ```

   Make sure the pod has a `READY` status of `1/1`:

   ```
   metrics-server-d9694457-mf69d           1/1     Running   0             39s
   ```

16.3 Troubleshooting the Metrics Server

If the Kubernetes Metric Server does not reach the `READY 1/1` state, run the following commands:

```
kubectl describe pod <metrics-server-pod> -n kube-system
```

```
kubectl logs <metrics-server-pod> -n kube-system
```

If you see errors such as:

```
Readiness probe failed: HTTP probe failed with statuscode: 500
```

and:

```
E0907 13:07:50.937308       1 scraper.go:140] "Failed to scrape node"
err="Get \"https://X.X.X.X:10250/metrics/resource\": x509: cannot validate
certificate for 100.105.18.113 because it doesn't contain any IP SANs"
node="worker-node1"
```

then you may need to install a valid cluster certificate for your Kubernetes cluster.

For testing purposes, you can resolve this issue by:

1. Delete the Kubernetes Metrics Server by running the following command:

   ```
   kubectl delete -f $WORKDIR/kubernetes/hpa/components.yaml
   ```

2. Edit the `$WORKDIR/hpa/components.yaml` and locate the `args:` section. Add `kubelet-insecure-tls` to the arguments. For example:

   ```
   spec:
     containers:
     - args:
       - --cert-dir=/tmp
       - --secure-port=4443
       - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
       - --kubelet-use-node-status-port
       - --kubelet-insecure-tls
       - --metric-resolution=15s
       image: registry.k8s.io/metrics-server/metrics-server:v0.6.4
    ...
   ```

3. Deploy the Kubernetes Metrics Server using the command:

   ```
   kubectl apply -f components.yaml
   ```

4. Run the following and make sure the `READY` status shows `1/1`:

   ```
   kubectl get pods -n kube-system | grep metric
   ```

The output should look similar to the following:

```
metrics-server-d9694457-mf69d           1/1     Running   0             40s
```

16.4 Deploying HPA

The steps below show how to configure and run Horizontal Pod Autoscaler (HPA) to scale Oracle Access Management (OAM), based on the CPU or memory utilization resource metrics.

The default OAM deployment creates the cluster `accessdomain-oam-cluster` which starts one OAM managed server (`oam_server1`). The deployment also creates, but doesn't start, four extra OAM Managed Servers (`oam-server2` to `oam-server5`).

In the following example an HPA resource is created, cluster resource `accessdomain-oam-cluster`. This resource will autoscale OAM managed server from a minimum of 1 cluster member up to 5 cluster members. Scaling up will occur when the average CPU is consistently over 70%. Scaling down will occur when the average CPU is consistently below 70%.

1. Navigate to the `$WORKDIR/kubernetes/hpa` and create an `autoscalehpa.yaml` file that contains the following:

```
#
#
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: accessdomain-oam-cluster-hpa
  namespace: oamns
spec:
  scaleTargetRef:
    apiVersion: weblogic.oracle/v1
    kind: Cluster
    name: accessdomain-oam-cluster
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 60
    scaleUp:
      stabilizationWindowSeconds: 60
  minReplicas: 1
  maxReplicas: 5
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
```

where:

- `accessdomain` is the `<domainUID>`
- `oamns` is the `<domain_namespace>`.
- `minReplicas` and `maxReplicas` should match your current domain setting.

> **Note**
>
> For setting HPA based on Memory Metrics, update the metrics block with the following content. Please note, Oracle recommends using only CPU or Memory, not both:
>
> ```
> metrics:
> - type: Resource
>   resource:
>     name: memory
>     target:
>       type: Utilization
>       averageUtilization: 70
> ```

2. Run the following command to create the autoscaler:

   ```
   kubectl apply -f autoscalehpa.yaml
   ```

   The output will look similar to the following:

   ```
   horizontalpodautoscaler.autoscaling/accessdomain-oam-cluster-hpa created
   ```

3. Verify the status of the autoscaler by running the following:

   ```
   kubectl get hpa -n oamns
   ```

   The output will look similar to the following:

   ```
   NAME                            REFERENCE
   TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
   accessdomain-oam-cluster-hpa   Cluster/accessdomain-oam-cluster
   5%/70%    1         5         1          21s
   ```

   In the example above, this shows that CPU is currently running at 5% for the `accessdomain-oam-cluster-hpa`.

16.5 Verifying HPA

To verify the Horizontal Pod Autoscaler (HPA) works, perform the following steps:

1. Check the current status of the Oracle Access Management (OAM) servers:

   ```
   kubectl get pods -n oamns
   ```

   The output will look similar to the following:

   ```
   NAME                                                    READY
   STATUS      RESTARTS        AGE
   accessdomain-adminserver                                0/1
   Running     0               141m
   ```

```
accessdomain-oam-policy-mgr1                                  0/1
Running     0               138m
accessdomain-oam-server1                                      1/1
Running     0               138m
```

In the above example only `accessdomain-oam-server1` is running.

2. To test HPA can scale up the WebLogic cluster `accessdomain-oam-cluster`, run the following commands:

```
kubectl exec --stdin --tty accessdomain-oam-server1 -n oamns -- /bin/bash
```

This will take you inside a bash shell inside the `oam_server1` pod:

```
[oracle@accessdomain-oam-server1 oracle]$
```

3. Inside the bash shell, run the following command to increase the load on the CPU:

```
[oracle@accessdomain-oam-server1 oracle]$ dd if=/dev/zero of=/dev/null
```

This command will continue to run in the foreground.

4. In a command window outside the bash shell, run the following command to view the current CPU usage:

```
kubectl get hpa -n oamns
```

The output will look similar to the following:

```
NAME                             REFERENCE
TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
accessdomain-oam-cluster-hpa   Cluster/accessdomain-oam-cluster
470%/70%    1         5         1        21s
```

In the above example the CPU has increased to 470%. As this is above the 70% limit, the autoscaler increases the replicas on the Cluster resource, and the operator responds by starting additional cluster members.

5. Run the following to see if any more OAM Managed Servers are started:

```
kubectl get pods -n oamns
```

The output will look similar to the following:

```
NAME                                                          READY
STATUS      RESTARTS        AGE
accessdomain-adminserver                                      0/1
Running                     143m
accessdomain-oam-policy-mgr1                                  0/1
Running     0               140m
accessdomain-oam-server1                                      1/1
Running     0               140m
accessdomain-oam-server2                                      1/1
```

```
Running     0               3m20s
accessdomain-oam-server3                                  1/1
Running     0               3m20s
accessdomain-oam-server4                                  1/1
Running     0               3m19s
accessdomain-oam-server5                                  1/1
Running     0               3m5s
```

In the example above four more OAM managed servers have been started (`oam-server2` - `oam-server5`).

> **Note**
>
> It may take some time for the server to appear and start. Once the servers are at `READY` status of `1/1`, the servers are started.

6. To stop the load on the CPU, in both bash shells, issue a Control C, and then exit the bash shell:

```
[oracle@accessdomain-oam-server1 oracle]$ dd if=/dev/zero of=/dev/null
^C
[oracle@accessdomain-oam-server1 oracle]$ exit
```

7. Run the following command to view the current CPU usage:

```
kubectl get hpa -n oamns
```

The output will look similar to the following:

```
NAME                            REFERENCE
TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
accessdomain-oam-cluster-hpa   Cluster/accessdomain-oam-cluster
19%/70%   1         5         5          19m
```

In the above example CPU has dropped to 19%. As this is below the 70% threshold, you should see the autoscaler scale down the servers:

```
kubectl get pods -n oamns
```

The output will look similar to the following:

```
NAME                                                      READY
STATUS        RESTARTS        AGE
accessdomain-adminserver                                  1/1
Running       0               152m
accessdomain-oam-policy-mgr1                              1/1
Running       0               149m
accessdomain-oam-server1                                  1/1
Running       0               149m
accessdomain-oam-server2                                  1/1
Running       0               14m
```

```
accessdomain-oam-server3                                  0/1
Terminating   0              14m
```

Eventually, all the servers except oam-server1 will disappear:

```
NAME                                                      READY
STATUS      RESTARTS       AGE
accessdomain-adminserver                                  1/1
Running     0              154m
accessdomain-oam-policy-mgr1                              1/1
Running     0              151m
accessdomain-oam-server1                                  1/1
Running     0              151m
```

16.6 Deleting HPA

If you need to delete the Horizontal Pod Autoscaler (HPA), you can do so by running the following commands:

```
cd $WORKDIR/kubernetes/hpa
```

```
kubectl delete -f autoscalehpa.yaml
```

16.7 Other Considerations for HPA

Administrators should be aware of the following considerations after deploying the Horizontal Pod Autoscaler (HPA):

- If HPA is deployed and you need to upgrade the Oracle Access Management (OAM) container image, then you must delete the HPA before upgrading. To delete the HPA, see Deleting HPA. Once the upgrade is successful you can deploy HPA again.
- If you choose to start/stop an OAM managed server manually as per Scaling OAM Pods, then it is recommended to delete the HPA before doing so.

17

Patching and Upgrading

This chapter includes the following topics:

- Patching and Upgrading Within 14.1.2
- Upgrading from Oracle Access Management 12.2.1.4 to 14.1.2

17.1 Patching and Upgrading Within 14.1.2

Learn how to patch or upgrade the Oracle Access Management (OAM) image used by an OAM 14.1.2 container.

This section contains the following topics:

- Patching a Container Image
- Upgrading WebLogic Kubernetes Operator

17.1.1 Patching a Container Image

The instructions in this section relate to patching or upgrading an existing 14.1.2.1.0 Oracle Access Management (OAM) deployment with a new OAM container image.

> **Note**
>
> Administrators should be aware of the following:
>
> - If you are not using Oracle Container Registry or your own container registry, then you must first load the new container image on all nodes in your Kubernetes cluster.
> - If you have Kubernetes Horizontal Pod Autoscaler (HPA) enabled, you must disable HPA before performing the steps below. See, Deleting HPA.
> - Updating the container image leads to a rolling restart of the pods. The administration server will terminate and restart, followed by any OAM managed servers and policy manager servers.

Choose one of the following options to update your OAM deployment with a new image:

- Running the `kubectl edit domain` command.
- Running the `kubectl patch domain` command.

Running the kubectl edit domain Command

To update the domain:

1. Run the following command:

   ```
   kubectl edit domain <domainname> -n <namespace>
   ```

   For example:

   ```
   kubectl edit domain accessdomain -n oamns
   ```

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

2. Update the `image` parameter to reference the new OAM container image:

   - If using Oracle Container Registry or your own container registry for your OAM container image, update the image to point at the location of the new image, for example:

     ```
     ...
     image: container-registry.oracle.com/middleware/oam_cpu:<new_tag>
       imagePullPolicy: IfNotPresent
       imagePullSecrets:
       - name: orclcred
     ...
     ```

   - If you are not using a container registry and have loaded the image on each of the worker nodes, update the image to point at the new image:

     ```
     ...
     image: container-registry.oracle.com/middleware/oam_cpu:<new_tag>
       imagePullPolicy: oracle/oam:<new_tag>
     ...
     ```

3. Save the file and exit (`:wq!`)

Running the kubectl patch Command

To update the domain:

1. Run the following command to set the `image` parameter to the location of the new image:

   ```
   kubectl patch domain <domain> -n <namespace> --type merge  -p '{"spec":
   {"image":"<repository>:<new_tag>"}}'
   ```

   For example:

   - If using Oracle Container Registry or your own container registry for your OAM container image:

     ```
     kubectl patch domain accessdomain -n oamns --type merge  -p '{"spec":
     {"image":"container-registry.oracle.com/middleware/oam_cpu:<new_tag>"}}'
     ```

- If you are not using a container registry and have loaded the image on each of the worker nodes:

  ```
  kubectl patch domain accessdomain -n oamns --type merge  -p '{"spec":
  {"image":"oracle/oam:<new_tag>"}}'
  ```

  The output will look similar to the following:

  ```
  domain.weblogic.oracle/accessdomain patched
  ```

Verifying the OAM Deployment is Using the New Image

After following the above steps, a rolling restart will be performed. The administration server will terminate and restart, followed by the OAM managed server(s) and policy manager server(s).

While the servers are restarting you can run the following command to view the status:

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

For example:

```
kubectl get pods -n oamns -w
```

Once the pods are up and running, you can run the following command to show the image is used by the pods:

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

For example:

```
kubectl describe pod accessdomain-oam-server1 -n oamns
```

The new image should be displayed in the following section:

```
...
Containers:
  weblogic-server:
    Container ID:  cri-o://
220fa83d079e079ac183c00f884b10ea30a794527dbb65e6964a035d450384f8
    Image:         container-registry.oracle.com/middleware/oam_cpu:<new_tag>
    Image ID:      container-registry.oracle.com/middleware/
oam_cpu@sha256:cdf51b6aa47cd05573bc53244681b193fb4e2f6db56e50d2251b9416bc68ebc
0
    Port:          14100/TCP
    Host Port:     0/TCP
    Command:
...
```

17.1.2 Upgrading WebLogic Kubernetes Operator

The instructions in this section relate to upgrading the WebLogic Kubernetes Operator used by an Oracle Access Management (OAM) deployment.

> **Note**
>
> This applies to WebLogic Kubernetes Operator in the 4.X release family as additional versions are released.

To upgrade the WebLogic Kubernetes Operator used by the OAM deployment, perform the following steps:

1. On the Kubernetes administrative host, download the new WebLogic Kubernetes Operator source code from the operator github project:

   ```
   mkdir <workdir>/weblogic-kubernetes-operator-4.X.X
   ```

   ```
   cd <workdir>/weblogic-kubernetes-operator-4.X.X
   ```

   ```
   git clone https://github.com/oracle/weblogic-kubernetes-operator.git --
   branch v4.X.X
   ```

   For example:

   ```
   mkdir /OAMK8S/weblogic-kubernetes-operator-4.X.X
   ```

   ```
   cd /OAMK8S/weblogic-kubernetes-operator-4.X.X
   ```

   ```
   git clone https://github.com/oracle/weblogic-kubernetes-operator.git --
   branch v4.X.X
   ```

2. Run the following helm commands to upgrade the operator::

   ```
   cd <workdir>/weblogic-kubernetes-operator-4.X.X/weblogic-kubernetes-
   operator
   ```

   ```
   helm upgrade --reuse-values \
   --set image=ghcr.io/oracle/weblogic-kubernetes-operator:4.X.X \
   --namespace <sample-kubernetes-operator-ns> \
   --wait weblogic-kubernetes-operator \
   kubernetes/charts/weblogic-operator
   ```

   For example:

   ```
   cd /OAMK8S/weblogic-kubernetes-operator-4.X.X/weblogic-kubernetes-operator
   ```

   ```
   helm upgrade --reuse-values \
   --set image=ghcr.io/oracle/weblogic-kubernetes-operator:4.X.X \
   ```

```
--namespace opns \
--wait weblogic-kubernetes-operator \
kubernetes/charts/weblogic-operator
```

The output will look similar to the following:

```
Release "weblogic-kubernetes-operator" has been upgraded. Happy Helming!
NAME: weblogic-kubernetes-operator
LAST DEPLOYED: <DATE>
NAMESPACE: opns
STATUS: deployed
REVISION: 2
TEST SUITE: None
```

3. Verify that the operator's pod and services are running by executing the following command:

```
kubectl get all -n <sample-kubernetes-operator-ns>
```

For example:

```
kubectl get all -n opns
```

The output will look similar to the following:

```
NAME                                                READY   STATUS
RESTARTS   AGE
pod/weblogic-operator-b7d6df78c-mfrc4               1/1     Running
0          40s
pod/weblogic-operator-webhook-7996b8b58b-frtwp      1/1     Running
0          42s

NAME                                        TYPE        CLUSTER-IP
EXTERNAL-IP   PORT(S)             AGE
service/weblogic-operator-webhook-svc       ClusterIP   10.106.51.57
<none>        8083/TCP,8084/TCP   42s

NAME                                           READY   UP-TO-DATE
AVAILABLE   AGE
deployment.apps/weblogic-operator              1/1     1
1           6d
deployment.apps/weblogic-operator-webhook      1/1     1
1           42s

NAME                                                 DESIRED   CURRENT
READY   AGE
replicaset.apps/weblogic-operator-5884685f4f         0         0
0       6d
replicaset.apps/weblogic-operator-b7d6df78c          1         1
1       40s
replicaset.apps/weblogic-operator-webhook-7996b8b58b   1         1
1       42s
```

17.2 Upgrading from Oracle Access Management 12.2.1.4 to 14.1.2

The instructions in this section are for upgrading an existing Oracle Access Management (OAM) 12.2.1.4 deployment on Kubernetes to OAM 14.1.2.1.0.

> **Note**
>
> Administrators should be aware that the upgrade requires a period of down time.

This section contains the following topics:

17.2.1 Upgrade Prerequisite Steps

Before upgrading Oracle Access Management (OAM) from 12c to 14c, you must meet the following prerequisites.

OAM 12c Prerequisites

It is recommended to be on the latest OAM 12c container image and supported WebLogic Kubernetes Operator, before upgrading to 14c. For further details on latest versions and supported operator versions, see, Oracle Access Management 12c on Kubernetes Release Notes.

Kubernetes Prerequisites

- The Kubernetes cluster and container engine must meet the minimum version requirements outlined in document ID 2723908.1 on My Oracle Support.
- Your `<persistent_volume>` must have enough free storage to back up the contents of the persistent volume and to allow for the upgrade. An upgrade can take around 1GB of storage space, so it's advisable to have at least 2GB spare before performing the upgrade.

Gathering Variables

You must gather the following information for your existing OAM 12c deployment. The values for these variables will be used in Creating the domainUpgradeResponse.txt File and Creating the domain-upgrade-pod.yaml.

Table 17-1 List of Variables

Variable	Description	Sample Value
%NAMESPACE%	The domain namespace used by the OAM 12c deployment.	`oamns`
%DOMAIN_UID%	This is the domain uid used by the OAM 12c deployment. To find the domain name, run: `kubectl get domain -n %NAMESPACE%`	`accessdomain`
%DOMAIN_MOUNT_PATH%	The mount path used by the OAM 12c deployment. To find the mount path, run: `kubectl describe domains %DOMAIN_UID% -n %NAMESPACE% | grep "Mount Path"`	• For WDT domains:`/u01/oracle/user_projects` • For WLST domains:`/u01/oracle/user_projects/domains`
%DOMAIN_HOME%	The domain home location used by the OAM 12c deployment. To find the mount path, run: `kubectl describe domains %DOMAIN_UID% -n %NAMESPACE% | grep "Domain Home:"`	`/u01/oracle/user_projects/domains/accessdomain`
%DOMAIN_ROOT_DIR%	For WLST created domains, this is the *%DOMAIN_MOUNT_PATH%/*. For WDT created domains, this is the *%DOMAIN_MOUNT_PATH%/domains* directory.	`/u01/oracle/user_projects/domains`
%CONNECTION_STRING%	The connection string for the database where the OAM 12c schemas reside, in the format: `<host.domain>:<db_port>/<db_service>`	`mydatabasehost.example.com:1521/orcl.example.com`
%RCU_PREFIX%	The RCU schema prefix for the OAM 12c deployment.	`OAMK8S`
%RCU_SCHEMA_PWD%	The password for *%RCUPREFIX%*.	<password>
%SYS_USERNAME%	The SYS username for the database where the OAM 12c schemas reside.	`sys`
%SYS_USERNAME_PWD%	The password for *%SYS_USERNAME%*.	`<password>`

Table 17-1 (Cont.) List of Variables

Variable	Description	Sample Value
%DOMAIN_PVC_NAME%	The persistent volume claim (PVC) for the OAM 12c deployment. To find the PVC, run: `kubectl get pvc -n %NAMESPACE%`	`accessdomain-domain-pvc`
%RCU_CREDENTIALS_SECRET_NAME%	The RCU secret for the OAM 12c deployment. To find the RCU secret, run: `kubectl get secrets -n %NAMESPACE% | grep rcu`	`accessdomain-rcu-credentials`
%WEBLOGIC_IMAGE%	The location of the OAM 14c container image.	`container-registry.oracle.com/middleware/oam_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD>`
%WEBLOGIC_IMAGE_PULL_POLICY%	The image pull policy you used in the OAM 12c deployment. To find the image pull policy, run: `kubectl describe domains %DOMAIN_UID% -n %NAMESPACE% | grep "Image Pull Policy"`	`IfNotPresent`

17.2.2 Creating the domainUpgradeResponse.txt File

Run the following steps to create the `domainUpgradeResponse.txt` file:

1. Create a working directory for the upgrade scripts and navigate to it:

   ```
   mkdir <workdir>/upgradescripts
   ```

   ```
   cd <workdir>/upgradescripts
   ```

   For example:

   ```
   mkdir /OAM12CUPG/upgradescripts
   ```

   ```
   cd /OAM12CUPG/upgradescripts
   ```

2. Create a `domainUpgradeResponse.txt` file and replace the environment variables listed with the corresponding values collected in Upgrade Prerequisite Steps:

```
[GENERAL]

fileFormatVersion = 3

#==============================================================
[UAWLSINTERNAL.UAWLS]
pluginInstance = 1

# Specifies the WebLogic Server domain directory:
UASVR.path = %DOMAIN_HOME%
UASVR.enforce = no

#==============================================================
[MDS.SCHEMA_UPGRADE]
pluginInstance = 2
MDS.databaseType = Oracle Database
MDS.databaseConnectionString = %CONNECTION_STRING%
MDS.schemaConnectionString = %CONNECTION_STRING%
MDS.schemaUserName = %RCU_PREFIX%_MDS
MDS.cleartextSchemaPassword = %RCU_SCHEMA_PWD%
MDS.dbaUserName = %SYS_USERNAME% as sysdba
MDS.cleartextDbaPassword = %SYS_USERNAME_PWD%

#==============================================================
[OPSS.OPSS_SCHEMA_PLUGIN]
pluginInstance = 3
OPSS.databaseConnectionString = %CONNECTION_STRING%
OPSS.schemaUserName = %RCU_PREFIX%_OPSS
OPSS.cleartextSchemaPassword = %RCU_SCHEMA_PWD%
OPSS.dbaUserName = %SYS_USERNAME% as sysdba
OPSS.cleartextDbaPassword = %SYS_USERNAME_PWD%

#==============================================================
[IAU.AUDIT_SCHEMA_PLUGIN]
pluginInstance = 4
IAU.databaseConnectionString = %CONNECTION_STRING%
IAU.schemaUserName = %RCU_PREFIX%_IAU
IAU.cleartextSchemaPassword = %RCU_SCHEMA_PWD%
IAU.dbaUserName = %SYS_USERNAME% as sysdba
IAU.cleartextDbaPassword = %SYS_USERNAME_PWD%

#==============================================================
[FMWCONFIG.CIE_SCHEMA_PLUGIN]
pluginInstance = 5
STB.databaseConnectionString = %CONNECTION_STRING%
STB.schemaUserName = %RCU_PREFIX%_STB
STB.cleartextSchemaPassword = %RCU_SCHEMA_PWD%
STB.dbaUserName = %SYS_USERNAME% as sysdba
STB.cleartextDbaPassword = %SYS_USERNAME_PWD%

#==============================================================
[WLS.WLS]
pluginInstance = 6
```

```
WLS.databaseConnectionString = %CONNECTION_STRING%
WLS.schemaUserName = %RCU_PREFIX%_WLS
WLS.cleartextSchemaPassword = %RCU_SCHEMA_PWD%
WLS.dbaUserName = %SYS_USERNAME% as sysdba
WLS.cleartextDbaPassword = %SYS_USERNAME_PWD%

# WLS_RUNTIME entries
WLS_RUNTIME.databaseConnectionString = %CONNECTION_STRING%
WLS_RUNTIME.schemaUserName = %RCU_PREFIX%_WLS_RUNTIME
WLS_RUNTIME.cleartextSchemaPassword = %RCU_SCHEMA_PWD%
WLS_RUNTIME.dbaUserName = %SYS_USERNAME% as sysdba
WLS_RUNTIME.cleartextDbaPassword = %SYS_USERNAME_PWD%

#=============================================================
[OAM.SCHEMAPLUGIN]
pluginInstance = 7
OAM.databaseConnectionString = %CONNECTION_STRING%
OAM.schemaUserName = %RCU_PREFIX%_OAM
OAM.cleartextSchemaPassword = %RCU_SCHEMA_PWD%
OAM.dbaUserName = %SYS_USERNAME% as sysdba
OAM.cleartextDbaPassword = %SYS_USERNAME_PWD%
```

For example:

```
# Copyright (c) 2024, 2025, Oracle and/or its affiliates.
# Licensed under the Universal Permissive License v 1.0 as shown at
https://oss.oracle.com/licenses/upl.
#
# This is a response file for the Fusion Middleware Upgrade Assistant.
# Individual component upgrades are performed in the order they are
described here.
# Each upgrade is introduced by a section header containing the name of the
# component and name of the upgrade plugin. The form of the section header
is
# [ComponentName.PluginName]
# These names can be found in the Upgrade Descriptor files for the
components.

# Individual input lines consist of a name, an equal sign, and a value.
# The name is in two parts separated by a period.  The first part is the
"name"
# attribute from the Descriptor File XML tag by which the plugin refers to
the value.
# The second part of the name identifies a field within that value.  Some
input
# types have only one field, while other types can have half a dozen.   Do
not
# intermix input lines that apply to different XML tags.

[GENERAL]

fileFormatVersion = 3

#=============================================================
[UAWLSINTERNAL.UAWLS]
```

```
pluginInstance = 1

# Specifies the WebLogic Server domain directory:
UASVR.path = /u01/oracle/user_projects/domains/accessdomain
UASVR.enforce = no

#====================================================================
[MDS.SCHEMA_UPGRADE]
pluginInstance = 2
MDS.databaseType = Oracle Database
MDS.databaseConnectionString = mydatabasehost.example.com:1521/
orcl.example.com
MDS.schemaConnectionString = mydatabasehost.example.com:1521/
orcl.example.com
MDS.schemaUserName = OAMK8S_MDS
MDS.cleartextSchemaPassword = <password>
MDS.dbaUserName = sys as sysdba
MDS.cleartextDbaPassword = <password>

#====================================================================
[OPSS.OPSS_SCHEMA_PLUGIN]
pluginInstance = 3
OPSS.databaseConnectionString = mydatabasehost.example.com:1521/
orcl.example.com
OPSS.schemaUserName = OAMK8S_OPSS
OPSS.cleartextSchemaPassword = <password>
OPSS.dbaUserName = sys as sysdba
OPSS.cleartextDbaPassword = <password>

#====================================================================
[IAU.AUDIT_SCHEMA_PLUGIN]
pluginInstance = 4
IAU.databaseConnectionString = mydatabasehost.example.com:1521/
orcl.example.com
IAU.schemaUserName = OAMK8S_IAU
IAU.cleartextSchemaPassword = <password>
IAU.dbaUserName = sys as sysdba
IAU.cleartextDbaPassword = <password>

#====================================================================
[FMWCONFIG.CIE_SCHEMA_PLUGIN]
pluginInstance = 5
STB.databaseConnectionString = mydatabasehost.example.com:1521/
orcl.example.com
STB.schemaUserName = OAMK8S_STB
STB.cleartextSchemaPassword = <password>
STB.dbaUserName = sys as sysdba
STB.cleartextDbaPassword = <password>

#====================================================================
[WLS.WLS]
pluginInstance = 6
WLS.databaseConnectionString = mydatabasehost.example.com:1521/
orcl.example.com
WLS.schemaUserName = OAMK8S_WLS
WLS.cleartextSchemaPassword = <password>
```

```
WLS.dbaUserName = sys as sysdba
WLS.cleartextDbaPassword = <password>

# WLS_RUNTIME entries
WLS_RUNTIME.databaseConnectionString = mydatabasehost.example.com:1521/
orcl.example.com
WLS_RUNTIME.schemaUserName = OAMK8S_WLS_RUNTIME
WLS_RUNTIME.cleartextSchemaPassword = <password>
WLS_RUNTIME.dbaUserName = sys as sysdba
WLS_RUNTIME.cleartextDbaPassword = <password>

#=====================================================================
[OAM.SCHEMAPLUGIN]
pluginInstance = 7
OAM.databaseConnectionString = mydatabasehost.example.com:1521/
orcl.example.com
OAM.schemaUserName = OAMK8S_OAM
OAM.cleartextSchemaPassword = <password>
OAM.dbaUserName = sys as sysdba
OAM.cleartextDbaPassword = <password>
```

17.2.3 Creating the domain-upgrade-pod.yaml

Run the following steps to create the `domain-upgrade-pod.yaml` file:

1. In the `<workdir>/upgradescripts` directory create a `domain-upgrade-pod.yaml` and replace the environment variables listed, with the corresponding values collected in Upgrade Prerequisite Steps:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    weblogic.domainUID: %DOMAIN_UID%
    weblogic.domainName: %DOMAIN_UID%
    app: %DOMAIN_UID%-domain-upgrade
  name: %DOMAIN_UID%-domain-upgrade
  namespace: %NAMESPACE%
spec:
  containers:
  - args:
    - sleep
    - infinity
    image: %WEBLOGIC_IMAGE%
    imagePullPolicy: %WEBLOGIC_IMAGE_PULL_POLICY%
    name: %DOMAIN_UID%-domain-upgrade
    volumeMounts:
    - mountPath: /u01/scripts
      name: domain-upgrade-cm-volume
    - mountPath: %DOMAIN_MOUNT_PATH%
      name: domain-storage-volume
    - mountPath: /weblogic-operator/rcu-secrets
      name: rcu-credentials-volume
    env:
    - name: DOMAIN_UID
```

```
        value: "%DOMAIN_UID%"
      - name: DOMAIN_ROOT_DIR
        value: "%DOMAIN_ROOT_DIR%"
      - name: DOMAIN_HOME_DIR
        value: "%DOMAIN_HOME%"
      - name: DOMAIN_NAME
        value: "%DOMAIN_UID%"
      - name: CONNECTION_STRING
        value: "%CONNECTION_STRING%"
      - name: RCUPREFIX
        value: "%RCU_PREFIX%"
      - name: DOMAIN_TYPE
        value: "OAM"
      - name: SECURE_ENABLED
        value: "false"
  volumes:
  - name: domain-upgrade-cm-volume
    configMap:
      name: %DOMAIN_UID%-domain-upgrade-pod-cm
  - name: domain-storage-volume
    persistentVolumeClaim:
      claimName: %DOMAIN_PVC_NAME%
  - name: rcu-credentials-volume
    secret:
      secretName: %RCU_CREDENTIALS_SECRET_NAME%
```

For example:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    weblogic.domainUID: accessdomain
    weblogic.domainName: accessdomain
    app: accessdomain-domain-upgrade
  name: accessdomain-domain-upgrade
  namespace: oamns
spec:
  containers:
  - args:
    - sleep
    - infinity
    image: container-registry.oracle.com/middleware/oam_cpu:14.1.2.1.0-
jdk17-ol8-<YYMMDD>
    imagePullPolicy: IfNotPresent
    name: accessdomain-domain-upgrade
    volumeMounts:
    - mountPath: /u01/scripts
      name: domain-upgrade-cm-volume
    - mountPath: /u01/oracle/user_projects
      name: domain-storage-volume
    - mountPath: /weblogic-operator/rcu-secrets
      name: rcu-credentials-volume
    env:
```

```
    - name: DOMAIN_UID
      value: "accessdomain"
    - name: DOMAIN_ROOT_DIR
      value: " /u01/oracle/user_projects/domains"
    - name: DOMAIN_HOME_DIR
      value: " /u01/oracle/user_projects/domains/accessdomain"
    - name: DOMAIN_NAME
      value: "accessdomain"
    - name: CONNECTION_STRING
      value: "mydatabasehost.example.com:1521/orcl.example.com"
    - name: RCUPREFIX
      value: "OAMK8S"
    - name: DOMAIN_TYPE
      value: "OAM"
    - name: SECURE_ENABLED
      value: "false"
  volumes:
  - name: domain-upgrade-cm-volume
    configMap:
      name: accessdomain-domain-upgrade-pod-cm
  - name: domain-storage-volume
    persistentVolumeClaim:
      claimName: accessdomain-domain-pvc
  - name: rcu-credentials-volume
    secret:
      secretName: accessdomain-rcu-credentials
```

17.2.4 Shutting Down the OAM Domain

Run the following commands to shutdown the Oracle Access Management (OAM) 12c deployment:

1. Shut down the OAM deployment using the following command:

   ```
    kubectl patch domain %DOMAIN_UID% -n %NAMESPACE% --type=merge --patch
   "{\"spec\": {\"serverStartPolicy\": \"Never\"}}"
   ```

   For example:

   ```
    kubectl patch domain accessdomain -n oamns --type=merge --patch
   "{\"spec\": {\"serverStartPolicy\": \"Never\"}}"
   ```

   The output will look similar to the following:

   ```
   domain.weblogic.oracle/accessdomain patched
   ```

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

   ```
   kubectl get pods -n %NAMESPACE%
   ```

For example:

```
kubectl get pods -n oamns
```

The output will look similar to the following:

```
NAME                                                              READY
STATUS        RESTARTS       AGE
accessdomain-adminserver                                          1/1
Running       0              21h
accessdomain-oam-policy-mgr1                                      1/1
Running       0              21h
accessdomain-oam-server1                                          0/1
Terminating   0              21h
```

The Administration Server pods and Managed Server pods will move to a `STATUS` of `Terminating`.

After a few minutes, run the command again and make sure the pods should have disappeared before continuing.

3. If a helper pod exists, then delete it:

```
kubectl delete pod helper -n %NAMESPACE%
```

For example:

```
kubectl delete pod helper -n oamns
```

17.2.5 Backing Up the Database and Persistent Volume

You must take a backup of the Oracle Database used by Oracle Access Management (OAM) 12c, and the persistent volume.

Backing Up the Oracle Database

Take a backup of the Oracle Database used by OAM 12c, using your usual Oracle Database backup procedure.

Backing Up the Persistent Volume

Take a backup of the persistent volume directory:

```
sudo cp -rp <persistent_volume>/accessdomainpv <persistent_volume>/
accessdomain_bkp12c
```

For example:

```
sudo cp -rp /nfs_volumes/oam/accessdomainpv /nfs_volumes/oam/
accessdomainpv_bkp12c
```

17.2.6 Creating an Upgrade ConfigMap

Create a ConfigMap for the upgrade by performing the following:

1. Run the following command to create the ConfigMap:

```
kubectl create configmap %DOMAIN_UID%-domain-upgrade-pod-cm -n %NAMESPACE%
\
--from-file <workdir>/upgradescripts --dry-run=client -o yaml
```

For example:

```
kubectl create configmap accessdomain-domain-upgrade-pod-cm -n oamns \
--from-file /OAM12CUPG/upgradescripts --dry-run=client -o yaml | kubectl
apply -f -
```

The output should look similar to the following:

```
configmap/accessdomain-domain-upgrade-pod-cm created
```

17.2.7 Performing the Upgrade

To perform the upgrade you must create an upgrade pod and run several upgrade commands.

> **Note**
>
> If the upgrade fails, see Restoring After a Failed Upgrade.

1. Run the following command to create the `domain-upgrade-pod`:

```
kubectl apply -f <workdir>/upgradescripts/domain-upgrade-pod.yaml
```

For example:

```
kubectl apply -f /OAM12CUPG//upgradescripts/domain-upgrade-pod.yaml
```

The output should look similar to the following:

```
pod/accessdomain-domain-upgrade created
```

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

```
kubectl get pods -n %NAMESPACE%
```

For example:

```
kubectl get pods -n oamns
```

The output will look similar to the following:

```
NAME                                                              READY
STATUS    RESTARTS   AGE
accessdomain-domain-upgrade                                       1/1
Running   0          2m3s
```

It may take a few minutes until the pod is in a status of `READY 1/1`.

3. If you are running Kubernetes 1.30 or higher, set the following environment variable:

```
export KUBECTL_REMOTE_COMMAND_WEBSOCKETS=false
```

4. Run the following command to enter a bash shell in the `domain-upgrade` pod:

```
kubectl exec -it %DOMAIN_UID%-domain-upgrade -n %NAMESPACE% -- /bin/bash
```

For example:

```
kubectl exec -it accessdomain-domain-upgrade -n oamns -- /bin/bash
```

This will take you into a bash shell in the `domain-upgrade` pod:

```
[oracle@accessdomain-domain-upgrade oracle]$
```

5. Inside the upgrade pod, navigate to the `/u01/scripts` directory:

```
cd /u01/scripts
```

6. Run the following command to run the Upgrade Assistant:

```
$ORACLE_HOME/oracle_common/upgrade/bin/ua -response /u01/scripts/
domainUpgradeResponse.txt -logLevel TRACE -logDir /tmp
```

The output should look similar to the following:

```
Oracle Fusion Middleware Upgrade Assistant 14.1.2.0.0
Log file is located at: /tmp/ua<DATE>.log
Reading installer inventory, this will take a few moments...
...completed reading installer inventory.
UPGAST-00238: Warning: A dependency by component OIM on component SOA is
unresolved. Upgrades of component OIM will be disabled.
A dependent component was not found while discovering upgrade components
and their dependencies.
Make sure the required component has been installed.
Using response file /u01/scripts/domainUpgradeResponse.txt for input
 Oracle Platform Security Services schema examine is in progress
 Oracle Access Management Suite (Schema Upgrade) schema examine is in
```

```
progress
 Oracle Audit Services schema examine is in progress
 Oracle Metadata Services schema examine is in progress
 Oracle Platform Security Services schema examine finished with status:
ready for upgrade
 Oracle Access Management Suite (Schema Upgrade) schema examine finished
with status: ready for upgrade
 Oracle WebLogicServer schema examine is in progress
 Common Infrastructure Services schema examine is in progress
 Oracle Metadata Services schema examine finished with status: ready for
upgrade
 Common Infrastructure Services schema examine finished with status: ready
for upgrade
 Oracle WebLogicServer schema examine finished with status: ready for
upgrade
 Oracle Audit Services schema examine finished with status: ready for
upgrade
Schema Version Registry saved to: /tmp/ua<DATE>.xml
 Oracle Audit Services schema upgrade is in progress
 Oracle Platform Security Services schema upgrade is in progress
 Oracle Access Management Suite (Schema Upgrade) schema upgrade is in
progress
 Oracle Metadata Services schema upgrade is in progress
 Oracle Access Management Suite (Schema Upgrade) schema upgrade finished
with status: succeeded
 Common Infrastructure Services schema upgrade is in progress
 Common Infrastructure Services schema upgrade finished with status:
succeeded
 Oracle WebLogicServer schema upgrade is in progress
 Oracle Audit Services schema upgrade finished with status: succeeded
 Oracle Metadata Services schema upgrade finished with status: succeeded
 Oracle WebLogicServer schema upgrade finished with status: succeeded
 Oracle Platform Security Services schema upgrade finished with status:
succeeded
```

> **Note**
>
> The following message above can be ignored:
>
> ```
> UPGAST-00238: Warning: A dependency by component OIM on component
> SOA is unresolved. Upgrades of component OIM will be disabled.
> A dependent component was not found while discovering upgrade
> components and their dependencies.
> Make sure the required component has been installed.
> ```

7. Enter a `wlst` prompt inside the `domain-upgrade` pod:

   ```
   $ORACLE_HOME/oracle_common/common/bin/wlst.sh
   ```

The output will look similar to the following:

```
Initializing WebLogic Scripting Tool (WLST) ...

Jython scans all the jar files it can find at first startup. Depending on
the system, this process may take a few minutes to complete, and WLST may
not return a prompt right away.

Welcome to WebLogic Server Administration Scripting Shell

Type help() for help on available commands

wls:/offline>
```

8. Run the following commands at the `wls:/offline>` prompt to perform domain reconfiguration:
   a. Set the `domainHome`:

      ```
      domainHome='/u01/oracle/user_projects/domains/%DOMAIN_UID%'
      ```

      For example:

      ```
      domainHome='/u01/oracle/user_projects/domains/accessdomain'
      ```

   b. Read the `domanHome`:

      ```
      readDomainForUpgrade(domainHome)
      ```

      The output will look similar to the following:

      ```
      wls:/offline/accessdomain>
      ```

   c. Update the domain:

      ```
      updateDomain()
      ```

      > **Note**
      >
      > This command can take approximately 25 minutes to complete.

      The output will look similar to the following:

      ```
      <DATE> 9:28:33 AM
      oracle.security.jps.internal.config.xml.XmlConfigurationFactory
      validateFileLocation
      INFO: JPS Config: /u01/oracle/user_projects/domains/accessdomain/config/
      fmwconfig/jps-config-jse.xml
      <DATE> 9:28:34 AM
      oracle.security.jps.internal.config.xml.XmlConfigurationFactory
      validateFileLocation
      INFO: JPS Config: /u01/oracle/user_projects/domains/accessdomain/config/
      ```

```
fmwconfig/jps-config.xml
<DATE> 9:28:34 AM
oracle.security.opss.internal.runtime.ServiceContextManagerImpl
getContext
WARNING: Bootstrap services are used by OPSS internally and clients
should never need to directly read/write bootstrap credentials. If
required, use Wlst or configuration management interfaces.
<DATE> 9:28:35 AM
oracle.security.jps.internal.config.xml.XmlConfigurationFactory
validateFileLocation
INFO: JPS Config: /u01/oracle/user_projects/domains/accessdomain/config/
fmwconfig/jps-config-jse.xml
<DATE> 9:28:35 AM
oracle.security.jps.internal.config.xml.XmlConfigurationFactory
validateFileLocation
INFO: JPS Config: /u01/oracle/user_projects/domains/accessdomain/config/
fmwconfig/jps-config.xml
<DATE> 9:54:05 AM
oracle.security.jps.az.internal.runtime.policy.AbstractPolicyImpl
initializeReadStore
INFO: Property for read store in parallel:
oracle.security.jps.az.runtime.readstore.threads = null
wls:/offline/accessdomain>
```

d. Close the domain:

```
closeDomain()
```

9. Run the following post upgrade steps at the `wls:/offline>` prompt:

a. Set the `domainHome`:

```
domainHome='/u01/oracle/user_projects/domains/%DOMAIN_UID%'
```

For example:

```
domainHome='/u01/oracle/user_projects/domains/accessdomain'
```

b. Set the `RCUPREFIX` and `RCU_SCHEMA_PWD` password:

```
wlsRuntimeUser='%RCUPREFIX%_WLS_RUNTIME'


schemaPassword='%RCU_SCHEMA_PWD%'
```

For example:

```
wlsRuntimeUser='OAMK8S_WLS_RUNTIME'


schemaPassword='<password>'
```

c. Read the domain:

```
readDomain(domainHome)
```

d. Change to the following directory:

```
cd('/JdbcSystemResource/WLSRuntimeSchemaDataSource/JdbcResource/
WLSRuntimeSchemaDataSource/JdbcDriverParams/NO_NAME_0')
```

e. Set the `CONNECTION_STRING`:

```
dbUrl="jdbc:oracle:thin:@%CONNECTION_STRING%"
```

For example:

```
dbUrl="jdbc:oracle:thin:@mydatabasehost.example.com:1521/
orcl.example.com"
```

f. Run the following to update the domain:

```
cmo.setUrl(dbUrl)


cmo.setDriverName('oracle.jdbc.OracleDriver')


set('PasswordEncrypted', schemaPassword)


cd('Properties/NO_NAME_0/Property/user')


cmo.setValue(wlsRuntimeUser)


cd('/')


updateDomain()
```

No output will be returned to the screen and you will just be returned to the prompt.

g. Close the domain and exit:

```
closeDomain()


exit()
```

10. Exit the `domain-upgrade` pod:

```
exit
```

17.2.8 Updating the OAM Container Image to 14c

You must update the deployment to use the Oracle Access Management (OAM) 14c container image:

> **Note**
>
> If the upgrade fails, see Restoring After a Failed Upgrade.

1. Run the following command to update the deployment with the OAM 14c container image:

   ```
   kubectl patch domain %DOMAIN_UID% -n %NAMESPACE% --type merge  -p '{"spec":
   {"image":"%WEBLOGIC_IMAGE%"}}'
   ```

   For example:

   - If using Oracle Container Registry or your own container registry for your OAM container image:

     ```
     kubectl patch domain accessdomain -n oamns --type merge  -p '{"spec":
     {"image":"container-registry.oracle.com/middleware/oam_cpu:<new_tag>"}}'
     ```

   - If you are not using a container registry and have loaded the image on each of the worker nodes:

     ```
     kubectl patch domain accessdomain -n oamns --type merge  -p '{"spec":
     {"image":"oracle/oam:14.1.2.1.0"}}'
     ```

     The output will look similar to the following:

     ```
     domain.weblogic.oracle/accessdomain patched
     ```

17.2.9 Updating the WebLogic Kubernetes Operator

You must update the deployment to use a WebLogic Kubernetes Operator version supported with 14c. The current supported version is 4.2.10.

1. Run the following command to see the current version of the WebLogic Kubernetes Operator:

   ```
   helm list -n opns
   ```

   The output will look similar to the following:

   ```
   NAME                            NAMESPACE       REVISION        UPDATED
   STATUS          CHART                                   APP VERSION
   weblogic-kubernetes-operator    opns            1               <DATE>
   ```

```
deployed        weblogic-operator-4.1.8-RELEASE-MARKER  4.1.8-RELEASE-
MARKER
```

In the above example the version is `4.1.8` and therefore the operator must be upgraded.

2. To upgrade the WebLogic Kubernetes Operator to 4.2.10, see Upgrading WebLogic Kubernetes Operator.
3. Once the operator is upgraded, continue with Starting the OAM 14c Deployment.

17.2.10 Starting the OAM 14c Deployment

Start the Oracle Access Management (OAM) 14c deployment.

1. Run the following command to start the OAM domain:

```
kubectl patch domain.v9.weblogic.oracle "%DOMAIN_UID%" -n "%NAMESPACE%" \
--type=merge --patch "{\"spec\": {\"serverStartPolicy\": \"IfNeeded\"}}"
```

For example:

```
kubectl patch domain.v9.weblogic.oracle "accessdomain" -n "oamns" \
--type=merge --patch "{\"spec\": {\"serverStartPolicy\": \"IfNeeded\"}}"
```

The output will look similar to the following:

```
domain.weblogic.oracle/accessdomain patched
```

2. Run the following command to view the pods:

```
kubectl get pods -n %NAMESPACE%
```

For example:

```
kubectl get pods -n oamns
```

The output will look similar to the following:

```
NAME                                                        READY
STATUS        RESTARTS   AGE
accessdomain-introspector-jwqxw                             1/1
Running     0          10s
```

The introspect job will start, followed by the Administration Server pod, and then the OAM server pods. This process will take several minutes, so keep executing the command until all the pods are running with `READY` status `1/1`:

> **Note**
>
> Alternatively, you can add the watch flag, `-w`, which allows you watch the status of the pods as they change.

```
NAME                                                     READY
STATUS      RESTARTS   AGE
accessdomain-adminserver                                 1/1
Running     0          10m
accessdomain-oam-policy-mgr1                             1/1
Running     0          7m35s
accessdomain-oam-server1                                 1/1
Running     0          7m35s
```

3. Once everything is running, check the consoles are accessible as per Validating the Domain URLs.
4. Once you are confident the upgrade is successful, delete the `domain-upgrade` pod and ConfigMap as follows:

   ```
   kubectl delete pod %DOMAIN_UID%-domain-upgrade -n %NAMESPACE%
   ```

   ```
   kubectl delete configmap %DOMAIN_UID%-domain-upgrade-pod-cm -n %NAMESPACE%
   ```

   For example:

   ```
   kubectl delete pod accessdomain-domain-upgrade -n oamns
   ```

   ```
   kubectl delete configmap accessdomain-domain-upgrade-pod-cm -n oamns
   ```

17.2.11 Upgrading the Ingress

In order to access the Oracle Access Management (OAM) 14c domain via WebLogic Remote Console, you must upgrade the ingress.

1. Download the latest code repository to a new directory and set the `$WORKDIR` to the new directory structure. See, Setting Up the Code Repository for OAM.

   > **Note**
   >
   > Make sure not to delete the original OAM 12c code repository as you will need the `values.yaml` used to create the original ingress.

2. Navigate to the following directory:

   ```
   cd $WORKDIR/kubernetes/charts/ingress-per-domain
   ```

3. Make a copy of the `values.yaml`:

   ```
   cp values.yaml $WORKDIR/
   ```

4. Copy over the `values.yaml` from the original OAM 12c code repository. For example:

   ```
   cp /OAMK8S/fmw-kubernetes/OracleAccessManagement/kubernetes/charts/ingress-
   per-domain \
   $WORKDIR/kubernetes/charts/ingress-per-domain
   ```

5. Upgrade the `oam-nginx` with the following commands:

   ```
   cd $WORKDIR
   ```

   ```
   helm upgrade oam-nginx kubernetes/charts/ingress-per-domain/ --namespace
   %NAMESPACE% \
   --values kubernetes/charts/ingress-per-domain/values.yaml --reuse-values
   ```

   For example:

   ```
   helm upgrade oam-nginx kubernetes/charts/ingress-per-domain/ --namespace
   oamns \
   --values kubernetes/charts/ingress-per-domain/values.yaml --reuse-values
   ```

   The output will look similar to the following:

   ```
   Release "oam-nginx" has been upgraded. Happy Helming!
   NAME: oam-nginx
   LAST DEPLOYED: <DATE>
   NAMESPACE: oamns
   STATUS: deployed
   REVISION: 2
   TEST SUITE: None
   ```

6. Check you can connect to the OAM 14c domain using the WebLogic Remote Console.

   > **Note**
   >
   > For more information about installing and configuring the console, see Getting Started Using Administration Console.

17.2.12 Restoring After a Failed Upgrade

If the upgrade fails at any point, you can restore back to the Oracle Access Management (OAM) 12c deployment using the following steps:

1. Shut down the OAM 14c deployment using the following command:

   ```
    kubectl patch domain <domain> -n <domain_namespace> --type=merge --patch
   "{\"spec\": {\"serverStartPolicy\": \"Never\"}}"
   ```

For example:

```
 kubectl patch domain accessdomain -n oamns --type=merge --patch
"{\"spec\": {\"serverStartPolicy\": \"Never\"}}"
```

The output will look similar to the following:

```
domain.weblogic.oracle/accessdomain patched
```

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

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

For example:

```
kubectl get pods -n oamns
```

The output will look similar to the following:

```
NAME                                                      READY
STATUS        RESTARTS   AGE
accessdomain-adminserver                                  1/1
Running       0          3h52m
accessdomain-oam-policy-mgr1                              1/1
Running       0          3h44m
accessdomain-oam-server1                                  1/1
Terminating   0          3h44m
```

The Administration Server pods and Managed Server pods will move to a `STATUS` of `Terminating`.

After a few minutes, run the command again and make sure the pods should have disappeared before continuing.

3. Restore the persistent volume from the backup taken before the upgrade:

```
sudo cp -rp <persistent_volume>/accessdomainpv <persistent_volume>/
accessdomain_bkp14c
```

```
sudo rm -rf <persistent_volume>/accessdomainpv
```

```
sudo cp -rp <persistent_volume>/accessdomainpv_bkp12c <persistent_volume>/
accessdomain
```

For example:

```
sudo cp -rp /nfs_volumes/oam/accessdomainpv /nfs_volumes/oam/
accessdomain_bkp14c


sudo rm -rf /nfs_volumes/oam/accessdomainpv


sudo cp -rp /nfs_volumes/oam/accessdomainpv_bkp12c /nfs_volumes/oam/
accessdomain
```

4. Restore the Oracle Database from the backup taken before the upgrade.
5. Run the following command to update the deployment with the OAM 12c container image used previously:

```
kubectl patch domain %DOMAIN_UID% -n %NAMESPACE% --type merge  -p '{"spec":
{"image":"%WEBLOGIC_IMAGE%"}}'
```

For example:

- If using Oracle Container Registry or your own container registry for your OAM container image:

```
kubectl patch domain accessdomain -n oamns \
--type merge  -p '{"spec":{"image":"container-registry.oracle.com/
middleware/oam_cpu:12.2.1.4-jdk8-ol8-<YYMMDD>"}}'
```

- If you are not using a container registry and have loaded the image on each of the worker nodes:

```
kubectl patch domain accessdomain -n oamns \
--type merge  -p '{"spec":{"image":"oracle/oam:12.2.1.4.0"}}'
```

The output will look similar to the following:

```
domain.weblogic.oracle/accessdomain patched
```

6. Downgrade the WebLogic Kubernetes Operator to a version supported by OAM 12c. Follow the instructions at, Updating the WebLogic Kubernetes Operator, but use a supported operator for OAM 12c.
7. Run the following command to start the OAM domain:

```
kubectl patch domain.v9.weblogic.oracle "%DOMAIN_UID%" -n "%NAMESPACE%" \
--type=merge --patch "{\"spec\": {\"serverStartPolicy\": \"IfNeeded\"}}"
```

For example:

```
kubectl patch domain.v9.weblogic.oracle "accessdomain" -n "oamns" \
--type=merge --patch "{\"spec\": {\"serverStartPolicy\": \"IfNeeded\"}}"
```

The output will look similar to the following:

```
domain.weblogic.oracle/accessdomain patched
```

8. Run the following command to view the pods:

```
kubectl get pods -n %NAMESPACE%
```

For example:

```
kubectl get pods -n oamns
```

The output will look similar to the following:

```
NAME                                                        READY
STATUS        RESTARTS   AGE
accessdomain-introspector-jwqxw                             1/1
Running     0          10s
```

The introspect job will start, followed by the Administration Server pod, and then the OAM server pods. This process will take several minutes, so keep executing the command until all the pods are running with `READY` status `1/1`:

> **Note**
>
> Alternatively, you can add the watch flag, `-w`, which allows you watch the status of the pods as they change.

```
NAME                                                        READY
STATUS      RESTARTS   AGE
accessdomain-adminserver                                    1/1
Running     0          10m
accessdomain-oam-policy-mgr1                                1/1
Running     0          7m35s
accessdomain-oam-server1                                    1/1
Running     0          7m35s
```

9. Once everything is running, check the consoles are accessible as per Validating the Domain URLs.

18
General Troubleshooting

This chapter includes the following topics:

- Viewing Pod Logs
- Viewing Pod Descriptions
- Known Issues

18.1 Viewing Pod Logs

To view logs for a pod use the following command:

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

For example:

```
kubectl logs accessdomain-oam-server1 -n oamns
```

> **Note**
>
> If you add `-f` to the command, then the log will be streamed.

18.2 Viewing Pod Descriptions

Details about a pod can be viewed using the `kubectl describe` command:

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

For example:

```
kubectl describe pod accessdomain-oam-server1 -n oamns
```

The output will look similar to the following:

```
Name:             accessdomain-oam-server1
Namespace:        oamns
Priority:         0
Service Account:  default
Node:             worker-node1/100.105.211.49
Start Time:       <DATE>
Labels:           weblogic.clusterName=oam_cluster
                  weblogic.clusterObservedGeneration=1
```

```
                    weblogic.createdByOperator=true
                    weblogic.domainName=accessdomain
                    weblogic.domainObservedGeneration=3
                    weblogic.domainUID=accessdomain
                    weblogic.operatorVersion=4.2.10
                    weblogic.serverName=oam_server1
Annotations:        prometheus.io/path: /wls-exporter/metrics
                    prometheus.io/port: 14100
                    prometheus.io/scrape: true
                    weblogic.sha256:
6be088360c66ca7e30d3e77399ca888b61a5c20d6c46939f59f00667067b3f3b
Status:             Running
SeccompProfile:     RuntimeDefault
IP:                 10.244.1.43
IPs:
  IP:           10.244.1.43
Controlled By:  Domain/accessdomain
Containers:
  weblogic-server:
    Container ID:  cri-o://
61135f79df711adddab9e935fd778df333ac4bd96695760ced658604a70d4d04
    Image:         container-registry.oracle.com/middleware/
oam_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD>
    Image ID:      container-registry.oracle.com/middleware/
oam_cpu@sha256:1c29d2688506f29be17a49a084e4bffaf8a224c1e328a8bc2224f8598a168e4
e
    Port:          14100/TCP
    Host Port:     0/TCP
    Command:
      /weblogic-operator/scripts/startServer.sh
    State:          Running
      Started:      <DATE>
    Last State:     Terminated
      Reason:       Error
      Exit Code:    137
      Started:      <DATE>
      Finished:     <DATE>
    Ready:          True
    Restart Count:  1
    Limits:
      cpu:     2
      memory:  8Gi
    Requests:
      cpu:      1
      memory:   4Gi
    Liveness:   exec [/weblogic-operator/scripts/livenessProbe.sh] delay=30s
timeout=5s period=45s #success=1 #failure=1
    Readiness:  http-get http://:14100/weblogic/ready delay=30s timeout=5s
period=5s #success=1 #failure=1
    Environment:
      USER_MEM_ARGS:                     -XX:+UseContainerSupport -
Djava.security.egd=file:/dev/./urandom -Xms8192m -Xmx8192m
      JAVA_OPTIONS:                      -Dweblogic.StdoutDebugEnabled=false
      DOMAIN_NAME:                       accessdomain
      DOMAIN_HOME:                       /u01/oracle/user_projects/domains/
accessdomain
```

```
      ADMIN_NAME:                          AdminServer
      ADMIN_PORT:                          7001
      SERVER_NAME:                         oam_server1
      DOMAIN_UID:                          accessdomain
      NODEMGR_HOME:                        /u01/nodemanager
      LOG_HOME:                            /u01/oracle/user_projects/domains/
logs/accessdomain
      SERVER_OUT_IN_POD_LOG:               true
      SERVICE_NAME:                        accessdomain-oam-server1
      AS_SERVICE_NAME:                     accessdomain-adminserver
      ADMIN_USERNAME:
      ADMIN_PASSWORD:
      LOCAL_ADMIN_PORT:                    14100
      LOCAL_ADMIN_PROTOCOL:                t3
      SHUTDOWN_TYPE:                       Graceful
      SHUTDOWN_TIMEOUT:                    30
      SHUTDOWN_IGNORE_SESSIONS:            false
      REPLACE_VARIABLES_IN_JAVA_OPTIONS:   false
      DYNAMIC_CONFIG_OVERRIDE:             true
      DOMAIN_SOURCE_TYPE:                  PersistentVolume
    Mounts:
      /u01/oracle/user_projects/domains from weblogic-domain-storage-volume
(rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-
pmvtq (ro)
      /weblogic-operator/debug from weblogic-domain-debug-cm-volume (ro)
      /weblogic-operator/introspector from weblogic-domain-introspect-cm-
volume (rw)
      /weblogic-operator/scripts from weblogic-scripts-cm-volume (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  weblogic-scripts-cm-volume:
    Type:      ConfigMap (a volume populated by a ConfigMap)
    Name:      weblogic-scripts-cm
    Optional:  false
  weblogic-domain-debug-cm-volume:
    Type:      ConfigMap (a volume populated by a ConfigMap)
    Name:      accessdomain-weblogic-domain-debug-cm
    Optional:  true
  weblogic-domain-introspect-cm-volume:
    Type:      ConfigMap (a volume populated by a ConfigMap)
    Name:      accessdomain-weblogic-domain-introspect-cm
    Optional:  false
  weblogic-domain-storage-volume:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim
in the same namespace)
    ClaimName:  accessdomain-domain-pvc
    ReadOnly:   false
  kube-api-access-pmvtq:
    Type:                    Projected (a volume that contains injected data
```

```
from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   Burstable
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists
for 300s
                             node.kubernetes.io/unreachable:NoExecute
op=Exists for 300s
Events:
  Type     Reason                          Age                    From
Message
  ----     ------                          ----                   ----
-------
```

18.3 Known Issues

This section contains information about known issues.

Domain Creation Failure With WLST

The instructions in this section relate to problems creating Oracle Access Management (OAM) domains using WLST. See, Creating OAM Domains Using WLST Offline Scripts.

If the OAM domain creation fails, run the following to diagnose the issue:

```
kubectl logs <domain_job> -n <domain_namespace>
```

For example:

```
kubectl logs accessdomain-create-fmw-infra-sample-domain-job-c6vfb -n oamns
```

Also run:

```
kubectl describe pod <domain_job> -n <domain_namespace>
```

For example:

```
kubectl describe pod accessdomain-create-fmw-infra-sample-domain-job-c6vfb -n
oamns
```

Using the output you should be able to diagnose the problem and resolve the issue.

If any of the above commands return the following error:

```
Failed to start container "create-fmw-infra-sample-domain-job": Error
response from daemon: error while creating mount source path
'/nfs_volumes/oam/accessdomainpv ': mkdir /nfs_volumes/oam/accessdomainpv :
permission denied
```

Then there is a permissions error on the directory for the PV and PVC and the following should be checked:

- The directory has 777 permissions: `chmod -R 777 <persistent_volume>/accessdomainpv`.
- If it does have the permissions, check if an oracle user exists and the uid is 1000 and gid is 0.
  Create the oracle user if it doesn't exist and set the uid to 1000 and gid to 0.
- Edit the `$WORKDIR/kubernetes/create-weblogic-domain-pv-pvc/create-pv-pvc-inputs.yaml` and add a slash to the end of the directory for the `weblogicDomainStoragePath` parameter:

  ```
  weblogicDomainStoragePath: /nfs_volumes/oam/accessdomainpv/
  ```

Once you have diagnosed the problem, clean down the failed domain creation by following:

- Deleting the OAM Domain
- Deleting RCU Schemas
- Deleting Persistent Volume Contents

Then follow the instructions again in Creating OAM Domains Using WLST Offline Scripts

Domain Creation Failure With WDT Models

The instructions in this section relate to problems creating OAM domains using WDT models. See, Creating OAM Domains Using WDT Models.

If the domain creation fails while creating domain resources using the `domain.yaml` file, run the following steps to diagnose the issue:

1. Check the domain events, by running the following command:

   ```
   kubectl describe domain <domain name> -n <domain_namespace>
   ```

   For example:

   ```
   kubectl describe domain accessdomain -n oamns
   ```

   Using the output, you should be able to diagnose the problem and resolve the issue.
2. If the instrospector job fails due to validation errors, then you can recreate the domain resources using the commands:

   ```
   kubectl delete -f domain.yaml
   ```

   ```
   kubectl create -f domain.yaml
   ```

3. If the domain creation fails because of database issues:
   a. Create a helper pod:

      ```
       kubectl run --image=container-registry.oracle.com/middleware/
      oam_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD> --image-pull-
      policy="IfNotPresent" --overrides='{"apiVersion": "v1","spec":
      ```

```
{"imagePullSecrets": [{"name": "orclcred"}]}}' helper -n oamns -- sleep
infinity
```

b. Once you have diagnosed the problem, clean down the failed domain creation by following:

- Deleting the OAM Domain
- Deleting RCU Schemas
- Deleting Persistent Volume Contents

c. Execute the steps in Creating OAM Domains Using WDT Models again.

> **Note**
>
> You might need to recreate the domain creation image depending upon the errors. Domain creation logs are stored in `<persistent_volume>/domains/wdt-logs`.

4. If there is any issues bringing up the administration server, OAM managed server pods or policy manager pods, you can run the following to check the logs:

```
kubectl logs <pod> -n <domain_namespace>
```

For example:

```
kubectl logs accessdomain-adminserver -n oamns
```

If the above does not give any information you can also run:

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

For example:

```
kubectl describe pod accessdomain-adminserver -n oamns
```

For more details related to debugging issues, refer to Domain Debugging.

Pods Restarting Due to LivenessProbe

If the server pods keep restarting due to `livenessProbe` or `readinessProbe` failure, then make the following changes in the `oam-cluster` and `policy-cluster` respectively.

If the restart is due to resource (CPU in this case) limit issue, then the CPU parameter need to be adjusted as needed:

```
kubectl edit cluster <cluster> -n <domain_namespace>
```

For example:

```
kubectl edit cluster accessdomain-oam-cluster -n oamns
```

In the edit session change the CPU parameter as follows:

```
...
spec:
  clusterName: oam_cluster
  replicas: 1
  serverPod:
    env:
    - name: USER_MEM_ARGS
      value: -XX:+UseContainerSupport -Djava.security.egd=file:/dev/./urandom
-Xms8192m
        -Xmx8192m
    livenessProbe:
      failureThreshold: 3
      periodSeconds: 60
      timeoutSeconds: 60
    readinessProbe:
      failureThreshold: 3
      periodSeconds: 60
      timeoutSeconds: 60
    resources:
      limits:
        cpu: 1700m
        memory: 6Gi
      requests:
        cpu: 500m
        memory: 4Gi
...spec:
```

19

Deleting an OAM Deployment

This chapter explains how to delete the Oracle Access Management (OAM) domain and other Kubernetes objects used by the OAM domain.

The instructions in this chapter should only be followed if you need to remove a certain part of the domain because of a deployment failure, or if you need clear the domain down completely for some other reason. If you are unsure consult Oracle Support.

This chapter includes the following topics:

19.1 Deleting the OAM Domain

The steps to delete an Oracle Access Management (OAM) domain depends on whether the domain was created with WLST or WDT.

Deleting WLST OAM Domains

1. Navigate to the `$WORKDIR/kubernetes/delete-domain` directory:

   ```
   cd $WORKDIR/kubernetes/delete-domain
   ```

2. Run the following command to delete the domain:

   ```
   ./delete-weblogic-domain-resources.sh -d <domain_uid>
   ```

   For example:

   ```
   ./delete-weblogic-domain-resources.sh -d accessdomain
   ```

Deleting WDT OAM Domains

1. Run the following command to delete the domain and clusters:

   ```
   kubectl delete -f $WORKDIR/yaml/domain.yaml
   ```

2. Navigate to the `$WORKDIR/kubernetes/delete-domain` directory:

   ```
   cd $WORKDIR/kubernetes/delete-domain
   ```

3. Run the following command to remove other domain objects:

```
./delete-weblogic-domain-resources.sh -d <domain_uid>
```

For example:

```
./delete-weblogic-domain-resources.sh -d accessdomain
```

19.2 Deleting RCU Schemas

To delete the RCU schemas, perform the following steps:

1. Check to see if the helper pod exists by running:

```
kubectl get pods -n <domain_namespace> | grep helper
```

For example:

```
kubectl get pods -n oamns | grep helper
```

The output should look similar to the following:

```
helper                                     1/1     Running     0          26h
```

If the helper pod doesn't exist, run the following:

- If using Oracle Container Registry or your own container registry for the Oracle Access Management (OAM) container image:

```
kubectl run --image=<image_name-from-registry>:<tag> \
--image-pull-policy="IfNotPresent" \
--overrides='{"apiVersion": "v1", "spec":{"imagePullSecrets": [{"name":
"orclcred"}]}}' \
helper -n <domain_namespace> \
-- sleep infinity
```

For example:

```
kubectl run --image=container-registry.oracle.com/middleware/
oam_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD> \
--image-pull-policy="IfNotPresent" \
--overrides='{"apiVersion": "v1","spec":{"imagePullSecrets": [{"name":
"orclcred"}]}}' \
helper -n oamns \
-- sleep infinity
```

- If you are not using a container registry and have loaded the image on each of the worker nodes, run the following command:

```
kubectl run helper --image <image>:<tag> -n oamns -- sleep infinity
```

For example:

```
kubectl run helper --image oracle/oam_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD>
-n oamns -- sleep infinity
```

The output will look similar to the following:

```
pod/helper created
```

2. Run the following command to start a bash shell in the helper pod:

```
kubectl exec -it helper -n <domain_namespace> -- /bin/bash
```

For example:

```
kubectl exec -it helper -n oamns -- /bin/bash
```

This will take you into a bash shell in the running helper pod:

```
[oracle@helper ~]$
```

3. In the helper bash shell run the following commands to set the environment:

```
export CONNECTION_STRING=<db_host.domain>:<db_port>/<service_name>


export RCUPREFIX=<rcu_schema_prefix>


echo -e <db_pwd>"\n"<rcu_schema_pwd> > /tmp/pwd.txt
```

Where:

- `<db_host.domain>:<db_port>/<service_name>` is your database connect string.
- `<rcu_schema_prefix>` is the RCU schema prefix.
- `<db_pwd>` is the SYS password for the database.
- `<rcu_schema_pwd>` is the password for the `<rcu_schema_prefix>`

For example:

```
export CONNECTION_STRING=mydatabasehost.example.com:1521/orcl.example.com


export RCUPREFIX=OAMK8S


echo -e <password>"\n"<password> > /tmp/pwd.txt


cat /tmp/pwd.txt
```

Ensure the `cat /tmp/pwd.txt` command shows the correct passwords.

4. In the helper bash shell, drop the RCU schemas as follows:

```
/u01/oracle/oracle_common/bin/rcu -silent -dropRepository -databaseType
ORACLE -connectString $CONNECTION_STRING \
-dbUser sys -dbRole sysdba -selectDependentsForComponents true -
schemaPrefix $RCUPREFIX \
-component MDS -component IAU -component IAU_APPEND -component IAU_VIEWER -
component OPSS \
-component WLS -component STB -component OAM -f < /tmp/pwd.txt
```

5. Exit the helper bash shell by issuing the command `exit`.

19.3 Deleting Persistent Volume Contents

Perform the following step to delete the persistent volume contents:

```
rm -rf <persistent_volume>/accessdomainpv/*
```

For example:

```
rm -rf /nfs_volumes/oam/accessdomainpv/*
```

19.4 Deleting the WebLogic Kubernetes Operator

To delete the WebLogic Kubernetes Operator, perform the following steps:

1. Run the following command to remove the operator:

```
helm delete weblogic-kubernetes-operator -n opns
```

2. Delete the label from the OAM namespace::

```
kubectl label namespaces <domain_namespace> weblogic-operator-
```

For example:

```
kubectl label namespaces oamns weblogic-operator-
```

3. Delete the service account for the operator:

```
kubectl delete serviceaccount <sample-kubernetes-operator-sa> -n
<domain_namespace>
```

For example:

```
kubectl delete serviceaccount op-sa -n opns
```

4. Delete the operator namespace:

```
kubectl delete namespace <sample-kubernetes-operator-ns>
```

For example:

```
kubectl delete namespace opns
```

19.5 Deleting the Ingress

Perform the following steps to delete the ingress and ingress controller:

1. To delete the ingress:

   ```
   helm delete oam-nginx -n <domain_namespace>
   ```

   For example:

   ```
   helm delete oam-nginx -n oamns
   ```

2. To delete the ingress controller:

   ```
   helm delete nginx-ingress -n <domain_namespace>
   ```

   For example:

   ```
   helm delete nginx-ingress -n mynginxns
   ```

3. Delete the namespace using the following command:

   ```
   kubectl delete namespace <domain_namespace>
   ```

   For example:

   ```
   kubectl delete namespace mynginxns
   ```

19.6 Deleting the OAM Namespace

Perform the following step to delete the Oracle Access Management (OAM) namespace:

1. Delete the helper pod if it is running:

   ```
   kubectl delete pod helper -n <domain_namepace>
   ```

   For example:

   ```
   kubectl delete pod helper -n oamns
   ```

2. Check to make sure all Kubernetes in the namespace are deleted:

   ```
   kubectl get all,domains -n <domain_namepace>
   ```

For example:

```
kubectl get all,domains -n oamns
```

If any objects remain, delete them manually.

3. Delete the namespace using the following command:

```
kubectl delete namespace <domain_namespace>
```

For example:

```
kubectl delete namespace oamns
```