Oracle® Communications Order and Service Management

Cloud Native Deployment Guide

Release 7.4.1
F32157-03
May 2021

Oracle Communications Order and Service Management Cloud Native Deployment Guide, Release 7.4.1

F32157-03

Contents

Preface

This document describes how to install and administer Oracle Communications Order and Service Management (OSM) Cloud Native Deployment.

Audience

This document is intended for DevOps administrators and those involved in installing and maintaining Oracle Communications Order and Service Management (OSM) Cloud Native Deployment.

Documentation Accessibility

1

Overview of the OSM Cloud Native Deployment

Get an overview of Oracle Communications Order and Service Management (OSM) cloud native deployment, architecture, and the OSM cloud native toolkit.

This chapter provides an overview of Oracle Communications Order and Service Management (OSM) deployed in a cloud native environment using container images and a Kubernetes cluster.

About the OSM Cloud Native Deployment

You can deploy OSM in a Kubernetes-based shared cloud (cluster) while implementing modern DevOps “Configuration as Code” principles to manage system configuration in a consistent manner. You can automate system lifecycle management. You set up your own cloud native environment and can then use the OSM cloud native toolkit to automate the deployment of OSM instances. By leveraging the pre-configured Helm charts, you can deploy OSM instances quickly ensuring your services are up and running in far less time than a traditional deployment.

OSM cloud native supports the following deployment models:

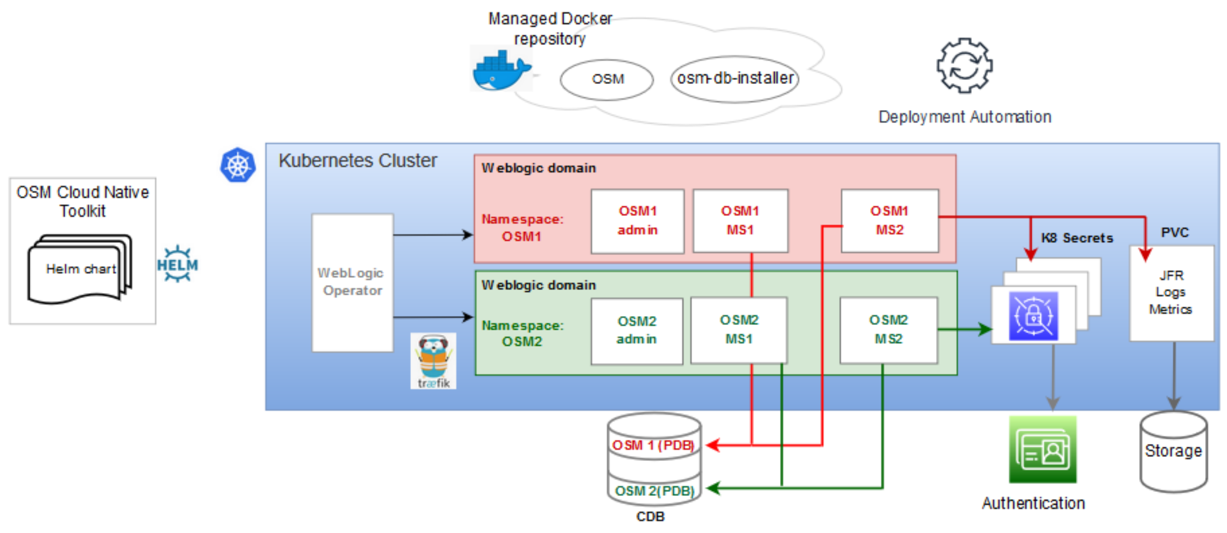
- **On Private Kubernetes Cluster**: OSM cloud native is certified for a general deployment of Kubernetes.
- **On Oracle Cloud Infrastructure Container Engine for Kubernetes (OKE)**: OSM cloud native is certified to run on Oracle's hosted Kubernetes OKE service.

OSM Cloud Native Architecture

This section describes and illustrates the OSM cloud native architecture and the deployment environment.

The following diagram illustrates the OSM cloud native architecture.

Figure 1-1   OSM Cloud Native Architecture

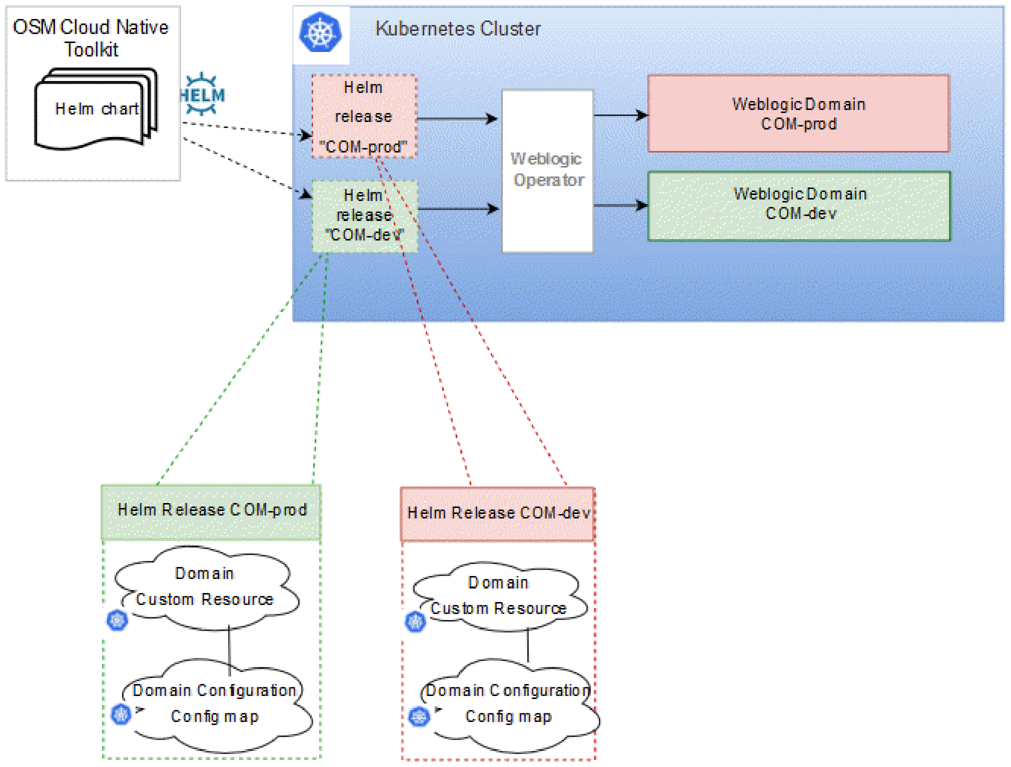


The OSM cloud native architecture requires components such as the Kubernetes cluster and WebLogic Kubernetes Operator, which are under your control to install and configure. A single WebLogic Operator can manage multiple OSM domains in multiple namespaces. Each domain is a dynamic cluster with multiple managed servers that is configured for integration with both optional and required components. The OSM cloud native artifacts include two container images built using Docker and the OSM cloud native toolkit.

About the WebLogic Domain

The following diagram illustrates the OSM cloud native deployment environment and important concepts about producing a WebLogic domain that is capable of supporting OSM cloud native.

Figure 1-2 OSM Cloud Native Deployment Environment



In the deployment environment, the Helm chart that is provided with the OSM cloud native toolkit is deployed into the Kubernetes cluster producing two Kubernetes resources. These resources are then consumed by the WebLogic Kubernetes Operator (WKO).

About Kubernetes Custom Resource Definitions (CRD) and Domain Configuration Config Map

The Kubernetes API provides extensions called custom resources. To understand more about a Custom Resource Definition (CRD) and why it might be used, see the Kubernetes CustomResourceDefinition (CRD) documentation at: https://kubernetes.io/docs/tasks/access-kubernetes-api/custom-resources/custom-resource-definitions/

To configure the operation of your WebLogic domain, you set up and configure your own domain resource. The domain resource does not replace the traditional configuration of the WebLogic domains found in the domain configuration files, but instead co-operates with those files to describe the Kubernetes artifacts of the corresponding domain. Refer to the Oracle WebLogic Kubernetes Operator User Guide to understand how to use a CRD to describe a WebLogic domain resource.

While the domain resource describes much of the operational details for a domain such as domain identification, secrets, pod creation, server instances, startup and shutdown, security, logging, clusters, admin and managed servers, and JVM options, the details about the more traditional configuration (deployed applications, JMS Queues, data sources and so on) are provided in a configuration map and are described using a metadata model specified by the Weblogic Deploy Tooling (WDT).

The OSM cloud native toolkit provides the base configuration to produce these resources.

About Oracle WebLogic Server Deploy Tooling (WDT)

The WebLogic Server Deploy Tooling (WDT) has the following main purposes:

- It provides a metadata model that describes a WebLogic Server domain configuration.
- It provides scripts that perform domain lifecycle operations, simplifying the definition and the creation of domains. This capability provides an alternative to programmatic ways of defining domain configuration such as WebLogic Scripting Tool (WLST) or Java Mbeans manipulation.

The OSM cloud native toolkit leverages the WDT metadata model only. It does not use the scripting capabilities directly.

The toolkit provides the WDT metadata for a domain that is capable of supporting OSM. The toolkit enables you to easily override much of the base configuration through the use of Helm charts. Additionally, the toolkit framework allows you to add supplementary WDT metadata fragments to the domain. WDT provides tools that help with this task by inspecting an existing domain to produce the WDT metadata required for the configuration.

For more details about WDT, see the Oracle WebLogic Server Deploy Tooling documentation on GitHub at: https://github.com/oracle/weblogic-deploy-tooling

About Projects and Instances

A project is a function of OSM. Examples of OSM functions include order management roles such as SOM and COM. For example, in a COM role, a solution cartridge contains configuration requirements that dictate how COM processes orders. This might include the JMS queues for messaging, credentials for communication with external systems, additional applications deployed to the WebLogic server (external system emulators), or SAF setup for connectivity to peer systems. All of these configuration requirements can be scoped to a project.

An instance is a specific flavor of OSM for a given project. Test, development, and production are all instances of an OSM COM project. Some bits of the configuration makes more sense to be applied on a per-instance basis. The production instance of OSM in a COM role uses different values for tuning parameters and may employ a different logging and metrics strategy than a development instance of COM.

In order to create a running WebLogic domain, the target project and instance must be determined so that the appropriate configuration can be assembled.

About Specification Layers

The OSM configuration defines the footprint, layout and tuning of OSM. Treating this as one monolithic configuration is not optimal for sustainability or risk management. The result is a layered approach to the configuration.

There are three layers defined, each scoping a set of values that are specific to the function of that layer:

- **Project**: The project layer contains configuration that is common and applicable for all instances of an OSM project. Examples of content in this layer are JMS Queues and external authentication details.
- **Instance**: The instance layer contains configuration that is unique to each OSM instance, such as database identity and cluster size.
- **Shape**: The shape layer defines the hardware resource utilization and the resulting tuning. Java Heap Size is an example of a configuration value found in the shape specification.

The layers are implemented as specification files written in YAML:

- *project-instance*.yaml
- *project*.yaml
- *shape*.yaml

You can build a palette of re-usable, common portions of a configuration for a shape and project. When a new environment is needed, you can pick from this palette, adding an instance specification, which is unique to a single instance of OSM.

About Helm Overrides

The specification files are consumed in a hierarchical fashion. If a value is found in multiple specification files (layers), the one further up the hierarchy takes precedence. This allows the instance specification to have the final control over its configuration by being able to override a value that is prescribed in either the shape or project specifications. This also allows Oracle to define sealed, base configuration, while still providing you the control over the values used for any specific OSM instance.

Following are the specification files, listed in the order of the highest priority to the lowest:

- *project-instance*.yaml
- *project*.yaml
- *shape*.yaml
- **values.yaml**

While the specification for an instance points to the specification for the shape to be used (implying the order here may be out of sequence), the values found in the specification for the shape are actually loaded for processing before the values in the specification for the instance.

The instance specification remains the final authority on any values that are found in multiple specification files.

About the OSM Cloud Native Toolkit

The OSM cloud native toolkit is an archive file that includes the default configuration files, utility scripts, and samples to deploy OSM in a cloud native environment. With OSM cloud native, managing the domain configuration as code (CaC) is paramount. OSM cloud native provides guidance on effective management of this configuration to ensure that instances can be created in a standardized and repeatable fashion.

Contents of the OSM Cloud Native Toolkit

The OSM cloud native toolkit contains the following artifacts:

- Helm charts for OSM and OSM database installer:
  - The Helm chart for OSM is located in **$OSM_CNTK/charts/osm**.
  - The Helm chart for the OSM DB Installer is located in **$OSM_CNTK/charts/osm-dbinstaller**.
- WebLogic Server Deploy Tooling (WDT) metadata model for an OSM WebLogic domain
- Mechanism to extend the domain and WDT samples and scripts for some common use cases
- Utility scripts to help with the lifecycle of WebLogic Kubernetes Operator
- Sample scripts to manage pre-requisite secrets. These are not pipeline-friendly.
- Scripts to manage the lifecycle of an OSM instance. These are pipeline friendly.

2

Planning and Validating Your Cloud Environment

In preparation for Oracle Communications Order and Service Management (OSM) cloud native deployment, you must set up and validate pre-requisite software. This chapter provides information about planning, setting up, and validating the environment for OSM cloud native deployment.

See the following topics:

- Required Components for OSM Cloud Native
- Planning Your Cloud Native Environment
- Planning Your Container Engine for Kubernetes (OKE) Cloud Environment
- Validating Your Cloud Environment

If you are already familiar with traditional OSM, for important information on the differences introduced by OSM cloud native, see "Differences Between OSM Cloud Native and OSM Traditional Deployments".

Required Components for OSM Cloud Native

In order to run, manage, and monitor the OSM cloud native deployment, the following components and capabilities are required. These must be configured in the cloud environment:

- Kubernetes Cluster
- Oracle Multitenant Container Database (CDB)
- Container Image Management
- Helm
- Oracle WebLogic Server Kubernetes Operator
- Load Balancer
- Domain Name System (DNS)
- Persistent Volumes
- Authentication
- Secrets Management
- Kubernetes Monitoring Toolchain
- Application Logs and Metrics Toolchain

For details about the required versions of these components, see *OSM Compatibility Matrix*.

In order to utilize the full flexibility, reliability and value of the deployment, the following aspects must also be set up:

- Continuous Integration (CI) pipelines for custom images and cartridges
- Continuous Delivery (CD) pipelines for creating, scaling, updating, and deleting instances of the cloud native deployment

Planning Your Cloud Native Environment

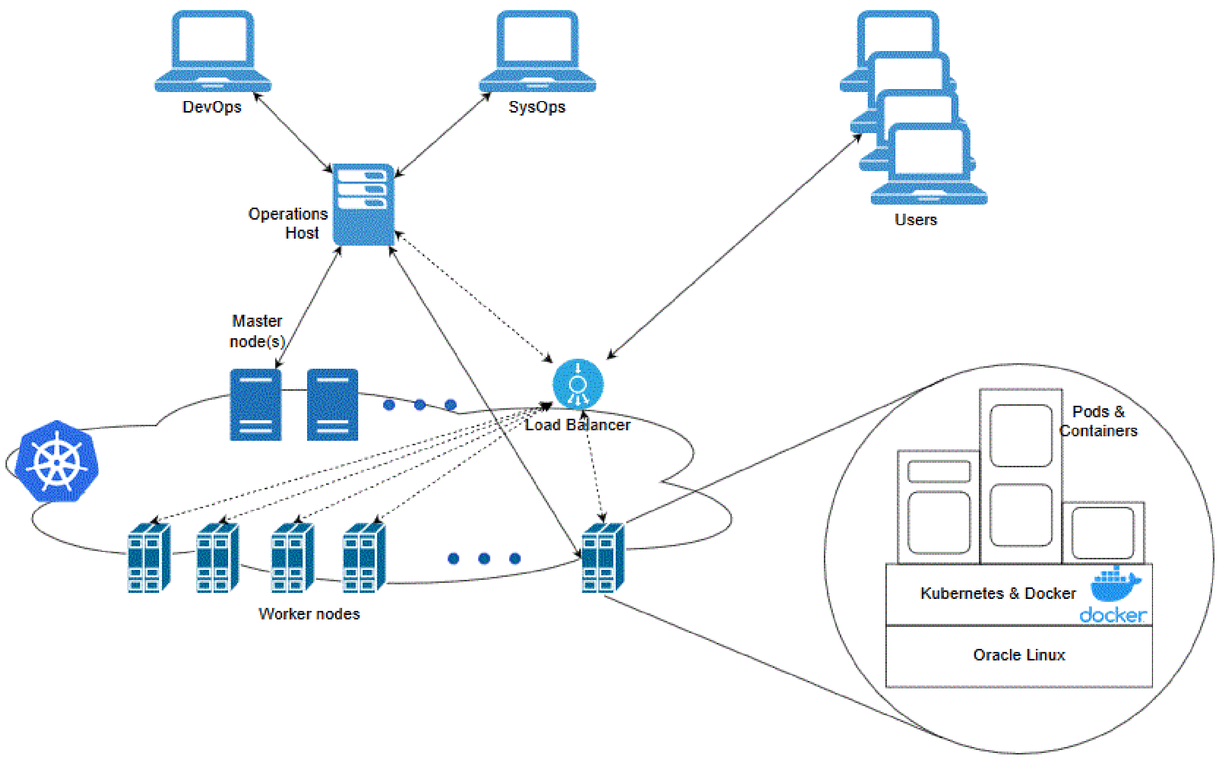
This section provides information about planning and setting up OSM cloud native environment. As part of preparing your environment for OSM cloud native, you choose, install, and set up various components and services in ways that are best suited for your cloud native environment. The following sections provide information about each of those required components and services, the available options that you can choose from, and the way you must set them up for your OSM cloud native environment.

Setting Up Your Kubernetes Cluster

For OSM cloud native, Kubernetes worker nodes must be capable of running Linux 7.x pods with software compiled for Intel 64-bit cores. A reliable cluster must have multiple worker nodes spread over separate physical infrastructure and a very reliable cluster must have multiple master nodes spread over separate physical infrastructure.

The following diagram illustrates Kubernetes cluster and the components that it interacts with.

Figure 2-1 Kubernetes Cluster



OSM cloud native requires:

- Kubernetes
  To check the version, run the following command:

  ```
  kubectl version
  ```

- Flannel
  To check the version, run the following command on the master node running the kube-flannel pod:

  ```
  docker images | grep flannel
  kubectl get pods --all-namespaces | grep flannel
  ```

- Docker
  To check the version, run the following command:

  ```
  docker version
  ```

Typically, Kubernetes nodes are not used directly to run or monitor Kubernetes workloads. You must reserve worker node resources for the execution of Kubernetes workload. However, multiple users (manual and automated) of the cluster require a point from which to access the cluster and operate on it. This can be achieved by using kubectl commands (either directly on command line and shell scripts or through Helm) or Kubernetes APIs. For this purpose, set aside a separate host or set of hosts. Operational and administrative access to the Kubernetes cluster can be restricted to these hosts and specific users can be given named accounts on these hosts to reduce cluster exposure and promote traceability of actions.

Typically, the Continuous Delivery pipeline automation deploys directly on a set of such operations hosts (as in the case of Jenkins) or leverage runners deployed on such operations hosts (as in the case of GitLab CI). These hosts must run Linux, with all interactive-use packages installed to support tools such as Bash, Wget, cURL, Hostname, Sed, AWK, cut, and grep. An example of this is the Oracle Linux 7.6 image (Oracle-Linux-7.6-2019.08.02-0) on Oracle Cloud Infrastructure.

In addition, you need the appropriate tools to connect to your overall environment, including the Kubernetes cluster. For instance, for a Container Engine for Kubernetes (OKE) cluster, you must install and configure the Oracle Cloud Infrastructure Command Line Interface.

Additional integrations may need to include LDAP for users to be able to login to this host, appropriate NFS mounts for home directories, security lists and firewall configuration for access to overall environment, and so on.

Kubernetes worker nodes should be configured with the recommended operating system kernel parameters listed in "Preparing the Operating System" in the *OSM Installation Guide*, or if they are engineered systems, "Installing OSM on Engineered Systems" of the *OSM Installation Guide*. Use the documented values as the minimum values to set for each parameter. Ensure that OS kernel parameter configuration is persistent, so as to survive a reboot.

The basic OSM cloud native instance, for which specification files are provided with the toolkit, requires up to 12 GB of RAM and 3 CPUs, in terms of Kubernetes worker node capacity. A small increment is needed for WebLogic Kubernetes Operator and Traefik. Refer to those projects for details. For detailed breakdown of CPU and memory capacity requirements, see "Working with Shapes."

Synchronizing Time Across Servers

It is important that you synchronize the date and time across all machines that are involved in testing, including client test drivers and Kubernetes worker nodes. Oracle recommends that you do this using Network Time Protocol (NTP), rather than manual synchronization, and strongly recommends it for Production environments. Synchronization is important in inter-component communications and in capturing accurate run-time statistics.

Provisioning Oracle Multitenant Container Database (CDB)

OSM cloud native architecture is best supported by the multitenant architecture that enables an Oracle database to function as a multitenant container database (CDB). A container database is either a Pluggable Database (PDB) or the root container. The root container is a collection of schemas, schema objects, and non-schema objects to which all PDBs belong. A PDB container for OSM cloud native contains the OSM schema and RCU schema. Each instance of OSM has its own PDB. OSM cloud native requires access to PDBs in an Oracle 19*c* Multitenant database. For more information about the benefits of Oracle Multitenant Architecture for database consolidation, see *Oracle Database Concepts*.

You can provision a CDB in an on-premise installation by following the instructions in *Oracle Database Installation Guide for Linux*. Alternatively, you can set it up as an Oracle Cloud Infrastructure DB system. For details on the supported versions, see *OSM Compatibility Matrix*. The provisioning process can vary based on the needs and the setup of your organization.

OSM cloud native requires certain settings to be configured at the CDB level. You can find those details in "Database Parameters" in *OSM Installation Guide*.

CDB hosts should be configured with OS kernel parameters as per Knowledge Article 1587357.1 on My Oracle Support. Use the recommended values specified in the KM article as the minimum values. Ensure that OS parameter configuration is persistent so as to survive a reboot.

Once the CDB is ready, you can follow one of the following strategies for the PDB:

Provisioning an Empty PDB

To create an empty PDB:

1. Run the following SQL commands using the sys dba account for the CDB:

```
CREATE PLUGGABLE DATABASE _replace_this_text_with_db_service_name_
ADMIN USER _replace_this_text_with_admin_name_ IDENTIFIED BY
"_replace_this_text_with_real_admin_password_" DEFAULT TABLESPACE
"USERS" DATAFILE '+DATA' SIZE 5M REUSE
AUTOEXTEND ON;
ALTER PLUGGABLE DATABASE _replace_this_text_with_db_service_name_
open instances = all;
ALTER PLUGGABLE DATABASE _replace_this_text_with_db_service_name_
save state instances = all;
alter session set
container=_replace_this_text_with_db_service_name_;
GRANT CREATE ANY CONTEXT TO SYS WITH ADMIN OPTION;
GRANT CREATE ANY CONTEXT TO _replace_this_text_with_admin_name_
WITH ADMIN OPTION;
GRANT CREATE ANY VIEW TO _replace_this_text_with_admin_name_ WITH
ADMIN OPTION;
GRANT CREATE SNAPSHOT TO _replace_this_text_with_admin_name_ WITH
ADMIN OPTION;
GRANT CREATE SYNONYM TO _replace_this_text_with_admin_name_ WITH
ADMIN OPTION;
GRANT CREATE TABLE TO _replace_this_text_with_admin_name_ WITH
ADMIN OPTION;
GRANT CREATE USER TO _replace_this_text_with_admin_name_ WITH ADMIN
OPTION;
GRANT CREATE VIEW TO _replace_this_text_with_admin_name_ WITH ADMIN
OPTION;
GRANT CREATE materialized view to
_replace_this_text_with_admin_name_;
GRANT GRANT ANY PRIVILEGE TO _replace_this_text_with_admin_name_
WITH ADMIN OPTION;
GRANT QUERY REWRITE TO _replace_this_text_with_admin_name_ WITH
ADMIN OPTION;
GRANT UNLIMITED TABLESPACE TO _replace_this_text_with_admin_name_
WITH ADMIN OPTION;
GRANT SELECT ON SYS.DBA_TABLESPACES TO
_replace_this_text_with_admin_name_ WITH GRANT OPTION;
GRANT SELECT ON SYS.V_$PARAMETER TO
_replace_this_text_with_admin_name_ WITH GRANT OPTION;
GRANT SELECT on SYS.dba_jobs to _replace_this_text_with_admin_name_
with grant option;
GRANT "CONNECT" TO _replace_this_text_with_admin_name_ WITH ADMIN
OPTION;
GRANT "DBA" TO _replace_this_text_with_admin_name_ WITH ADMIN
OPTION;
GRANT "EXP_FULL_DATABASE" TO _replace_this_text_with_admin_name_
WITH ADMIN OPTION;
GRANT "IMP_FULL_DATABASE" TO _replace_this_text_with_admin_name_
WITH ADMIN OPTION;
GRANT "RESOURCE" TO _replace_this_text_with_admin_name_ WITH ADMIN
OPTION;
GRANT EXECUTE ON SYS.DBMS_LOCK TO
_replace_this_text_with_admin_name_ WITH GRANT OPTION;
grant execute on utl_file to _replace_this_text_with_admin_name_
with grant option;
grant sysdba to _replace_this_text_with_admin_name_;
ADMINISTER KEY MANAGEMENT SET KEY USING TAG 'tag' FORCE
KEYSTORE IDENTIFIED BY "sys_password" WITH BACKUP USING
'db_service_name_backup';
```

2. Log into the PDB as the sys dba account for the PDB (defined by the "_replace_this_text_with_admin_name_" parameter in the above commands) and adjust the PDB tablespace by running the following command:

Note:

In the command, replace DATA with the proper name from v$asm_diskgroup.

```
create tablespace osm datafile '+DATA' size 1024m reuse autoextend on next
64m;
ALTER PLUGGABLE DATABASE DEFAULT TABLESPACE OSM;
```

Choosing Tablespaces

OSM cloud native supports the OSM best-practice of separate tablespaces for order data, order data indexes, OSM model data, and OSM model data indexes. Production and production-like instances must utilize this separation.

For a simple instance, such as a developer instance, separate tablespaces are not necessary. The default tablespace can be named as the tablespace for each of these categories in the OSM cloud native specification files.

To create PDBs for such instances, additional tablespaces can be added using the "sys dba" account for the PDB:

```
create tablespace osm_model datafile '+DATA' size 1024m reuse
autoextend on next 64m;
create tablespace osm_model_index datafile '+DATA' size 1024m reuse
autoextend on next 64m;
create tablespace osm_order datafile '+DATA' size 1024m reuse
autoextend on next 64m;
create tablespace osm_order_index datafile '+DATA' size 1024m reuse
autoextend on next 64m;
```

Choose tablespace names and datafiles as per your database management guidelines. Choose the initial tablespace size depending on the desired OSM partition size as per the following table:

Table 2-1 Partition Sizes and Tablespace Sizes

Partition Size	Tablespace Size
2000000 (2 million)	> or = 1024 MB
10000000 (10 million)	> or = 10240 MB
20000000 (20 million)	> or = 20480 MB

The tablespace names and the partition size chosen will be required to populate the OSM cloud native specification files for the instance that connects to this PDB.

Oracle recommends using the smaller partition size for developer instances and small test instances. Larger partition sizes are applicable for heavy-duty test instances (for example, for stress tests and performance tests) and production-grade instances.

If securing OSM data is a requirement, the recommended approach is to use transparent data encryption (TDE) to encrypt the tablespaces used to store OSM and WebLogic data. For more details, see *OSM - Encrypting Database Tablespaces and WebLogic Protocols* (Doc ID 2399723.1) knowledge article on My Oracle Support.

In that context, note that all OSM data is stored in tablespaces and, as a result, it is not necessary to supplement TDE encryption by setting the database parameter `db_securefile` to **PREFERRED**. While OSM supports **PREFERRED**, which has been the default since 12c, it is sufficient to set `db_securefile` to **PERMITTED**.

Provisioning a Seed OSM PDB

You can create a "master PDB" for OSM cloud native for a particular project or a subset of users by cloning a seed PDB and then running the OSM cloud native DB installer on it to deploy the OSM schema. At this point, you can deploy your cartridges to this PDB. The resulting PDB can serve as a master that you can clone for each instance that needs those set of cartridges.

You can also add the Fusion MiddleWare RCU DB schema to the master PDB. However, the master PDB must never be directly used in an OSM cloud native instance, as the RCU DB schema contents are inextricably linked to that instance. OSM cloud native instances must only use clones of the master PDB.

The advantage of a master PDB for OSM cloud native is that it standardizes a PDB for a significant number of users, and eliminates the need to perform some of the tasks related to creating instances in pipeline.

About Container Image Management

An OSM cloud native deployment generates container images for OSM and OSM database installer. Additionally, images are downloaded for WebLogic Kubernetes Operator and Traefik (depending on the choice of Ingress controllers).

Oracle highly recommends that you create a private container repository and ensure that all nodes have access to that repository. Images are saved in this repository and all nodes would then have access to the repository. This may require networking changes (such as routes and proxy) and include authentication for logging in to the repository. Oracle recommends that you choose a repository that provides centralized storage and management of not just container images, but also other artifacts such as OSM cartridge PAR files, Fusion MiddleWare patch ZIP files, and so on, as needed.

Failing to ensure that all nodes have access to a centralized repository will mean that images have to be synced to the hosts manually or through custom mechanisms (for example, using scripts), which are error-prone operations as worker nodes are commissioned, decommissioned or even rebooted. When an image on a particular worker node is not available, then the pods using that image are either not scheduled to that node, wasting resources, or fail on that node. If image names and tags are kept constant (such as myapp:latest), the pod may pick up a pre-existing image of the same name and tag, leading to unexpected and hard to debug behaviors.

Installing Helm

OSM cloud native requires Helm, which delivers reliability, productivity, consistency, and ease of use.

In an OSM cloud native environment, using Helm enables you to achieve the following:

- You can apply custom domain configuration by using a single and consistent mechanism, which leads to an increase in productivity. You no longer need to apply configuration changes through multiple interfaces such as WebLogic Console, WLST, and WebLogic Server MBeans.

- Changing the OSM domain configuration in the traditional installations is a manual and multi-step process which may lead to errors. This can be eliminated with Helm because of the following features:
  - Helm Lint allows pre-validation of syntax issues before changes are applied
  - Multiple changes can be pushed to the running instance with a single upgrade command
  - Configuration changes may map to updates across multiple Kubernetes resources (such as domain resources, config maps and so on). With Helm, you merely update the Helm release and its responsibility to determine which Kubernetes resources are affected.
- Including configuration in Helm charts allows the content to be managed as code, through source control, which is a fundamental principle of modern DevOps practices.

In order to co-exist with older Helm versions in production environments, OSM requires Helm 3.1.3 or later saved as **helm** in PATH.

The following text shows sample commands for installing and validating Helm:

```
$ cd some-tmp-dir
$ wget https://get.helm.sh/helm-v3.4.1-linux-amd64.tar.gz
$ tar -zxvf helm-v3.4.1-linux-amd64.tar.gz

# Find the helm binary in the unpacked directory and move it to its
desired destination. You need root user.
$ sudo mv linux-amd64/helm /usr/local/bin/helm

# Optional: If access to the deprecated Helm repository "stable" is
required, uncomment and run
# helm repo add stable https://charts.helm.sh/stable

# verify Helm version
$ helm version
version.BuildInfo{Version:"v3.4.1",
GitCommit:"c4e74854886b2efe3321e185578e6db9be0a6e29",
GitTreeState:"clean", GoVersion:"go1.14.11"}
```

Helm leverages **kubeconfig** for users running the `helm` command to access the Kubernetes cluster. By default, this is $HOME/.**kube/config**. Helm inherits the permissions set up for this access into the cluster. You must ensure that if RBAC is configured, then sufficient cluster permissions are granted to users running Helm.

Setting Up Oracle WebLogic Server Kubernetes Operator

Oracle WebLogic Server Kubernetes Operator provides WebLogic servers and clusters in a manner that is compatible with Kubernetes. The WebLogic Server Kubernetes Operator software is available as a container image. For OSM cloud native, you must download the WebLogic Server Kubernetes Operator container image and clone the WebLogic Server Kubernetes Operator GitHub repository.

To clone the repository, run the following commands:

```
$ cd path_to_wlsko_repository
$ git clone https://github.com/oracle/weblogic-kubernetes-operator.git
$ cd weblogic-kubernetes-operator
$ git checkout tags/v3.1.0
# This is the tag of v3.1.0 GA
```

For details about the required version of WKO, see *OSM Compatibility Matrix*.

After cloning the repository, set the `WLSKO_HOME` environment variable to the location of the WKO git repository, by running the following command:

```
$ export WLSKO_HOME=path-to-wlsko-repo/weblogic-kubernetes-operator
```

Note:

Developers can add the export command to **~/.bashrc** or **~/.profile** so that it is always set.

For more details on WKO, see Oracle WebLogic Kubernetes Operator User Guide.

For instructions on validating the operation of the WebLogic Server Kubernetes Operator on your Kubernetes cluster, see "Validating Your Cloud Environment".

About Load Balancing and Ingress Controller

Each OSM cloud native instance is a WebLogic cluster running in Kubernetes. To access application endpoints, you must enable HTTP/S connectivity to the cluster through an appropriate mechanism. This mechanism must be able to route traffic to the appropriate OSM cloud native instance in the Kubernetes cluster (as there can be many) and must be able to distribute traffic to the multiple Managed Server pods within a given instance. Each instance must be insulated from the traffic of the other instance. Distribution within an instance must allow for session stickiness so that OSM client UIs bind to a managed server wherever possible and therefore not require arbitrary re-authentication by the user. In the case of HTTPS, the load balance mechanism must enable TLS and handle it appropriately.

For OSM cloud native, an ingress controller is required to expose appropriate services from the OSM cluster and direct traffic appropriately to the cluster members. An external load balancer is an optional add-on.

The ingress controller monitors the ingress objects created by the OSM cloud native deployment, and acts on the configuration embedded in these objects to expose OSM HTTP and HTTPS services to the external network. This is achieved using NodePort services exposed by the ingress controller.

The ingress controller must support:

- Sticky routing (based on standard session cookie).
- Load balancing across the OSM managed servers (back-end servers).
- SSL termination and injecting headers into incoming traffic.

Examples of such ingress controllers include Traefik, Voyager, and Nginx. The OSM cloud native toolkit provides samples and documentation that use Traefik as the ingress controller.

An external load balancer serves to provide a highly reliable singe-point access into the services exposed by the Kubernetes cluster. In this case, this would be the NodePort services exposed by the ingress controller on behalf of the OSM cloud native instance. Using a load balancer removes the need to expose Kubernetes node IPs to the larger user base, and insulates the users from changes (in terms of nodes appearing or being decommissioned) to the Kubernetes cluster. It also serves to enforce access policies. The OSM cloud native toolkit includes samples and documentation that show integration with Oracle Cloud Infrastructure LBaaS when Oracle OKE is used as the Kubernetes environment.

Using Traefik as the Ingress Controller

If you choose to use Traefik as the ingress controller, the Kubernetes environment must have the Traefik ingress controller installed and configured.

For details about the required version of Traefik, see *OSM Compatibility Matrix*.

To install and configure Traefik, do the following:

> **Note:**
>
> Set **kubernetes.namespaces** and the chart version specifically using command-line.

1. Ensure that the following tasks are completed:
    - Docker daemons in your Kubernetes environment are configured for access to Docker Hub.
    - The Helm repository is updated successfully as per the Helm section in this chapter.
2. Run the following commands:

```
$ export TRAEFIK_NS=traefik
$ kubectl create namespace $TRAEFIK_NS
$ helm repo add traefik https://helm.traefik.io/traefik
$ helm install traefik-operator traefik/traefik \
 --namespace $TRAEFIK_NS \
 --version 9.11.0 \
 --values $OSM_CNTK/samples/charts/traefik/values.yaml \
  --set "kubernetes.namespaces={$TRAEFIK_NS}"
```

Once the installation of Helm succeeds, the Traefik operator monitors the namespaces listed in its `kubernetes.namespaces` field for Ingress objects.

Using Domain Name System (DNS)

A Kubernetes cluster can have many routable entrypoints. Common choices are:

- External load balancer (IP and port)

- Ingress controller service (master node IPs and ingress port)
- Ingress controller service (worker node IPs and ingress port)

You must identify the proper entrypoint for your Kubernetes cluster.

OSM cloud native requires hostnames to be mapped to routable entrypoints into the Kubernetes cluster. Regardless of the actual entrypoints (external load balancer, Kubernetes master node, or worker nodes), users who need to communicate with the OSM cloud native instances require name resolution.

The access hostnames take the *prefix*.*domain* form. *prefix* and *domain* are determined by the specifications of the OSM cloud native configuration for a given deployment. *prefix* is unique to the deployment, while *domain* is common for multiple deployments.

The default *domain* in OSM cloud native toolkit is `osm.org`.

For a particular deployment, as an example, this results in the following addresses:

- `dev1.wireless.osm.org` (for HTTP access)
- `admin.dev1.wireless.osm.org` (for WebLogic Console access)
- `t3.dev1.wireless.osm.org` (for T3 JMS/SAF access)

These "hostnames" must be routable to the entry point of your Ingress Controller or Load Balancer. For a basic validation, on the systems that access the deployment, edit the local hosts file to add the following entry:

Note:

The hosts file is located in **/etc/hosts** on Linux and MacOS machines and in **C:\Windows\System32\drivers\etc\hosts** on Windows machines.

```
ip_address  dev1.wireless.osm.org   admin.dev1.wireless.osm.org 
t3.dev1.wireless.osm.org
```

However, the solution of editing the hosts file is not easy to scale and co-ordinate across multiple users and multiple access environments. A better solution is to leverage DNS services at the enterprise level.

With DNS servers, a more efficient mechanism can be adopted. The mechanism is the creation of a domain level A-record:

```
A-Record: *.osm.org IP_address
```

If the target is not a load balancer, but the Kubernetes cluster nodes themselves, a DNS service can also insulate the user from relying on any single node IP. The DNS entry can be configured to map **.osm.org* to all the current Kubernetes cluster node IP addresses. You must update this mapping as the Kubernetes cluster changes with adding a new node, removing an old node, reassigning the IP address of a node, and so on.

With these two approaches, you can set up an enterprise DNS once and modify it only infrequently.

Configuring Kubernetes Persistent Volumes

Typically, runtime artifacts in OSM cloud native are created within the respective pod filesystems. As a result, they are lost when the pod is deleted. These artifacts include application logs, Fusion MiddleWare logs, and JVM Java Flight Recorder data.

While this impermanence may be acceptable for highly transient environments, it is typically desirable to have access to these artifacts outside of the lifecycle of the OSM could native instance. It is also highly recommended to deploy a toolchain for logs to provide a centralized view with a dashboard. To allow for artifacts to survive independent of the pod, OSM cloud native allows for them to be maintained on Kubernetes Persistent Volumes.

OSM cloud native does not dictate the technology that supports Persistent Volumes, but provides samples for NFS-based persistence. Additionally, for OSM cloud native on an Oracle OKE cloud, you can use persistence based on File Storage Service (FSS).

Regardless of the persistence provider chosen, persistent volumes for OSM cloud native use must be configured:

- With accessMode ReadWriteMany
- With capacity to support intended workload

Log size and retention policies can be configured as part of the shape specification.

About NFS-based Persistence

For use with OSM cloud native, one or more NFS servers must be designated.

It is highly recommended to split the servers as follows:

- At least one for the development instances and the non-sensitive test instances (for example, for Integration testing)
- At least one for the sensitive test instances (for example, for Performance testing, Stress testing, and production staging)
- One for the production instance

In general, ensure that the sensitive instances have dedicated NFS support, so that they do not compete for disk space or network IOPS with others.

The exported filesystems must have enough capacity to support the intended workload. Given the dynamic nature of the OSM cloud native instances, and the fact that the OSM logging volume is highly dependent on cartridges and on the order volume, it is prudent to put in place a set of operational mechanisms to:

- Monitor disk usage and warn when the usage crosses a threshold
- Clean out the artifacts that are no longer needed

If a toolchain such as ELK Stack picks up this data, then the cleanup task can be built into this process itself. As artifacts are successfully populated into the toolchain, they can be deleted from the filesystem. You must take care to only delete log files that have rolled over.

About Authentication

OSM cloud native requires the use of two-level LDAP with embedded first and then external next. All OSM system users are created in embedded LDAP during instance creation. It is highly recommended that all system users and all users configured for automation tasks and API servicing be created in embedded LDAP for performance and reliability reasons. Human users are recommended to be served via access to an external (corporate) LDAP system.

For complete details on the requirement of an external authenticator, see "Using WebLogic Server Authenticators with OSM" in *OSM System Administrator's Guide*. When OSM cloud instances use external authentication, ensure that you create separate users and groups for each environment (or class of environments) in the external LDAP service. The specifications of this depend on the LDAP service provider.

OSM cloud native toolkit provides a sample configuration that uses OpenLDAP to demonstrate how to integrate with external LDAP server for human users. For details on setting up the OpenLDAP server and the layout of the data within it, see "Setting Up Authentication."

Management of Secrets

OSM cloud native leverages Kubernetes Secrets to store sensitive information securely. This sensitive information is, at a minimum, the database credentials and the WebLogic administrator credentials. Additional credentials may be stored to authenticate with the external LDAP system. Your custom cartridges may need to communicate with other systems, such as Unified Inventory Management (UIM). The credentials for such systems too are managed as Kubernetes Secrets.

These secrets need to be secured over their lifecycle by the Kubernetes cluster administration. RBAC should be used to restrict the entities that can describe, view, or mount these credentials.

OSM cloud native scripts assume that a set of pre-requisite secrets exist when they are invoked. As such, creation of the secrets is a pre-requisite step in the pipeline. OSM cloud native toolkit provides a sample script to create some of the common secrets it needs, but this script is interactive and therefore not suitable for Continuous Delivery (CD) automation pipelines. The sample script serves to provide a basic mechanism to add secrets and illustrates the names and structure of the secrets that OSM cloud native requires.

You can create the secrets manually by using the sample script for each instance. The sample can be augmented to include additional custom secrets. This method requires exposing RBAC for creating secrets for a larger group of users, which might not be desirable. It can also result in human errors, such as mistyping a password, which will only be detected during the runtime of the OSM instance.

A more sustainable and scalable option is using a secrets management system. There are several secrets management systems available for use with Kubernetes. Choose a system that offers a secure API (to be called from the CD pipeline) and populates the sensitive information as secrets into Kubernetes, as opposed to populating into pods through environment variables. The installation, configuration, and validation of such a secrets management system is a pre-requisite to uptake OSM cloud native. For details on setting up the secrets management system, see the documentation of the system that you adopt.

Using Kubernetes Monitoring Toolchain

A multi-node Kubernetes cluster with multiple users and an ever-changing workload requires a capable set of tools to monitor and manage the cluster. There are tools that provide data, rich visualizations and other capabilities such as alerts. OSM cloud native does not require any particular system to be used, but recommends using such a monitoring, visualization and alerting capability.

For OSM cloud native, the key aspects of monitoring are:

- Worker capacity in CPU and memory. The pods take up non-trivial amount of worker resources. For example, pods configured for production performance use 32 GB of memory. Monitoring the free capacity leads to predictable OSM instance creation and scale-up.
- Worker node disk pressure
- Worker node network pressure
- Health of the core Kubernetes services
- Health of WebLogic Kubernetes Operator
- Health of Traefik (or other load balancer in the cluster)

The namespaces and pods that OSM cloud native uses provide a cross instance view of OSM cloud native.

About Application Logs and Metrics Toolchain

OSM cloud native generates all logs that traditional OSM and WebLogic Server typically generate. The logs can be sent to a shared filesystem for retention and for retrieval by a toolchain such as Elastic Stack.

In addition, OSM cloud native generates metrics and JVM Java Flight Recorder (JFR) data. OSM cloud native exposes metrics for scraping by Prometheus. These can then be processed by a metrics toolchain, with visualizations like Grafana dashboards. Dashboards and alerts can be configured to enable sustainable monitoring of multiple OSM cloud native instances throughout their lifecycles. The OSM JFR data can be retrieved by Java Mission Control or such similar tools to analyze the performance of OSM at the JVM level. Performance metrics include heap utilization, threads stuck, garbage collection, and so on.

Oracle highly recommends using a toolchain to effectively monitor OSM cloud native instances. The dynamic lifecycle in OSM cloud native, in terms of deploying, scaling and updating an instance, requires proper monitoring and management of the database resources as well. For non-sensitive environments such as development instances and some test instances, this largely implies monitoring the tablespace usage and the disk usage, and adding disk space as needed.

Another important facet is to track PDB usage to ensure PDBs that are no longer required are deleted so that the resources are freed up. Sensitive environments such as production and stress test instances require close monitoring of the database resources such as CPU, SGA/PGA, top-runner SQLs, and IOPS.

A key implication of the dynamic behavior of OSM cloud native on the database is when the instances are dehydrated. Very often, there is a requirement to have an OSM instance kept around even when it is not being actively used. This is because it captures a particular state (for example, cartridge lineup or order load) or is non-trivial to recreate. Such an environment lies idle until it is needed again. With OSM cloud native, there is no retained state within the run-time instance. The information on creating the instance is in the CD artifacts (the various specification files), and all the OSM application information is in the PDB. As a result, when the instance is not actively needed, all Kubernetes resources for it can be freed up by deleting the instance. This does not delete the PDB. The CD artifacts and the PDB can be used to rehydrate the instance when required. In the mean time, if the instance is not required for a while (or if there is database capacity pressure), the PDB can be unplugged to no longer consume any run-time resources. An unplugged PDB can even be transferred to another CDB and plugged in there.

Role of Continuous Integration (CI) Pipelines

The roles of CI pipelines in an OSM cloud native environment are as follows:

- To generate standard OSM cartridge PAR files and store them in a central location with appropriate path and naming convention for deployment. Developers run this automation as they modify cartridges for testing. Standalone mechanisms that generate "official" cartridge builds for testing and production use also run automation.
- To generate custom OSM cloud native images. The OSM cloud native images contain all the components needed to run OSM cloud native. However, you may require additional applications to be co-hosted by the OSM WebLogic cluster. Examples of such applications include MDBs to mediate communication with an external system and third-party Java EE monitoring tools. These applications must be layered on top of the OSM cloud native image to generate a custom image. Automation can accomplish this by using the file samples that are provided in the toolkit. The generated images must be uploaded to the internal container repository for use by deployment. The path and naming convention must be followed to designate images that are in development versus images that are ready for testing; and to version the images themselves.

OSM cloud native does not mandate the use of a specific set of tools for CI automation. Common choices are GitLab CI and Jenkins. As part of preparing for OSM cloud native, you must evaluate CI automation tools and choose one that fits your business needs and the desired source control mechanisms.

Role of Continuous Delivery (CD) Pipelines

The role of CD pipelines in an OSM cloud native environment is to perform operations on the target Kubernetes cluster to automate the full lifecycle of an OSM cloud native instance.

The following are the main operations you must implement:

- Create instance: This must drive off the source-controlled OSM cloud native specification files and run through the various stages (secrets creation, PDB creation, OSM database installation, OSM instance creation, load balancer creation, and cartridge deployment) to create a new OSM cloud native instance. Variability should be built in for some key phases as secrets may already exist and may need to be updated, or PDB may already exist with or without OSM schema, and so on. As a result, this automation is written to a "create-or-update" pattern.

- Update instance: This must be a variant of the instance creation automation, skipping the PDB creation and perhaps the load balancer (Ingress) creation. The automation takes the source-controlled OSM cloud native specification files, which have presumably been modified in some way since the instance was created, and runs through the steps to make those changes appear in the provisioned OSM instance. The specification changes could be as simple as a change in the number of desired Managed Servers, or could be as complex as introducing a new OSM container image. On the other hand, the only change might be a new version of the cartridge to be deployed.
- Delete instance: This must clean up the Kubernetes resources used by the instance. Typically, the PDB is left alone to be handled separately, but it is possible to chain its deletion to the clean up operation as well.

OSM cloud native does not mandate the use of a particular set of tools for CD automation. Common choices are GitLab CD and Jenkins. As part of preparing for OSM cloud native, you must evaluate CD automation tools and choose one that fits your business needs and the target Kubernetes environment.

Planning Your Container Engine for Kubernetes (OKE) Cloud Environment

This section provides information about planning your cloud environment if you want to use Oracle Cloud Infrastructure Container Engine for Kubernetes (OKE) for OSM cloud native. Some of the components, services, and capabilities that are required and recommended for a cloud native environment are applicable to the Oracle OKE cloud environment as well.

- **Kubernetes** and **Container Images**: You can choose from the version options available in OKE as long as the selected version conforms to the range described in the section about planning cloud native environment.
- **Container Image Management**: OSM cloud native recommends using Oracle Cloud Infrastructure Registry with OKE. Any other repository that you use must be able to serve images to the OKE environment in a quick and reliable manner. The OSM cloud native images are of the order of 3 GB each.
- **Oracle Multitenant Database**: It is strongly recommended to run Oracle DB outside of OKE, but within the same Oracle Cloud Infrastructure tenancy and the region as an Oracle DB service (BareMetal, VM, or ExaData). The database version should be 19c. You can choose between a standalone DB or a multi-node RAC.
- **Helm** and **Oracle WebLogic Kubernetes Operator**: Install Helm and Oracle WebLogic Kubernetes Operator as described for the cloud native environment into the OKE cluster.
- **Persistent Volumes**: Use NFS-based persistence. OSM cloud native recommends the use of Oracle Cloud Infrastructure File Storage service in the OKE context.
- **Authentication** and **Secrets Management**: These aspects are common with the cloud native environment. Choose your mechanisms to deliver these capabilities and implement them in your OKE instance.

- **Monitoring Toolchains**: While the Oracle Cloud Infrastructure Console provides a view of the resources in the OKE cluster, it also enables you to use the Kubernetes Dashboard. Any additional monitoring capability must be built up.
- **CI** and **CD Pipelines**: The considerations and actions described for CI and CD pipelines in the cloud native environment apply to the OKE environment as well.

Compute Disk Space Requirements

Given the size of the OSM cloud native container images (approximately 2 GB), the size of the OSM cloud native containers, and the volume of the OSM logs generated, it is recommended that the OKE worker nodes have at least 40 GB of free space that the **/var/lib** filesystem can use. Add disk space if the worker nodes do not have the recommended free space in the **/var/lib** filesystem.

Work with your Oracle Cloud Infrastructure OKE administrator to ensure worker nodes have enough disk space. Common options are to use Compute shapes with larger boot volumes or to mount an Oracle Cloud Infrastructure Block Volume to **/var/lib/docker**.

Note:

The reference to logs in this section applies to the container logs and other infrastructure logs. The space considerations still apply even if the OSM cloud native logs are being sent to an NFS Persistent Volume.

Connectivity Requirements

OSM cloud native assumes the connectivity between the OKE cluster and the Oracle CDBs is a LAN-equivalent in reliability, performance and throughput. This can be achieved by creating the Oracle CDBs within the same tenancy as the OKE cluster, and in the same Oracle Cloud Infrastructure region.

OSM cloud native allows for the full range of Oracle Cloud Infrastructure "cloud-to-ground" connectivity options for integrating the OKE cluster with on-premise applications and users. Selecting, provisioning, and testing such connectivity is a critical part of adopting Oracle Cloud Infrastructure OKE.

Using Load Balancer as a Service (LBaaS)

For load balancing, you have the option of using the services available in OKE. The infrastructure for OKE is provided by Oracle's IaaS offering, Oracle Cloud Infrastructure. In OKE, the master node IP address is not exposed to the tenants. The IP addresses of the worker nodes are also not guaranteed to be static. This makes DNS mapping difficult to achieve. Additionally, it is also required to balance the load between the worker nodes. In order to fulfill these requirements, you can use Load Balancer as a Service (LBaaS) of Oracle Cloud Infrastructure.

The load balancer can be created using the service descriptor in **$OSM_CNTK/samples/oci-lb-traefik.yaml**. The subnet ID referenced in this file must be filled in from your Oracle Cloud Infrastructure environment (using the subnet configured for your LBaaS). The port values assume you have installed Traefik using the unchanged sample values.

The configuration can be applied using the following command (or for traceability, by wrapping it into a Helm chart):

```
$ kubectl apply -f oci-lb-traefik.yaml
service/oci-lb-service-traefikconfigured
```

The Load Balancer service is created for Traefik pods in the Traefik namespace. Once the Load Balancer service is created successfully, an external IP address is allocated. This IP address must be used for DNS mapping.

```
$ kubectl get svc -n traefik oci-lb-service-traefik
NAME                      TYPE           CLUSTER-IP      EXTERNAL-
IP     PORT(S)
oci-lb-service-traefik    LoadBalancer   10.96.103.118
100.77.24.178   80:32006/TCP,443:32307/TCP
```

For additional details, see the following:

- "Creating Load Balancers to Distribute Traffic Between Cluster Nodes" in Oracle Cloud Infrastructure documentation.
- "Load Balancer Annotations" in Oracle GitHub documentation.

About Using Oracle Cloud Infrastructure Domain Name System (DNS) Zones

While a custom DNS service can provide the addressing needs of OSM cloud native even when OSM is running in OKE, you can evaluate the option of Oracle Cloud Infrastructure Domain Name System (DNS) zones capability. Configuration of DNS zones (and integration with on-premise DNS systems) is not within the scope of OSM cloud native.

Using Persistent Volumes and File Storage Service (FSS)

In the OKE cluster, OSM cloud native can leverage the high performance, high capacity, high reliability File Storage Service (FSS) as the backing for the persistent volumes of OSM cloud native. There are two flavors of FSS usage in this context:

- Allocating FSS by setting up NFS mount target
- Native FSS

To use FSS through an NFS mount target, see instructions for allocating FSS and setting up a Mount Target in "Creating File Systems" in the Oracle Cloud Infrastructure documentation. Note down the Mount Target IP address and the storage path and use these in the OSM cloud native instance specification as the NFS host and path. This approach is simple to set up and leverages the NFS storage provisioner that is typically available in all Kubernetes installations. However, the data flows through the mount target, which models an NFS server.

FSS can also be used natively, without requiring the NFS protocol. This can be achieved by leveraging the FSS storage provisioner supplied by OKE. The broad outline of how to do this is available in the blog post "Using File Storage Service with Container Engine for Kubernetes" on the Oracle Cloud Infrastructure blog.

Leveraging Oracle Cloud Infrastructure Services

For your OKE environment, you can leverage existing services and capabilities that are available with Oracle Cloud Infrastructure. The following table lists the Oracle Cloud Infrastructure services that you can leverage for your OKE cloud environment.

Table 2-2 Oracle Cloud Infrastructure Services for OKE Cloud Environment

Type of Service	Service	Indicates Mandatory / Recommended / Optional
Developer Service	Container Clusters	Mandatory
Developer Service	Registry	Recommended
Core Infrastructure	Compute Instances	Mandatory
Core Infrastructure	File Storage	Recommended
Core Infrastructure	Block Volumes	Optional
Core Infrastructure	Networking	Mandatory
Core Infrastructure	Load Balancers	Recommended
Core Infrastructure	DNS Zones	Optional
Database	BareMetal, VM, and ExaData	Recommended

Validating Your Cloud Environment

Before you start using your cloud environment for deploying OSM cloud native instances, you must validate the environment to ensure that it is set up properly and that any prevailing issues are identified and resolved. This section describes the tasks that you should perform to validate your cloud environment.

You can validate your cloud environment by:

- Performing a smoke test of the Kubernetes cluster
- Validating the common building blocks in the Kubernetes cluster
- Running tasks and procedures in Oracle WebLogic Kubernetes Operator Quickstart

Performing a Smoke Test

You can perform a smoke test of your Kubernetes cloud environment by running nginx. This procedure validates basic routing within the Kubernetes cluster and access from outside the environment. It also allows for initial RBAC examination as you need to have permissions to perform the smoke test. For the smoke test, you need nginx 1.14.2 container image.

Note:

The requirement of the nginx container image for the smoke test can change over time. See the content of the **deployment.yaml** file in step 3 of the following procedure to determine which image is required. Alternatively, ensure that you have logged in to Docker Hub so that the system can download the required image automatically.

To perform a smoke test:

1. Download the nginx container image from Docker Hub.

   For details on managing container images, see "Container Image Management."

2. After obtaining the image from Docker Hub, upload it into your private container repository and ensure that the Kubernetes worker nodes can access the image in the repository.

   Oracle recommends that you download and save the container image to the private Docker repository even if the worker nodes can access Docker Hub directly. The images in the OSM cloud native toolkit are available only through your private Docker repository.

3. Run the following commands:

   ```
   kubectl apply -f https://k8s.io/examples/application/
   deployment.yaml # the deployment specifies two replicas
   kubectl get pods     # Must return two pods in the Running state
   kubectl expose deployment nginx-deployment --type=NodePort --
   name=external-nginx
   kubectl get service external-nginx    # Make a note of the external
   port for nginx
   ```

   These commands must run successfully and return information about the pods and the port for nginx.

4. Open the following URL in a browser:

   ```
   http://master_IP:port/
   ```

   where:

   - *master_IP* is the IP address of the master node of the Kubernetes cluster or the external IP address for which routing has been set up
   - *port* is the external port for the **external-nginx** service

5. To track which pod is responding, on each pod, modify the text message in the web page served by nginx. In the following example, this is done for a deployment of two pods:

   ```
   $ kubectl get pods -o wide | grep nginx
   nginx-deployment-5c689d88bb-g7zvh   1/1     Running   0
   1d     10.244.0.149   worker1   <none>
   nginx-deployment-5c689d88bb-r68g4   1/1     Running   0
   1d     10.244.0.148   worker2   <none>
   ```

```
$ cd /tmp
$ echo "This is pod A - nginx-deployment-5c689d88bb-g7zvh -
worker1" > index.html
$ kubectl cp index.html nginx-deployment-5c689d88bb-g7zvh:/usr/
share/nginx/html/index.html
$ echo "This is pod B - nginx-deployment-5c689d88bb-r68g4 -
worker2" > index.html
$ kubectl cp index.html nginx-deployment-5c689d88bb-r68g4:/usr/
share/nginx/html/index.html
$ rm index.html
```

6. Check the index.html web page to identify which pod is serving the page.
7. Check if you can reach all the pods by running refresh (Ctrl+R) and hard refresh (Ctrl+Shift+R) on the index.html Web page.
8. If you see the default nginx page, instead of the page with your custom message, it indicates that the pod has restarted. If a pod restarts, the custom message in the page gets deleted.

   Identify the pod that restarted and apply the custom message for that pod.
9. Increase the pod count by patching the deployment.

   For instance, if you have three worker nodes, run the following command:

   Note:

   Adjust the number as per your cluster. You may find you have to increase the pod count to more than your worker node count until you see at least one pod on each worker node. If this is not observed in your environment even with higher pod counts, consult your Kubernetes administrator. Meanwhile, try to get as much worker node coverage as reasonably possible.

   ```
   kubectl patch deployment nginx-deployment -p '{"spec":
   {"replicas":3}}' --type merge
   ```
10. For each pod that you add, repeat step 5 to step 8.

Ensuring that all the worker nodes have at least one nginx pod in the Running state ensures that all worker nodes have access to Docker Hub or to your private Docker repository.

Validating Common Building Blocks in the Kubernetes Cluster

To approach OSM cloud native in a sustainable manner, you must validate the common building blocks that are on top of the basic Kubernetes infrastructure individually. The following sections describe how you can validate the building blocks.

Network File System (NFS)

OSM cloud native uses Kubernetes Persistent Volumes (PV) and Persistent Volume Claims (PVC) to use a pod-remote destination filesystem for OSM logs and performance data. By default, these artifacts are stored within a pod in Kubernetes and are not easily available for integration into a toolchain. For these to be available externally, the Kubernetes environment must implement a mechanism for fulfilling PV and PVC. The Network File System (NFS) is a common PV mechanism.

For the Kubernetes environment, identify an NFS server and create or export an NFS filesystem from it.

Ensure that this filesystem:

- Has enough space for the OSM logs and performance data.
- Is mountable on all the Kubernetes worker nodes

Create an nginx pod that mounts an NFS PV from the identified server. For details, see the documentation about "Kubernetes Persistent Volumes" on the Kubernetes website. This activity verifies the integration of NFS, PV/PVC and the Kubernetes cluster. To clean up the environment, delete the nginx pod, the PVC, and the PV.

Ideally, data such as logs and JFR data is stored in the PV only until it can be retrieved into a monitoring toolchain such as Elastic Stack. The toolchain must delete the rolled over log files after processing them. This helps you to predict the size of the filesystem. You must also consider the factors such as the number of OSM cloud native instances that will use this space, the size of those instances, the volume of orders they will process, and the volume of logs that your cartridges generate.

Validating the Load Balancer

For a development-grade environment, you can use an in-cluster software load balancer. OSM cloud native toolkit provides documentation and samples that show you how to use Traefik to perform load balancing activities for your Kubernetes cluster.

It is not necessary to run through "Traefik Quick Start" as part of validating the environment. However, if the OSM cloud native instances have connectivity issues with HTTP/HTTPS traffic, and the OSM logs do not show any failures, it might be worthwhile to take a step back and validate Traefik separately using Traefik Quick Start.

A more intensive environment, such as a test, a production, a pre-production, or performance environments can additionally require a more robust load balancing service to handle the HTTP/HTTPS traffic. For such environments, Oracle recommends using a load balancing hardware that is set up outside the Kubernetes cluster. A few examples of external load balancers are Oracle Cloud Infrastructure LBaaS for OKE, Google's Network LB Service in GKE, and F5's Big-IP for private cloud. The actual selection and configuration of an external load balancer is outside the scope of OSM cloud native itself, but is an important component to sort out in the implementation of OSM cloud native. For more details on the requirements and options, see "Integrating OSM."

To validate the ingress controller of your choice, you can use the same nginx deployment used in the smoke test described earlier. This is valid only when run in a Kubernetes cluster where multiple worker nodes are available to take the workload.

To perform a smoke test of your ingress setup:

1. Run the following commands:

```
kubectl apply -f https://k8s.io/examples/application/deployment.yaml
kubectl get pods -o wide    # two nginx pods in Running state;
ensure these are on different worker nodes
cat > smoke-internal-nginx-svc.yaml <<EOF
apiVersion: v1
```

```
kind: Service
metadata:
  name: smoke-internal-nginx
  namespace: default
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  sessionAffinity: None
  type: ClusterIP
EOF
kubectl apply -f ./smoke-internal-nginx-svc.yaml
kubectl get svc smoke-internal-nginx
```

2. Create your ingress targeting the **internal-nginx** service. The following text shows a sample ingress annotated to work with the Traefik ingress controller:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: traefik
  name: smoke-nginx-ingress
  namespace: default
spec:
  rules:
  - host: smoke.nginx.osmtest.org
    http:
      paths:
      - backend:
          serviceName: smoke-internal-nginx
          servicePort: 80
```

   If the Traefik ingress controller is configured to monitor the default namespace, then Traefik creates a reverse proxy and the load balancer for the nginx deployment. For more details, see Traefik documentation.

   If you plan to use other ingress controllers, refer to the documentation about the corresponding controllers for information on creating the appropriate ingress and make it known to the controller. The ingress definition should be largely reusable, with ingress controller vendors describing their own annotations that should be specified, instead of the Traefik annotation used in the example.

3. Create a local DNS/hosts entry in your client system mapping **smoke.nginx.osmtest.org** to the IP address of the cluster, which is typically the IP address of the Kubernetes master node, but could be configured differently.

4. Open the following URL in a browser:

```
http://smoke.nginx.osmtest.org:Traefik_Port/
```

   where *Traefik_Port* is the external port that Traefik has been configured to expose.

5. Verify that the web address opens and displays the nginx default page.

Your ingress controller must support session stickiness for OSM cloud native. To learn how stickiness should be configured, refer to the documentation about the ingress controller you choose. For Traefik, stickiness must be set up at the service level itself. For testing purposes, you can modify the **internal-nginx** service to enable stickiness by running the following commands:

```
kubectl delete ingress smoke-nginx-ingress
vi smoke-internal-nginx-svc.yaml
# Add an annotation section under the metadata section:
#   annotation:
#     traefik.ingress.kubernetes.io/affinity: "true"
kubectl apply -f ./smoke-internal-nginx-svc.yaml
# now apply back the ingress smoke-nginx-ingress using the above yaml
definition
```

Other ingress controllers may have different configuration requirements for session stickiness. Once you have configured your ingress controller, and the `smoke-nginx-ingress` and `smoke-internal-nginx` services as required, repeat the browser-based procedure to verify and confirm if nginx is still reachable. As you refresh (Ctrl+R) the browser, you should see the page getting served by one of the pods. Repeatedly refreshing the web page should show the same pod servicing the access request.

To further test session stickiness, you can either do a hard refresh (Ctrl+Shift+R) or restart your browser (you may have to use the browser in Incognito or Private mode), or clear your browser cache for the access hostname for your Kubernetes cluster. You may observe that the same nginx pod or a different pod is servicing the request. Refreshing the page repeatedly should stick with the same pod while hard refreshes should switch to the other pod occasionally. As the deployment has two pods, chances of a switch with a hard refresh are 50%. You can modify the deployment to increase the number of replica nginx pods (controlled by the `replicas` parameter under `spec`) to increase the odds of a switch. For example, with four nginx pods in the deployment, the odds of a switch with hard refresh rise to 75%. Before testing with the new pods, run the commands for identifying the pods to add unique identification to the new pods. See the procedure in "Performing a Smoke Test" for the commands.

To clean up the environment after the test, delete the following services and the deployment:

- `smoke-nginx-ingress`
- `smoke-internal-nginx`
- `nginx-deployment`

Running Oracle WebLogic Kubernetes Operator Quickstart

Oracle recommends that you validate your new Kubernetes environment for OSM cloud native by performing the procedures described in Oracle WebLogic Kubernetes Operator Quickstart available at: https://oracle.github.io/weblogic-kubernetes-operator/quickstart/

The quickstart guide provides instructions for creating a WebLogic deployment in a Kubernetes cluster with the Oracle WebLogic Kubernetes Operator. The guide also provides instructions for downloading and installing a load balancer, and a domain. Follow the instructions provided above for Helm 3.x.

When you run and complete the tasks in the quickstart successfully, the following aspects of the cloud environment are tested and verified:

- Private Docker repository (or procedures to sync per-node Docker cache on a multi-node Kubernetes cluster)
- Initial view of the chosen in-cluster load balancers
- RBAC for WebLogic Kubernetes Operator
- Procedure to introduce secrets into the cloud environment
- Basic compatibility of the cloud environment with WebLogic Kubernetes Operator

The quickstart also contains instructions for cleaning up the environment after you finish the validation and testing. Perform these clean-up procedures to return the environment to the original state for OSM cloud native.

After completing the clean-up procedures, ensure that the WebLogic Kubernetes Operator CustomResourceDefinition (CRD) is removed from your cluster by running the following commands:

```
$ kubectl get crd domains.weblogic.oracle
# if this returns an existing CRD even after WKO quickstart cleanup,
then run:
$ kubectl delete crd domains.weblogic.oracle
```

3

Creating OSM Cloud Native Images

OSM cloud native requires container images be made available to create and manage OSM cloud native instances. This chapter describes how to create those OSM cloud native images.

OSM cloud native requires two container images. The OSM DB Installer image is used to manage the OSM and Fusion MiddleWare schemas - create, delete, upgrade - as well as deploy and fast-undeploy OSM cartridges in the OSM schema. The other image is the OSM image itself. This image is the basis for all of the long running pods - the WebLogic admin server and all the Managed Servers that comprise an OSM cloud native instance. Each image is built on top of a Linux base image and adds Java, Fusion MiddleWare components and OSM product components on top.

OSM Cloud native images are created using the OSM cloud native builder toolkit and a dependency manifest file. The OSM cloud native Image Builder is intended to be run as part of a Continuous Integration process that generates images. It needs to run on Linux and have access to the local Docker daemon. The versions of these are as per the OSM statement of certification in the OSM documentation. The dependency manifest is a file that describes all the versions and patches required to build out the image.

See the following topics for further details:

- Downloading the OSM Cloud Native Image Builder
- Prerequisites for Creating OSM Images
- Specifying Configurations for the OSM Cloud Native Images
- Creating the OSM Cloud Native Images

Downloading the OSM Cloud Native Image Builder

You download the OSM cloud native image builder from My Oracle Support at: https://support.oracle.com

The OSM cloud native image builder is bundled with the following components:

- An unpatched dependency manifest file.

  This file does not include any artifacts that require contract-driven access to Oracle download sites (for example, for Fusion MiddleWare patches). Use this *unpatched* manifest file for evaluation purposes only.

  For production use (and throughout the adoption lifecycle leading up to production), obtain the latest dependency manifest file. See *OSM Compatibility Matrix* for details about the latest recommended manifest file for your OSM release.

- OSM cloud native builder kit. The kit contains:
  - The OSM Domain WDT Model.
  - The OSM DB Installer scripts and manifest files.

  - The WDT Deployment tool and the WebLogic Image tool.
- Staging directory structure.

Prerequisites for Creating OSM Images

The pre-requisites for building OSM cloud native images are:

- Docker client and daemon on the build machine.
- Installers for WebLogic Server and JDK. Download these from the Oracle Software Delivery Cloud:

  https://edelivery.oracle.com
- Oracle Instant Client. Download this from Oracle Software Downloads:

  https://www.oracle.com/downloads/
- Required patches. Download these from My Oracle Support:

  https://support.oracle.com/
- Java, installed with JAVA_HOME set in the environment.
- Bash, to enable the `<tab>` command complete feature.

See *OSM Compatibility Matrix* for details about the required and supported versions of these pre-requisite software.

Configuring the OSM Cloud Native Images

The dependency manifest file describes the input that goes into the OSM images. It is consumed by the image build process. The default configuration in the latest manifest file (from the Knowledge Article 2170105.1 on My Oracle Support) provides all the necessary components and required patches for creating the OSM cloud native images easily.

You can also modify the manifest file to extend it to meet your requirements. This enables you to:

- Specify any Linux image as the base, as long as its binary is compatible with Oracle Linux.
- Upgrade the Oracle Enterprise Linux version to a newer version to uptake a quarterly CPU.
- Upgrade the JDK version to a newer JDK version to uptake a quarterly CPU.
- Upgrade the Fusion Middleware version to a newer version. For example, you upgrade the Fusion Middleware version to a newer version when you initiate the upgrade to pick up new PSU or when Oracle recommends a new update.
- Change the set of patches applied on WebLogic Server, Coherence, Fusion Middleware, and OPatch to stay aligned with evolving OSM recommendations.
- Change the OSM artifacts to newer artifacts to uptake a new OSM patch.
- Choose a different **userid** and **groupid** for **oracle:oracle user:group** that the image specifies. The default is **1000:1000**.

The breakdown of each section in the dependency manifest file is as follows:

Note:

The `schemaVersion` and `date` parameters are maintained by Oracle. Do not modify these parameters.

Version numbers provided here are only examples. The manifest file used specifies the actual versions currently recommended.

- **OSM Cloud Native Infrastructure Image**

  While not required by OSM cloud native to create or manage OSM instances, this infrastructure image is a necessary building block of the final OSM container image.

  ```
  linux:
    vendor: Oracle
    version: 7.x
    image: oraclelinux:7-slim
  ```

  The `Linux` parameter specifies the base Linux image to be used as the base docker image. The version is the two-digit version from **/etc/redhat-release**.

  The vendor and version details are specified and used for:

  – Validation when an image is built.

  – Querying at run-time. To troubleshoot issues, Oracle support requires you to provide these details in the manifest file used to build the image.

  ```
  userGroup:
    username: oracle
    userid: 1000
    groupname: oracle
    groupid: 1000
  ```

  The `userGroup` parameter specifies the default userId and groupId for `oracle`

  ```
  jdk:
    vendor: Oracle
    version: 8u251
    path: $CN_BUILDER_STAGING/java/jdk-8u251-linux-x64.tar.gz
  ```

  The `jdk` parameter specifies the JDK vendor, version, and the staging path.

  ```
  fmw:
      version: 12.2.1.4.0
      path: $CN_BUILDER_STAGING/fmw/install/
  fmw_12.2.1.4.0_infrastructure_Disk1_1of1.zip
  ```

The `fmw` parameter specifies the Fusion Middleware version and staging path.

```
oPatch:
  description: Weblogic Opatch
  patchId: 28186730_13.9.4.2.2
  path: $CN_BUILDER_STAGING/fmw/patch/p28186730_139422_Generic.zip
```

The `oPatch` parameter specifies the Oracle Patch tool and staging path.

```
fmwPatch:
  - description: p28334768,p27184424(on top of 12.2.1.4.0)
    patchId: 28334768_12.2.1.4.0
    path: $CN_BUILDER_STAGING/fmw/patch/p28334768_122140_Generic.zip
  - description: p28334768,p27184424(on top of 12.2.1.4.0)
    patchId: 27184424_12.2.1.4.0
    path: $CN_BUILDER_STAGING/fmw/patch/p27184424_122140_Generic.zip
  - description: SAF to dynamic cluster
    patchId: 30656708_12.2.1.4.0
    path: $CN_BUILDER_STAGING/fmw/patch/p30656708_122140_Generic.zip
  - description: user-group association bug fix
    patchId: 30319071_12.2.1.4.0
    path: $CN_BUILDER_STAGING/fmw/patch/p30319071_122140_Generic.zip
  - description: PSU Coherence
    patchId: 31030896_12.2.1.4.0
    path: $CN_BUILDER_STAGING/fmw/patch/p31030896_122140_Generic.zip
  - description: PSU WLS
    patchId: 31101341_12.2.1.4.0
    path: $CN_BUILDER_STAGING/fmw/patch/p31101341_122140_Generic.zip
  - description: PSU WLS
    patchId: 30970477_12.2.1.4.0
    path: $CN_BUILDER_STAGING/fmw/patch/p30970477_122140_Generic.zip
  - description: PSU WLS
    patchId: 30761841_12.2.1.4.0
    path: $CN_BUILDER_STAGING/fmw/patch/p30761841_122140_Generic.zip
  - description: Opatch
    patchId: 31101362_12.2.1.4.0
    path: $CN_BUILDER_STAGING/fmw/patch/
p31101362_1394002_Generic.zip
```

The `fmwPatch` parameter specifies additional patches and their staging paths.

- **OSM Cloud Native Image**

> **Note:**
>
> Do not modify these parameters. These parameters are maintained by Oracle.

```
osmCnImage:
  name: osm-cn-base
  tag: 7.4.1
  wdt:
    version: 1.8.0
    path: $CN_BUILDER_STAGING/cnsdk/tools/weblogic-deploy.zip
  modelfiles: $CN_BUILDER_STAGING/cnsdk/osm_model/osm-domain-config/
osm-base-domain.yaml,$CN_BUILDER_STAGING/cnsdk/osm_model/osm-domain-
config/properties/docker-build/domain.properties
  application: $CN_BUILDER_STAGING/cnsdk/osm_model/osm-full-wdt-app-
archive.zip
  dockerExtension: $CN_BUILDER_STAGING/cnsdk/osm_model/
additionalBuildCommands.txt
```

The `osmCnImage` section specifies details about the OSM artifacts required to build the OSM base image. These include the oms.ear, cartridge management WS ear file, default cartridge par file, job control cartridge par file, WDT and base model files.

- **OSM Cloud Native DB Installer Image**

```
osmCnDbInstallerImage:
  name: osm-cn-db-installer
  tag: 7.4.1
```

The `osmCnDbInstallerImage` parameter specifies the DB Installer image name and version. This includes the transformed OSM DB model and Semele jar file.

```
oracleInstantClient:
   version: 19.8.0.0.0
   basic:
      path: $CN_BUILDER_STAGING/instant-client/oracle-instantclient-
basic-19.8.0.0.0-1.x86_64.rpm
    sqlplus:
      path: $CN_BUILDER_STAGING/instant-client/oracle-instantclient-
sqlplus-19.8.0.0.0-1.x86_64.rpm
    tools:
      path: $CN_BUILDER_STAGING/instant-client/oracle-instantclient-
tools-19.8.0.0.0-1.x86_64.rpm
```

The `oracleInstantClient` parameter specifies details about the Oracle Instant Client required by the DB installer.

Creating OSM Cloud Native Images

To create the OSM image, the image builder does the following:

- Starts with a base-level operating system image (for example, **oraclelinux:7-slim**).
- Creates user and group (for example, oracle:oracle).
- Updates the image with the necessary packages for installing Fusion Middleware.
- Installs Java, Fusion Middleware and applies patches.
- Installs the OSM application base on the WDT model.

To create the OSM DB Installer image, the image builder does the following:

- Starts with a base-level operating system image (for example, **oraclelinux:7-slim**).
- Creates a user and a group (for example, oracle:oracle)
- Updates the image with the necessary packages for installing Fusion Middleware.
- Installs Java, Fusion Middleware and applies the required patches.
- Installs SQL Plus and SQL Loader and the supporting libraries.
- Installs the OSM DB Installer.

You can specify any Linux image as the base, as long as its binary is compatible with Oracle Linux and conforms to the compatibility matrix. See *OSM Compatibility Matrix* for details about the supported software.

The following packages must be installed onto the given base image, or be already present:

- gzip
- tar
- unzip

Creating the OSM and OSM DB Installer Images

To create the OSM and OSM DB Installer images:

1. Create the workspace directory:

   ```
   mkdir workspace
   ```

2. Obtain and untar the OSM image builder file: **osm-image-builder.tar.gz** to the workspace directory:

   ```
   tar -xf ./osm-image-builder.tar.gz --directory workspace
   ```

3. (Optional) Download and copy a newer version of Oracle Instant Client to the **workspace/osm-image-builder/staging/instant-client** directory and update the version and the file names.

   > **Note:**
   >
   > Oracle Instant Client packages are included in the OSM Image Builder and can be used as-is without additional downloads.

   > **Note:**
   >
   > Follow your organization standard for $http_proxy.

   ```
   curl -x $http_proxy --output osm-image-builder/staging/
   instant-client/oracle-instantclient-basic-19.8.0.0.0-1.x86_64.rpm
   https://download.oracle.com/otn_software/linux/instantclient/19800/
   oracle-instantclient19.8-basic-19.8.0.0.0-1.x86_64.rpm
   ```

```
curl -x $http_proxy --output osm-image-builder/staging/
instant-client/oracle-instantclient-sqlplus-19.8.0.0.0-1.x86_64.rpm
https://download.oracle.com/otn_software/linux/instantclient/19800/
oracle-instantclient19.8-sqlplus-19.8.0.0.0-1.x86_64.rpm
curl -x $http_proxy --output osm-image-builder/staging/
instant-client/oracle-instantclient-tools-19.8.0.0.0-1.x86_64.rpm
https://download.oracle.com/otn_software/linux/instantclient/19800/
oracle-instantclient19.8-tools-19.8.0.0.0-1.x86_64.rpm
```

4. Download JDK to the **workspace/osm-image-builder/staging/java** directory. The JDK version to be downloaded is described in the dependency manifest file.

   ```
   cp jdk-8u251-linux-x64.tar.gz ./workspace/osm-image-builder/staging/
   java/jdk-8u251-linux-x64.tar.gz
   ```

5. From Oracle Software Delivery Cloud: https://edelivery.oracle.com, download Fusion Middleware Infrastructure installer and copy it to the **workspace/osm-image-builder/staging/fmw/install** directory. The Fusion Middleware Infrastructure installer version to be download is described in the dependency manifest file under the `fmw` section.

   ```
   cp fmw_12.2.1.4.0_infrastructure_Disk1_1of1.zip ./
   workspace/osm-image-builder/staging/fmw/install/
   fmw_12.2.1.4.0_infrastructure_Disk1_1of1.zip
   ```

6. Download all the listed patches to the **workspace/osm-image-builder/staging/fmw/patch** directory. The list of required patches is in the dependency manifest file in the `oPatch` and `fmwPatch` sections.

   Note:

   This step is not required if **osm_cn_ci_manifest_unpatched.yaml** is the manifest used.

   You can download the patches using any of the following options:

   - (Recommended) Manually search for and download each OPatch/FMW patches from Oracle Support to the current working directory and then copy to the staging directory.

     ```
     cp pxxxxxx_xxxxx_Generic.zip ./workspace/osm-image-builder/
     staging/fmw/patch
     ```

   - Provide your My Oracle Support account credentials when invoking the **build-osm-images.sh** script, and let the builder download the patches automatically:

> **Note:**
>
> Some patches may not be retrievable in this manner. If the image build process fails with errors about a missing patch, use the recommended option.

```
./workspace/osm-image-builder/bin/build-osm-images.sh -f
$DMANIFEST -s $STAGING -c osm -u MOS_username -p MOS_password
```

7. Run **build-osm-images.sh** and pass the dependency manifest file, staging path, and the type of image to be created.

   ```
   export DMANIFEST=./workspace/osm-image-builder/bin/
   osm_cn_ci_manifest_unpatched.yaml
   export STAGING=$(pwd)/workspace/osm-image-builder/staging
   ```

   - To create OSM image, use "-c osm" as shown:

     ```
     ./workspace/osm-image-builder/bin/build-osm-images.sh -f
     $DMANIFEST -s $STAGING -c osm
     ```

   - To create OSM DB installer image, use "-c dbinstaller" as shown:

     ```
     ./workspace/osm-image-builder/bin/build-osm-images.sh -f
     $DMANIFEST -s $STAGING -c dbinstaller
     ```

These steps can be included into your CI pipeline as long as the required components are already downloaded to the staging area.

Additional Considerations When Using the Unpatched Manifest File

When an OSM image is created by the image builder with the **osm_cn_ci_manifest_unpatched.yaml** file, the resulting image does not contain the Fusion Middleware patches that are required for proper OSM cloud native functioning. It is intended to be used only for evaluation purposes. One workaround is to manually establish the association between OSM users and groups.

OSM users and groups are not associated after the start up of the admin server, which results in OSM EJB failing to deploy to the managed server. You should manually associate users and the group before starting up the managed server.

To associate OSM users with a group when using the unpatched manifest file:

1. Create a new instance with only the admin server running. In the instance specification, change the value for `clusterSize` manually. This change would ultimately be performed by an automated CI/CD pipeline.

   ```
   vi $SPEC_PATH/project-instance.yaml

   # Change the cluster size to 0
   clusterSize: 0
   ```

   Create the OSM instance.

2. Run the **config-security.sh** script passing the domain namespace and domain UID.

   ```
   $OSM_CNTK/scripts/config-security.sh project project-instance
   ```

3. Start the managed servers.
   - In the instance specification, set `clusterSize` to the desired number of managed servers.

     ```
     vi $SPEC_PATH/project-instance.yaml
     # Change the cluster size to the desired number
     clusterSize: 8
     ```

   - Upgrade the OSM instance.

The associations are reset every time the Admin Server pod terminates or restarts. This can happen when the instance is deleted, or on an unexpected event (such as an hardware issue), or as a side-effect of an instance upgrade that involves a rolling restart. Regardless of the scenario that led to Admin Server pod being recreated, the associations must be set up afresh.

To recreate the user and group association:

1. Stop all the managed servers by setting the cluster size to 0 in the instance specification and upgrade the instance.
2. Run the **config-security.sh** script as described in step 2 in the above procedure.
3. Start the managed servers as described in step 3 in the above procedure.

Post-build Image Management

The OSM cloud native image builder creates images with names and tags based on the settings in the manifest file. By default, this results in the following images:

- osm-cn-base:7.4.1.0.0
- osm-cn-db-installer:7.4.1.0.0

Once images are built in a CI pipeline, the pipeline uniquely tags the images and pushes them to an internal Docker repository. An uptake process can then be triggered for the new images:

- Sanity Test
- Development Test (for explicit retesting of scenarios that triggered the rebuild, if any)
- System Test
- Integration Test
- Pre-Production Test
- Production

4

Creating a Basic OSM Cloud Native Instance

This chapter describes how to create a basic OSM cloud native instance in your cloud environment using the operational scripts and the base OSM configuration provided in the OSM cloud native toolkit. You can create an OSM instance quickly in order to become familiar with the process, explore the configuration, and structure your own project. This procedure is intended to validate that you are able to create a basic OSM instance in your environment. For information on creating your own project with custom configuration, see "Creating Your Own OSM Cloud Native Instance".

Before you can create an OSM instance, you must do the following:

- Download and extract the OSM cloud native toolkit archive file
- Clone the WebLogic Kubernetes Operator (WKO) GIT repository. This is required to be performed on each host that installs and runs the toolkit scripts when a Kubernetes cluster is shared by multiple hosts.
- Install the WKO and Traefik container images. These tasks are required to be performed for each cluster that has shared resources.

Installing the OSM Cloud Native Artifacts and the Toolkit

Build container images for the following using the OSM cloud native Image Builder:

- OSM core application
- OSM database installer

You must create a private Docker repository for these images, ensuring that all nodes in the cluster have access to the repository. See "About Container Image Management" for more details.

Download the OSM cloud native toolkit archive and do the following:

- **On Oracle Linux**: Where Kubernetes is hosted on Oracle Linux, download and extract the tar archive to each host that has connectivity to the Kubernetes cluster.
- **On OKE**: For an environment where Kubernetes is running in OKE, extract the contents of the tar archive on each OKE client host. The OKE client host is the bastion host/s that is set up to communicate with the OKE cluster.

Set the variable for the installation directory by running the following command, where *osm_cntk_path* is the installation directory of the OSM cloud native toolkit:

```
$ export OSM_CNTK=osm_cntk_path
```

Cloning the WebLogic Kubernetes Operator (WKO) GIT Repository

Some scripts in the OSM cloud native toolkit require access to the WebLogic Kubernetes Operator Helm chart. You must ensure that the repository is cloned on each host that will use the scripts in the toolkit. In environments where multiple hosts have access to the Kubernetes cluster, this would be done on each host.

To clone the repository, run the following commands:

```
$ cd path_to_wlsko_repository
$ git clone https://github.com/oracle/weblogic-kubernetes-operator.git
$ cd weblogic-kubernetes-operator
$ git checkout tags/v3.1.0
# This is the tag of v3.1.0 GA
```

For details on WKO 3.1.0, see https://github.com/oracle/weblogic-kubernetes-operator/releases/tag/v3.1.0.

After cloning the repository, set the `WLSKO_HOME` environment variable to the location of the WKO git repository, by running the following command:

```
$ export WLSKO_HOME=path-to-wlsko-repo/weblogic-kubernetes-operator
```

Note:

Developers can add the export command to **~/.bashrc** or **~/.profile** so that it is always set.

Installing WebLogic Kubernetes Operator (WKO) and Traefik Container Images

In a shared environment, multiple developers may create OSM instances in the same cluster, using a shared WebLogic Kubernetes Operator.

For each cluster in your environment, you download and install the following:

- WebLogic Kubernetes Operator (WKO) container image
- Traefik container image

Note:

These installations must be co-ordinated on large teams so that they occur in a controlled manner.

Before installing the WKO image and the Traefik image, do the following tasks:

- Remove the instances of the WKO and Traefik that you installed to validate your cloud environment.
- Ensure that you have cleaned up the environment. See "Validating Your Cloud Environment" for instructions on cleaning up.
- Ensure that there are no WebLogic Server Operator artifacts in the environment.

Installing the WebLogic Kubernetes Operator Container Image

To download and install the WKO container image:

1. Ensure that Docker in your Kubernetes cluster can pull images from Docker Hub. The WKO container image is at: **ghcr.io/oracle/weblogic-kubernetes-operator:3.1.0**.
2. Choose a namespace for the operator and set the `WLSKO_NS` environment variable to the Kubernetes namespace in which WKO will be deployed.
3. Run the following installation script to install the image:

   ```
   $OSM_CNTK/scripts/install-operator.sh -n $WLSKO_NS
   ```

4. Validate that the operator is installed by running the following command:

   ```
   kubectl get pods -n $WLSKO_NS
   ```

Installing the Traefik Container Image

To leverage the OSM cloud native samples that integrate with Traefik, the Kubernetes environment must have the Traefik ingress controller installed and configured.

If you are working in an environment where the Kubernetes cluster is shared, confirm whether Traefik has already been installed and configured for OSM cloud native. If Traefik is already installed and configured, set your `TRAEFIK_NS` environment variable to the appropriate namespace.

The instance of Traefik that you installed to validate your cloud environment must be removed as it does not leverage the OSM cloud native samples. Ensure that you have removed this installation in addition to purging the Helm release. Check that any roles and rolebindings created by Traefik are removed. There could be a **clusterrole** and **clusterrolebinding** called "traefik-operator". There could also be a **role** and **rolebinding** called "traefik-operator" in the $TRAEFIK_NS namespace. Delete all of these before you set up Traefik.

To download and install the Traefik container image:

1. Ensure that Docker in your Kubernetes cluster can pull images from Docker Hub. See *OSM Compatibility Matrix* for the required and supported versions of the Traefik image.
2. Run the following command to create a namespace ensuring that it does not already exist:

Note:

You might want to add the traefik namespace to the environment setup like .bashrc.

```
kubectl get namespaces
export TRAEFIK_NS=traefik
kubectl create namespace $TRAEFIK_NS
```

3. Run the following commands to install Traefik using the **$OSM_CNTK/samples/charts/traefik/values.yaml** file in the samples:

Note:

Set **kubernetes.namespaces** and the chart version specifically using command-line.

```
helm repo add traefik https://helm.traefik.io/traefik
helm install traefik-operator traefik/traefik \
 --namespace $TRAEFIK_NS \
 --version 9.11.0 \
 --values $OSM_CNTK/samples/charts/traefik/values.yaml \
  --set "kubernetes.namespaces={$TRAEFIK_NS}"
```

After the installation, Traefik monitors the namespaces listed in its `kubernetes.namespaces` field for Ingress objects. The scripts in the toolkit manage this namespace list as part of creating and tearing down OSM cloud native projects.

When the **values.yaml** Traefik sample in the OSM cloud native toolkit is used as is, Traefik is exposed to the network outside of the Kubernetes cluster through port 30305. To use a different port, edit the YAML file before installing Traefik. Traefik metrics are also available for Prometheus to scrape from the standard annotations.

Traefik function can be viewed using the Traefik dashboard. Create the Traefik dashboard by running the instructions provided in the **$OSM_CNTK/samples/charts/traefik/traefik-dashboard.yaml** file. To access this dashboard, the URL is: `http://traefik.osm.org`. This is if you use the **values.yaml** file provided with the OSM cloud native toolkit; it is possible to change the hostname as well as the port to your desired values.

Creating a Basic OSM Instance

This section describes how to create a basic OSM instance.

Setting Environment Variables

OSM cloud native relies on access to certain environment variables to run seamlessly. Ensure the following variables are set in your environment:

- Path to your private specification repository

- Location of the WebLogic Server Kubernetes Operator (WLSKO) GIT repository
- Traefik namespace

To set the environment variables:

1. Create a directory that serves as your specification repository, by running the following command, where *spec_repo_path* is the path to your private specification repository:

   Note:

   The scripts in the toolkit support multiple directories being supplied to the -s parameter in a colon separated list (path/one:path/two:path/three). For simplicity, the toolkit works with a single directory.

   ```
   $ export SPEC_PATH=spec_repo_path/quickstart
   ```

2. Set the `WLSKO_HOME` environment variable to the location of the WLSKO git repository that you cloned:

   ```
   $ export WLSKO_HOME=~/git/weblogic-kubernetes-operator
   ```

3. Set the `TRAEFIK_NS` variable for Traefik namespace.

Registering the Namespace

After you set the environment variables, register the namespace.

To register the namespace, run the following command:

```
$OSM_CNTK/scripts/register-namespace.sh -p sr -t targets
# For example, $OSM_CNTK/scripts/register-namespace.sh -p sr -t
wlsko,traefik
# Where the targets are separated by a comma without extra spaces
```

Note:

`wlsko` and `traefik` are the names of the targets for registration of the namespace. The script uses WLSKO_NS and TRAEFIK_NS to find these targets. Do not provide the "traefik" target if you are not using Traefik.

Creating Secrets

You must store sensitive data and credential information in the form of Kubernetes Secrets that the scripts and Helm charts in the toolkit consume. Managing secrets is out of the scope of the toolkit and must be implemented while adhering to your organization's corporate policies. Additionally, OSM cloud native does not establish password policies.

Note:

The passwords and other input data such as RCU schema prefix length that you provide must adhere to the policies specified by the appropriate component.

As a pre-requisite to using the toolkit for either installing the OSM database or creating an OSM instance, you must create secrets for access to the following:

- OSM database
- OSM system users
- RCU DB
- OPSS
- Operator artifacts for the instance
- WebLogic Server Admin credentials used when creating the domain

The toolkit provides sample scripts for this purpose. However, they are not pipeline-friendly. The scripts should be used for creating an instance manually and quickly, but not for any automated process for creating instances. The scripts also illustrate both the naming of the secret and the layout of the data within the secret that OSM cloud native requires. You must create secrets prior to running the **install-osmdb.sh** or **create-instance.sh** scripts.

Run the following script to create the required secrets:

```
$OSM_CNTK/scripts/manage-instance-credentials.sh -p sr -i quick \
 create \
 osmdb,rcudb,wlsadmin,osmldap,opssWP,wlsRTE
```

where:

- `osmdb` specifies the connectivity details and the credentials for connecting to the OSM PDB (OSM schema). This is consumed by the OSM DB installer and OSM runtime.

  Note:

  The osmdb secrets contain PDB sysdba user, osm main schema user, osm rule engine schema user, and osm report schema user. The names of these must be unique.

- `osmldap` is the credential for OSM system admin and internal users. The script prompts for passwords for the following users.
  - OSM admin user (username is **omsadmin**)
  - Design Studio admin user (username is **sceadmin**)
  - OSM internal user (username is **oms-internal**)
  - OSM automation user (username is **oms-automation**)

- `rcudb` specifies the connectivity details and the credentials for connecting to the OSM PDB (RCU schema). This is consumed by the OSM database installer and OSM and Fusion MiddleWare runtime.
- `wlsadmin` is the credential for the intended user that will be created with administrative access to the WebLogic domain.
- `opssWP` is the password for encrypting and decrypting the ewallet contents.
- `wlsRTE` is the password used to encrypt the operator artifacts for this instance. The merged domain model and the domain ZIP are available in the operator config map and are encoded using this password.

Verify that the following secrets are created:

```
sr-quick-database-credentials
sr-quick-embedded-ldap-credentials
sr-quick-weblogic-credentials
sr-quick-rcudb-credentials
sr-quick-opss-wallet-password-secret
sr-quick-runtime-encryption-secret
```

Additionally, the secret `opssWF` is created by the installation process and does not follow the same guidelines. It is therefore not a pre-requisite for creating a new instance. In scenarios where a database is being re-used for a different OSM instance, then this becomes a pre-requisite secret. For more details, see "Reusing the Database State".

Assembling the Specifications

To assemble the specifications:

1. Copy the instance specification to your $SPEC_PATH and rename:

   ```
   cp $OSM_CNTK/samples/instance.yaml $SPEC_PATH/sr-quick.yaml
   ```

2. Copy the project specification to your $SPEC_PATH and rename:

   ```
   cp $OSM_CNTK/samples/project.yaml $SPEC_PATH/sr.yaml
   ```

   You edit these files as per the instructions described in the sections that follow.

Installing the OSM and RCU Schemas

This procedure configures an empty PDB. Depending on the database strategy for your team, you may have already performed this procedure as described in "Planning Your Cloud Native Environment". Before continuing, confirm whether the PDB being used for creating the OSM instance has been cloned from a master PDB that includes the schema installation. If the PDB already has the schema installed, skip this procedure and proceed to the Creating OSM Users and Groups topic.

After the PDB is created, it is configured with the OSM schema, the RCU schema, and the cluster leasing table.

Note:

Before installing the OSM and RCU schemas, stop or interrupt the automatic optimizer statistics collection maintenance task. For more details, see the *New OSM Database Optimizer Statistics Management* knowledge article (Doc ID 1925539.1) on My Oracle Support.

To install the OSM and RCU schemas:

Note:

YAML formatting is case-sensitive. While the next step uses vi editor for editing, if you are not familiar with editing YAML files, use a YAML editor to ensure that the you do not make any syntax errors while editing. Follow the indentation guidelines for YAML, as incorrect spacing can lead to errors.

1. Edit the project specification file and update the DB installer image to point to the location of your image as shown below:

   Note:

   Before changing the default values provided in the specification file, confirm that they align with the values used during PDB creation. For example, the default tablespace name should match the value used when PDB is created.

   ```
   dbinstaller:
     image:  DB_installer_image_in_your_repo:<tag>
   ```

2. If your environment requires a password to download the container images from your repository, create a Kubernetes secret with the Docker pull credentials. See the Kubernetes documentation for details. Reference the secret name in the project specification.

   ```
   # The image pull access credentials for the "docker login" into
   Docker repository, as a Kubernetes secret.
   # Uncomment and set if required.
   # imagePullSecret: ""
   ```

3. Set the partition size to the actual tablespace size that was created. The default value for production sizing is 20000000 (20 million) and for development is 2000000 (2 million). These may need to be overridden for this instance. See the *OSM System Administrator's Guide* for guidelines on partition and tablespace sizing. If required, update `defaultPartitionSize` in the development shape in **$OSM_CNTK/charts/osm/shapes/dev.yaml**. The `defaultPartitionSize` parameter also impacts how `defaultSubPartitionCount` is calculated. Calculate `OSM_SUBPARTITION_COUNT` from `OSM_PARTITION_SIZE`.

Table 4-1 Calculating Sub-partitions

defaultPartitionSize	Calculated Sub-partitions
< = 2M	16
> 2M and < = 10M	32
> 10M	64

4. Run the following script to start the OSM DB installer, which instantiates a Kubernetes Pod resource. The pod resource lives until the DB installation operation completes.

   ```
   #(OSM Schema)
   $OSM_CNTK/scripts/install-osmdb.sh -p sr -i quick -s $SPEC_PATH -c
   1
    ## once finished
   # (RCU Schema)
   $OSM_CNTK/scripts/install-osmdb.sh -p sr -i quick -s $SPEC_PATH -c
   7
   ```

   You can invoke the script with `-h` to see the available options.
5. Check the console to see if the DB installer is installed successfully.
6. If the installation failed, run the following command to review the error message in the log:

   ```
   kubectl logs -n sr sr-quick-dbinstaller-osm-dbinstaller
   ```

7. Clean up the failed pod by running the following command:

   ```
   helm uninstall sr-quick-dbinstaller -n sr
   ```

8. Go back to step 4 and run the script again to install the OSM DB installer.

The following table lists the basic database parameters that are handled by the DB Installer:

Table 4-2 Database Parameters Handled by the DB Installer

Parameter	Value
cursor_sharing	FORCE
parallel_degree_policy	AUTO
deferred_segment_creation	By default, set to True. The DB Installer specification can override this to FALSE for production environments.
open_cursors	2000
optimizer_mode	ALL_ROWS
_optimizer_invalidation_period	600

OSM DB Installer Activities

The OSM DB Installer performs the following activities during OSM schema creation:

- **Automatic Optimizer Statistics Collection Maintenance Task**: The OSM DB Installer disables this task during the creation of OSM schema. This avoids race conditions when copying partition statistics as part of the OSM schema installation. This maintenance task is re-enabled after the partition statistics are copied. This is handled as part of the OSM schema installation.
- **Statistics gathering schedule**: The OSM DB Installer modifies the default statistics gathering schedule so that the weekend schedule is the same as the weekday schedule. By default, weekday maintenance windows start at 10 PM and are 4 hours long. The Saturday and Sunday maintenance windows are 20 hours long and start at 6 AM; this impacts order processing performance during peak weekend hours.

  See the following topics in *Oracle Database Administrator's Guide* for more details:
  - Predefined Maintenance Windows
  - Configuring Automated Maintenance Tasks

Configuring the Project Specification

This section provides instructions for creating a project that is configured to support the processing of the **SimpleRabbits** sample cartridge that is provided with the toolkit. This sample cartridge validates that OSM processes orders successfully. The project specification is a Helm override file that contains values that are scoped to a project. The values specified in the specification are shared by all the instances of a project, unless they are overridden in an instance specification. Review the content about Helm chart layering in "Overview of the OSM Cloud Native Deployment".

The toolkit provides a sample project specification by the name **sr** that you can use with minor adjustments.

To configure the project specification:

1. Edit the project specification to provide the image in your repository (name and tag) by running the following command:

   ```
   vi $SPEC_PATH/sr.yaml

   **  edit the image to reflect the OSM image name and location in
   your docker repository

   image: osm_image_in_your_repository
   ```

2. The test cartridge requires JMS Queue configuration, which is provided with the toolkit. Copy the JMS Queue configuration from the location shown below into the instance specification.

   ```
   vi $OSM_CNTK/samples/simpleRabbits/project_fragment.yaml

    ** Copy the queue content
     vi $SPEC_PATH/sr.yaml
     * find the existing placeholder for the queues and paste the
   content
   ```

The following text is an example of JMS Queue configuration:

```
# jms distributed queues
uniformDistributedQueues:
 - name: new_jms_queue_1
    jndiName: oracle.communication.ordermanagement.ppt.loopbackA
    jmsTemplate: defaultJmsTemplate

## first line is LEFT aligned with no leading spaces. each
subsequent indent is 2 spaces from the last
```

3. If your environment requires a password to download the container images from your repository, create a Kubernetes secret with the Docker pull credentials. See the Kubernetes documentation for details. Reference the secret name in the project specification.

```
# The image pull access credentials for the "docker login" into
Docker repository, as a Kubernetes secret.
# uncomment and set if required.
#imagePullSecret: ""
```

4. For your DNS resolution mechanism, change the default load balancer domain name as needed:

```
loadBalancerDomainName: "osm.org"
```

Tuning the Project Specification

This section provides instructions for tuning the project specification. The values specified in the specification are shared by all the instances of a project, unless they are overridden in an instance specification.

Do the following to tune the project specification:

- To configure the maximum number of bytes allowed in messages that are received over all WebLogic protocols, set the following parameter:

```
wlsMaxMsgSize: value_in_bytes
```

  For OSM cloud native, the default value is 300000000 bytes, which is much higher than the default value of 10000000 bytes in WebLogic. The low default value in WebLogic can cause errors when this limit is reached.

- To configure the tablespace name for OSM model and order tables and indexes, see the following parameters:

```
db:
  modelDataTablespace: string
  modelIndexTablespace: string
  orderDataTablespace: string
  orderIndexTablespace: string
```

  For each parameter, the default value is `OSM`.

- To configure the partition size for OSM order tables, see the following parameter:

  ```
  defaultPartitionSize: integer
  ```

  The default is 2,000,000 (2 million). Production shapes define a larger value of 20,000,000 (20 million), which is a better choice when combined with online purging.
- To configure the sub-partition count for partitioned OSM order tables, see the following parameter:

  ```
  defaultSubPartitionCount: integer
  ```

  The default value is undefined. Typical values are 16, 32 and 64. Leave this parameter undefined to allow the OSM cloud native database installer to choose a value appropriate for the partition size. For example, for a large 20 million partition, the installer will choose a value of 64 so as to minimize database contention.
- To configure whether database segment creation should be deferred, see the following parameter:

  ```
  deferredSegmentCreation: "TRUE" or "FALSE"
  ```

  The default value is `TRUE`. To minimize database contention, this should be set to `FALSE` for production systems.
- To configure OSM and infrastructure data source connection pool parameters, see the parameters under the jdbc element. For example, the maximum database connection pool capacity for the OSM application data sources and for the infrastructure data sources (which support JMS and tlog JDBC stores) can be set with:

  ```
  jdbc:
    app:
      maxCapacity: integer
    infra:
      maxCapacity: integer
  ```

  For more details on connection pool parameters, see *Oracle Fusion Middleware Administration Console Online Help for Oracle WebLogic Server 12.2.1.4.0*. Also refer to the production and development shapes for the full list of supported parameters and default values.
- To configure the message buffer cache size for individual JMS servers, see the following parameter:

  ```
  jmsMsgBufferSize: value_in_bytes
  ```

  The default value is approximately one-third of the maximum JVM heap size, or a maximum of 512 megabytes (536,870,912 bytes). For production environments, the recommended value is 1 giga byte (1,073,741,824 bytes) to reduce the possibility that WebLogic will start paging JMS message bodies to disk once the buffer is full.

- To configure whether database optimizer statistics should be loaded when creating OSM order table partitions, see the following parameter:

  ```
  loadPartitionStatistics: false
  ```

  The default value is `false`. This should be set to `true` for production systems.
- To configure logging options, see the following parameter:

  ```
  logging_options: string
  ```

  Refer to the production and development shapes for more details and the default values. The following is an example:

  ```
  logging_options: " -Dweblogic.log.FileMinSize=5000 -
  Dweblogic.log.FileCount=10 -Dweblogic.log.RotateLogOnStartup=false "
  ```

- To configure JVM parameters for the admin server or for managed servers, see the following parameter:

  ```
  user_mem_args: string
  ```

  Refer to the production and development shapes for sample values. The following is an example from the prodlarge shape:

  ```
  managedServers:
        shape:
          user_mem_args: "-XX:+UseG1GC -
  XX:G1HeapRegionSize=16m -XX:+ClassUnloadingWithConcurrentMark
  -XX:+UseStringDeduplication -XX:SurvivorRatio=3 -
  XX:CodeCacheMinimumFreeSpace=16m -XX:ReservedCodeCacheSize=512m
  -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps
  -XX:+PrintGCTimeStamps -XX:+PrintTenuringDistribution -
  XX:+PrintAdaptiveSizePolicy -Xloggc:/u01/oracle/user_projects/
  domains/domain/gc.log -XX:+DisableExplicitGC -
  XX:+ParallelRefProcEnabled -XX:+AlwaysPreTouch -Xms64g
  -Xmx64g -Xmn22g -XX:InitiatingHeapOccupancyPercent=50 -
  XX:ParallelGCThreads=13 "
  ```

  For more details, see the *OSM Memory Tuning Guidelines* (Doc ID: 2028249.1) knowledge article on My Oracle Support.

Configuring the Instance Specification

The instance specification is a Helm override file that contains values that are specific to a single instance. These values feed into the WDT model developed for the OSM WebLogic domain.

To configure the instance specification:

1. Edit the **sr-quick.yaml** file to specify the database details:

   ```
   db:
     datasourcesPrimary:
      port: 1521
       # If not using RAC, provide the DB server hostname/IP address
       # If using RAC, comment out "#host:"
       # host: dbserver-ip
       #
       # If using RAC, provide the list of SCAN hostname/IP addresses
       # If not using RAC, comment out "#scans:"
       #scans:
       # - scan1-ip
       # - scan2-ip
       #
       # If using RAC, provide either a list of VIP hostname/IP
   addresses
       # or a list of INSTANCE_NAMES
       # If not using RAC, comment out "#vips:" and "#instances:"
       #
       #vips:
       # - vip1-ip
       # - vip2-ip
       # --- OR ---
       #instances:
       # - instance-1
       # - instance-2
   ```

2. Assuming that `oci-lb-service-traefik` is the service created as part of the Oracle Cloud Infrastructure Load Balancer setup, run the following command to find the IP address of the Oracle Cloud Infrastructure LBaaS:

   ```
   kubectl get svc -n traefik oci-lb-service-traefik --
   output=jsonpath="{..status.loadBalancer.ingress[0].ip}"
   ```

3. Because an external load balancer is not required to be configured for the basic OSM instance, change the value of `loadBalancerPort` to the default Traefik NodePort of `30305` if you are not using Oracle Cloud Infrastructure LBaaS:

   ```
   loadBalancerPort: 30305
   ```

   If you use Oracle Cloud Infrastructure LBaaS, or any other external load balancer, set `loadBalancerPort` to 80, and uncomment and update the value for `externalLoadBalancerIP` appropriately:

   ```
   loadBalancerPort: load_balancer_port
   #externalLoadBalancerIP: IP_address_of_the_external_load_balancer
   ```

Creating an Ingress

An ingress establishes connectivity to the OSM instances.

To create an Ingress, run the following command:

```
$OSM_CNTK/scripts/create-ingress.sh -p sr -i quick -s $SPEC_PATH
Project Namespace : sr
Instance Fullname : sr-quick
LB_HOST           : quick.sr.osm.org
Ingress Controller: TRAEFIK
External LB IP    : 192.0.0.8

NAME: sr-quick-ingress
LAST DEPLOYED: Wed Jul  1 10:20:27 2020
NAMESPACE: sr
STATUS: deployed
REVISION: 1
TEST SUITE: None

Ingress created successfully...
```

Creating an OSM Instance

This procedure describes how to create an OSM instance in your environment using the scripts that are provided with the toolkit.

To create an OSM instance:

1. Run the following command:

   ```
   $OSM_CNTK/scripts/create-instance.sh -p sr -i quick -s $SPEC_PATH
   ```

   The **create-instance.sh** script uses the Helm chart located in the **charts/osm** directory to create and deploy the domain custom resource and the domain config map for your instance. If the scripts fails, see the **Troubleshooting Issues** section at the end of this topic, before you make additional attempts.

   The instance creation process creates the opssWF secret, which is required for access to the RCU DB. It is possible to handle the wallet manually if needed. To do so, pass `-w` to the **create-instance.sh** script, which creates the wallet file at a location you choose. You can then use this wallet file to create a secret by using the manage instance credentials script.

2. Validate the important input details such as Image name and tag, specification files used (Values Applied), hostname, and port for ingress routing:

   ```
   $OSM_CNTK/scripts/create-instance.sh -p sr -i quick -s $SPEC_PATH

   Calling helm lint
   ==> Linting ./scripts/../charts/osm
   [INFO] Chart.yaml: icon is recommended

   1 chart(s) linted, 0 chart(s) failed
   Project Namespace : sr
   Instance Fullname : sr-quick
   LB_HOST           : quick.sr.osm.org
   LB_PORT           : 30305
   Image             : osm:7.4.1.200504-0655-b1735.a0f9526f
   ```

```
Shape             : dev
Values Applied    : -f ./scripts/../charts/osm/values.yaml -f ./
scripts/../charts/osm/shapes/dev.yaml   -f /home/oracle/SmokeTest/
repo/sr.yaml -f /home/oracle/SmokeTest/repo/sr-quick.yaml
Output wallet     : n/a
```

After the script finishes executing, the log shows the following:

```
NAME              READY   STATUS              RESTARTS   AGE
sr-quick-admin    1/1     Running             0          2m12s
sr-quick-ms1      0/1     ContainerCreating   0          1s

Provide opss wallet File for 'sr-quick' ...
For example : '/path-to-osm-cntk/sr-quick.ewallet'
opss wallet File:
secret/sr-quick-opss-walletfile-secret created

Instance 'sr/sr-quick' admin server is now running.
Creation of instance 'sr/sr-quick' has completed successfully.
```

The **create-instance.sh** script also provides some useful commands and configuration to inspect the instance and access it for use.

3. If you query the status of the pods, the **READY** state of the managed servers may display **0/1** for several minutes while the OSM application is starting.
   When the **READY** state shows **1/1**, your OSM instance is up and running. You can then validate the instance by deploying a sample cartridge and submitting orders.

The base hostname required to access this instance using HTTP is `quick.sr.osm.org`. See "Planning and Validating Your Cloud Environment" for details about hostname resolution.

The **create-instance** script prints out the following valuable information that you can use while working with your OSM domain:

- The T3 URL: `http://t3.quick.sr.osm.org` This is required for external client applications such as JMS and WLST.
- The URL for access to the WebLogic UI, which is provided through the ingress controller at host:`http://admin.quick.sr.osm.org:30305/console`.
- The URL for access to the OSM UIs, which is provided through the ingress controller that requires the host to be specified as: `http://quick.sr.osm.org:30305/OrderManagement/Login.jsp`.

Validating the OSM Instance

After creating an instance, you can validate it by checking the domain configuration and the client UIs.

Run the following command to display the domain configuration details of the OSM instance that you have created:

```
kubectl describe domain sr-quick -n sr
```

The command displays the domain configuration information.

To verify the client UIs:

- Log into the WebLogic console using the URL specified in the output of the **create-instance** script: http://admin.quick.sr.osm.org:30305/console

  You can use the console to verify the configuration that has been applied and to see that the OSM application is in a good state.
- Log into the OSM Task Web client user interface with the OSM administrator login credentials created as part of "Creating Secrets" using the URL (http://quick.sr.osm.org:30305/OrderManagement/Login.jsp) specified in the output of the **create-instance** script.

> **Note:**
>
> After an OSM instance is created, it may take a few minutes for the OSM user interface to become active.

Scaling the OSM Application Cluster

Now that your OSM instance is up and running, you can explore the ability to dynamically scale the application cluster.

To scale the OSM application cluster, edit the configuration:

1. In the instance specification, change the value for `clusterSize` manually. This change would ultimately be performed by an automated CI/CD pipeline.

   ```
   vi $SPEC_PATH/sr-quick.yaml

   # Change the cluster size to a value not larger than 18

    #cluster size
   clusterSize: 2
   ```

   > **Note:**
   >
   > You can watch the Kubernetes pods in your namespace shrink or grow in real-time. To watch the pods shrink or grow, in a separate terminal window, run the following command:
   >
   > ```
   > kubectl get pods -n sr --watch
   > ```

2. Upgrade the deployed Helm release:

   ```
   $OSM_CNTK/scripts/upgrade-instance.sh -p sr -i quick -s $SPEC_PATH
   ```

   This pushes the new configuration to the deployed Helm release so the operator can take the necessary steps.

The WebLogic operator monitors changes to `clusterSize` and results in the operator spinning up or tearing down managed servers to align with the requested cluster size.

Deploying the Sample Cartridge

By deploying the sample cartridge that is provided with the toolkit, you can validate order processing in the OSM instance that you created.

Before deploying the cartridge, you must bring down the running domain. You can do this by scaling the cluster size down to **0**.

To deploy the sample cartridge:

1. Scale down the cluster:
   a. Reduce the cluster size in the configuration:

      ```
      vi $SPEC_PATH/sr-quick.yaml

      # Change the cluster size to 0

      #cluster size
      clusterSize: 0
      ```

   b. Push the configuration to the runtime environment:

      ```
      $OSM_CNTK/scripts/upgrade-instance.sh -p sr -i quick -s
      $SPEC_PATH
      ```

   The operator terminates the managed server.
2. Deploy the **SimpleRabbits** sample cartridge by running the following command:

   ```
   ./scripts/manage-cartridges.sh \
    -p sr -i quick -s $SPEC_PATH
    -f $OSM_CNTK/samples/simpleRabbits/SimpleRabbits.par -c parDeploy
   ```

3. Scale up the cluster:
   a. Increase the cluster size in the configuration:

      ```
      vi $SPEC_PATH/sr-quick.yaml

      # Change the cluster size to 1

      #cluster size
      clusterSize: 1
      ```

   b. Push the configuration to the runtime environment:

      ```
      $OSM_CNTK/scripts/upgrade-instance.sh -p sr -i quick -s
      $SPEC_PATH
      ```

   The operator terminates the managed server.

Submitting Orders

The OSM cloud native toolkit provides a sample order that you can submit to validate order processing in OSM. The sample order is available at: $OSM_CNTK/**samples/simpleRabbits/sample.xml**.

To submit OSM orders over HTTP, use an external client such as SoapUI. The endpoint is the same as the URL used to verify the OSM Task Web client.

When using SoapUI, a Soap Envelope element needs to wrap **CreateOrderBySpecification** that is provided in $OSM_CNTK/**samples/simpleRabbits/sample.xml**

To submit OSM orders over JMS, use an external client such as Hermes JMS. The endpoint must be as follows:

```
jms://OSM_1::queue_oracle/communications/ordermanagement/
WebServiceQueue::queue_oracle/communications/ordermangement/
SoapUIResponseQueue
```

The connection factory's providerURL must be as follows:

```
http://t3.quick.sr.osm.org:30305
```

Deleting and Recreating Your OSM Instance

Deleting Your OSM Instance

To delete your OSM instance, run the following command:

```
$OSM_CNTK/scripts/delete-instance.sh -p sr -i quick
```

Re-creating Your OSM Instance

When you delete an OSM instance, the database state for that instance still remains unaffected. You can re-create an OSM instance with the same project and the instance names, pointing to the same database.

Note:

Ensure that you use the same specifications that you used for creating the instance and that the following secrets have not been deleted:

- osmdb
- osmldap
- rcudb
- opssWF
- opssWP
- wlsRTE

To re-create an OSM instance, run the following command:

```
$OSM_CNTK/scripts/create-instance.sh -p sr -i quick -s $SPEC_PATH
```

Note:

After re-creating an instance, client applications such as SoapUI and HermesJMS may need to be restarted to avoid using expired cache information.

Cleaning Up the Environment

To clean up the environment:

1. Delete the instance:

   ```
   $OSM_CNTK/scripts/delete-instance.sh -p sr -i quick
   ```

2. Delete the ingress:

   ```
   $OSM_CNTK/scripts/delete-ingress.sh -p sr -i quick
   ```

3. Delete the namespace, which in turn deletes the Kubernetes namespace and the secrets:

   ```
   $OSM_CNTK/scripts/unregister-namespace.sh -p sr -d -t targets
   ```

   Note:

   `wlsko` and `traefik` are the names of the targets for registration of the namespace. The script uses WLSKO_NS and TRAEFIK_NS to find these targets. Do not provide the "traefik" target if you are not using Traefik.

4. Drop the PDB.

Troubleshooting Issues with the Scripts

This section provides information about troubleshooting some issues that you may come across when running the scripts.

If you experience issues when running the scripts, do the following:

- Check the operator logs to find out the details about the issue:

  ```
  kubectl get pods -n $WLSKO_NS
  # get the operator pod name to be used in the next command
  kubectl logs -n $WLSKO_NS operator_pod
  ```

- Check the "Status" section of the domain to see if there is useful information:

  ```
  kubectl describe domain -n sr sr-quick
  ```

"Timeout" Issue

In the logs, you may sometimes see the word "timeout" when the **create-instance** script fails. When you run the **create-instance** script, it may take a long time to pull the image, if you are doing it for the first time. In such a scenario, the script may fail and display the text "timeout" in the log.

To resolve this issue, try increasing the `podStartupDeadlineSeconds` parameter. The `podStartupDeadlineSeconds` parameter is a configurable parameter exposed in the instance specification that can be increased if required. Start with a very high timeout value and then monitor the average time it takes, because it depends on the speed with which the images are downloaded and how busy your cluster is. Once you have a good idea of the average time, you can reduce the timeout value accordingly to something that considers both the average time and some buffer.

```
# Modify the timeout value to start introspector pod. Mainly
# when using against slow DB or pulling image first time.
podStartupDeadlineSeconds: 800
```

After adjusting the parameter, clean up the failed instance and re-create the instance.

Cleanup Failed Instance

When a create-instance script fails, you must clean up the instance before making another attempt at instance creation.

> **Note:**
>
> Do not retry running the **create-instance** script or the **upgrade-instance** script immediately to fix any errors, as they would return errors. The **upgrade-instance** script may work but re-running it does not complete the operation.

To clean up the failed instance:

1. Delete the instance:

   ```
   $OSM_CNTK/scripts/delete-instance.sh -p sr -i quick
   ```

2. Delete and recreate the RCU schema:

   ```
   $OSM_CNTK/scripts/install-osmdb.sh -p sr -i quick -s $SPEC_PATH -c 5
   ```

Recreating an Instance

If you face issues when creating an instance, do not try to re-run the **create-instance.sh** script as this will fail. Instead, perform the cleanup activities and then run the following command:

```
$OSM_CNTK/scripts/create-instance.sh -p sr -i quick -s $SPEC_PATH
```

Next Steps

A basic OSM cloud native instance should now be running in your environment. This process exposed you to some of the base functionality and concepts that are new to OSM cloud native. You can continue in your sandbox environment learning about more OSM cloud native capabilities by following the learning path.

If, however, your first priority is to understand details on infrastructure setup and structuring of OSM instances for your organization, then you should follow the infrastructure path.

To follow the infrastructure path, proceed to "Planning Infrastructure".

To follow the learning path, proceed to "Creating Your Own OSM Cloud Native Instance".

5

Planning Infrastructure

In "Creating a Basic OSM Cloud Native Instance", you learned how to create a basic OSM instance in your cloud native environment. This chapter provides details about setting up infrastructure and structuring OSM instances for your organization. However, if you want to continue in your sandbox environment learning about more OSM cloud native capabilities, then proceed to "Creating Your Own OSM Cloud Native Instance".

See the following topics:

- Sizing Considerations
- Managing Configuration as Code
- Setting Up Automation
- Securing Operations in Kubernetes

Sizing Considerations

The hardware utilization for an OSM cloud native deployment is approximately the same as that of an OSM traditional deployment.

Consider the following when sizing for your cloud native deployment:

- For OSM cloud native, ensure that the database is sized to account for the WLS Persistent Store workload residing in the database. For details, see the "Persistent Store Configuration & Operational Considerations for JMS, SAF & WebLogic tlogs in OSM (Doc ID 2469767.1)" knowledge article on My Oracle Support.
- Oracle recommends sizing using a given production shape as a building block, adjusting the OSM cluster size to meet target order volumes.
- In addition to planning hardware for a production instance, Oracle recommends planning for a Disaster Recovery size and key non-production instances to support functional, integration and performance tests The Disaster Recovery instance can be created against an Active Data Guard Standby database when needed and terminated when no longer needed to improve hardware utilization.
- Non-production instances can likewise be created when needed, either against new or existing database instances.

Contact Oracle Support for further assistance with sizing.

Managing Configuration as Code

Managing Configuration as Code involves the following tasks:

- Creating Source Control Repository
- Managing OSM instances
- Deciding on the Scope

- Deployment Considerations
- Creating an Instance Using the Repository

Creating Source Control Repository

Managing Configuration as Code (CAC) is a central tenet of using OSM cloud native. You must create a source control repository to store all configuration that is necessary to re-create an OSM instance (or PDB) if it is lost. This does not include the toolkit scripts.

You must also set up a Docker repository for the OSM and OSM DB Installer images, as well as any custom versions of the OSM image for your use cases. For example, custom images are required to deploy a custom application **.ear** file. For more details on custom images, see "Extending the WebLogic Server Deploy Tooling (WDT) Model".

Managing OSM Instances

To extract the full benefits of OSM cloud native, it is imperative that you consider the management of the OSM instances before making potential configuration changes. The sections that follow describe how to structure your repositories to group project level artifacts, while allowing for other artifacts to be re-used (if needed) by the multiple OSM instances within a project.

Example Scenario

This section describes a scenario to help illustrate the concepts.

Let us assume that in an organization, OSM is used for two business purposes each of which is handled by two separate teams. The first team uses OSM to orchestrate wire line (triple play) orders for residential customers, and a second team uses OSM to process mobile orders for business customers.

Deciding on the Scope

You must first decide on the scope of the project including how many instances are required. Choose meaningful names for your project and instance.

The organization in our example will have two projects named **resiwireline** and **bizwireless**. We can assume that each project team has a predefined "pre-production" instance for final validation or production changes, a geo-redundant production instance for disaster recovery, a final User Acceptance Testing (UAT) instance for business testing, a few small Quality Assurance (QA) systems and many small development instances.

The directory structure for your configuration repository should reflect the hierarchical relationship of the project/instance relationship as well as isolating different projects from each other.

About the Repository Directory Structure

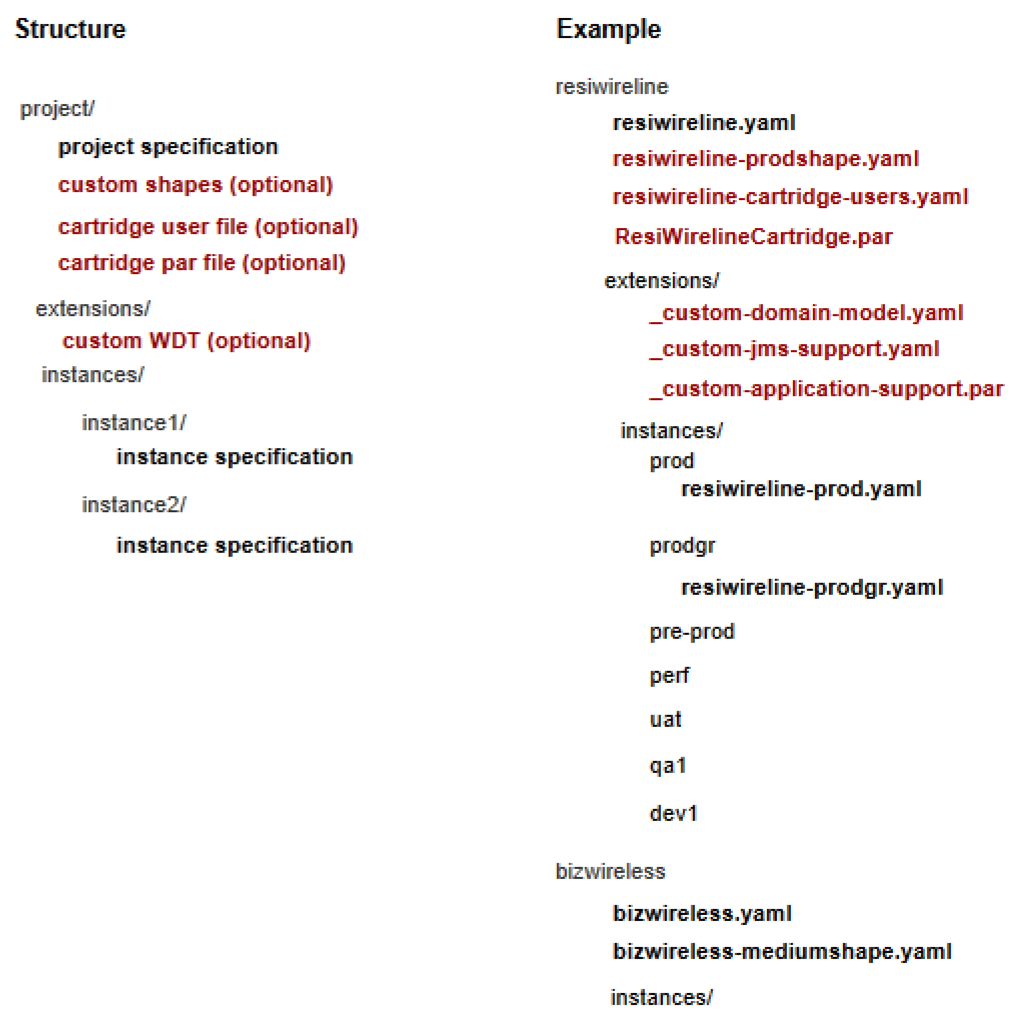
The project directory includes the project specification as well as configuration that is common to all instances, whereas instance specifications reside in a specific instance directory.

- Each project requires its own project specifications (YAML files).
- Optional artifacts such as the list of users and credentials used by the cartridges are also located under the top level project directory.
- All artifacts under the project are shared across the instances. Instance directories contain the instance specification.

> **Note:**
>
> While cartridge par files are shown to reside in this repository, you may consider using a separate repository for cartridges as described in "Working with Cartridges".

The following illustration shows the structure and hierarchy of the project directory with an example.

Figure 5-1 Project Directory Structure



```
Structure

project/
    project specification
    custom shapes (optional)
    cartridge user file (optional)
    cartridge par file (optional)
  extensions/
      custom WDT (optional)
  instances/
      instance1/
          instance specification
      instance2/
          instance specification
```

```
Example

resiwireline
    resiwireline.yaml
    resiwireline-prodshape.yaml
    resiwireline-cartridge-users.yaml
    ResiWirelineCartridge.par
  extensions/
      _custom-domain-model.yaml
      _custom-jms-support.yaml
      _custom-application-support.par
   instances/
      prod
          resiwireline-prod.yaml
      prodgr
          resiwireline-prodgr.yaml
      pre-prod
      perf
      uat
      qa1
      dev1

bizwireless
    bizwireless.yaml
    bizwireless-mediumshape.yaml
    instances/
```

Deployment Considerations

As the scenario shows, there will be many bits of configuration that may mix and match in different ways to produce a specific OSM instance. While all of these instances are pre-defined in the source control repository, they need not be deployed all the time.

Consider the following:

- For each project, one or more production instances may be deployed.
- It would be reasonable for pre-production to be deployed only when needed while first cloning the production DB.
- Likewise, the performance instance could also be deployed only when needed. Its PDB could be cloned from a specially generated PDB with synthetic test data, providing a consistent starting point.
- Likewise, the UAT instance could be deployed when needed, starting from similarly saved UAT PDB.
- The GR instance application would not be pre-deployed, but its database would be created in a DR site and synchronized from production via Active Data Guard.

Setting the Repository Path During Instance Creation

To offer flexibility in how the repository directory structure develops, the **create-instance** script takes as input, the path to the specification files.

The **-s specPath** parameter is mandatory in **create-instance.sh** and can point to several directories at once (directories are separated by a colon).

specPath would contain all the directories that contain specification files used for creating an instance:

- *repo*/**resiwireline**
- *repo*/**resiwireline**:*repo***/resiwireline**/**instances**/**qa**. (This will include all specification files at the resiwireline project level, as well as the specification files in the **qa** instance directory.)

Additionally, a separate parameter is used to point to the directory where custom extensions are found.

The **-m customExtPath** parameter is an optional parameter that can be passed into the **create-instance.sh** script.

customExtPath would point to all the directories where custom template files reside for the instance being created: *fileRepo*/**resiwireline**/**extensions**

Setting Up Automation

This section describes the complete sequence of activities for setting up an OSM environment with the aim of grouping repeatable steps into high-level categories. You should start to plan the steps that you can automate to some degree. This section does not include details on the changes that must be made to the specification files, which is described in "Creating a Basic OSM Instance".

Note:

These steps exclude any one-time setup activities. For details on one-time setup activities, see the tasks you must do before creating an OSM instance in "Creating a Basic OSM Cloud Native Instance".

Where pre-requisite secrets are required, the toolkit provides sample scripts for this activity. However, the scripts are not pipeline-friendly. Use the scripts for manually standing up an instance quickly and not for any automated process for creating instances. These scripts are also important because they illustrate both the naming of the secret and the layout of the data within the secret that OSM cloud native requires. You must replace references to toolkit scripts for creating secrets with your own mechanism in your DevOps process.

Configuring Code for Creating an OSM Instance

To configure code for creating an instance, you assemble the configuration at the project and the instance levels. While some of these activities could be automated, much of the work is manual in nature.

1. Assemble the configuration.
   To assemble the configuration at the project level:

   Note:

   These steps should be performed once per project and then re-used for each instance.

   a. Copy **$OSM_CNTK/samples/project.yaml** to your file repository and rename to align with your project naming decisions made earlier (for example, *project*.yaml).
   b. Assemble the optional configuration files as needed. These files include custom WDT fragments, custom shapes, cartridge user files, and par files for deployment.

   To assemble the configuration at the instance level, copy **$OSM_CNTK/samples/instance.yaml** to your file repository and rename to align with your project naming decisions made earlier (for example, *project-instance*.yaml).
2. Create pre-requisite secrets for OSM DB access, RCU DB access, OSM system users, OPSS, Introspector and the WLS Admin credentials used when creating the domain.

```
$OSM_CNTK/scripts/manage-instance-credentials.sh -p project -i
instance \
 create \
 osmdb,rcudb,wlsadmin,osmldap,opssWP,wlsRTE
```

> **Note:**
>
> Passwords and other secret input must adhere to the rules specified of the corresponding component.

3. Create custom secrets as required by your OSM solution cartridges.

   ```
   $OSM_CNTK/samples/credentials/manage-cartridge-credentials.sh -p
   project -i instance \
     -c create \
     -f user information file

   ** $OSM_CNTK/samples/credentials/manage-cartridge-credentials.sh -h
   for help
   ```

4. Create other custom secrets as required by optional configuration.
5. Populate the embedded LDAP with all the cartridge users (only those from prefix/map name osm) under the `cartridgeUsers` section in the *project*.yaml file. During the creation of the OSM server instance, for all the users listed, an account is created in embedded LDAP with the same username and password as the Kubernetes secret:

   ```
   cartridgeUsers:
     - osm
     - osmoe
     - osmde
     - osmfallout
     - osmoelf
     - osmlfaop
     - osmlf
     - tomadmin
   ```

After the configuration and the input are available, the remaining activities are focused on Continuous Delivery, which can be automated.

1. Register a namespace per project:

   ```
   $OSM_CNTK/scripts/register-namespace.sh -p project -t namespaces
   # For example, $OSM_CNTK/scripts/register-namespace.sh -p sr -t
   wlsko,traefik
   # where the namespaces are separated by a comma without spaces
   ```

   > **Note:**
   >
   > `wlsko` and `traefik` are examples of namespaces. Do not provide details about Traefik if you are not using it.

2. Create one OSM PDB per instance:
   - If the master OSM PDB exists in the CDB, clone the PDB. In this scenario, a master PDB is created by cloning a seed PDB, deploying the OSM/RCU

schema, and then optionally deploying cartridges. This master is only valid for a specific OSM schema version.

- If the master CDB does not have the schema provisioned, do the following:
  a. Clone the seed PDB and then run the DB installer to create OSM and the RCU schema:

     ```
     $OSM_CNTK/scripts/install-osmdb.sh -p sr -i quick -s
     $SPEC_PATH -c 1 (OSM Schema)
     $OSM_CNTK/scripts/install-osmdb.sh -p sr -i quick -s
     $SPEC_PATH -c 7 (RCU Schema)
     ```

  b. Deploy the cartridges:

     ```
     ./scripts/manage-cartridges.sh -p project_name -i
     instance_name -s $SPEC_PATH
      -f par_file -c parDeploy
     ```

- If you want to use a PDB from another instance in order to reuse the OSM data, do the following:
  a. Clone the existing PDB.
  b. Drop the existing RCU:

     ```
     $OSM_CNTK/scripts/install-osmdb.sh -p project -i instance -s
     $SPEC_PATH -c 8
     ```

  c. Recreate the RCU:

     ```
     $OSM_CNTK/scripts/install-osmdb.sh -p project -i instance -s
     $SPEC_PATH -c 7
     ```

  Alternatively, the RCU schema can be re-used. This use case has additional CaC changes as discussed in the Re-using PDB topic.

3. Create the Ingress:

   ```
   $OSM_CNTK/scripts/create-ingress.sh -p project -i instance -s
   $SPEC_PATH
   ```

4. Create the instance.

   ```
   $OSM_CNTK/scripts/create-instance.sh -p project -i instance -s
   $SPEC_PATH
   ```

Deleting an Ingress

To delete an ingress, run the following command:

```
$OSM_CNTK/scripts/delete-ingress.sh -p project -i instance
```

Deleting an Instance

This section describes the sequence of activities for deleting and cleaning up various aspects of the OSM environment.

To delete the application instance:

1. Run the following command:

```
$OSM_CNTK/scripts/delete-instance.sh -p project -i instance
```

2. Remove the instance content manually from the LDAP server using your LDAP Admin client. Specify ou=*project-instance*.

To clean up the PDB, drop it.

To clean up the configuration as code:

1. Delete the OSM instance and the database instance specification files.
2. Delete the secrets:

```
$OSM_CNTK/scripts/manage-instance-credentials.sh -p project -i
instance \
 delete \
 osmldap,osmdb,rcudb,wlsadmin,opssWP,wlsRTE
```

3. Delete any additional custom secrets using kubectl.

Trying to streamline the processes and identifying when to omit certain activities and where other activities must be repeated can be challenging. For instance, dropping the OSM RCU schema is independent of deleting an instance, which happens through different script invocations. While the life-cycle of the OSM instance and the PDB should be aligned, there are also use cases where the business data in a PDB (cartridges or orders) is required for re-use by a different OSM instance. For details on specific use cases, see "Reusing the Database State".

Securing Operations in Kubernetes Cluster

This section describes how to secure the operations of OSM cloud native users in a Kubernetes cluster. A well organized deployment of OSM cloud native ensures that individual users have specific privileges that are limited to the requirements for their approved actions. The Kubernetes objects concerned are service accounts and RBAC objects.

All OSM cloud native users fall into the following three categories:

- Infrastructure Administrator
- Project Administrator
- OSM User

Infrastructure Administrator

Infrastructure Administrators perform the following operations:

- Install WebLogic Kubernetes Operator in its own namespace
- Create a project for OSM cloud native and configure it
- After creating a new project, run the **register-namespace.sh** script provided with the OSM cloud native toolkit
- Before deleting an OSM cloud native project, run the **unregister-namespace.sh** script

- Delete an OSM cloud native project
- Manage the lifecycle of WebLogic Kubernetes Operator (restarting, upgrading, and so on)

Project Administrator

Project Administrators can perform all the tasks related to an instance level OSM cloud native deployment within a given project. This includes creating, updating, and deleting secrets, OSM cloud native instances, OSM cloud native DB Installer, and so on. A project administrator can work on one specific project. However, a given human user may be assigned Project Administrator privileges on more than one project.

OSM User

This class of users corresponds to the users described in the context of traditionally deployed OSM. These users can log into the user interfaces (UI) of OSM and can call the OSM APIs. These users are not Kubernetes users and have no privileges outside that granted to them within the OSM application. For details about user management, see the *OSM Cloud Native System Administrator's Guide* and "Setting Up Authentication" in this guide.

About Service Accounts

Installing the WebLogic Kubernetes Operator requires the presence of a service account that is set up appropriately. The **install-operator.sh** script requires a service account called *wlsko-ns*-**sa** in the *wlsko-ns* namespace. For example, if the namespace where WKO is to be installed is called wlsko, then the expected service account is **wlsko-sa**. If a service account is found with the correct name, the script uses it. Otherwise, the script creates a service account by that name. The WKO pods need to be installed by the Infrastructure Administrator with cluster-admin privileges, but at runtime, they use this service account and its associated privileges as described in the WKO documentation.

The pods that comprise each OSM cloud native instance (including the transient OSM DB Installer pod and the transient WKO Introspector pod) within a given project namespace use the "default" service account in that namespace. This is created at the time of namespace creation, but can be modified by the Infrastructure Administrator later.

RBAC Requirements

The RBAC requirements for the WebLogic Kubernetes Operator are documented in its user guide. The privileges of the Infrastructure Administrator have to include these. In addition, the Infrastructure Administrator must be able to create and delete namespaces, operate on the WebLogic Kubernetes Operator's namespace and also on the Traefik namespace (if Traefik is used as the ingress controller). Depending on the specifics of your Kubernetes cluster and RBAC environment, this may require cluster-admin privileges.

The Project Administrator's RBAC can be much more limited. For a start, it would be limited to only that project's namespace. Further, it would be limited to the set of actions and objects that the instance-related scripts manipulate when run by the Project Administrator. This set of actions and objects is documented in the OSM cloud native toolkit sample located in the **samples/rbac** directory.

Structuring Permissions Using the RBAC Sample Files

There are many ways to structure permissions within a Kubernetes cluster. There are clustering applications and platforms that add their own management and control of these permissions. Given this, the OSM cloud native toolkit provides a set of RBAC files as a sample. You will have to translate this sample into a configuration that is appropriate for your environment. These samples are in **samples/rbac** directory within the toolkit.

The key files are **project-admin-role.yaml** and **project-admin-rolebinding.yaml**. These files govern the basic RBAC for a Project Administrator.

Do the following with these files:

1. Make a copy of both these files for each particular project, renaming them with the *project/namespace* name in place of "project". For example, for a project called "biz", these files would be **biz-admin-role.yaml** and **biz-admin-rolebinding.yaml**.
2. Edit both the files, replacing all occurrences of *project* with the actual *project/namespace* name.

   For the *project*-**admin-rolebinding.yaml** file, replace the contents of the "subjects" section with the list of users who will act as Project Administrators for this particular project.

   Alternatively, replace the contents with reference to a group that contains all users who will act as Project Administrators for this project.
3. Once both files are ready, they can be activated in the Kubernetes cluster by the cluster administrator using **kubectl apply -f** *filename*.

   It is strongly recommended that these files be version controlled as they form part of the overall OSM cloud native configuration.

The Project Administrator role specification contains the **pods/exec** resource. This is required for only one specific scenario - using the OSM DB Installer to deploy a cartridge from the local file system (where the **install-osmdb.sh** script is being run). This particular resource can be removed, forcing cartridge deployment to only happen from a repository. It is highly recommended to remove this resource for production environments. The resource may be retained for development environment, as it eases the code-build-deploy-test cycle for OSM cartridge development.

In addition to the main Project Administrator role and its binding, the samples contain two additional and optional role-rolebinding sets:

- **project-admin-addon-role.yaml** and **project-admin-addon-rolebinding.yaml**: This role is per project and is an optional adjunct to the main Project Administrator role. It contains authorization for resources and actions in the project namespace that are not required by the toolkit, but might be of some use to the Project Administrator for debugging purposes.
- **wko-read-role.yaml** and **wko-read-rolebinding.yaml**: This role is available in the WebLogic Kubernetes Operator's namespace, and is an optional add-on to the Project Administrator's capabilities. It lets the user list the WKO pods and view their logs, which can be useful to debug issues related to instance startup and upgrade failures. This is suitable only for sandbox or development environments. It is strongly recommended that, even in these environments, WKO logs be exposed via a logs toolchain. The WebLogic Kubernetes Operator's Helm chart comes with the capability to interface with an ELK stack. For details, see https://oracle.github.io/weblogic-kubernetes-operator/userguide/managing-operators/#optional-elastic-stack-elasticsearch-logstash-and-kibana-integration.

6

Creating Your Own OSM Cloud Native Instance

This chapter provides information on creating your own OSM instance. While the "Creating a Basic OSM Cloud Native Instance" chapter provides instructions for creating a basic OSM instance that is capable of processing orders for the **SimpleRabbits** sample cartridge that is provided with the OSM cloud native toolkit, this chapter provides information on how you can create an OSM instance that is tailored to the business requirements of your organization. However, if you want to first understand details on infrastructure setup and structuring of OSM instances for your organization, then see "Planning Infrastructure".

Before proceeding with creating your own OSM instance, you can look at the alternate and optional configuration options described in "Exploring Alternate Configuration Options".

When you created a basic instance, you used the operational scripts and the base configuration provided with the toolkit.

Creating your own instance involves various activities spanning both instance management and instance configuration and includes some of the following tasks:

- Configuring OSM Runtime Parameters
- Preparing Cartridges for OSM Cloud Native
- Extending the WDT Model. See "Extending the WebLogic Server Deploy Tooling (WDT) Model".
- Working with Kubernetes Secrets
- Adding JMS Queues and Topics
- Creating a JMS template
- Working with Cartridges
- Provisioning Cartridge User Accounts

Configuring OSM Runtime Parameters

You can control various OSM runtime parameters using the **oms-config.xml** file. See "Configuring OSM with oms-config.xml" in *OSM Cloud Native System Administrator's Guide* for details.

This configuration is managed differently in OSM cloud native. While all the parameters described in the *OSM Cloud Native System Administrator's Guide* are still valid, the method of specifying them is different for a cloud native deployment.

Each of the three specification file tiers - shape, project, and instance - has a section called **omsConfig**. For example, the project specification has the following section:

```
omsConfig:
  project:

com.mslv.oms.handler.cluster.ClusteredHandlerFactory.HighActivityOrder.C
ollectionCycle.Enabled: true
   oracle.communications.ordermanagement.cache.UserPerferenceCache: near
```

Some parameters have been laid out for you in the pre-configured shape specification files and in the sample project and instance specification files. When you wish to change the value of a parameter to a different one from the documented default value, you must add that parameter and its custom value to an appropriate specification file.

For values that depend on (or contribute to) the footprint of the OSM Managed Server, the shape specification would be best. For values that are common across instances for a given project, the project specification would be best. Values that vary for each instance would be appropriate in the instance specification.

Any parameter specified in the instance specification overrides the same parameter specified in the project or shape specification. Any parameter specified in the project specification overrides the same parameter in the shape specification.

Any parameter that is not present in all three specification files (shape, project, instance) automatically has its default value as documented in *OSM Cloud Native System Administrator's Guide*.

Note:

All pre-defined shape specifications have the `omsConfig` parameters flagged as `do NOT delete`. These must not be edited and must be copied as-is to custom shapes. See "Working with Shapes" for details about custom shapes.

Preparing Cartridges

Existing OSM cartridges that run on a traditional OSM deployment can still be used with OSM cloud native, but you prepare and deploy those cartridges differently. Instead of using multiple interfaces to persist the WebLogic domain configuration (WebLogic Admin console and WLST), the configuration is added into the files that feed into the instance creation mechanism. With OSM cloud native, you use the WebLogic Admin Console only for validation purposes.

Before proceeding, you must determine which OSM solution cartridge you want to use to validate your OSM cloud native environment. For simplicity, use a setup where any communication with OSM is restricted to an application running in the same instance of the WebLogic domain.

Identify the following requirements for your cartridge:

- The list of JMS queues and topics that the cartridge needs.
- The list of credentials stored in the OSM Credential Store.

- Users that the cartridge requires.
- Applications that need to be deployed to the WebLogic server. This can include system emulators for stubbing out communication to external peer systems.

About OSM Cloud Native Cartridges and Design Studio

Existing cartridges do not always need to be rebuilt for use with OSM cloud native. As long as they were built with an OSM 7.4.0.x SDK, using the Design Studio target OSM version of 7.4.0, their existing par files can be deployed.

If cartridges have to be built afresh or re-built, use the OSM SDK packaged with OSM 7.4.1 release, and set the Design Studio target OSM version as 7.4.0. In general, use the Design Studio target OSM version that is closest to the actual OSM version but not newer than it.

About Domain Configuration Restrictions

Some restrictions on the domain configuration are necessary to keep the process simple for creating and validating your basic OSM cloud native instance. As you build confidence in the tooling and the extension mechanisms, you can remove the restrictions and include additional configuration in your specifications to support advanced features.

Ensure that you restrict the domain configuration to the following:

- Instance with no SAF setup.
- Re-directing logs (to live outside the pods) will not be configured at this time.

Changing the Default Values

The project and the instance specification templates in the toolkit contain the values used in the out-of-the-box domain configuration. These files are intended for editing, as the required information such as the PDB host needs updating and the flags controlling the optional features such as NFS need to be turned on or off, and the default values such as Java options and cluster size can be changed. If you find that the existing values need to be updated for your OSM instance, update the values in your specification files.

Change the default values as per the following guidelines:

- `NFS`: As per the restrictions, leave `nfs` disabled in the instance specification
- `Shape`: The provided configuration uses tuning parameters that are appropriate for a development environment. This is done through the use of a shape specification that is specified in the instance specification.

Creating an instance with the default shape is recommended. For details on how you can provide a custom shape if necessary, see "Working with Shapes".

Adding New WDT Metadata

The OSM cloud native toolkit provides the base WDT metadata in **$OSM_CNTK/charts/osm/templates**. As the OSM application requires this WDT metadata for the proper functioning, this must not be edited. Instead, the toolkit provides a mechanism whereby new pieces of WDT metadata can be included in the final description of the domain.

See "Extending the WebLogic Server Deploy Tooling (WDT) Model" for complete details on the general process for providing custom WDT. The steps described must be repeated for a variety of WDT use cases.

You must specify the JMS Queues required for your new using the WDT metadata.

There are two options for providing the required configuration for JMS Queues:

- Re-using the OSM JMS Resources as described in "Adding JMS Queues and Topics". This is the suggested mechanism for your first attempt at configuring your customized OSM instance.
- Creating custom JMS Resources as described in "Adding a JMS System Resource".

Handling of sensitive data from within the WDT metadata fragment is supported as described in the "Accessing Kubernetes Secrets from WDT Metadata".

Other Customizations

To support a custom OSM solution cartridge, not all changes are done using the WDT metadata. Depending on the processing needs of your OSM solution cartridge, there are other changes that are likely required:

This topic describes how to use the following methods for supporting a custom solution cartridge:

- Credential Store
- Custom Application EAR
- Cartridge Users

Credential Store

For traditional installations, if a solution cartridge has automation plugins that needed to retrieve external system credentials, it did so by storing those credentials in the WebLogic Credential Store.

In OSM cloud native, if your cartridge uses the credential store framework of OSM, then you must provision cartridge user accounts. See "Provisioning Cartridge User Accounts" for details.

Custom Application Ear

If there are additional applications that need to be deployed to WebLogic to support the processing of your OSM solution cartridge, see "Deploying Entities to an OSM WebLogic Domain".

This method requires both WDT metadata as well as the custom OSM images. Supplemental scripts and WDT fragments are provided as samples in the **$OSM_CNTK/samples/customExtensions**

Cartridge Users

Cartridges may also define users who need access to OSM APIs. These user credentials need to be available in the right locations as described in "Provisioning Cartridge User Accounts". These credentials must then be made available through the configuration to OSM.

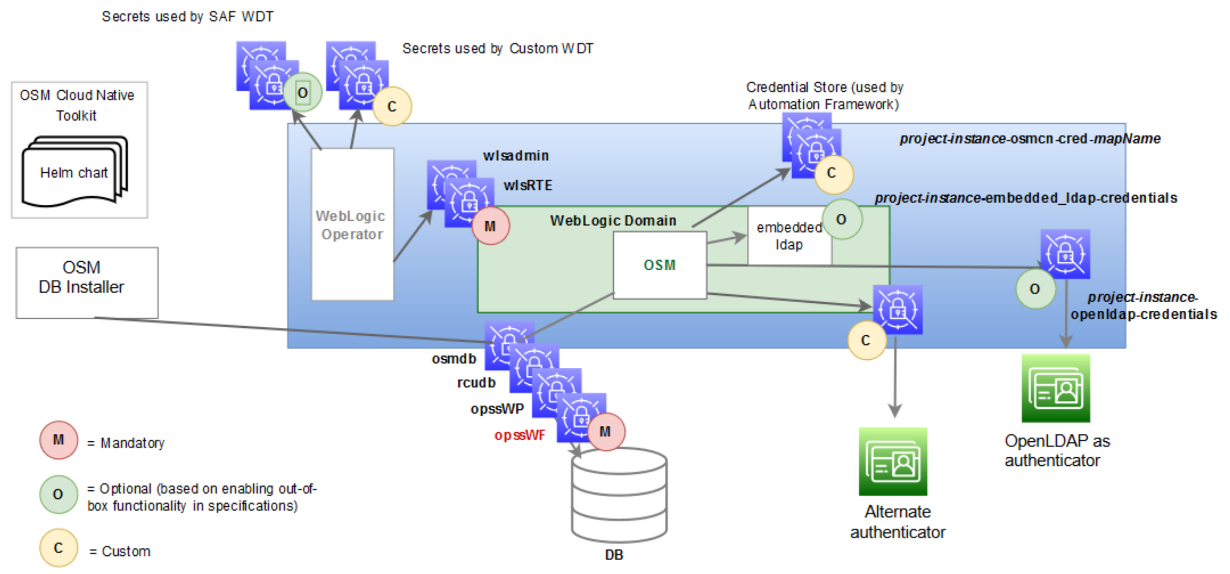
Working with Kubernetes Secrets

Secrets are Kubernetes objects that you must create in the cluster through a separate process that adheres to your corporate policies around managing secure data. Secrets are then made available to OSM cloud native by declaring them in your configuration.

When the OSM cloud native sample scripts are not used for creating secrets, the secrets you create must align to what is expected by OSM. The sample scripts contain guidelines for creating secrets.

The following diagram illustrates the role of Kubernetes Secrets in an OSM cloud environment:

Figure 6-1 Kubernetes Secrets in OSM Cloud Environment


Secrets used by SAF WDT
Secrets used by Custom WDT
OSM Cloud Native Toolkit
Helm chart
OSM DB Installer
WebLogic Operator
Credential Store (used by Automation Framework)
project-instance-osmcn-cred-mapName
project-instance-embedded_ldap-credentials
wlsadmin
wlsRTE
WebLogic Domain
OSM
embedded ldap
project-instance-openldap-credentials
osmdb
rcudb
opssWP
opssWF
DB
Alternate authenticator
OpenLDAP as authenticator
M = Mandatory
O = Optional (based on enabling out-of-box functionality in specifications)
C = Custom

There are three classifications of secrets, as shown in the above illustration:

- Mandatory (Pre-requisite) Secrets
- Optional Secrets
- Custom Secrets

About Mandatory Secrets

Mandatory secrets must be created prior to running the cartridge management scripts or the instance creation script.

The toolkit provides the sample script: **$OSM_CNTK/scripts/manage-instance-credentials.sh** to create the secrets for you. Refer to the script code to see the naming and internal structure required for each of these secrets.

See the following topics for more details about Kubernetes Secrets:

- Creating Secrets
- Management of Secrets

About Optional Secrets

Optional secrets are dictated by enabling the out-of-the-box configuration. There is some functionality that is pre-configured in OSM cloud native and can be enabled or disabled in the specification files. When the functionality is enabled, these secrets must be created in the cluster before an OSM instance is created.

- If you use OpenLDAP for authentication, OSM cloud native relies on the following secret to have been created:

  ```
  project-instance-openldap-credentials
  ```

  The toolkit provides a sample script to create these secrets for you (**$OSM_CNTK/samples/credentials/manage-osm-ldap-credentials.sh** by passing in "-o secret").
- With Credential Store, the secrets hold credentials for external systems that the automation plug-ins access. These secrets are a pre-requisite to running cartridges that rely on this mechanism and must adhere to a naming convention. See "Provisioning Cartridge User Accounts" for more details.
- When SAF is configured, SAF secrets are used. SAF secrets are similar to custom secrets and are declared in a specialized area within the instance specification that feeds into the SAF-specific WDT.

  ```
  safConnectionConfig:
    - name: external_system_identifier
      t3Url: t3_url
      secretName: secret_t3_user_pass
  ```

About Custom Secrets

OSM cloud native provides a mechanism where WDT metadata can access sensitive data through a custom secret that is created in the cluster and then declared in the configuration. See "Accessing Kubernetes Secrets from WDT Metadata" to familiarize yourself with this process.

This class of secrets are required only if you need secrets for this mechanism.

To use custom secrets with WDT metadata:

Note:

As an example, this procedure uses a WDT snippet for authentication.

1. Add secret usage in the WDT metadata fragment:

   ```
   Host: '@@SECRET:authentication-credentials:host@@'
   Port: '@@SECRET:authentication-credentials:port@@'
   ControlFlag: SUFFICIENT
   Principal: '@@SECRET:authentication-credentials:principal@@'
   CredentialEncrypted: '@@SECRET:authentication-
   credentials:credential@@'
   ```

2. Add the secret to the project specification.

   ```
   #Custom secrets
   # Multiple secret names can be provided
   customSecrets:
     enabled: true
     secretNames:
       - authentication-credentials
   ```

3. Create the secret in the cluster, by using any one of the following methods:
   - Using OSM cloud native toolkit scripts
   - Using a Template
   - Using the Command-line Interface

   In the example metadata shown in step 1, the secret must capture host, port, principal, and credential.

See "Mechanism for Creating Custom Secrets" for details about the methods.

Accommodating the Scope of Secrets

The WDT metadata fragments are defined at the project level as the project typically owns the solution definition. Accommodating this is a simple task. However, the scenario becomes complicated when you consider that there may be project level configuration that needs to allow for instance level control over the secret contents.

To walk through this, we will use authentication as an example and introduce a COM project that includes three instances: development, test, and production. The production environment has a dedicated authentication system, but the development and test instances use a shared authentication server.

To accommodate this scenario, the following changes must be made to each of the basic steps:

1. Define a naming strategy for the secrets that introduce scoping. For instance, secrets that need instance level control could prepend the instance name. In the example, this results in the following secret names:
   - `COM-dev-authentication-credentials`
   - `COM-test-authentication-credentials`
   - `COM-prod-authentication-credentials`
2. Include the secret in the WDT fragment. In order for this scenario to work, a generic way is required to declare the "scope" or instance portion of the secret name. To do this, use the built-in Helm values:

   ```
   .Values.name - references the full instance name (project-instance)
   .Values.namespace - references the project name (project)
   ```

   If the fragment needs to support instance-level control, derive the instance name portion of the secret name.

   ```
   Host: '@@SECRET:{{ .Values.name }}-authentication-
   credentials:host@@'
   ```

```
Port: '@@SECRET:{{ .Values.name }}-authentication-
credentials:port@@'
ControlFlag: SUFFICIENT
Principal: '@@SECRET:{{ .Values.name }}-authentication-
credentials:principal@@'
CredentialEncrypted: '@@SECRET:{{ .Values.name }}-authentication-
credentials:credential@@'
```

3. Add the secret to the instance specification. The secret name must be provided in the instance specification as opposed to the project specification.

   ```
   ## Dev Instance Spec

    #Custom secrets
   # Multiple secret names can be provided
   customSecrets:
     enabled: true
     secretNames:
      - COM-dev-authentication-credentials

    ## Test Instance spec

    #Custom secrets
   # Multiple secret names can be provided
   customSecrets:
     enabled: true
     secretNames:
      - COM-test-authentication-credentials

    ## Prod Instance Spec

   #Custom secrets
   # Multiple secret names can be provided
   customSecrets:
     enabled: true
     secretNames:
      - COM-prod-authentication-credentials
   ```

4. Create the secret in the cluster by following any one of the methods described in the **Mechanism for Creating Custom Secrets** topic. In our example, the secret would need to capture host, port, principal and credential. Each instance would need a secret created, but the values provided depend on which authentication system is being used.

   ```
   # Dev secret creation

    kubectl create secret generic COM-dev-authentication-credentials \
   -n COM \
   --from-literal=principal=<value1> \
   --from-literal=credential=<value2> \
   --from-literal=host=<value3> \
   --from-literal=port=<value4>

    # Test secret creation
   ```

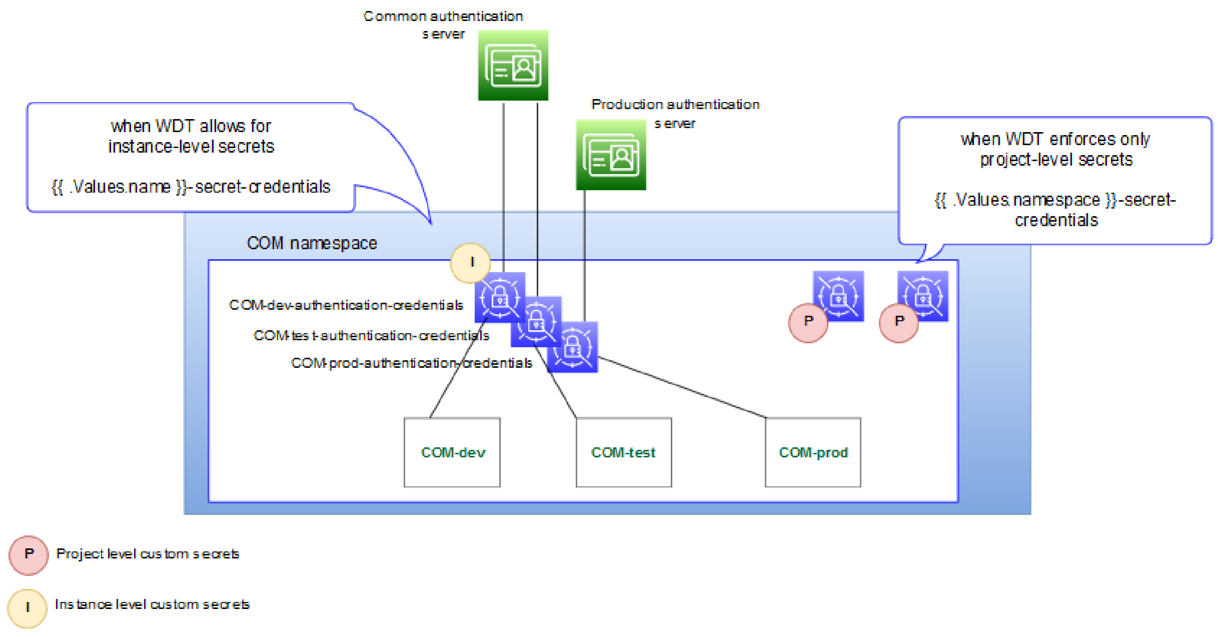
```
kubectl create secret generic COM-test-authentication-credentials \
-n COM \
--from-literal=principal=<value1> \
--from-literal=credential=<value2> \
--from-literal=host=<value3> \
--from-literal=port=<value4>

  ##Production secret creation

 kubectl create secret generic COM-prod-authentication-credentials \
-n COM \
--from-literal=principal=<prodvalue1> \
--from-literal=credential=<prodvalue2> \
--from-literal=host=<prodvalue3> \
--from-literal=port=<prodvalue4>
```

The following diagram illustrates the secret landscape in this example:

Figure 6-2 Landscape of Secrets


Common authentication server
Production authentication server
when WDT allows for instance-level secrets
{{ .Values.name }}-secret-credentials
when WDT enforces only project-level secrets
{{ .Values.namespace }}-secret-credentials
COM namespace
I
COM-dev-authentication-credentials
COM-test-authentication-credentials
COM-prod-authentication-credentials
P
P
COM-dev
COM-test
COM-prod
P
Project level custom secrets
I
Instance level custom secrets

Mechanism for Creating Custom Secrets

You can create custom secrets in any of the following ways:

- Using Scripts
- Using a Template
- Using the Command-line Interface

Using Scripts to Create Secrets

Functionality such as OpenLDAP, NFS, and Credential Store that can be enabled or disabled in OSM cloud native relies on pre-requisite secrets to be created. In such cases, the toolkit provides sample scripts that can create the secrets for you. While these scripts are useful for configuring instances quickly in development situations, it is important to remember that they are sample scripts, and not pipeline friendly. These scripts are also essential because when the secret is mandated by OSM cloud native, both the secret name and the secret data are available in the sample script that populates it.

As an example, the secrets used by the Credential Store mechanism must follow a specific naming convention:

```
projectName-instanceName-osmcn-cred-mapName
```

Using a Template

To create custom secrets using a template:

1. Save the secret details into a template file.

   ```
   apiVersion: v2
   kind: Secret
   metadata:
    labels:
    weblogic.resourceVersion: domain-v2
    weblogic.domainUID: project-instance
    weblogic.domainName: project-instance
    namespace: project
    name: secretName
   type: Opaque
   stringData:
   password_key: value1
   user_key: value2
   ```

2. Run the following command to create the secret:

   ```
   kubectl apply -f templateFile
   ```

Using the Command-line Interface

You can also specify the secret name and the details directly on the command-line interface:

```
kubectl create secret generic secretName \
-n project \
--from-literal=password_key=value1 \
--from-literal=user_key=value2
```

Adding JMS Queues and Topics

JMS queues and topics are unique because the base JMS resources (JMS server and JMS subdeployments) already exist in the domain as the OSM core application requires them. You can add custom queues and topics to the OSM JMS resources by specifying the appropriate content in the project specification file.

To add queues or topics, uncomment the sample in your specification file, providing the values necessary to align with your requirements.

Consider the following points:

- The only mandatory values are 'name' and 'jndiName'.
- Text in angular brackets do not have a default value. You must supply an actual value per your requirements.
- The remaining parameters are set to their default values if omitted. When a different value is supplied in the specification file, it is used as an override to the default value.

> **Note:**
>
> There should only be one list of `uniformDistributedQueues` and one list of `uniformDistributedTopics` in the specification. When copying the content from the samples, ensure that you do not replicate these sections more than once.

To add JMS distributed queues:

```
# jms distributed queues
uniformDistributedQueues:
 - name: custom-queue-name
 jndiName: custom-queue-jndi
 resetDeliveryCountOnForward: false
 deliveryFailureParams:
 redeliveryLimit: 10
 deliveryParamsOverrides:
 timeToLive: -1
 priority: -1
 redeliveryDelay: 1000
 deliveryMode: 'No-Delivery'
 timeToDeliver: '-1'
```

To add JMS distributed topics:

```
# jms distributed topics
uniformDistributedTopics:
 - name: custom-topic-name
 jndiName: custom-topic-jndi
 resetDeliveryCountOnForward: false
 deliveryFailureParams:
 redeliveryLimit: 10
 deliveryParamsOverrides:
 timeToLive: -1
 priority: -1
 redeliveryDelay: 1000
 deliveryMode: 'No-Delivery'
 timeToDeliver: '-1'
```

Generating Error Queues for Custom Queues and Topics

You can generate error queues for all custom queues and topics automatically.

To generate error queues automatically, configure the following parameters in the **project.yaml** file:

```
errorQueue:
    autoGenerate: false
    expirationPolicy: "Redirect"
    redeliveryLimit: 15
```

By default, the `autoGenerate` parameter is set to **false**. To generate error queues for all JMS queues automatically, set this parameter to **true**.

When `autoGenerate` is set to **true**, all custom queues and topics will have their own error queues.

The following sample shows the error queue generated for a custom queue:

```
'jms_queue_name_ERROR':
    ResetDeliveryCountOnForward: false
    SubDeploymentName: osm_jms_server
    JNDIName: error/ jms_queue_jndiName
    IncompleteWorkExpirationTime: -1
    LoadBalancingPolicy: 'Round-Robin'
    ForwardDelay: -1
    Template: osmErrorJmsTemplate
```

Note:

- All error queues have _**ERROR** as the suffix.
- For internal queues and topics in OSM, generation of error queues is always enabled. Each queue and topic has its own _**ERROR** queue. Messages that cannot be delivered are redirected accordingly.
- Disable this feature for O2A 2.1.2.1.0 cartridges used in an OSM cloud native environment. The O2A build generates its own project specification fragment, which must be used instead.

Creating a JMS Template

A JMS template provides an efficient means of defining multiple destinations with similar attribute settings.

You can add one or more JMS templates if required in addition to the one provided. To create additional JMS templates, copy the **customJmsTemplate** definition and rename it:

```
# JMS Template (optional). Uncomment to define "customJmsTemplate"
# Alternatively use the built-in template "customJmsTemplate"
#jmsTemplate:
#  customJmsTemplate:
#    DeliveryFailureParams:
#      RedeliveryLimit: 10
#      ExpirationPolicy: Discard
#    DeliveryParamsOverrides:
#      RedeliveryDelay: 1000
#      TimeToLive: -1
#      Priority: -1
#      TimeToDeliver: -1
```

To use a JMS template for a queue or topic definition, you can specify the template name, as well as the unique JNDI name:

```
# jms distributed queues. Uncomment to define one or more JMS queues
under a
# single element uniformDistributedQueues.
uniformDistributedQueues: {} # This empty declaration should be removed
if adding items here.
#uniformDistributedQueues:
#  - name: jms_queue_name
#    jndiName: jms_queue_jndiName
#    jmsTemplate: customJmsTemplate

# jms distributed topic. Uncomment to define one or more JMS Topics
under a
# single element uniformDistributedTopics.
uniformDistributedTopics: {} # This empty declaration should be removed
if adding items here.
#uniformDistributedTopics:
#  - name: jms_topic_name
#    jndiName: jms_topic_jndiName
#    jmsTemplate: customJmsTemplate
```

If the queues and topics need to be created under custom JMS resources, then the OSM cloud native WDT extension mechanism should be employed as described in "Adding a JMS System Resource".

Working with Cartridges

This section describes how you build, deploy, and undeploy OSM cartridges in a cloud native environment.

OSM cartridges are built using either Design Studio or build scripts, which are the methods used for building cartridges in traditional environments.

This topic covers the following operations:

- Deploying Cartridges Using the OSM Cloud Native Toolkit
- Deploying Cartridges Using Design Studio

Deploying Cartridges Using the OSM Cloud Native Toolkit

To deploy cartridge par files, OSM cloud native employs a mechanism using the OSM cloud native toolkit's **manage-cartridges.sh** script. You can deploy cartridge par files in offline or online modes.

Use the following commands with the **manage-cartridges.sh** script:

- **-p** *projectName*: Mandatory. Name of the project.
- **-i** *instanceName*: Mandatory. Name of the instance.
- **-s** *specPath*: Mandatory. The location of the specification files. A colon(:) delimited list of directories.
- **-m** *customExtPath*: Use this to specify the path of custom extension files. Takes a colon(:) delimited list of directories. If the path provided is empty with the custom flag enabled as **true** in the specifications, then the script is stopped.
- **–o**: Enables online cartridge deployment.
- **-c** *commandName*: Mandatory. Use the following command names:
  - **parDeploy**: Use this to deploy a cartridge par file from your local file system. Use this for development environments only.
  - **sync**: Use this to synchronize cartridges using the project specification and remote repository. Use this for all controlled environments.
- **-f** *parPath*: Mandatory if **parDeploy** is used. This specifies the path of the cartridge par file that you want to deploy.
- **-q**: Optional. Disables verbose progress indicators.

The **manage-cartridges.sh** script spins up a pod to perform the requested deployment activities:

- If **parDeploy** is chosen, the script must be run such that it has access to the specified cartridge par file as well as the "kubectl cp" privileges on the pod that is spun up.
- If **sync** is chosen, the script compares the list of cartridges and versions in the project specification file against those that are present in the OSM cloud native database and performs the necessary synchronization actions. The list in the project specification file must depict the desired target state:

  Note:

  In the actions listed below, "cartridge" refers to "cartridge+version".

  - If a cartridge is listed as **deployed**, but is not deployed in the database: it is deployed.
  - If a cartridge is listed as **deployed** and the same version exists in the database, the two are compared; if there is a difference, the new par file is redeployed.

  - If a cartridge is listed with a **default** setting that does not match with what is in the database, the **default** setting in the database is updated to match; no change is done to this setting if they already match.
  - If a cartridge is listed as **fastundeployed** and it exists as **active** in the database, it is fast-undeployed in the database. If the cartridge is already fast-undeployed in the database, nothing is done. If the cartridge does not exist in the database, nothing is done.

  The OSM cloud native toolkit ignores the "**default**" flag in the par file when the **sync** command is used - it enforces the list as specified in the project specification. For each cartridge, the **sync** validation ensures that exactly one version is tagged as **default**.

Using a remote repository is the recommended approach as all aspects of an OSM instance, including the cartridge set deployed, remain in source controlled configuration.

Each entry in the list of cartridges describes a specific cartridge using the name of the cartridge, its version, the intended deployment state and the intended default state. In addition, it specifies a URL that can be used to download the cartridge par file into the cartridge management pod. This URL would be pointing to a remote repository that may require authentication or other parameters. The cartridge entry has fields that can be used to provide parameters (in the form of "curl" command line parameters) as well as a secret that carries the username and password information.

```
cartridges:
  - name: name_of_the_cartridge - Mandatory, (must match the cartridge
name in the par file)
    url: URL_of_the_location_where_to_download_the_cartridge_par_file -
Mandatory.
    secret: a Kubernetes_secret_in_the_project_namespace - Optional,
only required if remote URL server requires authentication.
    params: Commandline_parameters_will_be_passed_to_curl - Optional,
user can provide additional parameters like proxy settings for curl.
    version: cartridge version, example 1.0.0.0.0 - Mandatory,
cartridge version (must match the cartridge version in the par file)
    default: true|false - Mandatory, set this cartridge as default
cartridge or not.
    deploymentState: deployed|fastundeployed - Mandatory, indicate the
desired target state of the cartridge
```

Offline Cartridge Deployment

This deployment mode supports deployment of new cartridges, deployment of new versions of existing cartridges, and redeployment of existing cartridge versions with changes.

For offline cartridge deployment, all managed servers in your environment must be shut down. The script stops running if there are managed servers up and running. Rolling restart of managed servers is not performed for offline deployment.

When using the toolkit for deploying cartridges in offline mode, the running instance of OSM must be shut down first by scaling down the cluster size to 0:

```
vi spec_Path/project-instance.yaml
```

```
# Change the cluster size to 0

#cluster size
clusterSize: 0

$OSM_CNTK/scripts/upgrade-instance.sh -p project -i instance -s
spec_Path
```

Run the following command to deploy cartridges in offline mode:

```
$OSM_CNTK/scripts/manage-cartridges.sh -p project -i instance -s
spec_Path -c sync [-o]
```

Online Cartridge Deployment

This deployment mode supports deployment of new cartridges and deployment of new versions of existing cartridges.

Deploying cartridges in an OSM cloud native environment provides the following key benefits:

- You can deploy the cartridges without needing to isolate OSM from order processing at the JMS/HTTP level.
- You can describe the cartridges for an environment in a declarative fashion.

In online mode, you can deploy cartridges to your OSM cloud native running instance while orders from a cartridge that you deployed earlier are still being processed. To achieve this, you should have a minimum of two managed servers on which your OSM cloud native instance is running. In such an environment, when you deploy cartridges, OSM availability is uninterrupted and ongoing order processing continues.

You use the **manage-cartridges.sh** script with the **-o** option to enable online deployment of cartridges. After deploying the cartridges, the script performs a rolling restart of all the managed servers in your environment.

When deploying cartridges in online mode, the running instance of OSM must continue to run and the required cluster size is at least 2.

```
vi spec_Path/project-instance.yaml

# Change the cluster size to a minimum of 2

#cluster size
clusterSize: 2

$OSM_CNTK/scripts/upgrade-instance.sh -p project -i instance -s
spec_Path
```

Run the following command to deploy cartridges in online mode:

```
$OSM_CNTK/scripts/manage-cartridges.sh -p project -i instance -s
spec_Path -c sync [-o]
```

Consider the following when deploying cartridges in online mode:

- If no managed servers are running, a warning is shown that no managed server is up and running and that the deployment mode is switching to offline deployment. The script continues with offline deployment.
- If only one managed server is running, then the script fails to perform the deployment.

The OSM cloud native deployment has two different methods of providing cartridge par files based on the following types of the environment they are being deployed to:

- Open Environments
- Controlled Environments

Deploying Cartridges in Open Environments

Open environments are mostly development and some test environments. To deploy cartridges to a running instance of OSM cloud native in an open environment, you can use any of the following options:

- **Local par file**
  Run the script as follows:

  ```
  $OSM_CNTK/scripts/manage-cartridges.sh -p projectName -i
  instanceName -s spec_Path -f cartridge_par_file -c parDeploy
  ```

- **Remote Repository (Unsecured)**
  This approach could be suitable for test environments.

  1. Edit the project specification in your file repository to add entries for each cartridge to be deployed:

     ```
     ## Unsecured repository
     cartridges:
      - name: OracleComms_OSM_O2A_COMSOM_CSO_Solution
        version: 2.1.2.0.0
        url:
     http://example.com/Repo/OracleComms_OSM_O2A_COMSOM_CSO_Solution/
     OracleComms_OSM_O2A_COMSOM_CSO_Solution.par
        default: true
        deploymentState: deployed
      - name: SimpleRabbits
        version: 1.0.0.0.0
        url: http://example.com/Repo/SimpleRabbits/1.0/
     SimpleRabbits.par
        default: false
        deploymentState: fastundeployed
      - name: SimpleRabbits
        version: 2.0.0.0.0
        url: http://example.com/Repo/SimpleRabbits/2.0/
     SimpleRabbits.par
        default: true
        deploymentState: deployed
     ```

  2. Run the script as follows:

     ```
     $OSM_CNTK/scripts/manage-cartridges.sh -p project_name -i
     instance_name -s spec_path -c sync [-o]
     ```

- **Remote Repository - Disabling Verification**
  To disable host verification:

  1. Pass in the `curl -k` option as follows.

     > **Note:**
     >
     > Disabling the verification on a secured repository is a security risk.

     ```
     ## secured repository, disabling host verification
     cartridges:
      - name: OracleComms_OSM_O2A_COMSOM_CSO_Solution
        version: 2.1.2.0.0
        url:
     http://example.com/Repo/OracleComms_OSM_O2A_COMSOM_CSO_Solution/
     OracleComms_OSM_O2A_COMSOM_CSO_Solution.par
        default: true
        deploymentState: deployed
        params: -k
      - name: SimpleRabbits
        version: 1.0.0.0.0
        url: http://example.com/Repo/SimpleRabbits/1.0/
     SimpleRabbits.par
        default: false
        deploymentState: fastundeployed
        params: -k
      - name: SimpleRabbits
        version: 2.0.0.0.0
        url: http://example.com/Repo/SimpleRabbits/2.0/
     SimpleRabbits.par
        default: true
        deploymentState: deployed
        params: -k
     ```

  2. Run the script as follows:

     ```
     $OSM_CNTK/scripts/manage-cartridges.sh -p project_name -i
     instance_name -s specification_path -c sync [-o]
     ```

Deploying Cartridges in Controlled Environments

To install cartridges in controlled environments such as UAT, pre-production, and production, use only the declarative approach. Rather than copying the par files into the cartridge management pod, they are "pulled" from a URL.

The cartridge list is defined in the project specification, ensuring that the cartridge load is also under version control.

- **Using a Remote Repository**

  To use a remote repository to deploy cartridges in a controlled environment:

1. Edit the project specification in your file repository as follows:

   ```
   ## Credentials required
   cartridges:
    - name: OracleComms_OSM_O2A_COMSOM_CSO_Solution
      version: 2.1.2.0.0
      url:
   http://example.com/Repo/OracleComms_OSM_O2A_COMSOM_CSO_Solution/
   OracleComms_OSM_O2A_COMSOM_CSO_Solution.par
      default: true
      deploymentState: deployed
      secret: solution_cartridge_secret_name_in_lowercase
    - name: SimpleRabbits
      version: 1.0.0.0.0
      url: http://example.com/Repo/SimpleRabbits/1.0/
   SimpleRabbits.par
      default: false
      deploymentState: fastundeployed
      secret: solution_cartridge_secret_name_in_lowercase
    - name: SimpleRabbits
      version: 2.0.0.0.0
      url: http://example.com/Repo/SimpleRabbits/2.0/
   SimpleRabbits.par
      default: true
      deploymentState: deployed
      secret: solution_cartridge_secret_name_in_lowercase
   ```

   The secret would contain any authentication credentials required to download the par file from the remote repository. The toolkit relies on the secret having the entries for the username and password set to the appropriate values. These are used by curl.

   An example of creating the secret using kubectl on the command line is as follows:

   ```
   kubectl create secret generic
   solution_cartridge_secret_name_in_lowercase \
    -n project \
    --from-literal=username='remoteRepoUsername' \
    --from-literal=password='remoteRepoPassword'
   ```

2. Run the script as follows:

   ```
   ./scripts/manage-cartridges.sh -p project_name -i instance_name
   \ -s spec_path -c sync [-o]
   ```

- **Using a Remote Repository - TLS/SSL**
  For HTTPS, the SSL certificate of the repository server must be exposed to the cartridge management pod and then passed as a command line parameter `-cacert`*path_to_repo_server_ssl_certificate* to curl. The *path_to_repo_server_ssl_certificate* is the path within the pod.

  To allow curl access to the SSL certificate within the cartridge management pod:

1. Obtain the server certificate by running the following command:

   ```
   echo quit | openssl s_client -showcerts -
   servername repo_server_hostname -connect repo_server_url
   path_to_repo_server_ssl_name.pem
   ```

2. Run the **register-certificate.sh** script to create a Kubernetes secret that contains the SSL certificate:

   ```
   $OSM_CNTK/scripts/register-certificate.sh -p project_name -n
   secret_name -f path_to_repo_server_ssl_name.pem
   ```

3. Add the following fragment to the project specification to enable the secret to be mounted at the path **/etc/ssl/certs/** within the cartridge management pod. The name is the *secret_name* created in step 2 and *type* is the file extension of the certificate file:

   ```
   certificates:
     - name: secret_name
       type: file_type

   #example
   certificates:
     - name: mySecret
       type: pem
   ```

4. Add the parameter **--cacert /etc/ssl/certs/*secret_name*.*file_type*** to the `cartridges: params` parameter in the project specification:

   ```
   cartridges:
    - name: OracleComms_OSM_O2A_COMSOM_CSO_Solution
      version: 2.1.2.0.0
      url:
   http://example.com/Repo/OracleComms_OSM_O2A_COMSOM_CSO_Solution/
   OracleComms_OSM_O2A_COMSOM_CSO_Solution.par
      default: true
      deploymentState: deployed
      params: --cacert /etc/ssl/certs/secret_name.file_type
   ```

You use Design Studio or build scripts to undeploy (fast undeploy and full undeploy) OSM cartridges.

Deploying Cartridges Using Design Studio

You can deploy cartridges directly from Design Studio using the Eclipse user interface or headless Design Studio. However, use Design Studio for deploying cartridges in scenarios where there is a lot of churn in the build, deploy and test cycle, but not for production environments. If used in conjunction with the OSM cloud native cartridge management mechanism, then the deployed cartridges become out of sync with what is listed in the source controlled specification file. For this reason, deploying cartridges using Design Studio is not recommended for environments where the specification file is considered the single source of truth for the set of deployed cartridges.

In order to incorporate Design Studio into the larger OSM cloud native ecosystem, you need to have previously taken care of the mapping of the hostname to the Kubernetes cluster or the load balancer as described in "Planning and Validating Your Cloud Environment".

After confirming that this has been done, do the following in Design Studio:

- Ensure that the connection URL of the Design Studio environment project matches your OSM cloud native environment. This is likely: `http://`*instance.project*`.osm.org:30305/cartridge/wsapi`. The suffix `osm.org` is configurable.
- In the Design Studio workspace, depending on your network setup, you may need to set the **Proxy bypass** field in the **Network Connection** Preferences to: *instance.project*`.osm.org`

Provisioning Cartridge User Accounts

This section describes how to use the sample scripts to create credential store secrets and provide the instance configuration so that OSM cloud native can access the credentials.

The sample scripts also provide the ability to populate the OpenLDAP server so that OSM can authenticate any cartridge users. In this way, provisioning a cartridge user account uses the same mechanism regardless of the end location for the credentials. In this way, provisioning a cartridge user account uses the same mechanism regardless of the end location for the credentials.

This section covers the following topics:

- Creating Credential Store Secret
- Declaring the Secret
- Configuring LDAP Systems

OSM solution cartridges have complex requirements around user credentials:

- Automation plugins that handle communication with external systems need a programmatic way to access credentials so that outgoing requests can supply the appropriate credentials for the requested operation. To meet this requirement, a credential store mechanism is required. Credentials must be populated into a central repository for storing usernames and passwords, and OSM must be able to access this repository to pass credentials to the plugin code when requested.
- Additionally, if a cartridge defined user (non-human) account is accessing an OSM API, then the credentials for this user account also need to exist in the embedded LDAP so that OSM can authenticate the user. Also, the cartridge human user account needs to exist in the external authentication system (OpenLDAP).

In summary, some cartridge defined users need to be provisioned in a credential store, some in OpenLDAP or other LDAP provider, and some users need to be defined in both.

The following table summarizes the system that cartridge user accounts need to be provisioned to:

Note:

When the same credentials need to exist in both the LDAP server and as a Kubernetes secret, care must be taken to ensure the credentials remain in sync.

Table 6-1 Cartridge User Accounts

User Credential Usage	LDAP	Kubernetes Secret	Description
OSM UI	Required	Not Required	Normal manual OSM user
OSM Web Service API	Required	Required	The cartridge code generates an OSM **create order** request or other OSM Web Service payload.
OSM XML API	Required	Required if API access is to another instance of OSM	Normal manual OSM user
OSM Automation	Required	Not Required	OSM Automation Plugin Run as user
OSM REST API	Required	Required if API access is to another instance of OSM	REST API User
External Systems (Web Services, APIs and so on)	Not Required	Required	The cartridge code generates a request for external systems that require authentication.

Creating Credential Store Secret

In a traditional deployment, OSM uses the Fusion Middleware Credential Store framework and provides tooling for creating and populating the credential store through the XMLIE's "credStoreAdmin" operation. OSM cloud native uses Kubernetes Secrets as the credential store and the OSM cloud native toolkit provides sample scripts that create credential store secrets and populate them with the required credentials.

Note:

If you use custom code that relies on the OPSS Keystore Service, you need to make changes for OSM cloud native as that mechanism is no longer supported. For details, see "Differences Between OSM Cloud Native and OSM Traditional Deployments".

A text file is used to describe the details required to provision the user accounts properly. Each user is captured in one line and has the following format:

```
map_name:key_name:username:credential-system[:osm-groups]
```

$OSM_CNTK/samples/credentials/osm_users.txt is used to define OSM human users for external LDAP but can be used as a template for other user credentials that need to be created.

Copy this file to your private specification repository under the instance specific directory and rename it to something meaningful. For example, rename the file as repo/*cartridge_user_text_file*.txt.

The **mapName** parameter is a mandatory parameter. If <credential-system> contains "secret", then this value is used as the prefix of the secret name to be created.

Note:

If only LDAP is required, use "osm" for the secret prefix. This value is not used anywhere, but enables the sample to extract the remaining data properly.

The choice of map name and key name affects which OSM automation framework API can be used to retrieve the value within the automation plugin:

- Use "osm" as map name and _sysgen_ as key name. The credential record is accessed with the `context:getOsmCredentialPassword` API.
- Any other map name and key name needs access with the `context:getCredentialAsXML` or `context:getCredential` APIs. Refer to the OSM SDK for more details.

The **credential-system** parameter is a mandatory parameter and must be at least one of the following values:

- **ldap**: Creates the OSM human user against the external LDAP server.

  Note:

  The cartridge automation user account should be created in embedded LDAP by specifying the list of usernames in cartridgeUsers in *projectName*.yaml. Do not create them in external LDAP.

- **secret**: Creates the human user or automation user against the Kubernetes Secret.

Note:

Use a comma to separate the values if creation in both the systems is required.

The **osm-groups** parameter represents a list of OSM groups to associate the user to either the embedded or external LDAP server.

The valid values for the *osm-groups* parameter are:

- OMS_client

- OMS_designer
- OMS_user_assigner
- OMS_workgroup_manager
- OMS_xml_api
- OMS_ws_api
- OMS_ws_diag
- OMS_log_manager
- OMS_cache_manager
- Cartridge_Management_WebService
- OSM_automation
- osmEntityClientGroup
- osmRestApiGroup

Refer to *OSM System Administrator's Guide* for details on OSM user group mapping.

The following text shows a sample user information text file:

```
Line 1
osm:_sysgen_:osmfallout:ldap,secret:OMS_client,OMS_xml_api,OSM_automatio
n,OMS_ws_api
Line 2 osm:_sysgen_:webuser:ldap,secret:OMS_client
Line 3 uim:_sysgen_:uimuser:secret
Line 4 uim:_sysgen_:uimadmin:secret
Line 5
osm:_sysgen_:osmlf:secret:OMS_client,OMS_xml_api,OSM_automation,OMS_ws_a
pi
Line 6 # Guidelines
Line 7 mapName:keyName:userName:credentialSystem:OsmGroup
```

Note:

The secret contains username, password, and the groups.

In the above example:

- Line 1 creates a user "osmfallout" in OpenLDAP and associates that user against the groups listed.
- Line 2 creates a user "webuser" in OpenLDAP and associates this user to the OSM_client group.
- Line 3 and 4 create users "uimadmin" and "uimuser" in the "uim" credential secret.
- Line 5 creates user "osmlf" in the "osm" credential secret.

The secrets that the **manage-cartridge-credentials.sh** script creates are named *project-instance*`-osmcn-cred-`*mapName* as per the naming conventions required by OSM. For each unique mapName that you provide, the script creates one secret. This means if five user entries exist for "uim", each entry will be available in a single secret named *project-instance*`-osmcn-cred-uim`. The script prompts for passwords interactively.

To create the credential store secret:

1. Run the following script:

   ```
   $OSM_CNTK/samples/credentials/manage-cartridge-credentials.sh \
   -p project \
   -i instance \
   -c create \
   -f fileRepo/customSolution_users.txt


   # You will see the following output
   secret/project-instance-osmcn-cred-uim created
   ```

2. Validate that the secrets are created:

   ```
   kubectl get secret -n project

   NAME
   project-instance-osmcn-cred-uim
   ```

Creating Cartridge User Accounts in Embedded LDAP

To create accounts for cartridge users in embedded LDAP, under the `cartridgeUsers` section in *project*.**yaml**, add all the cartridge users (only those from the prefix/map name **osm**). During the creation of the OSM server instance, for all the cartridge users listed, an account is created in embedded LDAP with the same username and password and groups as the Kubernetes secret.

```
cartridgeUsers:
  - osm
  - osmoe
  - osmde
  - osmfallout
  - osmoelf
  - osmlfaop
  - osmlf
  - tomadmin
```

Declaring the Secret

After the secret is created, declare the secret used by the credential store mechanism by editing your project specification. In the project specification, specify only *mapName*. The prefix *project-instance-osmcn-cred* is derived during the instance creation.

To declare the secrets, edit the project specification:

```
#External Credentials Store
externalCredStore:
 secrets:
```

```
mapNames:
  -mapName
```

The OSM cloud native configuration provides a start-up parameter that allows the OSM core application to determine whether the credentials are held in a WebLogic Credential Store (for traditional deployments) or in a Kubernetes Secret Credential Store (for cloud native) so that the configuration is set for you. Cartridges that rely on accessing these credentials are now enabled for execution.

Configuring Other LDAP Systems

The **manage-cartridge-credentials.sh** script supports the OpenLDAP system. To provide support for a different LDAP provider, you must modify the script. Also, the corresponding LDAP client or the API must be installed on the system where the script is executed.

You must modify the following functions within this script:

- `create_ldap_account`. This function creates the user account in the LDAP system and associates the user to the specified groups.
- `update_ldap_account`. This function updates the user password.
- `delete_ldap_account`. This function deletes the user from the LDAP system and disassociates this user from the specified group.
- `verify_ldap_account`. This function verifies that the specified user exists in the LDAP server.

For details on developing the functions, see the developer's guide of the target LDAP server that you want to use.

7 Extending the WebLogic Server Deploy Tooling (WDT) Model

While the OSM cloud native toolkit provides a domain model that is sufficient to support the operation of the OSM application, there are a few aspects that you can customize to meet your business requirements. This chapter provides the general mechanism that OSM cloud native provides for how custom WebLogic Server Deploy Tooling (WDT) metadata can be used.

The following sections enable you to familiarize yourself with the basic extension mechanism. For details on using the sample scripts to add custom WDT metadata, see "Using the Sample Scripts to Extend the WDT Model".

About the Custom WDT Extension Mechanism

The OSM cloud native toolkit exposes an extension mechanism to extend the base WDT domain configuration. For better management practices, you must specify different WDT model fragments in multiple **.tpl** files that can be included in instances as necessary.

All extensions must be located in your source control repository in a directory referred to as *customExtPath*, which is provided during instance creation. This does not need to be the same location as *specPath* that contains the specification files. See the illustration about the directory structure in "Managing Configuration as Code".

Using the WDT Model Tools

This section describes the WDT model tools that you can use when extending the WDT model.

The WDT model tools are available at: https://github.com/oracle/weblogic-deploy-tooling. The documentation available on GitHub describes various tools, which are included in the OSM cloud native toolkit.

For a developer trying to modify or extend the WDT model for a custom OSM instance, the following tools are most useful:

- WDT Discover Domain
- WDT Validate Model

WDT Discover Domain Tool

One way to generate the desired custom model is to manually create a WLS domain (using legacy installers, wlst scripts, console UI changes, and so on) that contains all the constructs that are required and is known to work, in terms of the custom use case. The WDT Discover Domain tool can be pointed at this WLS domain to generate a set of model files. These can be scanned and pruned to get the portions that are of custom interest. They can further be parameterized using WDT's properties files or using Helm values.

If WDT properties are used to parameterize, ensure that you add that properties file to the extension point in the custom implementation.

If Helm values are used to parameterize, ensure that you add these values to the appropriate location - project/instance/shape yamls.

To discover a domain, run the following commands on the prepared WLS admin server or standalone server:

```
# ensure ORACLE_HOME is properly set
cd $ORACLE_HOME
mkdir wdt && cd wdt
wget https://github.com/oracle/weblogic-deploy-tooling/releases/
download/weblogic-deploy-tooling-1.6.0/weblogic-deploy.zip
# Replace 1.6.0 with the actual WDT version as per OSM documentation
unzip weblogic-deploy.zip
cd weblogic-deploy/bin
./discoverDomain.sh -oracle_home $ORACLE_HOME \
 -domain_home domain-home \
 -archive_file archive \
 -model_file model \
 -domain_type domain-type \
 -admin_user admin-user \
 -admin_url t3-admin-url
```

where:

- *archive* and *model* are the directory+name of the files that the discovery tool creates. The model file is of primary importance in this situation.
- *domain-type* is JRF for OSM applications

The command extracts the model from the running WLS instance. Alternatively, if it is sufficient to extract the model from the domain configuration files, the `admin_user` and `admin_url` parameters can be left out.

WDT Validate Model Tool

This tool is useful in the following scenarios:

- When there is a need to see what attributes and sub-fields are available for a model element
- When there is a need to see if a model fragment is valid

Trying to test a newly written or even a modified model file by incorporating it into an instance creation is cumbersome and often an inefficient way to test your changes. You need to check the Introspector logs to see the details of any errors.

With the Validate Model tool, it is easier to validate the model file, especially if you are building the model iteratively.

Common WDT Extension Mechanism

This section describes the extension mechanism that is generic and common to all methods of extending WDT metadata.

Enabling the Extension Mechanism

To enable the extension mechanism:

1. Copy **$OSM_CNTK/samples/_custom-domain-model.tpl** to your source control repository *customExtPath*. This file is a single location where other template files, which store specific WDT metadata fragments, can be included for an OSM instance. This sets up the WDT fragments for re-use across a project, while allowing conditional inclusion based on instance level values in the specification files.
2. Enable the extension mechanism by setting the **custom** flag to `true` in the project specification and including `_custom-domain-model.tpl`:

   ```
   custom:
     enabled: true
     #wdtFiles: {}
     wdtFiles:
      - _custom-domain-model.tpl
   ```

   The basic extension mechanism is now enabled.

For each WDT fragment that is destined for inclusion, perform the following additional steps:

- Provide the WDT fragment
- (Optional) Parameterize the WDT Fragment
- Load the WDT Fragment
- List the .tpl files
- Debug the changes in the Helm chart

Providing the WDT Fragment

Naming convention dictates that the template files start with an underscore _. For example, `_custom-extension-support.tpl`.

You can copy any one of the WDT fragments provided in the samples, or you can create your own. If you provide your own WDT fragment, then you will need to reverse engineer the required metadata using the WDT tooling. For these samples, see "Using the WDT Model Tools".

If you create your own .tpl file, ensure that the WDT fragment is enclosed in a `define` block as follows:

```
{{- define "osm-domain.custom-extension-support" -}}
custom model fragment goes here
{{- end }}
```

(Optional) Parameterizing the WDT Fragment

Instead of hard coding the values into the WDT, you can parameterize the content so that specific values can be driven from the Helm chart. Determine which values fall into this category and then apply the following changes:

To parameterize the WDT fragment:

1. Update the WDT to use a parameter as illustrated in the following example:

   ```
   Host: 'external.provider.hostname'

   becomes....

   Host: '{{ .Values.custom.extension.host }}'
   ```

2. Add values to the application instance in either the project specification or the instance specification found in the source control at *spec_Path*.

   ```
   custom:
    enabled: true
    <extension>:
      host: provide_explicit_value_here
   ```

The custom area of the specification file is where you can add as much content as needed for your extension use cases. Oracle recommends that you keep the yaml structure as flat as possible.

Loading the WDT Fragment

The sample `_custom-domain-model.tpl` already has conditional inclusions for some of the samples provided in the toolkit. JMS, JDBC, and custom application archives can be enabled by providing the appropriate flag in the instance specification and including the specific **.tpl** file in the project specification. For the samples, you do this task as described in "Using the Sample Scripts to Extend the WDT Model".

Load the model fragment into *extension_Directory*/_custom-domain-model.tpl as follows:

```
{{- define "osm-domain.custom-domain-model" -}}
{{- $root := . }}
custom-<extension>-support.<index>.yaml: |+
 {{- include "osm-domain.custom-extension-support" $root | nindent 2 }}
{{- end }}
```

Note:

See the yaml naming convention that is specified by **wdt - filename.yaml**. The index used determines the loading order when there are multiple yaml files. Indexes below 70 are reserved for internal Oracle use.

The WDT may only need to be used conditionally. It is important to be able to exclude the fragment based on the values provided in the project specification. In this case, `_custom-domain-model.tpl` should include the condition that needs to be met for the WDT to be included.

> **Note:**
>
> Including the WDT in *extension_Directory*, which makes it available during instance creation, but not used does not pose any problems for Helm.

```
{{- define "osm-domain.custom-domain-model" -}}
{{- $root := . }}
{{- if .Values.custom.<extension>.enabled }}
custom-extension-support.index.yaml: |+
 {{- include "osm-domain.custom-extension-support" $root | nindent 2 }}
{{- end }}
{{- end }}
```

Listing the TPL Files in the Project

For each WDT fragment that is created in a .tpl file, it needs to be listed in the project specification.

```
custom:
  enabled: true
  #wdtFiles: {}
  wdtFiles:
   - _custom-domain-model.tpl
   - new_wdt.tpl
```

Debugging Helm Chart Changes

When making changes to existing yaml files or creating new WDT fragments, it is useful to test the changes before attempting to create an instance.

You can use the following scripts provided with the toolkit to debug Helm chart changes:

- **$OSM_CNTK/scripts/lint-osm-instance-chart.sh**
- **$OSM_CNTK/scripts/create-instance-dry-run.sh**

You can now create an OSM instance.

Using the Sample Scripts to Extend the WDT Model

This section provides instructions for extending the WDT model by using the sample scripts that are provided with the toolkit. You add custom WDT metadata to create your own OSM instances.

The toolkit includes sample scripts for the following:

- Adding a JDBC Datasource
- Adding a JMS System Resource
- Deploying a Custom Application ear to an OSM WebLogic domain
- Extending the WDT Metadata for an External Authenticator

The general and common extension process described in "Common WDT Extension Mechanism" must be repeated for each of the use cases described in this section.

Adding a JDBC Datasource

The WDT fragment describing a **JDBCSystemResource** is provided in the **$CNTK/samples/customExtension/_custom-jdbc-support.tpl** sample file.

To incorporate this fragment into your OSM instance:

1. Enable the extension mechanism by setting the `custom` flag to **true** and add the **custom-domain-model** to the list of included wdtFiles in the project specification:

   ```
   custom:
     enabled: true
     wdtFiles:
      - _custom-domain-model.tpl
   ```

2. Provide the WDT fragment by copying **$CNTK/samples/customExtensions/_custom-jdbc-support.tpl** to the *customExtPath* in your source control repository.
3. Parameterize the WDT fragment. The fragment has already been parameterized and uses values specified in the shape file. You must update the remaining values enclosed in angular brackets. By default, this WDT reads the JDBC values from the shape that is provided during instance creation.

   > **Note:**
   >
   > Kubernetes Secrets can also be used to provide sensitive data such as username and password. See "Accessing Kubernetes Secrets from WDT Metadata" for details.

   ```
   resources:
    JDBCSystemResource:
    '<custom-conn-pool>':
    JdbcResource:
    JDBCDriverParams:
    URL: 'jdbc:oracle:thin:@<db-host>:<db-port>/<db-service>'
    PasswordEncrypted: '<password>'
    #PasswordEncrypted: '@@SECRET:my_secret_name:my_db_password@@'
    Properties:
    user:
    Value: '<user>'
    #Value: '@@SECRET:my_secret_name:my_db_user@@'
    oracle.net.CONNECT_TIMEOUT:
    Value: {{ default "10000" .Values.jdbc.oracleNetConnectTimeout }}
    oracle.jdbc.ReadTimeout:
    Value: {{ default "3660000" .Values.jdbc.oracleJdbcReadTimeout }}
    JDBCConnectionPoolParams:
    InitialCapacity: {{ default "0" .Values.jdbc.initialCapacity }}
    MaxCapacity: {{ default "15" .Values.jdbc.maxCapacity }}
    MinCapacity: {{ default "0" .Values.jdbc.minCapacity }}
    ShrinkFrequencySeconds: {{ default
   ```

```
"900" .Values.jdbc.shrinkFrequencySeconds }}
 TestFrequencySeconds: {{ default
"300" .Values.jdbc.testFrequencySeconds }}
 TestConnectionsOnReserve: {{ default
"true" .Values.jdbc.testConnectionsOnReserve }}
 SecondsToTrustAnIdlePoolConnection: {{ default
"10" .Values.jdbc.secondsToTrustAnIdlePoolConnection }}
 StatementCacheSize: {{ default
"30" .Values.jdbc.statementCacheSize }}
 ConnectionCreationRetryFrequencySeconds: {{ default
"30" .Values.jdbc.connectionCreationRetryFrequencySeconds }}
 IgnoreInUseConnectionsEnabled: {{ default
"true" .Values.jdbc.ignoreInUseConnectionsEnabled }}
 InactiveConnectionTimeoutSeconds: {{ default
"0" .Values.jdbc.inactiveConnectionTimeoutSeconds }}
 StatementCacheType: '{{ default
"LRU" .Values.jdbc.statementCacheType }}'
 CountOfTestFailuresTillFlush: {{ default
"5" .Values.jdbc.countOfTestFailuresTillFlush }}
 CountOfRefreshFailuresTillDisable: {{ default
"5" .Values.jdbc.countOfRefreshFailuresTillDisable }}
 RemoveInfectedConnections: {{ default
"false" .Values.jdbc.removeInfectedConnections }}
 ConnectionReserveTimeoutSeconds: {{ default
"10" .Values.jdbc.connectionReserveTimeoutSeconds }}
 StatementTimeout: {{ default
"3630" .Values.jdbc.statementTimeout }}
```

4. The fragment is already configured for conditional loading based on the presence of the `jdbc` flag in the project specification. Set the `jdbc` flag to `true`.

   ```
   custom:
     enabled: true
     jdbc: true
   ```

5. Add the JDBC .tpl file to the project specification:

   ```
   custom:
     enabled: true
     jdbc: true
     wdtFiles:
      - _custom-domain-model.tpl
      - _custom-jdbc-support.tpl
   ```

   You can now create the OSM instance.

Adding a JMS System Resource

The WDT fragment describing a JMS System Resource is provided in the **$CNTK/samples/customExtension/_custom-jms-support.tpl** sample file.

To incorporate this fragment into your OSM instance:

1. Enable the extension mechanism by setting the `custom` flag to **true** and add the **custom-domain-model** to the list of included wdtFiles in the project specification:

   ```
   custom:
     enabled: true
     wdtFiles:
      - _custom-domain-model.tpl
   ```

2. Provide the WDT fragment by copying **$CNTK/samples/customExtensions/_custom-jms-support.tpl** to the *customExtPath* in your source control repository. While this sample shows WDT for a JMS Queue and JMS Topic, any other JMS entity can be supplied instead. See "Using the WDT Model Tools" for details on establishing the correct WDT.
3. Parameterize the WDT fragment. The fragment has not been parameterized. The text enclosed in angular brackets must be replaced with specific values. Alternatively, update the WDT to parameterize content and provide actual values in the project specification.
4. The fragment is already configured for conditional loading based on the presence of the `jms` flag in the project specification. See the **$CNTK/charts/osm/templates/_custom-domain-model.tpl** template. Set the `jms` flag to `true`.

   ```
   custom:
     enabled: true
     jms: true
   ```

5. Add the jms tpl file to the project specification:

   ```
   custom:
     enabled: true
     jms: true
     wdtFiles:
      - _custom-domain-model.tpl
      - _custom-jms-support.tpl
   ```

   You can now create the OSM instance.

Deploying Entities to an OSM WebLogic Domain

You can deploy any WebLogic Server deployable entity, such as an application EAR or WAR to an OSM WebLogic domain.

To deploy an entity to an OSM WebLogic Domain:

1. Package the entity, for example, the application ear into an archive file and place it inside the container image used for creating OSM instances.

> **Note:**
>
> The WebLogic domain tooling expects application binaries to be available at the correct path within the archive. A script is provided for your convenience that packages the application into the correct path.

```
cp application.ear samples/customExtensions
cd samples/customExtensions
./make-custom-archive.sh archive_file_name.zip application.ear
```

2. Build a new container image:

   ```
   cd samples/customExtensions
   docker build -t "image_name:tag" --
   build-arg base_image=osm_base_image --build-arg
   archive=archive_file_name.zip .
   ```

3. Upload the generated image to your private Docker repository.
4. Add the domain configuration.
   In addition to copying the archive file into the base image, you must supply custom configuration, which can be passed in by any one of following two mechanisms:
   - Inside the container image.
     This mechanism keeps the ear file together with the domain configuration in one location. This is best suited to applications that can be considered standard or fixed for all variants of a domain that are required (test, development, and production).

     Advantage: You do not need to add the custom domain configuration every time you create a domain.

     Disadvantage: If you want to change the configuration, it requires a change to the base image. In domains that are already up, an image change triggers a full restart of the domain.

     To add the domain configuration using this mechanism:

     a. Save your fragment in a YAML file that includes an index 70 or above. For example, **custom-application-extension.70.yaml**.

     b. Edit **Dockerfile** to copy the YAML file to the **u01/wdt/models** directory along with the archive.
   - Using the extension mechanism.
     This approach allows for per instance control over the application. This is best suited to situations where the application configuration needs to be dictated by the specific domain instance (for example, test vs. production).

     Advantage: Keeps all "variable" (per instance) configuration in one place at domain creation.

     Disadvantage: Domain creation for every instance that uses the application must remember to add the configuration.

     To use the extension mechanism:

a. Enable the extension mechanism by setting the `custom` flag to **true** and add the **custom-domain-model** to the list of included wdtFiles in the project specification:

```
custom:
  enabled: true
  wdtFiles:
   - _custom-domain-model.tpl
```

b. Provide the WDT fragment by copying **$CNTK/samples/customExtensions/_custom-application-support.tpl** to the *customExtPath* in your source control repository.

c. Parameterize the WDT fragment. The fragment has already been parameterized.

```
appDeployments:
 Application:
 {{- .Values.custom.application_name }}:
 SourcePath: 'wlsdeploy/
applications/{{- .Values.custom.binary_name }}.ear'
 ModuleType: ear
 StagingMode: nostage
 PlanStagingMode: nostage
 Target: '@@PROP:CLUSTER_NAME@@'
```

d. Provide the values in the instance specification:

```
custom:
  enabled: true
  application: true
  #additional values here
  application_name: myApplication
  binary_name: myApp
```

e. Add the `application` flag and set it to `true`. The fragment is already configured for conditional loading based on the presence of the `application` flag in the project specification. See **$CNTK/charts/osm/templates/_custom-domain-model.tpl** in the toolkit.

```
custom:
  enabled: true
  application: true
```

f. Add the application tpl file to the project specification:

```
custom:
  enabled: true
  application: true
  wdtFiles:
   - _custom-domain-model.tpl
   - _custom-application-support.tpl
```

You can now create the OSM instance.

Extending the WDT Metadata for an External Authenticator

The OSM cloud native toolkit provides out-of-the-box configuration for a WebLogic domain using OpenLDAP as the authenticator. Using a different provider (even a different LDAP provider) requires different WDT metadata, which is a significant undertaking. The configuration required to support an alternate WLS provider would need to be investigated and developed independently using an existing WebLogic domain. Oracle's WDT Discover Domain Tool can analyze an existing domain and generate the corresponding WDT model. The WDT model fragment can then be used to configure the OSM domain using the toolkit extension mechanism.

See the following documentation for information on configuring a WebLogic domain with alternative authentication providers:

- Configuring WebLogic to use LDAP
- Configuring Active Directory (AD) as an Authentication Provider in WebLogic

After the WDT is determined, it is provided during the creation process in the same way as other WDT metadata fragments. This section describes the process for setting up external authentication for OSM cloud native.

To set up external authentication:

1. Disable OpenLDAP by editing the project specification in *customExPath*:

```
authentication:
  openldap:
    enabled: false
```

2. Copy **$OSM_CNTK/samples/_custom-domain-model.tpl** to your source control repository at *customExtPath*.
3. Enable the extension mechanism by setting the `custom` flag to `true` in the project specification and including the `_custom-domain-model.tpl`

```
custom:
  enabled: true
  wdtFiles:
   - _custom-domain-model.tpl
```

4. Determine and provide the WDT model fragment for the security provider in the WebLogic domain. Once you know the WDT fragment that needs to be supplied, save it into a file in your source control repository at the *customExtPath* (_custom-*provider*-support.tpl).

```
{{- define "osm.custom-provider-support" -}}
topology:
 SecurityConfiguration:
 Realm:
 myrealm:
 AuthenticationProvider:
 '!DefaultAuthenticator':
 '!DefaultIdentityAsserter':
 YouLDAPProviderStartHere:
 .... <specific details here>
 DefaultAuthenticator:
 DefaultAuthenticator:
 ControlFlag: SUFFICIENT
 UseRetrievedUserNameAsPrincipal: true
 DefaultIdentityAsserter:
 DefaultIdentityAsserter:
{{- end }}
```

Note:

You can review the fragment for an OpenLDAP provider that is included in the toolkit: **$OSM_CNTK/charts/osm/templates/_osm-openldap-support.tpl**

The security configuration WDT should respect sensitive data by using secrets. See "Accessing Kubernetes Secrets from WDT Metadata" for details on how to access secret data from within your WDT fragment.

5. (Optional) Update any parameters that should not be hard coded in the WDT fragment. Add these values to the project specification under the "custom" section.
6. Load the model fragment by editing your *custom_extension_path***/ _custom-domain-model.tpl** file:

```
{{- define "osm.custom-domain-model" -}}
{{- $root := . }}
custom-provider-support.index.yaml: |+
  {{- include "osm.custom-provider-support" $root | nindent 2 }}
{{- end }}


## If you would like conditional inclusion of the
fragment...something like this instead


{{- define "osm.custom-domain-model" -}}
{{- $root := . }}
{{- if .Values.custom.provider.flag}}
custom-provider-support.index.yaml: |+
  {{- include "osm.custom-<provider>-support" $root | nindent 2 }}
{{- end }}
{{- end }}
```

Note:

Remember the yaml naming convention that is specified by wdt - filename.yaml. The index used determines the loading order when there are multiple yaml files. Indexes below 70 are reserved for internal Oracle use.

7. Add the tpl file that has the authentication provider WDT into the project specification:

```
custom:
  enabled: true
  wdtFiles:
   - _custom-domain-model.tpl
   - _custom-provider-support.tpl
```

   You can now create an OSM instance.

Accessing Kubernetes Secrets from WDT Metadata

The process of handling sensitive data inside a WDT fragment involves the following:

- Creating Kubernetes secrets
- Declaring the secrets in the specification file
- Referencing the secrets from the WDT fragment

To access Kubernetes secrets from WDT metadata:

1. Create the secret.
   Secrets must be created in the correct Kubernetes namespace. The namespace is already created when registering the namespace and aligns to your project name.

   To create the secret using the command line, run the following command:

```
$kubectl -n project_Name create secret generic secret_Name \
 --from-literal=key1=$value \
 --from-literal=key2=$value
```

2. Add the secret in the `custom` section of the instance specification in your source repository:

```
# Custom secrets
# replace the empty secret names with one or more secrets
instance:
  customSecrets:
    enabled: true
    secretNames:
     - mysecret1
     - mysecret2
```

   Once you have created and declared your custom secrets, they can be referenced from elsewhere in the WDT model.

3. Access the secret from inside a WDT fragment:

```
Field1: '@@SECRET:secret_name:key1@@'
Field2: '@@SECRET:secret_name:key2@@'
```

   where *secret_name* represents the secret name and *key* represents one of the keys in the secret.

Troubleshooting WDT Issues

This section provides details about some procedures that you may have to run in order to resolve issues with WDT.

Starting and Terminating a WDT Pod

The OSM image includes the WDT tools that are often needed to debug or discover a WDT fragment. You can start a temporary pod that provides access to these tools. Before starting the pod, download the container image of the OSM base image to ensure that the download time does not exceed the duration of the Kubernetes pod creation timeout.

```
kubectl run wdt --generator=run-pod/v1 \
 --image OSM_base_image -- sleep infinity
```

When the pod is no longer needed, you can delete it:

```
kubectl delete pod wdt
```

Validating a Model YAML File

To validate a model YAML file:

1. Copy a model yaml into your temporary pod:

   ```
   kubectl cp model_file wdt:/tmp/model_file
   ```

2. Run the following command and wait for the prompt:

   ```
   kubectl exec -ti wdt /bin/bash
   ```

3. Validate the model file you copied:

   ```
   cd /u01/wdt/weblogic-deploy/bin
   ./validateModel.sh -oracle_home $ORACLE_HOME -model_file /tmp/
   model_file
   ```

4. When you are done validating, exit the pod:

   ```
   exit
   ```

The line numbers returned by the **validateModel** script are exclusive of the comment lines. Either strip the comments first or do the calculation to get the "real" line number in the file.

This process can be iterated by first reviewing the WDT errors and warnings, fixing the YAML file, and then re-running the above procedure. Repeat this as required.

Note:

Model files can contain fragments of models, but each model element must have its full parentage, starting from `section`. For example, following is the sample if the fragment is the model element **JmsResource**:

```
resources:
 JMSSystemResource:
 JmsResource:
 model-fragment-to-validate
```

Displaying Valid Attributes and Child Attributes of a WDT Model

To display the attributes of a WDT model, run the following commands:

```
kubectl exec -ti wdt /bin/bash
# wait for prompt
cd /u01/wdt/weblogic-deploy/bin
./validateModel.sh -oracle_home $ORACLE_HOME \
 -print-usage path
exit
```

The *path* here is the WDT path to the model element of interest. For example, to see all the attributes and child attributes for SAFImportedDestinations, the path is `resources:/JMSSystemResource/JmsResource/SAFImportedDestinations`.

A common way to construct the path is to look for the element in a discovered model file and determine its yaml path. Another way is to start off with a path of `section:`, where `section` is one of "domainInfo", "topology", "resources" or "appDeployments". By iteratively discovering the child attributes, the final path can be built-up.

To shorten this search process, add the `-recursive` flag to the validateModel.sh script command line. Care should be taken as the output can be quite large at the higher levels.

8

Exploring Alternate Configuration Options

The OSM cloud native toolkit provides samples and documentation for setting up your OSM cloud native environment using standard configuration options. However, you can choose to explore alternate configuration options for setting up your environment, based on your requirements. This chapter describes alternate configurations you can explore, allowing you to decide how best to configure your OSM cloud native environment to suit your needs.

You can choose alternate configuration options for the following:

- Setting Up Authentication
- Working with Shapes
- Injecting Custom Configuration Files
- Choosing Worker Nodes for Running OSM Cloud Native
- Working with Ingress, Ingress Controller, and External Load Balancer
- Using an Alternate Ingress Controller
- Reusing the Database State
- Setting Up Persistent Storage
- Setting Up Database Optimizer Statistics
- Leveraging Oracle WebLogic Server Active GridLink
- Managing Logs
- Managing OSM Cloud Native Metrics

The sections that follow provide instructions for working with these configuration options.

Setting Up Authentication

By default, OSM uses the WebLogic embedded LDAP as the authentication provider and all OSM system users are created in embedded LDAP during instance creation. For human users, you may set up an optional authentication for the users who access OSM through user interfaces. See "Planning and Validating Your Cloud Environment" for information on the components that are required for setting up your cloud environment. The OSM cloud native toolkit provides samples that you use to integrate components such as OpenLDAP, WebLogic Kubernetes Operator (WKO), and Traefik. This section describes the tasks you must do for configuring optional authentication for OSM cloud native human users.

Perform the following tasks using the samples provided with the OSM cloud native toolkit:

- Install and configure OpenLDAP. This is required to be done once for your organization.

- Install OpenLDAP clients. This is required to be performed on each host that installs and runs the toolkit scripts and when a Kubernetes cluster is shared by multiple hosts.
- In the OpenLDAP server, create the root node for each OSM instance

Installing and Configuring OpenLDAP

OpenLDAP enables your organization to handle authentication for all instances of OSM. You install and configure OpenLDAP once for your organization.

To install and configure OpenLDAP:

1. Run the following command, which installs OpenLDAP:

   ```
   $ sudo -s  yum -y install "openldap" "migrationtools"
   ```

2. Specify a password by running the following command:

   ```
   $ sudo -s slappasswd
   New password:
   Re-enter new password:
   ```

3. Configure OpenLDAP by running the following commands:

   ```
   $ sudo -s
   $ cd /etc/openldap/slapd.d/cn=config
   $ vi olcDatabase\=\{2\}hdb.ldif
   ```

4. Update the values for the following parameters:

   Note:

   Ignore the warning about editing the file manually.

   - `olcSuffix: dc=osmcn-ldap,dc=com`
   - `olcRootDN: cn=Manager,dc=osmcn-ldap,dc=com`
   - `olcRootPW:`*ssha*
     where *ssha* is the SSHA that is generated

5. Update the **dc** values for the **olcAccess** parameter as follows:

   ```
   olcAccess: {0}to * by
   dn.base="gidNumber=0+uidNumber=0,cn=peercred,cn=external, cn=auth"
   read by dn.base="cn=Manager,dc=osmcn-ldap,dc=com" read by * none
   ```

6. Test the configuration by running the following command:
   `sudo -s slaptest -u`

   Ignore the checksum warnings in the output and ensure that you get a success message at the end.

7. Run the following commands, which restart and enable LDAP:

   ```
   sudo -s systemctl restart slapd
   sudo -s systemctl enable slapd
   sudo -s cp -rf /usr/share/openldap-servers/
   DB_CONFIG.example /var/lib/ldap/DB_CONFIG
   ldapadd -Y EXTERNAL -H ldapi:/// -f /etc/openldap/schema/cosine.ldif
   ldapadd -Y EXTERNAL -H ldapi:/// -f /etc/openldap/schema/nis.ldif
   ldapadd -Y EXTERNAL -H ldapi:/// -f /etc/openldap/schema/
   inetorgperson.ldif
   ```

8. Create a root node named **domain**, which will be the top parent for all OSM instances.
9. Run the following command to create a new file named **base.ldif**:

   ```
   sudo -s  vi /root/base.ldif
   ```

10. Add the following entries to the **base.ldif** file:

    ```
    dn: ou=Domains,dc=osmcn-ldap,dc=com
    objectClass: top
    objectClass: organizationalUnit
    ou: Domains
    ```

11. Run the following commands to update the values in the **base.ldif** file:

    ```
    ldapadd -x -W -D "cn=Manager,dc=osmcn-ldap,dc=com" -f /root/
    base.ldif
    ldapsearch -x cn=Manager -b dc=osmcn-ldap,dc=com
    ```

12. Open the LDAP port 389 on all Kubernetes nodes in the cluster.

Installing OpenLDAP Clients

In environments where the Kubernetes cluster is shared by multiple hosts, you must install the OpenLDAP clients on each host. You use the scripts in the toolkit to populate the LDAP server with users and groups.

On the host on which you want to create a basic OSM instance, run the following command, which installs the OpenLDAP clients:

```
sudo -s yum -y install openldap-clients
```

Creating the Root Node

You must create the root node for each OSM instance before additional OSM non-automation user and OSM group can be created.

The toolkit provides a sample script (**$OSM_CNTK/samples/credentials/manage-osm-ldap-credentials.sh**) that you can use to create the root node in the LDAP tree for the OSM instance.

Run the **$OSM_CNTK/samples/credentials/manage-osm-ldap-credentials.sh** script by passing in **-o account**.

Working with Shapes

The OSM cloud native toolkit provides the following pre-configured shapes:

- **charts/osm/shapes/dev.yaml**. This can be used for development, QA and user acceptance testing (UAT) instances.
- **charts/osm/shapes/devsmall.yaml**. This can be used to reduce CPU requirements for small development instances.
- **charts/osm/shapes/prod.yaml**. This can be used for production, pre-production, and disaster recovery (DR) instances.
- **charts/osm/shapes/prodlarge.yaml**. This can be used for production, pre-production and disaster recovery (DR) instances that require more memory for OSM cartridges and order caches.
- **charts/osm/shapes/prodsmall.yaml**. This can be used to reduce CPU requirements for production, pre-production and disaster recovery (DR) instances. For example, it can be used to deploy a small production cluster with two managed servers when the order rate does not justify two managed servers configured with a **prod** or **prodlarge** shape. For production instances, Oracle recommends two or more managed servers. This provides increased resiliency to a single point of failure and can allow order processing to continue while failed managed servers are being recovered.

You can create custom shapes using the pre-configured shapes. See "Creating Custom Shapes" for details.

The pre-defined shapes come in standard sizes, which enable you to plan your Kubernetes cluster resource requirement.

The following table lists the sizing requirements of the shapes for a managed server:

Table 8-1 Sizing Requirements of Shapes for a Managed Server

Shape	Kube Request	Kube Limit	JVM Heap (GB)
prodlarge	80 GB RAM, 15 CPU	80 GB RAM, 15 CPU	64
prod	48 GB RAM, 15 CPU	48 GB RAM, 15 CPU	31
prodsmall	48 GB RAM, 7.5 CPU	48 GB RAM, 7.5 CPU	31
dev	8 GB RAM, 2 CPU	8 GB RAM	5
devsmall	8 GB RAM, 0.5 CPU	8 GB RAM	5

The following table lists the sizing requirements of the shapes for an admin server:

Table 8-2 Sizing Requirements of Shapes for an Admin Server

Shape	Kube Request	Kube Limit	JVM Heap (GB)
prodlarge	8 GB RAM, 2 CPU	8 GB RAM	4
prod	8 GB RAM, 2 CPU	8 GB RAM	4
prodsmall	8 GB RAM, 2 CPU	8 GB RAM	4
dev	3 GB RAM, 1 CPU	4 GB RAM	1
devsmall	3 GB RAM, 0.5 CPU	4 GB RAM	1

These values are encoded in the specifications and are automatically part of the individual pod configuration. The Kubernetes schedulers evaluate the Kube request settings to find space for each pod in the worker nodes of the Kubernetes cluster.

To plan the cluster capacity requirement, consider the following:

- Number of development instances required to be running in parallel: D
- Number of managed servers expected across all the development instances: Md (Md will be equal to D if all the development instances are 1 MS instances)
- Number of production (and production-like) instances required to be running in parallel: P
- Number of managed servers expected across all production instances: Mp
- Assume use of "dev" and "prod" shapes
- CPU requirement (CPUs) = D * 1 + Md * 2 + P * 2 + Mp * 15
- Memory requirement (GB) = D * 4 + Md * 8 + P * 8 + Mp * 48

Note:

The production managed servers take their memory and CPU in large chunks. Kube scheduler requires the capacity of each pod to be satisfied within a particular worker node and does not schedule the pod if that capacity is fragmented across the worker nodes.

The shapes are pre-tuned for generic development and production environments. You can create an OSM instance with either of these shapes, by specifying the preferred one in the instance specification.

```
# Name of the shape. The OSM cloud native shapes are devsmall, dev,
prodsmall, prod, and prodlarge.
# Alternatively, custom shape name can be specified (as the filename
without the extension)
```

Creating Custom Shapes

You create custom shapes by copying the provided shapes and then specifying the desired tuning parameters. Do not edit the values in the shapes provided with the toolkit.

In addition to processor and memory sizing parameters, a custom shape can be used to tune:

- The number of threads allocated to OSM work managers
- OSM connection pool parameters
- Order cache sizes and inactivity timeouts

For more details on the recommend approach to tune these parameters, see "OSM Pre-Production Testing and Tuning" in *OSM Installation Guide*.

To create a custom shape:

1. Copy one of the pre-configured shapes and save it to your source repository.

2. Rename the shape and update the tuning parameters as required.
3. In the instance specification, specify the name of the shape you copied and renamed:

   ```
   shape: custom
   ```

4. Create the domain, ensuring that the location of your custom shape is included in the comma separated list of directories passed with `-s`.

   ```
   $OSM_CNTK/scripts/create-instance.sh -p project -i instance -s
   spec_Path
   ```

> **Note:**
>
> While copying a pre-configured shape or editing your custom shape, ensure that you preserve any configuration that has comments indicating that it must not be deleted.

Injecting Custom Configuration Files

Sometimes, a solution cartridge may require access to a file on disk. A common example is for reading of property files or mapping rules.

A solution may also need to provide configuration files for reference via parameters in the **oms-config.xml** file for OSM use (for example, for operational order jeopardies and OACC runtime configuration).

To inject custom configuration files:

1. Make a copy of the **OSM_CNTK/samples/customExtensions/custom-file-support.yaml** file.
2. Edit it so that it contains the contents of the files. See the comments in the file for specific instructions.
3. Save it (retaining its name) into the directory where you save all extension files. Say *extension_directory*. See "Extending the WebLogic Server Deploy Tooling (WDT) Model" for details.
4. Edit your project specification to reference the desired files in the `customFiles` element:

   ```
   #customFiles:
   # - mountPath: /some/path/1
   #   configMapSuffix: "path1"
   # - mountPath: /some/other/path/2
   #   configMapSuffix: "path2"
   ```

When you run **create-instance.sh** or **upgrade-instance.sh**, provide the *extension_directory* in the "-m" command-line argument. In your **oms-config.xml** file or in your cartridge code, you can refer to these custom files as *mountPath/filename*, where *mountPath* comes from your project specification and *filename* comes from your **custom-file-support.yaml** contents. For example, if your **custom-file-support.yaml** file contains a file called **properties.txt** and you have a mount path of **/mycompany/mysolution/config**, then you can refer to this file in your cartridge or in the **oms-config.xml** file as **/mycompany/mysolution/config/properties.txt**.

While working with custom configuration files, consider the following usage guidelines:

- The files created are read-only for OSM and for the cartridge code.
- The *mountPath* parameter provided in the project specification should point to a new directory location. If the location is an existing location, all of its existing content will occlude with the files you are injecting.
- Do not provide the same *mountPath* more than once in a project specification.
- The **custom-file-support.yaml** file in your *extension_directory* is part of your configuration-as-code, and must be version controlled as with other extensions and specifications.

To modify the contents of a custom file, update your **custom-file-support.yaml** file in your *extension_directory* and invoke **upgrade-instance.sh**. Changes to the contents of the existing files are immediately visible to the OSM pods. However, you may need to perform additional actions in order for these changes to take effect. For example, if you changed a property value in your custom file, that will only be read the next time your cartridge runs the appropriate logic.

If you wish to add files for a running OSM cloud native instance, update your **custom-file-support.yaml** file as described above and invoke **upgrade-instance.sh**. While this same procedure can work when you need to remove custom files for a running OSM instance, it is strongly recommended that you do this as described in the following procedure to avoid "file not found" type of errors:

1. Update the instance specification to set the size to **0** and then run **upgrade-instance.sh**.
2. Update the instance specification to set the size to the initial value and remove the file from your **custom-file-support.yaml** file.
3. Update the `customFiles` parameter in your project specification and run **upgrade-instance.sh**.

Choosing Worker Nodes for Running OSM Cloud Native

By default, OSM cloud native has its pods scheduled on all worker nodes in the Kubernetes cluster in which it is installed. However, in some situations, you may want to choose a subset of nodes where pods are scheduled.

For example, these situations include:

- Licensing restrictions: Coherence could be limited to be deployed on specific shapes. Also, there could be a limit on the number of CPUs where Coherence is deployed.
- Non license restrictions: Limitation on the deployment of OSM on specific worker nodes per each team for reasons such as capacity management, chargeback, budgetary reasons, and so on.

To choose a subset of nodes where pods are scheduled, you can use the configuration in the project specification yaml file.

```
# If OSM cloud native instances must be targeted to a subset of worker
nodes in the
# Kubernetes cluster, tag those nodes with a label name and value, and
choose
# that label+value here.
#   key    : any node label key
#   values : list of values to choose the node.
#            If any of the values is found for the above label key,
then that
#            node is included in the pod scheduling algorithm.
#
# This can be overriden in instance specification if required.
osmcnTargetNodes: {} # This empty declaration should be removed if
adding items here.
#osmcnTargetNodes:
#  nodeLabel:
##   oracle.com/licensed-for-coherence is just an indicative example,
any label and its values can be used for choosing nodes.
#    key: oracle.com/licensed-for-coherence
#    values:
#      - true
```

Consider the following when you update the configuration:

- There is no restriction on node label key. Any valid node label can be used.
- There can be multiple valid values for a key.
- You can override this configuration in the instance specification yaml file, if required.

Working with Ingress, Ingress Controller, and External Load Balancer

A Kubernetes ingress is responsible for establishing access to back-end services. However, creating an ingress is not sufficient. An Ingress controller connects the back-end services with the front-end services based on Ingress rules. In OSM cloud native, an ingress controller can be configured in the project specification.

```
# valid values are TRAEFIK, GENERIC, OTHER
ingressController: "TRAEFIK"
```

To create an ingress, run the following:

```
$OSM_CNTK/scripts/create-ingress.sh -p project -i instance -s $SPEC_PATH
```

To delete an ingress, run the following:

```
$OSM_CNTK/scripts/delete-ingress.sh -p project -i instance
```

The Traefik ingress controller works by creating an operator in its own "traefik" namespace and exposing a NodePort service. However, all ingress controllers do not behave the same way. In order to accommodate all types of ingress controllers, by default, the instance.yaml file provides the `loadBalancerPort` parameter.

If an external load balancer is used, it needs to be connected to the NodePort service of the Ingress controller. Hence, `externalLoadBalancerIP` also needs to be present in instance.yaml.

For the Traefik ingress controller, do the following:

- If an external load balancer is not configured, fetch `loadBalancerPort` by running the following command:

  ```
  $kubectl -n $TRAEFIK_NS get service traefik-operator --
  output=jsonpath="{..spec.ports[?(@.name=='http')].nodePort}"
  ```

- If an external load balancer is used, fetch `loadBalancerPort` by running the following command:

  ```
  kubectl -n $TRAEFIK_NS get service traefik-operator --
  output=jsonpath="{..spec.ports[?(@.name=='http')].port}"
  ```

Populate the values in **instance.yaml** before invoking **create-instance.sh** command to create an instance:

```
# If external hardware or software load balancer is used, set this
value to that frontend host IP.
# If OCI load balancer is used, then set externalLoadBalancerIP from
OCI LBaaS
#externalLoadBalancerIP: ""

# For Traefik Ingress Controller:
# If external load balancer is used, then this would be 80, else
traefik pod's Nodeport (30305)
loadBalancerPort: 80
```

Note:

If you choose Traefik or any other ingress controller such as Traefik, you can move the `loadBalancerPort` and `externalLoadBalancerIP` parameters to **project.yaml**.

Using an Alternate Ingress Controller

By default, OSM cloud native supports Traefik and provides sample files for integration. However, you can use any Ingress controller that supports host-based routing and session stickiness with cookies. OSM cloud native uses the term "generic" ingress for scenarios where you want to leverage the Ingress capabilities that the Kubernetes platform may provide.

To use a generic ingress controller, you must create the ingress object and configure your OSM instance to use it. The toolkit uses an ingress Helm chart (**$OSM_CNTK/samples/charts/ingress-per-domain/templates/traefik-ingress.yaml**) and scripts for creating the ingress objects. If you want to use a generic ingress controller, these samples can be used as a reference and customized as necessary.

If your OSM cloud native instance needs to secure incoming communications, then look at the **$OSM_CNTK/samples/charts/ingress-per-domain/templates/traefik-ingress.yaml** file. This file demonstrates the configuration for a TLS-enabled Traefik ingress that can be used as a sample.

The host-based rules and the corresponding back-end Kubernetes service mapping are provided using the following definitions:

- domainUID: Combination of *project-instance*. For example, **sr-quick**.
- clusterName: The name of the cluster in lowercase. Replace any hyphens "-" with underscore "_". The default name of the cluster in **values.yaml** is **c1**.

The following table lists the service name and service ports for Ingress rules:

Table 8-3 Service Name and Service Ports for Ingress Rules

Rule	Service Name	Service Port	Purpose
instance.project.loadBalancerDomainName	*domainUID*-cluster-*clusterName*	8001	For access to OSM through UI, XMLAPI, Web Services, and so on.
t3.*instance.project.loadBalancerDomainName*	t3.*instance.project.loadBalancerDomainName*	30303	OSM T3 Channel access for WLST, JMS, and SAF clients.
admin.*instance.project.loadBalancerDomainName*	*domainUID*-admin	7001	For access to OSM WebLogic Admin Console UI.

Ingresses need to be created for each of the above rules per the following guidelines:

- Before running **create-instance.sh**, ingress must be created.
- After running **delete-instance.sh**, ingress must be deleted.

You can develop your own code to handle your ingress controller or copy the sample `ingress-per-domain` chart and add additional template files for your ingress controller with a new value for the type (NGINX for example).

- The reference sample for creation is: **$OSM_CNTK/scripts/config-ingress.sh**
- The reference sample for deletion is: **$OSM_CNTK/scripts/delete-ingress.sh**

You must update the value of the `ingressController` parameter in the instance specification at **$SPEC_PATH/***project-instance*.yaml

```
#valid values are TRAEFIK, GENERIC, OTHER
ingressController: "GENERIC"
```

If any of the supported Ingress controllers or even a generic ingress does not meet your requirements, you can choose "OTHER".

By choosing this option, OSM cloud native does not create or manage any ingress required for accessing the OSM cloud native services. However, you may choose to create your own ingress objects based on the service and port details mentioned in the above table.

Note:

Regardless of the choice of Ingress controller, it is mandatory to provide the value of `loadBalancerPort` in one of the specification files. This is used for establishing front-end cluster.

Reusing the Database State

When an OSM instance is deleted, the state of the database remains unaffected, which makes it available for re-use. This is common in the following scenarios:

- When an instance is deleted and the same instance is re-created using the same project and the instance names, the database state is unaffected. For example, consider a performance instance that does not need to be up and running all the time, consuming resources. When it is no longer actively being used, its specification files and PDB can be saved and the instance can be deleted. When it is needed again, the instance can be rebuilt using the saved specifications and the saved PDB. Another common scenario is when developers delete and re-create the same instance multiple times while configuration is being developed and tested.
- When a new instance is created to point to the data of another instance with a new project and instance names, the database state is unaffected. A developer, who might want to create a development instance with the data from a test instance in order to investigate a reported issue, is likely to use their own instance specification and the OSM data from PDB of the test instance.

Additionally, consider the following components when re-using the database state:

- The OSM DB (schema and data)
- The RCU DB (schema and data)

Recreating an Instance

You can re-create an OSM instance with the same project and instance names, pointing to the same database. In this case, both the OSM DB and the RCU DB are re-used, making the sequence of events for instance re-creation relatively straightforward.

To recreate an instance, the following pre-requisites must be available from the original instance and made available to the re-creation process:

- PDB
- The project and instance specification files

Reusing the OSM Schema

To reuse the OSM DB, the secret for the PDB must still exist:

```
project-instance-database-credentials
```

project-instance`-database-credentials.`

This is the `osmdb` credential in the **manage-instance-credentials.sh** script.

Reusing the RCU

To reuse the RCU, the following secrets for the RCU DB must still exist:

- *project-instance*`-rcudb-credentials`. This is the `rcudb` credential.
- *project-instance*`-opss-wallet-password-secret`. This is the `opssWP` credential.
- *project-instance*`-opss-walletfile-secret`. This is the `opssWF` credential.

If the `opssWP` and `opssWF` secrets no longer exist and cannot be re-created from offline data, then drop the RCU schema and re-create it using the OSM DB Installer.

Create the instance as you would normally do:

```
$OSM_CNTK/scripts/create-instance.sh -p project -i instance -s spec_Path
```

Creating a New Instance

If the original instance does not need to be retained, then the original PDB can be re-used directly by a new instance. If however, the instance needs to be retained, then you must create a clone of the PDB of the original instance. This section describes using a newly cloned PDB for the new instance.

If possible, ensure that the images specified in the project specification (*project*.yaml) match the images in the specification files of the original instance.

Reusing the OSM Schema

To reuse the OSM DB, the following secret for the PDB must be created using the new project and instance names. This is the `osmdb` credential in **manage-instance-credentials.sh** and points to your cloned PDB:

```
project-instance-database-credentials
```

If your new instance must reference a newer OSM DB installer image in its specification files than the original instance, it is recommended to invoke an in-place upgrade of OSM schema before creating the new instance.

To upgrade or check the OSM schema:

```
# Upgrade the OSM schema to match new instance's specification files
# Do nothing if schema already matches
$OSM_CNTK/scripts/install-osmdb.sh -p project -i instance -s spec_path
-c 1
```

You can choose a strategy for the RCU DB from one of the following options:

- Create a new RCU
- Reuse RCU

Creating a New RCU

If you only wish to retain the OSM schema data (cartridges and orders), then you can create a new RCU schema.

The following steps provide a consolidated view of RCU creation described in "Managing Configuration as Code".

To create a new RCU, create the following secrets:

- *project-instance*-`rcudb-credentials`. This is the `rcudb` credential and describes the new RCU schema you want in the clone.
- *project-instance*-`opss-wallet-password-secret`. This is the `opssWP` credential unique to your new instance

After these credentials are in place, prepare the cloned PDB:

```
# Create a fresh RCU DB schema while preserving OSM schema data
$OSM_CNTK/scripts/install-osmdb.sh -p project -i instance -s spec_path
-c 7
```

With this approach, the RCU schema from the original instance is still available in the cloned PDB, but is not used by the new instance.

Reusing the RCU

Using the **manage-instance-credentials.sh** script, create the following secret using your new project and instance names:

```
project-instance-rcudb-credentials
```

The secret should describe the old RCU schema, but with new PDB details.

- **Reusing RCU Schema Prefix**

  Over time, if PDBs are cloned multiple times, it may be desirable to avoid the proliferation of defunct RCU schemas by re-using the schema prefix and re-initializing the data. There is no OSM metadata or order data stored in the RCU DB so the data can be safely re-initialized.

  project-instance-`opss-wallet-password-secret`. This is the `opssWP` credential unique to your new instance.

  To re-install the RCU, invoke DB Installer:

  ```
  $OSM_CNTK/scripts/install-osmdb.sh -p project -i instance -s
  spec_path -c 5
  ```

- **Reusing RCU Schema and Data**

  In order to reuse the full RCU DB from another instance, the original `opssWF` and `opssWP` must be copied to the new environment and renamed following the convention: *project-instance*-**opss-wallet-password-secret** and *project-instance*-**opss-walletfile-secret**.

  This directs Fusion MiddleWare OPSS to access the data using the secrets.

Create the instance as you would normally do:

```
$OSM_CNTK/scripts/create-instance.sh -p project -i instance -s spec_path
```

Setting Up Persistent Storage

OSM cloud native can be configured to use a Kubernetes Persistent Volume to store data that needs to be retained even after a pod is terminated. This data includes application logs, JFR recordings and DB Installer logs, but does not include any sort of OSM state data. When an instance is re-created, the same persistent volume need not be available. When persistent storage is enabled in the instance specification, these data files, which are written inside a pod are re-directed to the persistent volume.

Data from all instances in a project may be persisted, but each instance does not need a unique location for logging. Data is written to a *project-instance* folder, so multiple instances can share the same end location without destroying data from other instances.

The final location for this data should be one that is directly visible to the users of OSM cloud native. The development instances may simply direct data to a shared file system for analysis and debugging by cartridge developers. Whereas, formal test and production instances may need the data to be scraped by a logging toolchain such as EFK, that can then process the data and make it available in various forms. The recommendation therefore is to create a PV-PVC pair for each class of destination within a project. In this example, one for developers to access and one that feeds into a toolchain.

A PV-PVC pair would be created for each of these "destinations", that multiple instances can then share. A single PVC can be used by multiple OSM domains. The management of the PV and PVC lifecycles is beyond the scope of OSM cloud native.

The OSM cloud native infrastructure administrator is responsible for creating and deleting PVs or for setting up dynamic volume provisioning.

The OSM cloud native project administrator is responsible for creating and deleting PVCs as per the standard documentation in a manner such that they consume the pre-created PVs or trigger the dynamic volume provisioning. The specific technology supporting the PV is also beyond the scope of OSM cloud native. However, samples for PV supported by NFS are provided.

Creating a PV-PVC Pair

The technology supporting the Kubernetes PV-PVC is not dictated by OSM cloud native. Samples have been provided for NFS and can either be used as is, or as a reference for other implementations.

To create a PV-PVC pair supported by NFS:

1. Edit the sample PV and PVC yaml files and update entries with enclosing brackets

   > **Note:**
   >
   > PVCs need to be ReadWriteMany.

   ```
   vi $OSM_CNTK/samples/nfs/pv.yaml
     vi $OSM_CNTK/samples/nfs/pvc.yaml
   ```

2. Create the Kubernetes PV and PVC.

   ```
   kubectl create -f $OSM_CNTK/samples/nfs/pv.yaml
   kubectl create -f $OSM_CNTK/samples/nfs/pvc.yaml
   ```

Enable storage in the instance specification and specify the name of the PVC created:

```
# The storage volume must specify the PVC to be used for persistent
storage.
storageVolume:
  enabled: true
  pvc: storage-pvc
```

After the instance is created, you should see the following directories in your PV mount point, if you have enabled logs:

```
[oracle@localhost project-instance]$ dir
db-installer  logs  performance
```

Setting Up Database Optimizer Statistics

As part of the setup of a highly performant database for OSM, it is necessary to set up database optimizer statistics. OSM DB Installer can be used to set up the database partition statistics, which ensures a consistent source of statistics for new partitions so that the database generates optimal execution plans for queries in those partitions.

About the Default Partition Statistics

The OSM DB Installer comes with a set of default partition statistics. These statistics come from an OSM system running a large number of orders (over 400,000) for a cartridge of reasonable complexity. These partition statistics are usable as-is for production.

Setting Up Database Partition Statistics

To use the provided default partition statistics, no additional input, in terms of specification files, secrets or other runtime aspects, is required for the OSM cloud native DB Installer.

The OSM cloud native DB Installer is invoked during the OSM instance creation, to either create or update the OSM schema. The installer is configured to automatically populate the default partition statistics (to all partitions) for a newly created OSM schema when the "prod", "prodsmall", or "prodlarge" (Production) shape is declared in the instance specification. The **statistics.loadPartitionStatistics** field within these shape files is set to **true** to enable the loading.

If you want to load partition statistics for a non-production shape, or if you want to reload statistics due to a DB or schema upgrade, use the command with **11** to load the statistics to all existing partitions in the OSM schema:

```
$OSM_CNTK/scripts/install-osmdb.sh -p project -i instance -s $SPEC_PATH
-c 11
```

If you create new partitions, to import the default partition statistics to these new partitions, run the following command on the DB Installer.

> **Note:**
>
> The partition name is specified in **-b** parameter with a comma delimited list of partition names.

```
$OSM_CNTK/scripts/install-osmdb.sh -p project -i instance -s $SPEC_PATH
-b the_newly_created_partition_1,the_newly_created_partition_2 -c 11
```

If you create new partitions, and want to copy or load the partition statistics data from an existing partition to these new partitions, run the following command on the DB Installer.

```
$OSM_CNTK/scripts/install-osmdb.sh -p project -i instance
-s $SPEC_PATH -a existing_partition_name -b
the_newly_created_partition_1,the_newly_created_partition_2 -c 11
```

Leveraging Oracle WebLogic Server Active GridLink

If you are using a RAC database for your OSM cloud native instance, by default, OSM uses WebLogic Multi-DataSource (MDS) configurations to connect to the database.

If you are licensed to use Oracle WebLogic Server Active GridLink (AGL) separately from your OSM license (consult any additional WebLogic licenses you possess that may apply), you can configure OSM cloud native to use AGL configurations where possible. This will better distribute load across RAC nodes.

To enable the use of AGL, find the "db:" section in your instance specification YAML file and add the "aglLincensed" line as shown below and then create or upgrade your instance as usual:

```
db:
  aglLicensed: true
```

Managing Logs

OSM cloud native generates traditional textual logs. By default, these log files are generated in the managed server pod, but can be re-directed to a Persistent Volume Claim (PVC) supported by the underlying technology that you choose. See "Setting Up Persistent Storage" for details.

By default, logging is enabled. When persistent storage is enabled, logs are automatically re-directed to the Persistent Volume.

```
# The storage volume must specify the PVC to be used for persistent
storage. If enabled, the log, metric and JFR data will be directed here.
storageVolume:
  enabled: true
  pvc: storage-pvc
```

- The OSM application logs can be found at: *pv-directory*/*project-instance*/logs
- The OSM DB Installer logs can be found at: *pv_directory*/*project-instance*/db-installer

Managing OSM Cloud Native Metrics

All managed server pods running OSM cloud native carry annotations added by WebLogic Operator and an additional annotation by OSM cloud native.

```
osmcn.metricspath: /OrderManagement/metrics
osmcn.metricsport: 8001
prometheus.io/scrape: true
```

Configuring Prometheus for OSM Cloud Native Metrics

Configure the scrape job in Prometheus as follows:

```
- job_name: 'osmcn'
    kubernetes_sd_configs:
    - role: pod
    relabel_configs:
    - source_labels:
['__meta_kubernetes_pod_annotationpresent_osmcn_metricspath']
      action: 'keep'
      regex: 'true'
    - source_labels:
[__meta_kubernetes_pod_annotation_osmcn_metricspath]
      action: replace
      target_label: __metrics_path__
      regex: (.+)
    - source_labels:
['__meta_kubernetes_pod_annotation_prometheus_io_scrape']
      action: 'drop'
      regex: 'false'
    - source_labels: [__address__,
__meta_kubernetes_pod_annotation_osmcn_metricsport]
      action: replace
      regex: ([^:]+)(?::\d+)?;(\d+)
      replacement: $1:$2
      target_label: __address__
    - action: labelmap
      regex: __meta_kubernetes_pod_label_(.+)
    - source_labels: [__meta_kubernetes_pod_name]
      action: replace
      target_label: pod_name
    - source_labels: [__meta_kubernetes_namespace]
      action: replace
      target_label: namespace
```

Note:

OSM cloud native has been tested with Prometheus and Grafana installed and configured using the Helm chart **prometheus-community/kube-prometheus-stack** available at: https://prometheus-community.github.io/helm-charts.

Viewing OSM Cloud Native Metrics Without Using Prometheus

The OSM cloud native metrics can also be viewed at:

```
http://instance.project.domain_Name:LoadBalancer_Port/OrderManagement/
metrics
```

By default, `domain_Name` is set to **osm.org** and can be modified in ***project*.yaml**. This only provides metrics of the managed server that is serving the request. It does not provide consolidated metrics for the entire cluster. Only Prometheus Query and Grafana dashboards can provide consolidated metrics.

Viewing OSM Cloud Native Metrics in Grafana

OSM cloud native metrics scraped by Prometheus can be made available for further processing and visualization. The OSM cloud native toolkit comes with sample Grafana dashboards to get you started with visualizations.

Import the dashboard JSON files from **$OSM_CNTK/samples/grafana** into your Grafana environment.

The sample dashboards are:

- OSM by Instance: Provides a view of OSM cloud native metrics for one or more instances in the selected project namespace.
- OSM by Server: Provides a view of OSM cloud native metrics for one or more managed servers for a given instance in the selected project namespace.
- OSM by Order Type: Provides a view of OSM cloud native metrics for one or more order types for a given cartridge version in the selected instance and project namespace.

Exposed OSM Order Metrics

The following OSM metrics are exposed via Prometheus APIs.

Note:

- All metrics are per managed server. Prometheus Query Language can be used to combine or aggregate metrics across all managed servers.
- All metric values are short-lived and indicate the number of orders (or tasks) in a particular state since the managed server was last restarted.
- When a managed server restarts, all the metrics are reset to **0**. These metrics do not refer to the exact values, which can be queried via OSM APIs such as Web Services and XML API.

Order Metrics

The following table lists order metrics exposed via Prometheus APIs.

Table 8-4 Order Metrics Exposed via Prometheus APIs

Name	Type	Help Text	Notes
osm_orders_created	Counter	Counter for the number of Orders Created	N/A
osm_orders_completed	Counter	Counter for the number of Orders Completed	N/A
osm_orders_failed	Counter	Counter for the number of Orders Failed	N/A
osm_orders_cancelled	Counter	Counter for the number of Orders Cancelled	N/A
osm_orders_aborted	Counter	Counter for the number of Orders Aborted	N/A
osm_orders_in_progress	Gauge	Gauge for the number of orders currently in the In Progress state	N/A
osm_orders_amending	Gauge	Gauge for the number of orders currently in the Amending state	N/A
osm_short_lived_orders	Histogram	Histogram that tracks the duration of all orders in seconds with buckets for 1 second, 3 seconds, 5 seconds, 10 seconds, 1 minute, 3 minutes, 5 minutes, and 15 minutes. Enables focus on short-lived orders.	Buckets for 1 second, 3 seconds, 5 seconds, 10 seconds, 1 minute, 3 minutes, 5 minutes, and 15 minutes.
osm_medium_lived_orders	Histogram	Histogram that tracks the duration of all orders in minutes with buckets for 5 minutes, 15 minutes, 1 hour, 12 hours, 1 day, 3 days, 1 week, and 2 weeks. Enables focus on medium-lived orders.	Buckets for 5 minutes, 15 minutes, 1 hour, 12 hours, 1 day, 3 days, 7 days, and 14 days.
osm_long_lived_orders	Histogram	Histogram that tracks the duration of all orders in days with buckets for 1 week, 2 weeks, 1 month, 2 months, 3 months, 6 months, 1 year and 2 years. Enables focus on long-lived orders.	Buckets for 7 days, 14 days, 30 days, 60 days, 90 days, 180 days, 365 days, and 730 days.

Table 8-4 (Cont.) Order Metrics Exposed via Prometheus APIs

Name	Type	Help Text	Notes
osm_order_cache_entries_total	Gauge	Gauge for the number of entries in the cache of type order, orchestration, historical order, closed order, and redo order	N/A
osm_order_cache_max_entries_total	Gauge	Gauge for the maximum number of entries in the cache of type order,orchestration, historical order, closed order, and redo order	N/A

Labels For All Order Metrics

The following table lists labels for all order metrics.

Table 8-5 Labels for All Order Metrics

Label Name	Sample Value	Notes	Source of the Label
cartridge_name_version	SimpleRabbits_1.7.0.1.0	Combined Cartridge Name and Version	OSM Metric Label Name/Value
order_type	SimpleRabbitsOrder	OSM Order Type	OSM Metric Label Name/Value
server_name	ms1	Name of the Managed Server	OSM Metric Label Name/Value
instance	10.244.0.198:8081	Indicates the Pod IP and Pod port from which this metric is being scraped.	Prometheus Kubernetes SD
job	omscn	Job name in Prometheus configuration which scraped this metric.	Prometheus Kubernetes SD
namespace	quick	Project Namespace	Prometheus Kubernetes SD
pod_name	quick-sr-ms1	Name of the Managed Server Pod	Prometheus Kubernetes SD
weblogic_clusterName	c1	OSM Cloud Native WebLogic Cluster Name	WebLogic Operator Pod Label
weblogic_clusterRestartVersion	v1	OSM Cloud Native WebLogic Operator Cluster Restart Version	WebLogic Operator Pod Label
weblogic_createdByOperator	true	WebLogic Operator Pod Label to identify operator created pods	WebLogic Operator Pod Label
weblogic_domainName	domain	WebLogic Operator pod label	WebLogic Operator pod label
weblogic_domainRestartVersion	v1	OSM Cloud Native WebLogic Operator Domain Restart Version	WebLogic Operator Pod Label
weblogic_domainUID	quick-sr	OSM Cloud Native WebLogic Operator Domain UID	WebLogic Operator Pod Label
weblogic_modelInImageDomainZipHash	md5.3d1b561138f3ae3238d67a023771cf45.md5	Image md5 hash	WebLogic Operator Pod Label
weblogic_serverName	ms1	WebLogic Operator Pod Label for Name of the Managed Server	WebLogic Operator Pod Label

Task Metrics

The following metrics are captured for Manual or Automated Task Types only. All other Task Types are currently not being captured.

Table 8-6 Task Metrics Captured for Manual or Automated Task Types Only

Name	Type	Help Text
osm_tasks_created	Counter	Counter for the number of Tasks Created
osm_tasks_completed	Counter	Counter for the number of Tasks Completed

Labels for all Task Metrics

A task metric has all the labels that an order metric has. In addition, a task metric has two more labels.

Table 8-7 Labels for All Task Metrics

Label	Sample Value	Notes	Source of Label
task_name	RabbitRunTask	Task Name	OSM Metric Label Name/ Value
task_type	A	**A** for Automated **M** for Manual	OSM Metric Label Name/ Value

Managing WebLogic Monitoring Exporter (WME) Metrics

OSM cloud native provides a sample Grafana dashboard that you can use to visualize WebLogic metrics available from a Prometheus data source.

You use the WebLogic Monitoring Exporter (WME) tool to expose WebLogic server metrics. WebLogic Monitoring Exporter is part of the WebLogic Kubernetes Toolkit. It is an open source project, based at: https://github.com/oracle/weblogic-monitoring-exporter. You can include WME in your OSM cloud native images. Once an OSM cloud native image with WME is generated, creating an OSM cloud native instance with that image automatically deploys a WME WAR file to the WebLogic server instances. While WME metrics are available through WME Restful Management API endpoints, OSM cloud native relies on Prometheus to scrape and expose these metrics. This version of OSM supports WME 1.3.0. See WME documentation for details on configuration and exposed metrics.

The following topics provide a sample integration:

- Generating the WME WAR File
- Deploying the WME WAR File
- Enabling Prometheus for WebLogic Monitoring Exporter (WME) Metrics
- Configuring the Prometheus Scrape Job for WME Metrics
- Viewing WebLogic Monitoring Exporter Metrics in Grafana

Generating the WME WAR File

To generate the WME WAR file, run the following commands, which update the **wls-exporter.war** WAR file with the **exporter-config.yaml** configuration file.

```
mkdir -p ~/wme
cd ~/wme

curl -x $http_proxy -L https://github.com/oracle/weblogic-monitoring-
exporter/releases/download/v1.3.0/wls-exporter.war -o wls-exporter.war
curl -x $http_proxy https://raw.githubusercontent.com/
oracle/weblogic-monitoring-exporter/v1.3.0/samples/kubernetes/end2end/
dashboard/exporter-config.yaml -o exporter-config.yaml

jar -uvf wls-exporter.war exporter-config.yaml
```

Deploying the WME WAR File

After the WME WAR file is generated and updated, you can deploy it as a custom application archive.

For details about deploying entities, see "Deploying Entities to an OSM WebLogic Domain".

You can use the following sample to deploy the WME WAR file to the admin server and the managed servers in a cluster:

```
appDeployments:
    Application:
        'wls-exporter':
            SourcePath: 'wlsdeploy/applications/wls-exporter.war'
            ModuleType: war
            StagingMode: nostage
            PlanStagingMode: nostage
            Target: '@@PROP:ADMIN_NAME@@ , @@PROP:CLUSTER_NAME@@'
```

Enabling Prometheus for WebLogic Monitoring Exporter (WME) Metrics

To enable Prometheus for gathering and exposing WebLogic Monitoring Exporter metrics on server pods running OSM cloud native, add the following to your custom shape specification or project specification:

```
#### For AdminServer pod
prometheus.io/path: /wls-exporter/metrics
prometheus.io/port: 7001
prometheus.io/scrape: true

#### For Managed Server pods
prometheus.io/path: /wls-exporter/metrics
prometheus.io/port: 8001
prometheus.io/scrape: true
```

Configuring the Prometheus Scrape Job for WME Metrics

Configure the scrape job in Prometheus as follows in the **scrapeJobConfiguration.yaml** file:

Note:

In the `basic_auth` section, specify the WebLogic username and password.

```
- job_name: 'basewls'
    kubernetes_sd_configs:
    - role: pod
    relabel_configs:
    - source_labels:
['__meta_kubernetes_pod_annotation_prometheus_io_scrape']
      action: 'keep'
      regex: 'true'
    - source_labels:
[__meta_kubernetes_pod_label_weblogic_createdByOperator]
      action: 'keep'
      regex: 'true'
    - source_labels:
[__meta_kubernetes_pod_annotation_prometheus_io_path]
      action: replace
      target_label: __metrics_path__
      regex: (.+)
    - source_labels: [__address__,
__meta_kubernetes_pod_annotation_prometheus_io_port]
      action: replace
      regex: ([^:]+)(?::\d+)?;(\d+)
      replacement: $1:$2
      target_label: __address__
    - action: labelmap
      regex: __meta_kubernetes_pod_label_(.+)
    - source_labels: [__meta_kubernetes_pod_name]
      action: replace
      target_label: pod_name
    - source_labels: [__meta_kubernetes_namespace]
      action: replace
      target_label: namespace
    basic_auth:
      username: weblogic_username
      password: weblogic_password
```

Viewing WebLogic Monitoring Exporter Metrics in Grafana

WebLogic Monitoring Exporter metrics scraped by Prometheus can be made available for further processing and visualization. The OSM cloud native toolkit comes with sample Grafana dashboards to get you started with visualizations. The **OSM and WebLogic by Server** sample dashboard provides a combined view of OSM cloud native and WebLogic Monitoring Exporter metrics for one or more managed servers for a given instance in the selected project namespace.

Import the dashboard JSON file from **$OSM_CNTK/samples/grafana** into your Grafana environment, selecting Prometheus as the data source.

9
Integrating OSM

Typical usage of OSM involves the OSM application coordinating activities across multiple peer systems. Several systems interact with OSM for various purposes. This chapter examines the considerations involved in integrating OSM cloud native instances into a larger solution ecosystem.

This section describes the following topics and tasks:

- Connectivity with traditional OSM instances
- Connectivity with OSM cloud native instances
- Configuring SAF
- Applying the WebLogic patch for external systems
- Configuring SAF for External Systems
- Setting up Secure Communication with SSL/TLS

Connectivity With Traditional OSM Instances

OSM interacts with external systems that fall broadly in the following categories:

- Human user interaction
- Upstream systems that inject orders and check status
- Peer systems and downstream systems that receive requests and provide updates

Human User Interaction

Human users interact with OSM using the following user interfaces:

- Task Web Client
- Order Management Web Client

These user interfaces connect to OSM through HTTP and HTTPS. Some deployments involve custom user interfaces built for specific purposes. These too interact with OSM using the Web Services API (WSAPI) or XML API (XMLAPI), with requests and responses transmitted over HTTP and HTTPS.

Order Submission and Status Check

Order capture systems, CRM systems, and middleware applications such as Application Integration Architecture (AIA) submit orders into OSM. They can sign up for order updates through the event/milestone framework. This interaction can theoretically happen through Web Services API,XML API calls over HTTP/HTTPS. However, for reasons of scalability, resilience and load management, the strong recommendation is to conduct this interaction over JMS. This typically involves SAF as well, to avoid foreign JMS injection. JMS, whether native or with SAF, runs over the T3 protocol.

OSM itself can be the upstream system here. For instance, consider an OSM instance functioning as Central Order Management (COM). This would need to send orders to another OSM instance functioning as Service Order Management (SOM) and receive updates from it. This too would be via JMS with SAF, running over T3.

There are additional use cases where monitoring systems (or similarly tasked components) can query OSM. These typically take the form of searches for orders that fit some business criteria, and reporting back status and perhaps some additional operationally significant information. OSM is optimized to process orders and therefore processes such requests at some impact. However, many deployments still opt for such interactions. These typically happen as WSAPI or XMLAPI calls over HTTP/HTTPS.

Connectivity with Peer Systems

As OSM processes orders, the logic encoded in the cartridges drives requests to other systems, such as those for billing or inventory or work-force management. These requests can be one-way messages but are much more likely to follow a "request - response" pattern, where the remote system sends one or more responses back to OSM. These responses can arrive immediately or at a later (perhaps much later) time. The communication model OSM recommends for this is JMS (with SAF), which runs over T3.

Technical Connectivity

Over the three categories of interaction, we can distill the following connectivity types:

- OSM APIs invoked via HTTP/HTTPS
- OSM APIs invoked via JMS and SAF
- OSM conversing via JMS and SAF

OSM initiates HTTP/HTTPS messages if explicitly coded to do so in cartridges. This is an anti-pattern for OSM cartridge development as it causes high impact to the throughput capability of OSM. Normally, OSM responds to incoming requests over HTTP/HTTPS (API call responses).

With JMS messages, OSM can be both the originator of a "request-response" transaction or the recipient of one. To support this, OSM can host SAF agents that provide the ability to send JMS messages to remote systems, and OSM can host queues that are targeted by SAF agents on those remote systems.

Security Requirements

OSM Cloud Native supports HTTP and T3. In addition, SAF configuration from one WebLogic domain to another domain very often requires additional security arrangements, including the availability of credentials to authenticate such a connection.

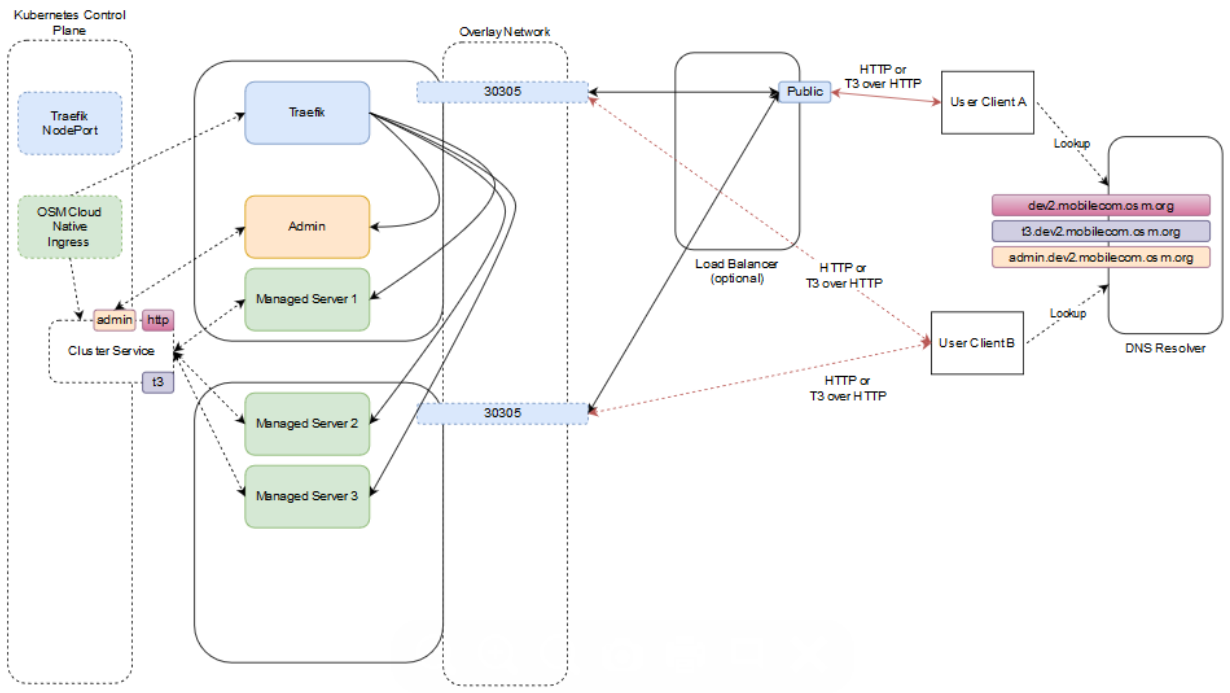
Connectivity With OSM Cloud Native

Functionally, the interaction requirements of OSM do not change when OSM is run in a cloud native environment. All of the categories of interaction that are applicable for connectivity with traditional OSM instances are applicable and must be supported for OSM cloud native.

Connectivity Between the Building Blocks

The following diagram illustrates the connectivity between the building blocks in an OSM cloud native environment using an example:

Figure 9-1 Connectivity Between Building Blocks in OSM Cloud Native Environment



Invoking the OSM cloud native Helm chart creates a new OSM instance. In the above illustration, the name of the instance is "dev2" in the project "mobilecom". The instance consists of the WebLogic cluster that has one Admin Server and three Managed Servers and a Kubernetes Cluster Service.

The Cluster Service contains endpoints for both HTTP and T3 traffic. The instance creation script creates the OSM cloud native Ingress object. The Ingress object has metadata to trigger the Traefik ingress controller as a sample. Traefik responds by creating new front-ends with the configured "hostnames" for the cluster (**dev2.mobilecom.osm.org** and **t3.dev2.mobilecom.osm.org** in the illustration) and the admin server (**admin.dev2.mobilecom.osm.org**) and links them up to new back-end constructs. Each back-end routes to each member of the Cluster Service (MS1, MS2, and MS3 in the example) or to the Admin Server. The **dev2.mobilecom.osm.org** front-end is linked to the back-end pointing to the HTTP endpoint of each managed server, while the **t3.dev2.mobilecom.osm.org** front-end links to the back-end pointing to the T3 endpoint of each managed server.

The prior installation of Traefik has already exposed Traefik itself via a selected port number (30305 in the example) on each worker node.

Inbound HTTP Connectivity

An OSM instance is exposed outside of the Kubernetes cluster for HTTP access via an Ingress Controller and potentially a Load Balancer.

Because the Traefik port (30305) is common to all OSM cloud native instances in the cluster, Traefik must be able to distinguish between the incoming messages headed for different instances. It does this by differentiating on the basis of the "hostname" mentioned in the HTTP messages. This means that a client (User Client B in the illustration) must believe it is talking to the "host" **dev2.mobilecom.osm.org** when it sends HTTP messages to port 30305 on the access IP. This might be the Master node IP, or IP address of one of the worker nodes, depending on your cluster setup. The "DNS Resolver" provides this mapping.

In this mode of communication, there are concerns around resiliency and load distribution. For example, If the DNS Resolver always points to the IP address of Worker Node 1 when asked to resolve **dev2.mobilecom.osm.org**, then that Worker node ends up taking all the inbound traffic for the instance. If the DNS Resolver is configured to respond to any ***.mobilecom.osm.org** requests with that IP, then that worker node ends up taking all the inbound traffic for all the instances. Since this latter configuration in the DNS Resolver is desired, to minimize per-instance touches, the setup creates a bottleneck on Worker node 1. If Worker node 1 were to fail, the DNS Resolver would have to be updated to point ***.mobilecom.osm.org** to Worker node 2. This leads to an interruption of access and requires intervention. The recommended pattern to avoid these concerns is for the DNS Resolver to be populated with all the applicable IP addresses as resolution targets (in our example, it would be populated with the IPs of both Worker node 1 and node 2), and have the Resolver return a random selection from that list.

An alternate mode of communication is to introduce a load balancer configured to balance incoming traffic to the Traefik ports on all the worker nodes. The DNS Resolver is still required, and the entry for ***.mobilecom.osm.org** points to the load balancer. Your load balancer documentation describes how to achieve resiliency and load management. With this setup, a user (User Client A in our example) sends a message to **dev2.mobilecom.osm.org**, which actually resolves to the load balancer - for instance, **http://dev2.mobilecom.osm.org:8080/OrderManagement/Login.jsp**. Here, 8080 is the public port of the load balancer. The load balancer sends this to Traefik, which routes the message, based on the "hostname" targeted by the message to the HTTP channel of the OSM cloud native instance.

By adding the hostname resolution such that **admin.dev2.mobilecom.osm.org** also resolves to the Kubernetes cluster access IP (or Load Balancer IP), User Client B can access the WebLogic console via **http://admin.dev2.mobilecom.osm.org/console** and the credentials specified while setting up the "wlsadmin" secret for this instance.

Note:

Access to the WebLogic Admin console is provided for review and debugging use only. Do not use the console to change the system state or configuration. These are maintained independently in the WebLogic Operator, based on the specifications provided when the instance was created or last updated by the OSM cloud native toolkit. As a result, any such manual changes (whether using the console or using WLST or other such mechanisms) are liable to be overwritten without notice by the Operator. The only way to change state or configuration is through the tools and scripts provided in the toolkit.

Inbound JMS Connectivity

JMS messages use the T3 protocol. Since Ingress Controllers and Load Balancers do not understand T3 for routing purposes, OSM cloud native requires all incoming JMS traffic to be "T3 over HTTP". Hence, the messages are still HTTP, but contain a T3 message as payload. OSM cloud native requires the clients to target the "t3 hostname" of the instance - **t3.dev2.mobilecom.osm.org**, in the example. This "t3 hostname" should behave identically as the regular "hostname" in terms of the DNS Resolver and the Load Balancer. Traefik however not only identifies the instance this message is meant for (dev2.mobilecom) but also that it targets the T3 channel of instance.

The "T3 over HTTP" requirement applies for all inbound JMS messages - whether generated by direct or foreign JMS API calls or generated by SAF. The procedure in SAF QuickStart explains the setup required by the message producer or SAF agent to achieve this encapsulation. If SAF is used, the fact that T3 is riding over HTTP does not affect the semantics of JMS. All the features such as reliable delivery, priority, and TTL, continue to be respected by the system. See "Applying the WebLogic Patch for External Systems".

An OSM instance can be configured for secure access, which includes exposing the T3 endpoint outside the Kubernetes cluster for HTTPS access. See "Configuring Secure Incoming Access with SSL" for details on enabling SSL.

Inbound JMS Connectivity Within the Same Kubernetes Cluster

For all inbound JMS connectivity, use the T3 hostname: `t3.dev2.mobilecom.osm.org`. This URL applies to clients outside of the Kubernetes cluster in which OSM cloud native is deployed. This requires configuring Ingress Controller and DNS Resolver to access the URL.

However, there can be situations where OSM cloud native needs to be accessed from within the same Kubernetes cluster where it is deployed. For example, an upstream application sending orders or a downstream application sending status updates could be deployed in the same Kubernetes cluster. It could also be another OSM cloud native instance deployed in the same Kubernetes cluster either sending or receiving Create Order requests. For such requirements, there is no need for the request to be routed via an Ingress Controller or a load balancer and resolved via a DNS Resolver.

OSM cloud native exposes a T3 channel exclusively for such connections and can be accessed via **t3://***project*-*instance***-cluster-c1.***project***.svc.cluster.local:31313**.

This saves the various network hops typically involved in routing a request from an external client to OSM cloud native deployed in a Kubernetes cluster.

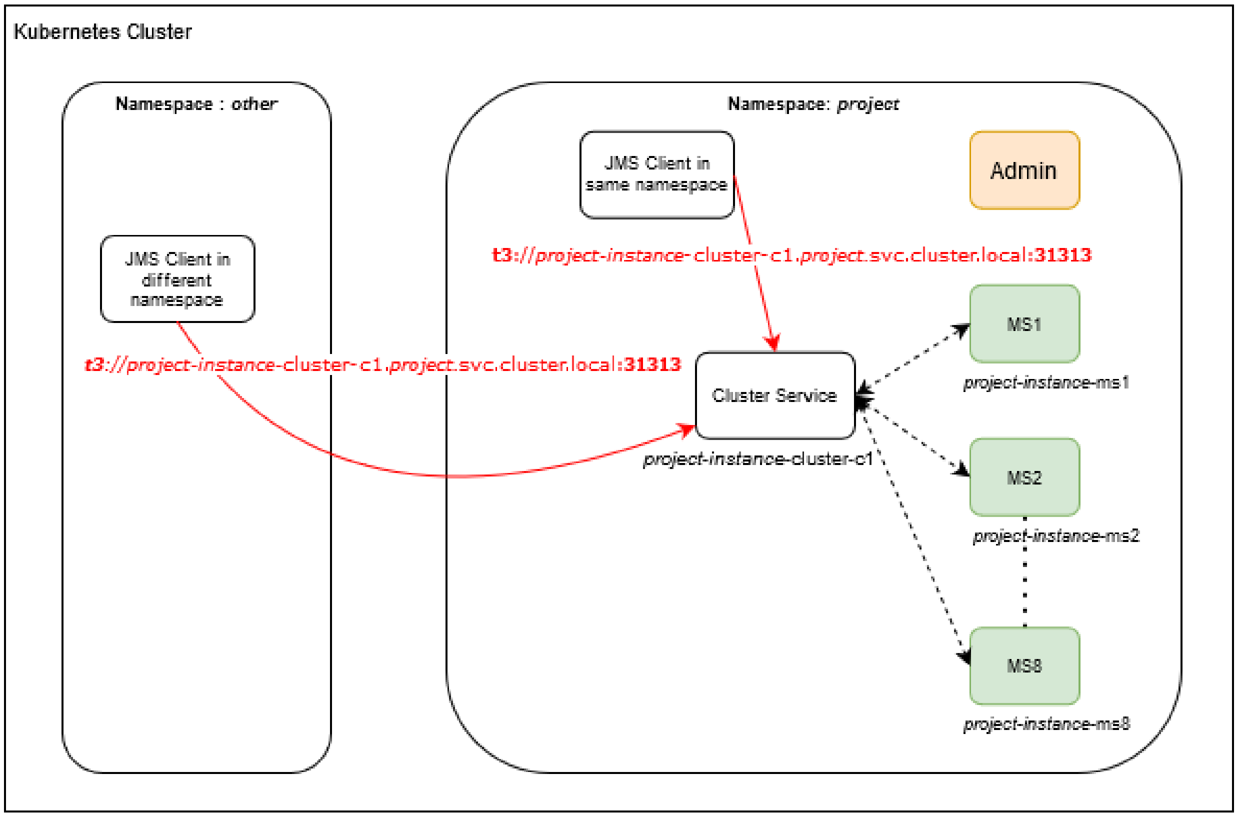
The following diagram illustrates inbound JMS connectivity within the same Kubernetes cluster using an example.

For the example, the URL is **t3://mobilecom-dev2-cluster-c1.mobilecom.svc.cluster.local:31313**.

Note:

The protocol is T3 as there is no need for wrapping in HTTP. Note that the port is different.

Figure 9-2 Inbound JMS Connectivity in a Kubernetes Cluster


Kubernetes Cluster
Namespace : other
JMS Client in different namespace
t3://project-instance-cluster-c1.project.svc.cluster.local:31313
Namespace: project
JMS Client in same namespace
Admin
t3://project-instance-cluster-c1.project.svc.cluster.local:31313
Cluster Service
project-instance-cluster-c1
MS1
project-instance-ms1
MS2
project-instance-ms2
MS8
project-instance-ms8

If SSL is enabled for domains, communication between the domains within the Kubernetes cluster is not secured because the ingress is not involved. See "Setting Up Secure Communication with SSL" for further details.

Outbound HTTP Connectivity

No specific action is required to ensure the HTTP messages from OSM cloud native instance reach out of the Kubernetes Cluster.

When a domain inside a Kubernetes cluster sends REST API or Web Service requests over HTTP to a domain that is outside the cluster that is enabled with SSL, then you should set up some required configuration. For instructions, see "Configuring Access to External SSL-Enabled Systems".

Outbound JMS Connectivity

JMS messages originating from the OSM cloud native instance such as requests to peer systems from cartridge automation plug-ins or event notifications to upstream system from notification plug-ins, always end up on local queues. The OSM cloud native Helm chart allows for the specification of SAF connections to remote systems in order to get these messages to their destinations. The project specification contains all the SAF connections that must exist for the cartridge(s) to do their job. The instance specification provides a specific endpoint for each of these SAF connections. This allows for a canonical expression of the SAF connectivity requirements, which are uniquely fulfilled by each instance by pointing to the appropriate upstream, downstream, peer systems or emulators, and so on.

When a domain inside a Kubernetes cluster sends JMS messages to a domain that is outside the cluster that is SSL-enabled, then see "Configuring Access to External SSL-Enabled Systems" for instructions on setting up some required configuration.

Configuring SAF

OSM cloud native requires SAF for the OSM cartridge automation functionality to send messages to external systems through JMS. The SAF configuration in OSM cloud native has two distinct aspects - the project and the instance. At the project level, the project specification can be used to define all the SAF connections that any OSM cloud native instance must make. This list is governed by the cartridges that constitute the project. At the instance level, each of these SAF connections must be given a specific remote endpoint.

Configuring the Project Specification

The project specification lists out all the SAF connections that are required for the set of solution cartridges that the project requires in order to function. These are listed under the `safDestinationConfig` element of the project specification.

The following sample shows a basic SAF specification that describes the need to interact via SAF with `external-system-identifier`. It specifies that the project is interested in accessing two queues on that remote system: `remote-queue-1` and `remote-queue-2`. On that system, these queues can be addressed using the JNDI prefix `prefix-1`. Further, `remote-queue-1` is also mapped locally as `local-queue-1`. Whether this is necessary or not depends on the addressing system coded into the OSM cartridge's external sender automation plugins. OSM cloud native supports both local names and remote names for SAF destinations.

```
safDestinationConfig:
  - name: external_system_identifier
    destinations:
      - jndiPrefix: prefix_1
        queues:
          - queue:
              remoteJndi: remote_queue_1
              localJndi: local_queue_1
          - queue:
              remoteJndi: remote_queue_2
```

If the queues of an external system are spread across more than one JNDI prefix, the `jndiPrefix` element can be repeated as many times as necessary. In this example, `prefix_1` applies to `remote_queue_1` and `remote_queue_2`, while `prefix_2` applies to `remote_queue_3`.

The following sample shows SAF project specification with multiple JNDIs:

```
safDestinationConfig:
  - name: external_system_identifier
    destinations:
      - jndiPrefix: prefix_1
        queues:
          - queue:
              remoteJndi: remote_queue_1
              localJndi: local_queue_1
          - queue:
              remoteJndi: remote_queue_2
      - jndiPrefix: prefix_2
        queues:
          - queue:
              remoteJndi: remote_queue_3
```

It is possible for an external system to not use a JNDI prefix, which is configured by leaving the value empty for `jndiPrefix`. However, at most, one of the `jndiPrefix` entries in a destinations list can be empty, as the `jndiPrefixes` in this list have to be unique. If there are more than one external system that the project's solution cartridges interact with via SAF, these can be named and listed as follows:

```
safDestinationConfig:
  - name: external_system_identifier_1
    destinations:
      - jndiPrefix: prefix_1
        queues:
          - queue:
              remoteJndi: remote_queue_1
  - name: external_system_identifier_2
    destinations:
      - jndiPrefix: prefix_2
        queues:
          - queue:
              remoteJndi: remote_queue_2
```

Note:

Using the provided configuration, OSM cloud native automatically computes `names` for some entities required for completing the SAF setup. You may find such entities when you log into WebLogic Administration Console for troubleshooting purposes and are not to be confused.

Configuring the Instance Specification

The project specification lays out the connectivity requirements of the solution cartridges in the project. However, each instance needs to provide its own set of endpoints to satisfy those connections. For example, the project specification may require connectivity to a remote UIM system to send inventory related commands via JMS and SAF. It is the instance specification that directs this requirement to a specific UIM installation valid for use with this instance. Another instance of the same project might target a different UIM installation or an emulator.

The instance specification contains the T3 URL of the external system along with the name of a Kubernetes secret that provides the credentials required to interact with that system. The T3 URL can be specified using any of the standard mechanisms supported by WebLogic. The Kubernetes secret must contain the fields username and password, carrying credentials which have permission to inject JMS messages into the remote system.

```
safConnectionConfig:
  - name: external_system_identifier
    t3Url: t3_url
    secretName: secret_t3_user_pass
```

Here, the `external_system_identifier` needs to match the `external_system_identifier` specified in the project specification. The instance specification must have an entry for each of the `external_system_identifier` entries listed in the project specification.

If the external system is an OSM cloud native instance deployed in the same Kubernetes cluster, use the T3 URL as described in "Inbound JMS Connectivity Within the Same Kubernetes Cluster".

If SSL is enabled for the external system, use the T3 URL as described in "Configuring Access to External SSL-Enabled Systems".

Configuring Domain Trust

For details about global trust, see "Enabling Global Trust" in *Oracle Fusion Middleware Administering Security for Oracle WebLogic Server*.

Because the shared password provides access to all domains that participate in the trust, strict password management is critical. Trust should be enabled when SAF is configured as it is needed for inter-domain communication using distributed destinations. In a Kubernetes cluster where the pods are transient, it is possible that a SAF sender will not know where it can forward messages unless domain trust is configured.

If trust is not configured when using SAF, you may experience unstable SAF behavior when your environment has pods that are growing, shrinking, or restarting.

To enable domain trust, in your instance specification file, for `domainTrust`, change the default value to **true**:

```
domainTrust:
 enabled: true
```

If you are enabling domain trust, then you must create a Kubernetes secret (exactly as specified) to store the shared trust password by running the following command:

> **Note:**
>
> This step is not required if you are not enabling domain trust in the instance specification.

```
kubectl create secret generic -n project project-instance-global-trust-
credentials --from-literal=password=pwd
```

The same password must be used in all domains that connect to this one through SAF.

Usage in OSM Cartridge Automation

The OSM cartridge automation external sender plugins are unaffected by the switch to OSM cloud native. The plugins continue to address their destinations as before, using JNDI prefix and remote queue name, or JNDI prefix and local queue name. The project specification must reflect what the cartridge developer has actually coded into the automation plug-in in Design Studio.

Inbound SAF Requirements

The OSM cloud native Helm charts create all the entities required for inbound SAF to be processed as T3 over HTTP. No additional configuration is required in the OSM cloud native specification files. However, if the OSM cartridge automation receiver plugins are set up to read from local JNDI prefix and queue name, these must be added to the project specification as standard solution queues under `uniformDistributedQueues` (not as `safConnectionConfig`).

Applying the WebLogic Patch for External Systems

When an external system is configured with a SAF sender towards OSM cloud native, using HTTP tunneling, a patch is required to ensure the SAF sender can connect to the OSM cloud native instance. This is regardless of whether the connection resolves to an ingress controller or to a load balancer. Each such external system that communicates with OSM through SAF must have the WebLogic patch 30656708 installed and configured, by adding `-Dweblogic.rjvm.allowUnknownHost=true` to the WebLogic startup parameters.

For environments where it is not possible to apply and configure this patch, a workaround is available. On each host running a Managed Server of the external system, add the following entries to the **/etc/hosts** file:

```
0.0.0.0 project-instance-ms1
0.0.0.0 project-instance-ms2
0.0.0.0 project-instance-ms3
0.0.0.0 project-instance-ms4
0.0.0.0 project-instance-ms5
0.0.0.0 project-instance-ms6
0.0.0.0 project-instance-ms7
0.0.0.0 project-instance-ms8
0.0.0.0 project-instance-ms9
0.0.0.0 project-instance-ms10
0.0.0.0 project-instance-ms11
0.0.0.0 project-instance-ms12
```

```
0.0.0.0 project-instance-ms13
0.0.0.0 project-instance-ms14
0.0.0.0 project-instance-ms15
0.0.0.0 project-instance-ms16
0.0.0.0 project-instance-ms17
0.0.0.0 project-instance-ms18
```

You should add these entries for all the OSM cloud native instances that the external system interacts with. Set the IP address to 0.0.0.0. All the eight managed servers possible in the OSM cloud native instance must be listed regardless of how many are actually configured in the instance specification.

Configuring SAF On External Systems

To create SAF and JMS configuration on your external systems to communicate with the OSM cloud native instance, use the configuration samples provided as part of the SAF sample as your guide.

It is important to retain the "Per-JVM" and "Exactly-Once" flags as provided in the sample.

All connection factories must have the "Per-JVM" flag, as must SAF foreign destinations.

Each external queue that is configured to use SAF must have its QoS set to "Exactly-Once".

Enabling Domain Trust

To enable domain trust, in your domain configuration, under **Advanced**, edit the **Credential** and **ConfirmCredential** fields with the same password you used to create the global trust secret in OSM cloud native.

Setting Up Secure Communication with SSL

When OSM cloud native is involved in secure communication with other systems, either as the server or as the client, you should additionally configure SSL/TLS. The configuration may involve the WebLogic domain, the ingress controller or the URL of remote endpoints, but it always involves participating in an SSL handshake with the other system. The procedures for setting up SSL use self-signed certificates for demonstration purposes. However, replace the steps as necessary to use signed certificates.

If an OSM cloud native domain is in the role of the client and the server, where secure communications are coming in as well as going out, then both of the following procedures need to be performed:

- Configuring Secure Incoming Access with SSL
- Configuring Access to External SSL-enabled Systems

Configuring Secure Incoming Access with SSL

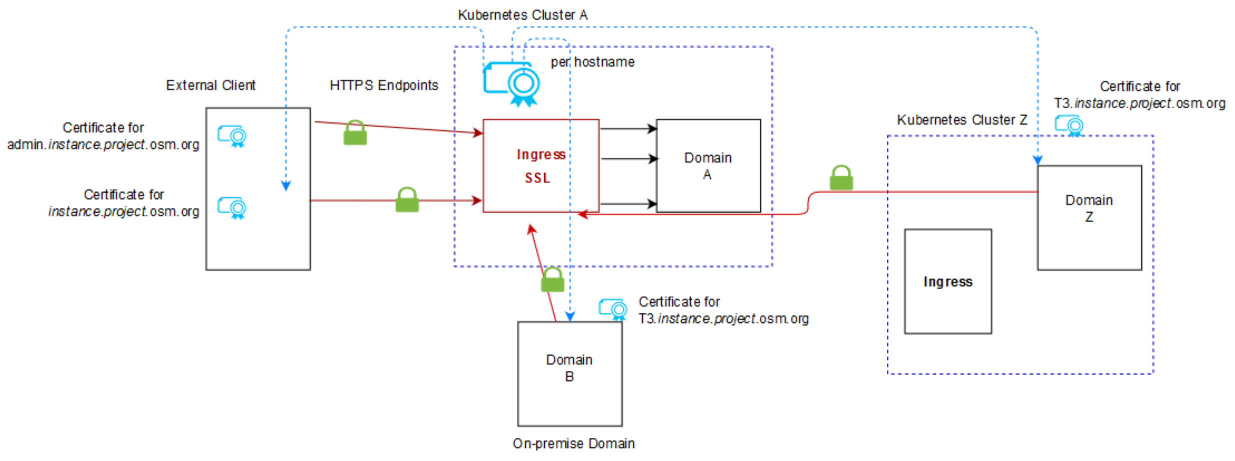
This section demonstrates how to secure incoming access to OSM cloud native. In this scenario, SSL termination happens at the ingress. The traffic coming in from external clients must use one of the HTTPS endpoints. When SSL terminates at the ingress, it also means that communication within the cluster, such as SAF between the OSM cloud native instances, is not secured.

The OSM cloud native toolkit provides the sample configuration for Traefik ingress. If you use Voyager or other ingress, you can look at the **$OSM_CNTK/samples/charts/ingress-per-domain/templates/traefik-ingress.yaml** file to see what configuration is applied.

Generating SSL Certificates for Incoming Access

The following illustration shows when certificates are generated.

Figure 9-3 Generating SSL Certificates


Kubernetes Cluster A
per hostname
External Client
HTTPS Endpoints
Certificate for admin.*instance.project*.osm.org
Certificate for *instance.project*.osm.org
Ingress SSL
Domain A
Kubernetes Cluster Z
Certificate for T3.*instance.project*.osm.org
Domain Z
Ingress
Certificate for T3.*instance.project*.osm.org
Domain B
On-premise Domain

When OSM cloud native dictates secure communication, then it is responsible for generating the SSL certificates. These must be provided to the appropriate client. When an OSM cloud native instance in a different Kubernetes cluster acts as the external client (Domain Z in the illustration), it loads the T3 certificate from Domain A as described in "Configuring Access to External SSL-Enabled Systems".

Setting Up OSM Cloud Native for Incoming Access

The ingress controller routes unique hostnames to different backend services. You can see this if you look at the ingress controller YAML file (obtained by running **kubectl get ingress -n** *project ingress_name* **-o yaml**):

Note:

Traefik 2.x moved to using IngressRoute (a CustomResourceDefinition) instead of the Ingress object. If you are using Traefik, in the following commands, change all references of `ingress` to `ingressroute`.

```
rules:
- host: instance.project.osm.org
```

```
  http:
    paths:
    - backend:
        serviceName: project-instance-cluster-c1
        servicePort: 8001
- host: t3.instance.project.osm.org
  http:
    paths:
    - backend:
        serviceName: project-instance-cluster-c1
        servicePort: 30303
- host: admin.instance.project.osm.org
  http:
    paths:
    - backend:
        serviceName: project-instance-admin
        servicePort: 7001
```

To set up OSM cloud native for incoming access:

1. Generate key pairs for each hostname corresponding to an endpoint that OSM cloud native exposes to the outside world:

   ```
   # Create a directory to save your keys and certificates. This is
   for sample only. Proper management policies should be used to store
   private keys.

   mkdir $SPEC_PATH/ssl

   # Generate key and certificates
   openssl req -x509 -nodes -days 365 -newkey rsa:2048 -
   keyout $SPEC_PATH/ssl/osm.key -out $SPEC_PATH/ssl/osm.crt -subj "/
   CN=instance.project.osm.org"
   openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout
   $SPEC_PATH/ssl/admin.key -out $SPEC_PATH/ssl/admin.crt -subj "/
   CN=admin.instance.project.osm.org"
   openssl req -x509 -nodes -days 365 -newkey rsa:2048 -
   keyout $SPEC_PATH/ssl/t3.key -out $SPEC_PATH/ssl/t3.crt -subj "/
   CN=t3.instance.project.osm.org"

   # Create secrets to hold each of the certificates. The secret name
   must be in the format below. Do not change the secret names

   kubectl create secret -n project tls project-instance-osm-tls-cert
   --key $SPEC_PATH/ssl/osm.key --cert $SPEC_PATH/ssl/osm.crt
   kubectl create secret -n project tls project-instance-admin-tls-
   cert --key $SPEC_PATH/ssl/admin.key --cert $SPEC_PATH/ssl/admin.crt
   kubectl create secret -n project tls project-instance-t3-tls-cert
   --key $SPEC_PATH/ssl/t3.key --cert $SPEC_PATH/ssl/t3.crt
   ```

2. Edit the instance specification and set `incoming` to **true**:

   ```
   ssl:
     incoming: true
   ```

3. After running **create-ingress.sh**, you can validate the configuration by describing the ingress controller for your instance. You should see each of the certificates you generated, terminating one of the hostnames:

   ```
   kubectl get ingress -n project
   ```

   Once you have the name of your ingress, run the following command:

   ```
   kubectl describe ingress -n project ingress

   TLS:
     project-instance-osm-tls-cert terminates instance.project.osm.org
     project-instance-t3-tls-cert terminates 
   t3.instance.project.osm.org
     project-instance-admin-tls-cert terminates 
   admin.instance.project.osm.org
   ```

4. Create your instance as usual.

Configuring Incoming HTTP and JMS Connectivity for External Clients

This section describes how to configure incoming HTTP and JMS connectivity for external clients.

Note:

Remember to have your DNS resolution set up on any remote hosts that will connect to the OSM cloud native instance.

Incoming HTTPS Connectivity

External Web clients that are connecting to OSM cloud native must be configured to accept the certificates from OSM cloud native. They will then connect using the HTTPS endpoint and port **30443**.

Incoming JMS Connectivity

For external servers that are connected to OSM cloud native through SAF, the certificate for the t3 endpoint needs to be copied to the host where the external domain is running.

If your external WebLogic configuration uses "CustomIdentityAndJavaSTandardTrust", then you can follow these instructions exactly to upload the certificate to the Java Standard Trust. If, however, you are using a CustomTrust, then you must upload the certificate into the custom trust keystore.

The keytool is found in the **bin** directory of your jdk installation. The alias should uniquely describe the environment where this certificate is from.

```
./keytool -importcert -v -trustcacerts -alias alias 
-file /path-to-copied-t3-certificate/t3.crt -keystore /path-to-jdk/
jdk1.8.0_202/jre/lib/security/cacerts -storepass default_password
```

```
# For example
./keytool -importcert -v -trustcacerts -alias osmcn -file /scratch/
t3.crt -keystore /jdk1.8.0_202/jre/lib/security/cacerts -storepass
default_password
```

Update the SAF remote endpoint (on the external OSM instance) to use HTTPS and 30443 port (still t3 hostname).

From the SAF sample provided with the toolkit, the external system would configure the following remote endpoint URL:

```
https://t3.dev.supracom.osm.org:30443/
oracle.communications.ordermanagement.SimpleResponseQueue
```

Configuring Access to External SSL-Enabled Systems

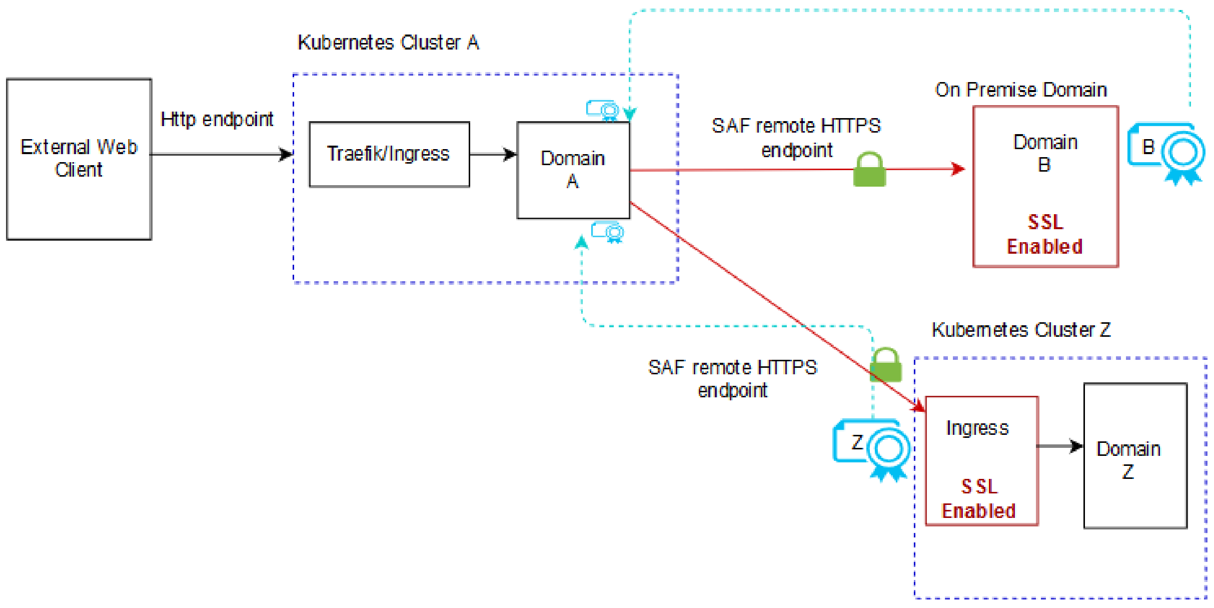
In order for OSM cloud native to participate successfully in a handshake with an external server for SAF connectivity, the SSL certificates from the external domain must be made available to the OSM cloud native setup. See "Enabling SSL on an External WebLogic Domain" for details about how you could do this for an on-premise WebLogic domain. If you have an external system that is already configured for SSL and working properly, you can skip this procedure and proceed to "Setting Up OSM Cloud Native for Outgoing Access".

Loading Certificates for Outgoing Access

In outgoing SSL, the certificates come from the external domain, whether on-premise or in another Kubernetes cluster. These certificates are then loaded into the OSM cloud native trust.

The following illustration shows information about loading certificates into OSM cloud native setup.

Figure 9-4 SSL Certificates for Outgoing Connectivity



Enabling SSL on an External WebLogic Domain

These instructions are specific to enabling SSL on a WebLogic domain that is external to the Kubernetes cluster where OSM cloud native is running.

To enable SSL on an external WebLogic domain:

1. Create the certificates. Perform the following steps on the Linux host that has the on-premise WebLogic domain:

   a. Use the Java keytool to generate public and private keys for the server. When the tool asks for your username, use the FQDN for your server.

   ```
   path-to-jdk/jdk1.8.0_202/bin/keytool -genkeypair -keyalg RSA -
   keysize 1024 -alias alias -keystore keystore file -keypass
   private key password -storepass keystore password -validity 360
   ```

   b. Export the public key. This certificate will then be used in the OSM cloud native setup.

   ```
   path-to-jdk/jdk1.8.0_202/bin/keytool -exportcert -rfc -alias
   alias -storepass password -keystore keystore -file certificate
   ```

2. Configure WebLogic server for SSL. Follow steps 3 to 17 (skip step 7) in the OSM - Encrypting Database Tablespaces and WebLogic Protocols (Doc ID 2399723.1) KM note on My Oracle Support.

3. Validate that SSL is configured properly on this server by importing the certificate to a trust store. For this example, the Java trust store is used.

   ```
   path-to-jdk/jdk1.8.0_202/bin/keytool -importcert -trustcacerts
   -alias alias -file certificate -keystore path-to-jdk/
   jdk1.8.0_202/jre/lib/security/cacerts -storepass default_password
   ```

4. Verify that t3s over the specified port is working by connecting using WLST. Navigate to the directory where the WLST scripts are located.

   ```
   # Set the environment variables. Some shells don't set the
   variables correctly so be sure to check that they are set afterward
   path-to-FMW/Oracle/Middleware/Oracle_Home/oracle_common/common/bin/
   setWlsEnv.sh

   # ensure CLASSPATH and PATH are set
   echo $CLASSPATH

   java -
   Dweblogic.security.JavaStandardTrustKeyStorePassPhrase=default_passw
   ord weblogic.WLST

   # once wlst starts, connect using t3s
   wls:offline> connect('<admin user>','<admin password>','t3s://
   <server>:7002')

   # If successful you will see the prompt
   wls:>domain_name/serverConfig>

   #when finished disconnect
   disconnect()
   ```

Setting Up OSM Cloud Native for Outgoing Access

To set up OSM cloud native for outgoing access:

1. Set up custom trust using the following steps:

   a. Load the certificate from your remote server into a trust store and make it available to the OSM cloud native instance.
      Use the Java keytool to create a jks file (truststore) that holds the certificate from your SSL server:

      ```
      keytool -importcert -v -alias alias -file /path-to/
      certificate.cer -keystore /path-to/truststore.jks -storepass
      password
      ```

      > **Note:**
      >
      > Repeat this step to add as many trusted certificates as required.

b. Create a Kubernetes secret to hold the truststore file and the passphrase. The secret name should match the truststore name.

```
# manually
kubectl create secret generic trust_secret_name -n project --
from-file=truststore.jks --from-literal=passphrase=password

# verify
k get secret -n project trust_secret_name -o yaml
```

c. Edit the instance specification, setting the trust name.

```
# SSL trust and identity
ssl:
  trust:
    name: trust_secret_name    # The name of the secret holding
the remote server truststore contents and passphrase
  identity:
    useDemoIdentity: true

# leave remaining fields commented out
```

When custom trust is enabled, the **useDemoIdentity** field can be left to **true** for development instances. This configures the WebLogic server to use the demo identity that is shipped with WebLogic. For production instances, follow the additional steps for custom identity in the next step.

2. (Optional) Set up custom identity using the following steps:

a. Create the keystore.

```
keytool -genkeypair -keyalg RSA -keysize 1024 -alias <alias>
-keystore identity.jks -keypass private_key_password -storepass
keystore_password -validity 360
```

b. Create the secret.

```
kubectl create secret generic secretName -n project --from-
file=keystore.jks --from-literal=passphrase=password

# verify
k get secret -n project secretName -o yaml
```

c. Edit the specification file:

```
identity:
  useDemoIdentity: false
  name: alias        # only valid when useDemoIdentity is false.
Secret name that contains the identity store file.
  alias: secretName  # only valid when useDemoIdentity is false.
```

3. Configure SAF by updating the SAF connection configuration in the OSM cloud native instance specification file to reflect t3s and the SSL port:

```
safConnectionConfig:
  - name: simple
    t3Url: t3s://remote_server:7002
    secretName: simplesecret
```

4. Create the OSM cloud native instance as usual.

Adding Additional Certificates to an Existing Trust

You can add additional certificates to an existing trust while an OSM cloud native instance is up and running.

To add additional certificates to an existing trust:

1. Set up OSM cloud native for outgoing access. See "Configuring Access to External SSL-Enabled Systems" for instructions.
2. Copy the certificates from your remote server and load them into the existing *truststore*.**jks** file you had created:

```
keytool -importcert -v -alias alias -file /path-to/certificate.cer
-keystore /path-to/truststore.jks -storepass password
```

3. Re-create your Kubernetes secret using the same name as you did previously:

```
# manually
kubectl create secret generic trust_secret_name -n project --from-
file=truststore.jks --from-literal=passphrase=password

# verify
k get secret -n project trust_secret_name -o yaml
```

4. Upgrade the instance to force WebLogic Operator to re-evaluate:

```
$OSM_CNTK/scripts/upgrade-instance.sh -p project -i instance -s
$SPEC_PATH
```

Debugging SSL

To debug SSL, do the following:

- Verify Hostname
- Enable SSL logging

Verifying Hostname

When the keystore is generated for the on-premise server, if FQDN is not specified, then you may have to disable hostname verification. This is not secure and should only be done in development environments.

To do so, add the following Java option to the managed server in the project specification:

```
managedServers:

  project:
    #JAVA_OPTIONS for all managed servers at project level
    java_options: "-
Dweblogic.security.SSL.ignoreHostnameVerification=true"
```

Enabling SSL Logging

When trying to establish the handshake between servers, it is important to enable SSL specific logging.

Add the following Java options to your managed server in the project specification. This should be done for your external server as well.

```
managedServers:

  project:
    #JAVA_OPTIONS for all managed servers at project level
    java_options: "-Dweblogic.StdoutDebugEnabled=true
-Dssl.debug=true -Dweblogic.security.SSL.verbose=true -
Dweblogic.debug.DebugSecuritySSL=true -Djavax.net.debug=ssl"
```

10
Running the SAF Sample for OSM Cloud Native

It is highly recommended that you explore OSM cloud native support of SAF using a predefined set of configurations and instructions. This activity not only serves to quickly identify issues with your cloud environment but also enables you to familiarize yourself with setting up the connectivity for your own projects, which are likely to be more complex than the SAF sample this section describes.

This chapter describes how to run the SAF sample for OSM cloud native.

The SAF sample for OSM cloud native consists of the following components:

- **SimpleProvisioningCartridge** sample cartridge available as a par file, ready to be deployed using the OSM cloud native DB Installer. This cartridge implements a flow that consists of sending a JMS message to a remote system and receiving a JMS message in response. The order then ends.
- Configuration fragments for a project and an instance. These can be added to your project and instance specifications and contain all the SAF connection specifications as well as endpoint identification.
- A simple emulator that is available as a JAR file, along with instructions and configuration samples. This emulator can be set up on a WebLogic system outside the Kubernetes cluster and functions as a "remote system" in the SAF communication. The emulator simply echos the message given to it.

The SAF sample can be run as a separate project and instance, derived from the samples in the OSM cloud native toolkit. Alternatively, it can be added on to the specifications of a basic OSM instance. A project can consist of multiple cartridges. If you add the specifications to a basic OSM instance, the project consists of **SimpleRabbits** and **SimpleProvisioningCartridge**; instances of this project can consume both types of orders.

For the SAF sample, you need the following:

- A Linux host capable of running WebLogic Server 12.2.1.4 outside of the Kubernetes cluster.
- Traffic should be routable between the Kubernetes cluster and this host.
- If you are not using a centralized DNS resolution server, edit the /**etc/hosts** file of the Linux host to add resolution for your OSM cloud native instance. For example, use *kubernetes access IP address* **quick.sr.osm.org t3.quick.sr.osm.org admin.quick.sr.osm.org**.

For further details, see "Planning and Validating Your Cloud Environment".

Running the SAF sample involves the following tasks:

- Preparing the WebLogic system to run the emulator
- Deploying the emulator on the WebLogic system
- Deploying the SimpleProvisioning sample cartridge

- Preparing the OSM instance
- Validating the SAF endpoints
- Submitting OSM orders

Preparing the WebLogic System to Run the Emulator

Install WebLogic 12.2.1.4 on the Linux host. The specific patchset does not matter as long as it contains the patch referenced in "Applying the WebLogic Patch for External Systems".

To prepare the WebLogic system to run the emulator:

1. Start WebLogic server and create a domain accepting all the default settings. Do not enable JRF or any other Fusion MiddleWare capabilities for this sample. Name the domain `simple`.
2. Stop the WebLogic server and find the domain home for `simple`.
3. Edit the ***domain-home*/config/config.xml** file and delete the line: *admin-server-name*`AdminServer/`*admin-server-name*.
4. Locate and open the **samples/saf-sample/emulated-weblogic-resources/config/config_fragment.xml** configuration fragment XML file in the OSM cloud native toolkit.
5. Copy the contents under the `domain` element and append them to the end of the `domain` element in the ***domain-home*/config/config.xml** file just before `</domain>`.
   This creates a persistent store for JMS as well as a JMS server and a SAF agent. The SAF agent is used in sending emulator responses back to the OSM cloud native instance.
6. Copy the **samples/saf-sample/emulated-weblogic-resources/config/jms** folder in the toolkit to **<domain-home>/config**. This creates a folder `jms` under the target **config** directory with the specific JMS configuration. This also creates JMS queues and SAF entities.
7. Configure the SAF system to connect to your OSM cloud native instance. The instance does not need to be up at this point, but you should have decided on a project name, instance name, and the WebLogic username and password. If you want to reuse the basic OSM instance, you should already have these ready.
   Edit the **<domain-home>/config/jms/simple_osm_jms_module-jms.xml** file and update the fields underlined in the following fragment. The password is entered as plain text and gets auto-encrypted during WLS startup:

```
    <saf-login-context>
        <loginURL>{osm_cn_t3_url}</loginURL>
        <username>{osm_weblogic_username}</username>
        <password-encrypted>{osm_weblogic_password}</password-
encrypted>
    </saf-login-context>
```

   `osm_cn_t3_url` is:

   - If Oracle Cloud Infrastructure Load Balancer is not used: `http://t3.`*instance*.*project*`.osm.org:30305`

- If Oracle Cloud Infrastructure Load Balancer is used: `http://t3.`*instance*.*project*`.osm.org:80`

8. Start WebLogic. At this point, if you see errors from SAF/JMS about your OSM cloud native instance, you can ignore them. These errors go away once the OSM cloud native instance is up and configured for the SAF sample.

Deploying the Emulator on the WebLogic System

To deploy the emulator on the WebLogic system:

1. Find the **samples/saf-sample/emulator-mdb/emulator-mdb-1.0.0.jar** emulator MDB jar file in the OSM cloud native toolkit.
2. Open the WebLogic Console for the `simple` domain.
3. In Deployments, upload the emulator MDB jar file.
4. Complete the deployment using the defaults and ensure that the MDB file is shown with State "Active" and Health "OK".

Deploying the SimpleProvisioning Sample Cartridge

The **SimpleProvisioning** sample cartridge contains the following:

- **process_1** process
- A manual creation task
- An automation task with the following:
  - **qQuerySender** XQuery Sender
  - **Receiver** XQuery Automator

To deploy the **SimpleProvisioning** cartridge:

1. Identify a PDB for use with the SAF sample.
   This must be ready to host an OSM cloud native instance with RCU DB schema and OSM DB schema in place. You can use a fresh PDB and run the OSM cloud native DB Installer, or reuse or clone the PDB from the basic OSM cloud native instance. If you reuse the PDB in the basic OSM cloud native instance, you must use the basic OSM cloud native project and instance specification files in subsequent steps and delete the basic OSM cloud native instance.
2. Deploy the **SimpleProvisioning** cartridge using the script in the toolkit:

```
./scripts/manage-cartridges.sh -p project_name -i instance_name
-s $SPEC_PATH -f $OSM_CNTK/samples/saf-sample/cartridge-resources/
cartridge-par/SimpleProvisioning.par -c parDeploy
```

Preparing the OSM Cloud Native Instance

To prepare the OSM cloud native instance for the SAF sample:

1. Obtain a starter project specification. This can be the **samples/project.yaml** sample in the toolkit or you can reuse the project specification created for the basic OSM cloud native instance.

a. Configure a UDQ (SimpleResponseQueue) to receive the response from an external WebLogic domain by replacing the following line:

```
uniformDistributedQueues: {}
```

with the following:

```
uniformDistributedQueues:
  - name: SimpleResponseQueue
    jndiName: 
oracle.communications.ordermanagement.SimpleResponseQueue
    resetDeliveryCountOnForward: false
    deliveryFailureParams:
      expirationPolicy: Discard
      redeliveryLimit: 10
    deliveryParamsOverrides:
      timeToLive: -1
      priority: -1
      redeliveryDelay: 1000
      deliveryMode: 'No-Delivery'
```

If `uniformDistributedQueues` already exists in your ***project*.yaml** file, do not create a new element. Instead, append the item `SimpleResponseQueue` from the above snippet to the end of the existing list of items for `uniformDistributedQueues`.

b. Configure the SAF Queue (RequestQueue) by replacing the following line:

```
safDestinationConfig: {}
```

with the following:

```
safDestinationConfig:
  - name: simple
    destinations:
      - jndiPrefix: simple.
        queues:
          - queue:
             localJndi: RequestQueue
             remoteJndi: RequestQueue
```

The cartridge deployed for this sample uses this SAF queue to send messages to the external WebLogic domain.
If `safDestinationConfig` already exists in your ***project*.yaml** file, do not create a new element. Instead, append the item `simple` from above to the end of the existing list of items for `safDestinationConfig`.

2. Obtain a starter instance specification. This can be the **samples/instance.yaml** sample in the toolkit or you can reuse the instance specification created for your basic OSM instance.

   a. If you start with the **instance.yaml** sample, you must use your experience with creating a basic OSM cloud native instance to set up the DB server, NFS for logs (optional), authentication, and so on.

b. Configure the connection to the external OSM WebLogic domain by replacing the following line:

```
safConnectionConfig: {}
```

with the following:

```
safConnectionConfig:
  - name: simple
    t3Url: t3://{simple_weblogic_hostname}:{simple_weblogic_port}
    secretName: simplesecret
```

Replace the value of `{simple_weblogic_hostname}` and `{simple_weblogic_port}` with the hostname and port where `simple` WebLogic domain is installed. If `safConnectionConfig` already exists in your ***project-instance*.yaml**, do not create a new element. Instead, append the item `simple` from the above to the end of the existing list of items for `safConnectionConfig`.

3. Create a secret to contain the credentials for the `simple` WebLogic domain by running the following command. Name the secret as `simpleSecret` as specified in the above steps for the SAF connection and Replace the username and password with the values for the `simple` WebLogic domain.

```
kubectl -n project create secret generic simplesecret
--from-literal=username='simple_domain_weblogic_username' --from-
literal=password='simple_domain_weblogic_password'
```

4. Bring up the OSM cloud native instance. If you are not reusing the basic OSM instance, you will have to first create all the required secrets.
5. If you used a clone of the PDB of the basic OSM cloud native instance, you must replicate the `opssWF` and `opssWP` secrets from your basic OSM instance and set **rcu.db.preexisting** to **true** in your instance specification file. Failing to do this results in your new instance not being able to process the cloned PDB.
6. Once the OSM cloud native instance is up, do the following:
   a. Log in to the OSM Orchestration UI.
   b. Go to Administer Workgroups.
   c. Choose the OSM user you will be using to inject orders and add this user to the "SimpleProvisioningRole" workgroup.

   This allows your chosen user to create orders in the **SimpleProvisioning** cartridge.
   Both SAF endpoints, one on `simple` and one in this OSM cloud native instance should now be active. You can confirm this by validating the setup.

Validating the SAF Endpoints

To validate the SAF endpoints:

1. On the `simple` WebLogic domain, log in to the WebLogic console and do the following:

   a. Navigate to **Remote Endpoints**. You should see a remote endpoint called `simple_osm_saf_agent` with the URL pointing to your OSM cloud native instance.
   b. Navigate to **Deployments**. You should see the emulator MDB shown with State "Active" and Health "OK".
2. On the OSM cloud native instance, log in to the WebLogic console and navigate to **Remote Endpoints**. You should see the following remote endpoints pointing to the `simple` WebLogic domain:

   ```
   osm_simple_jms_module!osm_saf_destinations_simple.!<saf_queuex|
   saf_topicx>@osm_saf_agent@ms1
   ```

Submitting Orders

You can submit orders with HTTP and T3 over HTTP.

Submitting Orders with HTTP

To submit orders with HTTP:

1. Submit orders using the OSM Task Web Client or SoapUI:
   a. If you wish to use SoapUI, find the sample order payload for the **SimpleProvisioning** cartridge in the toolkit at **samples/saf-sample/cartridge-resources/CreateOrderBySpec.xml**.
   b. In the OSM Task Web client, create a new order of type **SimpleProvisioning**.
2. In the order data, find the "data" element and replace `MsgText` with a unique value.
3. Submit the order.
4. Examine the order in OSM Task Web Client of the OSM cloud native instance. It is very likely that the order completes very quickly and therefore does not appear in the Worklist. Use Query to look for completed orders and find the order. The completed order should show the response from the emulator.

If there are any issues with connectivity, the order does not complete successfully. Examine the message count on the queues in both OSM cloud native and in the **simple** WebLogic domain to see where the sequence was disrupted.

Submitting Orders with T3 over HTTP

To submit orders with T3 over HTTP, install SoapUI and HermesJMS and set them up to connect to your cloud native environment. SoapUI uses plain HTTP to submit orders. By using SoapUI with HermesJMS, orders can also be submitted as JMS messages using T3 over HTTP.

Consider the following when setting up SoapUI and HermesJMS:

- **Java 8**: If you use Java 8, you must find your Hermes installation location and the **hermes.sh** file within that location in the bin directory. Edit this file to replace the existing invocation of JAVACMD with the following:

```
"$JAVACMD" -
Dorg.xml.sax.parser=com.sun.org.apache.xerces.internal.parsers.SAXPa
rser
-
Djavax.xml.parsers.DocumentBuilderFactory=com.sun.org.apache.xerces.
internal.jaxp.DocumentBuilderFactoryImpl
-
Djavax.xml.parsers.SAXParserFactory=com.sun.org.apache.xerces.intern
al.jaxp.SAXParserFactoryImpl
-XX:NewSize=512m -Xmx2048m $HERMES_OPTS -
Dlog4j.configuration=file:$HERMES_HOME/bin/log4j.props
-Dhermes.home=$HERMES_HOME -Dhermes=$HERMES_CFG -
Dhermes.libs=$HERMES_LIB -classpath
$LOCALCLASSPATH hermes.browser.HermesBrowser
```

- **WebLogic Libraries**: When you create a session in HermesJMS preferences, create a Classpath Group that includes the WebLogic jar files `weblogic.jar`, `wlclient.jar` and `wlthint3client.jar`. These jar files are found in the standard WebLogic installation. Provide the full path to each jar file in the HermesJMS preferences.
- **Connection Properties**: When you set up a Connection Factory in your HermesJMS Session, add the following properties to it to point to your OSM cloud native instance:

Table 10-1 Connection Properties for HermesJMS Session

Property	Value
providerURL	http://t3.*instance.project*.osm.org:*access-port* *access-port* is the Traefik (ingress controller) NodePort or the Load Balancer port.
binding	oracle/communications/ordermanagement/osm/ExternalClientConnectionFactory
initialContextFactory	weblogic.jndi.WLInitialContextFactory
securityPrincipal	*osm-user-name*
securityCredentials	*password-for-osm-user-name*

With these in place, you should now be able to discover the JMS queues and topics from your OSM cloud native instance

- **Target JMS Queue**: Add a JMS endpoint using the Session created. Specify the "Send/Publish Destination" as `oracle/communications/ordermanagement/WebServiceQueue`. If you wish to see the responses, specify the "Receive/Subscribe Destination" as `oracle/communications/ordermanagement/SoapUIResponseQueue`. You need to have first specified this response queue as an additional queue in your project specification.

HermesJMS submits the order requests into the OSM cloud native `WebServiceQueue` distributed queue and optionally shows you responses in the `SoapUIResponseQueue` distributed queue.

- **SoapUI Test Case**: When you create a test case with the sample order payload, do the following:
  - Choose Basic authentication and specify the *osm-user-name* and *password-for-osm-user-name* as earlier.
  - If you use responses, set the `JMSReplyTo` JMS Property to `oracle/communications/ordermanagement/SoapUIResponseQueue`
  - Add "JMS Headers" properties with name-value as:

```
_wls_mimehdrContent_Type : text/xml; charset="utf-8"
URI                      : /osm/wsapi
```

This setup can now submit orders into the OSM cloud native instance as JMS messages. The SoapUI project and configuration can be saved to serve as a template for future reuse.

11

Upgrading the OSM Cloud Native Environment

This chapter describes the tasks you perform in order to apply a change or upgrade to a component in the cloud native environment.

Creating a detailed upgrade plan can be a complex process. It is useful to start by mapping your use case to an upgrade path. These upgrade paths identify a set of sequenced activities that align to a CD stage. Once you know the activity sequence, you can then look for the detailed steps involved in each to come up with the comprehensive set of steps to be performed.

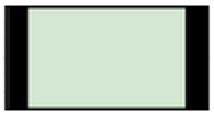
Upgrade paths consist of activities that fall into the following two main categories:

- Operational Procedures
- Component Upgrade Procedures

Operational Procedures

There are many different operational procedures and all of these affect the operating state of OSM. OSM cloud native provides the mechanism to change the operational state as described in "Running Operational Procedures".

The flowcharts in this chapter use the following image to depict an operational procedure:



Component Upgrade Procedures

These are the actual set of steps to perform a component upgrade and can be one of the following types:

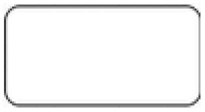
- **OSM Cloud Native Procedures**: OSM cloud native owns the component and therefore the upgrade procedure for that component. OSM cloud native provides the mechanism to perform the upgrade via the scripts that are bundled with the OSM cloud native toolkit.
  An example of this is a change to a value in an OSM cloud native specification file (shape, project, and instance).

  The flowcharts in this chapter use the following image to depict an OSM cloud native owned procedure.

- **External Procedures**: These procedures are for components that are part of the OSM cloud native operating environment, but are out of the control of OSM cloud native. OSM cloud native does not determine how to apply the upgrade,

but provides recommendations on the operational state of OSM accompanying the upgrade.
An example would be updating the operating system on a worker node.

The flowcharts in this chapter use the following image to depict an external upgrade procedure.



- Miscellaneous upgrade procedures: There are some procedures that require special handling and are not captured in any of the upgrade paths. These are described in "Miscellaneous Upgrade Procedures".

Rolling Restart

Occasionally, you may need to restart OSM managed servers in a rolling fashion, one at a time. This does not result in downtime, but only reduced capacity for a limited period. A rolling restart can be triggered by invoking the **restart-instance.sh** script. This script can restart the whole instance in a rolling fashion, or only the admin server or all the managed servers in a rolling fashion. Some operations may automatically trigger rolling restart. These include online cartridge deployment and certain changes (image updates, tuning parameter changes, and so on) pushed via the **upgrade-instance.sh** script.

Identifying Your Upgrade Path

In order to prepare your detailed plan for an upgrade, you need to be able to map your upgrade use case to an upgrade path. Some common use cases are detailed in the following charts. If your use case is not listed, see "Upgrade Path Flow Chart", which guides you through the decision making process to prepare a specific upgrade path.

Table 11-1 Common Upgrade Paths

Upgrade Type	Component	Upgrade Path	Requires Changing Image?
Cartridge Management	Deploy new cartridge version	Online change, online cartridge deployment OR Offline change, offline cartridge deployment	No
Cartridge Management	Redeploy a cartridge against an existing cartridge version	Offline change, offline cartridge deployment	No
Cartridge Management	Fast undeploy cartridge version	Offline change, offline cartridge deployment OR Online change, online cartridge deployment	No
Cartridge Management	Purge cartridge version	Online Change, external procedure, Manual restart	No

Table 11-1 (Cont.) Common Upgrade Paths

Upgrade Type	Component	Upgrade Path	Requires Changing Image?
Configuration and Tuning	OSM cluster size (scaling up or down)	Online change, application upgrade	Not applicable
Configuration and Tuning	Java parameters (memory, GC, and so on)	Online change, application upgrade	Not applicable
Configuration and Tuning	WebLogic domain configuration (WDT such as JMS Queue configuration)	Online change, application upgrade	No
Configuration and Tuning	OSM configuration parameters (traditionally, oms-config.xml)	Online change, application upgrade	No
Database Storage Management	Create partition and clone database statistics	Offline Change, PDB upgrade	No
Database Storage Management	Online row-based order purge	Online Change, external procedure	No
Database Storage Management	Purge partition	Online Change, external procedure	No
Security parameters	New, renamed or deleted secrets passed to cartridges	Online change, application upgrade	No
Security parameters	Secrets value (For example, changing password)	Online change, external procedure, Manual restart	No
Software Upgrade and Patching	OSM release or patch upgrade with Database change	Offline change, PDB upgrade	Yes
Software Upgrade and Patching	Fusion MiddleWare upgrade	Online change, application upgrade (some exceptions needing offline change)	Yes
Software Upgrade and Patching	OSM patch upgrade without Database change	Online Change, application upgrade (some exceptions needing offline change)	Yes
Software Upgrade and Patching	Fusion MiddleWare overlay patches (for example, PSU or one-off patch)	Online Change, application upgrade (some exceptions needing offline change)	Yes
Software Upgrade and Patching	Java upgrade	Online Change, application upgrade	Yes
Software Upgrade and Patching	Linux	Online Change, application upgrade	Yes
Software Upgrade and Patching	Custom code or third-party tool (custom image)	Online Change, application upgrade (some exceptions needing offline change)	Yes
Software Upgrade and Patching	OSM cloud native toolkit	The release dictates the constraints.	Not applicable
Shared infrastructure	Operating system or hardware on worker node	Online change, external procedure	No

Table 11-1 (Cont.) Common Upgrade Paths

Upgrade Type	Component	Upgrade Path	Requires Changing Image?
Shared infrastructure	Docker	Online change, external procedure	No
Shared infrastructure	WebLogic Operator minor upgrade (backward compatible)	Online change, external procedure	No
Shared infrastructure	WebLogic Operator major upgrade (non-backward compatible)	Online change, external procedure	No

Once you understand the activities in your upgrade path, you can begin to map out the sequence of activities that you need to perform.

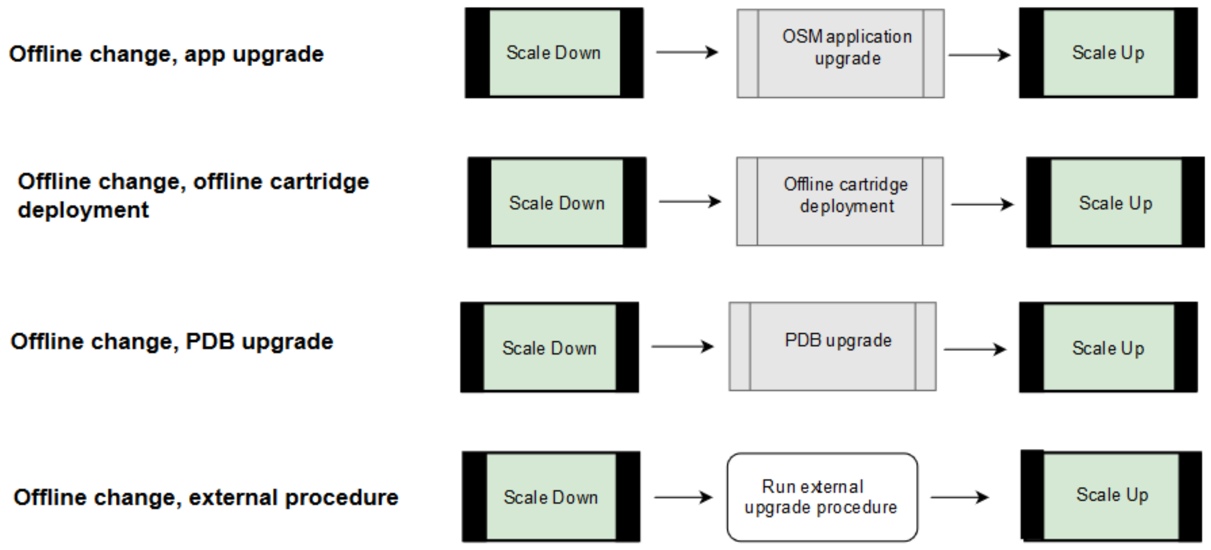
Offline Change Upgrade Paths

Offline changes are defined as those requiring OSM to be shutdown before the change can be applied.

All offline upgrades must start with a Scale Down procedure and end with a Scale Up procedure. You can find the explicit steps to perform these activities in Running Operational Procedures.

Once the cluster has been scaled down, you will need to perform either an external procedure (referencing documentation for the component) or follow an OSM cloud native owned procedure. See "OSM Cloud Native Upgrade Procedures" for details.

Figure 11-1 Offline Change Upgrade Paths


Offline change, app upgrade
Scale Down
OSM application upgrade
Scale Up
Offline change, offline cartridge deployment
Scale Down
Offline cartridge deployment
Scale Up
Offline change, PDB upgrade
Scale Down
PDB upgrade
Scale Up
Offline change, external procedure
Scale Down
Run external upgrade procedure
Scale Up

As an example, if your use case is to re-deploy an existing cartridge version, then the upgrade path would be "Offline change, offline cartridge deployment", the second flow in the above flow chart. The actual steps involve the following:

- Scale Down
  - Edit the instance specification file to set cluster size to **0**.
  - Run **upgrade-instance.sh**.
- Offline cartridge deployment
  - Edit the project specification file to change the cartridge version.
  - Run **manage-cartridges.sh** with option **sync**.
- Scale Up
  - Edit the instance specification file to return cluster size to original (1-18).
  - Run **upgrade-instance.sh**.

Online Change Upgrade Paths

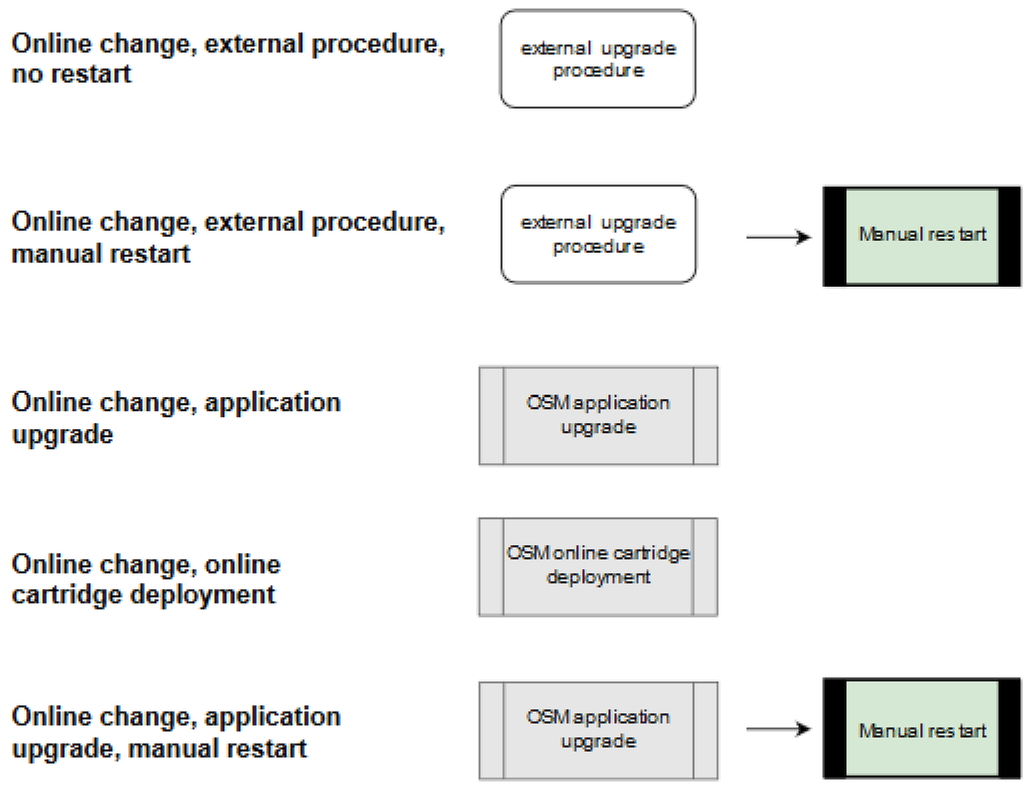
Online changes are changes for which OSM can remain running while the component upgrade is performed. There is, therefore, no operational procedure at the start of the flow, but some paths include a rolling restart after the upgrade procedure is performed.

The component upgrade will either be an external procedure (referencing documentation for the component) or follow an OSM cloud native owned procedure described in "OSM Cloud Native Upgrade Procedures".

If explicit post-upgrade operational activities are required, you can find details in "Running Operational Procedures".

The following flowchart illustrates online change upgrade paths:

Figure 11-2 Online Change Upgrade Paths


Online change, external procedure, no restart
external upgrade procedure
Online change, external procedure, manual restart
external upgrade procedure
Manual restart
Online change, application upgrade
OSM application upgrade
Online change, online cartridge deployment
OSM online cartridge deployment
Online change, application upgrade, manual restart
OSM application upgrade
Manual restart

Exceptions

The following require shutdown:

- Some OSM patches
- Some Oracle Fusion MiddleWare overlay patches
- Some custom code or 3rd party
- Oracle Fusion MiddleWare version upgrades

Unsupported Tasks

Adding, modifying, and deleting users or groups from embedded LDAP are not supported through an upgrade procedure.

To make changes to users and groups, the instance must be deleted and re-created.

OSM Cloud Native Upgrade Procedures

The OSM cloud native owned upgrade procedures are:

- PDB upgrade
- OSM application upgrade
- Online cartridge deployment

- Offline cartridge deployment

Change or upgrade procedures that are dictated by OSM cloud native are applied using the scripts and the configuration provided in the toolkit.

PDB Upgrade Procedure

Changes impacting the PDB can be found in any of the specification files - project, instance or shape.

Examples include updating the OSM DB Installer image.

To perform a PDB upgrade procedure:

1. Make the necessary modifications in your specification files.
2. Invoke **$OSM_CNTK/scripts/install-osmdb.sh** with the command appropriate for your use case.
   To see a list of options, invoke with **-h**.

OSM Application Upgrade

Changes impacting the OSM application can be found in any of the specification files - project, instance or shape.

Examples include changing an existing value, changing the OSM image or supplying something new such as a secret or a new WDT extension.

To perform OSM application upgrade:

1. Make the necessary modifications in your specification files.
2. Invoke **$OSM_CNTK/scripts/upgrade-instance.sh** to push out the changes you just made to the running instance. This also triggers introspection for upgrade paths where introspection is required.
3. In upgrade paths where a manual restart is required, restart the instance. See "Restarting the Instance" for details.

Offline Cartridge Deployment

Offline deployment mode supports deployment of new cartridges, deployment of new versions of existing cartridges, fast undeploy and re-deployment of existing cartridge versions with changes.

Changes impacting the cartridges can be found in the project specification file.

In order to perform an offline deployment, you must not have managed servers running.

To perform an offline cartridge deployment:

1. Scale down your managed server count. See “Scaling Down the Cluster” for more details.
2. Deploy the cartridges:
   - Make the necessary modifications in your project specification.

- Run the following command:

```
$OSM_CNTK/scripts/manage-cartridges.sh -p project -i instance -s
spec_Path –c sync
```

3. Scale up your managed server count. See "Scaling Up the Cluster" for more details.

Online Cartridge Deployment

Online deployment mode supports deployment of new cartridges, deployment of new versions of existing cartridges and fast undeploy. It does not support re-deployment of existing cartridges.

The changes impacting the cartridges can be found in the project specification file.

In order to perform an online deployment, you must have a minimum of two managed servers running.

To perform an online cartridge deployment:

1. If necessary, scale up your managed server count (2 or more). See "Scaling Up the Cluster" for more details.
2. Deploy the cartridges:
   - Make the necessary modifications in your project specification.
   - Invoke the following script:

```
$OSM_CNTK/scripts/manage-cartridges.sh -p project -i instance -s
spec_Path -c sync -o
```

> **Note:**
>
> If the changes to the cartridges in the project specification include more than one kind of update (new cartridge, new version, existing version, undeploy), if it includes redeploy of existing versions, then you must use offline cartridge deployment. Alternatively, if possible, break up the operational activity into two parts: one set of changes that satisfy the online deployment and then following that, a second set with all the cartridge redeployment changes to be done offline.

Upgrades to Infrastructure

From the point of view of OSM instances, upgrades to the cloud infrastructure fall into two categories: rolling upgrades and one-time upgrades.

> **Note:**
>
> All infrastructure upgrades must continue to meet the supported types and versions listed in the OSM documentation's certification statement.

Rolling upgrades are where, with proper high-availability planning (like anti-affinity rules), the instance as a whole remains available as parts of it undergo temporary outages. Examples of this are Kubernetes worker node OS upgrades, Kubernetes version upgrades and Docker version upgrades.

One-time upgrades affect a given instance all at once. The instance as a whole suffers either an operational outage or a control outage. Examples of this are WebLogic Operator upgrade and perhaps Ingress Controller upgrade.

Kubernetes and Docker Infrastructure Upgrades

Follow standard Kubernetes and Docker practices to upgrade these components. The impact at any point should be limited to one node - master (Kubernetes and OS) or worker (Kubernetes, OS, and Docker). If a worker node is going to be upgraded, drain and cordon the node first. This will result in all pods moving away to other worker nodes. This is assuming your cluster has the capacity for this - you may have to temporarily add a worker node or two. For OSM instances, any pods on the cordoned worker will suffer an outage until they come up on other workers. However, their messages and orders are redistributed to surviving managed server pods and processing continues at a reduced capacity until the affected pods relocate and initialize. As each worker undergoes this process in turn, pods continue to terminate and start up elsewhere, but as long as the instance has pods in both affected and unaffected nodes, it will continue to process orders.

WebLogic Operator Upgrade

To upgrade the WebLogic Operator, follow the Operator documentation. As long as the target version can co-exist in a Kubernetes cluster with the current version, a phased cutover can be performed. In this, you will perform a fresh install of the new version of the Operator into a new namespace. RBAC will be arranged here, identical to your existing Operator namespace. Once the new Operator is functioning, for each OSM cloud native project, un-register it from the old Operator and register it with the new Operator. This can be done at your convenience on a per-project basis. When all projects have been switched to the new Operator, the old Operator can be safely deleted.

```
export WLSKO_NS=old-namespace $OSM_CNTK/scripts/unregister-namespace -p
project -t wlsko
export WLSKO_NS=new-namespace $OSM_CNTK/scripts/register-namespace -p
project -t wlsko
```

All instances with the transitioned project are impacted by this operation. However, there is no order processing outage during the transition. There is a control outage - where no changes can be pushed to the instances (**upgrade-instance.sh** or **delete-instance.sh**). Also, during the control outage, the termination of a pod does not immediately trigger healing. However, once the transition of the project is complete, the new Operator will react to any changed state (whether in the cluster, like pod termination, or in pushed changes, like instance upgrades) and run the required actions.

Ingress Controller Upgrade

Follow the documentation of your chosen Ingress Controller to perform an upgrade. Depending on the Ingress Controller used and its deployment in your Kubernetes environment, the OSM instances it serves may see a wide set of impacts, ranging from no impact at all (if the Ingress Controller supports a clustered approach and can be upgraded that way) to a complete outage.

To take the sample of Traefik that OSM cloud native toolkit uses as an Ingress Controller illustration:

An approach identical to that of WebLogic Operator upgrade can be followed for Traefik upgrade. The new Traefik can be installed into a new namespace, and one-by-one, projects can be unregistered from the old Traefik and registered with the new Traefik.

```
export TRAEFIK_NS=old-namespace $OSM_CNTK/scripts/unregister-namespace
-p project -t traefik
export TRAEFIK_NS=new-namespace $OSM_CNTK/scripts/register-namespace -p
project -t traefik
```

During this transition, there will be an outage in terms of the outside world interacting with OSM. Any data that flows through the ingress will be blocked until the new Traefik takes over. This includes GUI traffic, order injection, API queries, and SAF responses from external systems. This outage will affect all the instances in the project being transitioned.

Miscellaneous Upgrade Procedures

This section describes miscellaneous upgrade scenarios.

Network File System (NFS)

If an instance is created successfully, but a change to the NFS configuration is required, then the change cannot be made to a running OSM instance. In this case, the procedure is as follows:

1. Perform a fast delete. See "Running Operational Procedures" for details.
2. Update the `nfs` details in the instance specification.
3. Start the instance.

Running Operational Procedures

This section describes the tasks you perform on the OSM server in response to a planned upgrade to the OSM cloud native environment. You must consider if the change in the environment fundamentally affects OSM processing to the extent that OSM should not run when the upgrade is applied or OSM can run during the upgrade but must be restarted to properly process the change.

The operational procedures are performed using the OSM cloud native specification files and scripts.

The operational procedures you perform for upgrading your cloud environment are:

- Trigger introspection
- Scaling down the cluster
- Scaling up the cluster
- Restarting the cluster
- Fast delete
  - Shutting down the cluster

- Starting up the cluster

Triggering Introspection

When any of the specification files have changed, invoke the **upgrade-instance.sh** script to trigger the operator's introspector to examine the change and apply it to the running instance.

Scaling Down the Cluster

The scaling down procedure described here is only in the context of the upgrade flow diagram. Hence, scaling down is down to 0 managed servers. A generalized scaling can change the cluster size down to a value between 0 and 18 (both inclusive) in any desired increment or decrement.

To scale down the cluster, edit the instance specification and change the `clusterSize` parameter to `0`. This terminates all the managed server pods, but leaves the admin server up and running.

Apply the change to the running Helm release by running the upgrade script:

```
$OSM_CNTK/scripts/upgrade-instance.sh -p project -i instance -s
$SPEC_PATH
```

Scaling Up the Cluster

The scaling up procedure described here is only in the context of the upgrade flow diagram. Hence, scaling up is up to the initial cluster size. A generalized scaling can change the cluster size up to a value between 0 and 18 (both inclusive) in any desired increment or decrement.

To scale up the cluster, edit the instance specification and change the value of the `clusterSize` parameter to its original value to return the cluster to its previous operational state.

Apply the change to the running Helm release by running the upgrade script:

```
$OSM_CNTK/scripts/upgrade-instance.sh -p project -i instance -s
$SPEC_PATH
```

Restarting the Instance

The OSM cloud native toolkit provides a script (**restart-instance.sh**) that you can use to perform different flavors of restarts on a running instance of OSM cloud native.

Following is the usage of the **restart-instance.sh** script

```
restart-instance.sh parameters
      -p projectName : mandatory
      -i instanceName : mandatory
      -s specPath : mandatory; locations of specification files
      -m customExtPath : optional; locations of custom extension files
      -r restartType : mandatory; what kind of restart is requested
```

```
    # specPath and customExtPath take a colon(:) delimited list of
directories
    # restartType can take the following values:
      * full: Restarts the whole instance (rolling restart)
      * admin: Restarts the WebLogic Admin Server only
      * ms: Restarts all the Managed Servers (rolling restart)

    # or just -h for help
```

For example, to restart a complete cluster, run the following command:

```
$OSM_CNTK/scripts/restart-instance.sh -p project -i instance -s
$SPEC_PATH -r full
```

Fast Delete

When the entire domain, including the admin server, needs to be taken offline, then the full shutdown and full startup procedures follow. This can be used to perform a "fast delete" or "dehydration" of the domain, instead of a full **delete-instance** operation where you may have to be concerned about the secrets and other pre-requisites being deleted. To quickly restore the domain, simply perform the startup procedure.

Shutting Down the Cluster

To shut down the cluster, edit the instance specification and add or modify the value of the `serverStartPolicy` parameter to **NEVER**. This terminates all the pods.

```
# Operational control parameters
# scope - domain or cluster
serverStartPolicy: NEVER
```

Apply the change to the running Helm release by running the upgrade script:

```
$OSM_CNTK/scripts/upgrade-instance.sh -p project -i instance -s
$SPEC_PATH
```

Starting Up the Cluster

To start up the cluster, edit the instance specification and comment out or modify the value of the `serverStartPolicy` parameter to **IF_NEEDED**. This starts up all the pods.

```
# Operational control parameters
# scope - domain or cluster
serverStartPolicy: IF_NEEDED
```

Apply the change to the running Helm release by running the upgrade script:

```
$OSM_CNTK/scripts/upgrade-instance.sh -p project -i instance -s
$SPEC_PATH
```

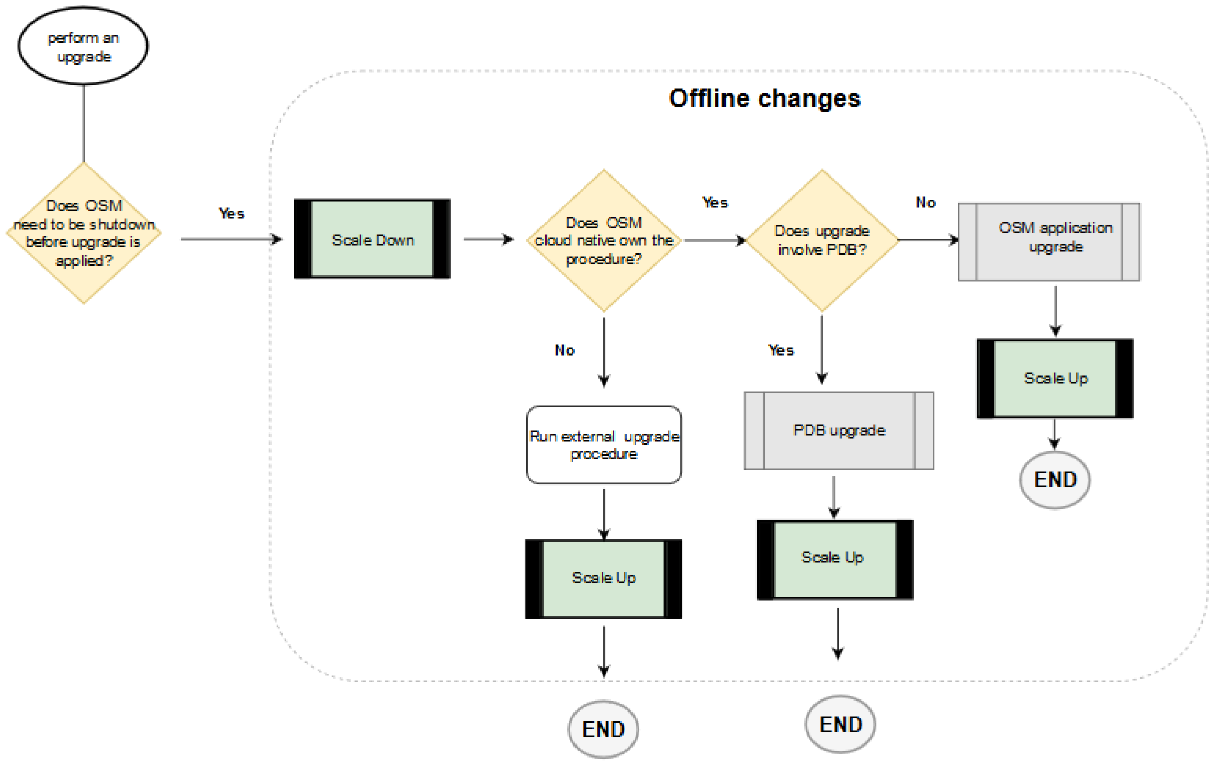
Upgrade Path Flow Chart

When comparing and contrasting the different flows, identifying common steps or divergences, it can be useful to have a combined view of the flowcharts along with the main decision points. This can be useful when trying to automate parts of the process.

The first decision to make is whether OSM can be running when you apply the change. Typically, OSM needs to be shutdown for PDB impacting scenarios and the exceptions listed in the "Exceptions" section.

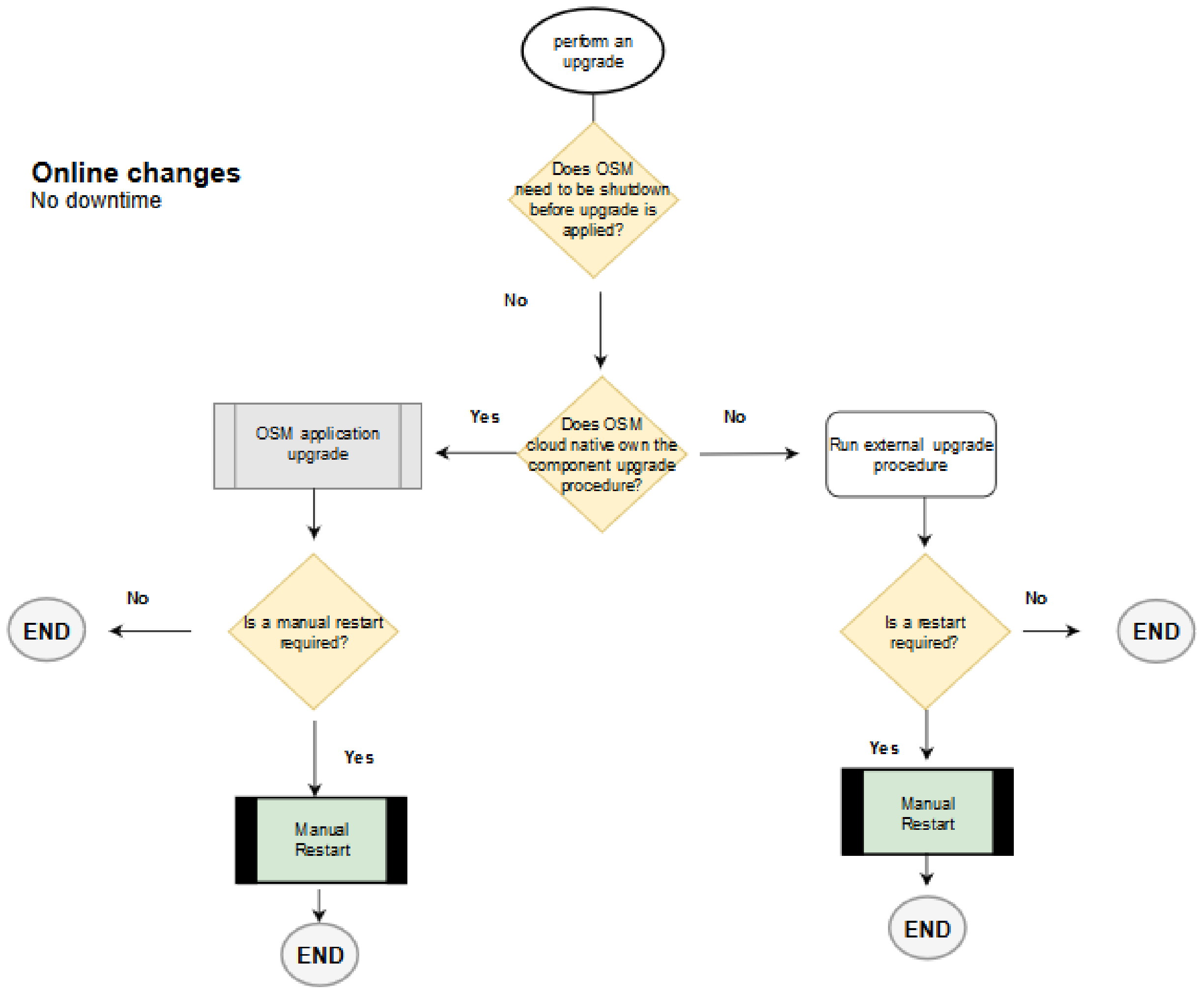
The following flowchart illustrates the flow for offline upgrades and various scenarios.

Figure 11-3 Upgrade Path Flow for Offline Changes


perform an upgrade
Does OSM need to be shutdown before upgrade is applied?
Yes
Offline changes
Scale Down
Does OSM cloud native own the procedure?
Yes
Does upgrade involve PDB?
No
OSM application upgrade
No
Yes
Scale Up
Run external upgrade procedure
PDB upgrade
END
Scale Up
Scale Up
END
END

The following flowchart illustrates the flow for online upgrades and various scenarios.

Figure 11-4   Upgrade Path Flow for Online Changes


Online changes
No downtime
perform an upgrade
Does OSM need to be shutdown before upgrade is applied?
No
Does OSM cloud native own the component upgrade procedure?
Yes
No
OSM application upgrade
Run external upgrade procedure
Is a manual restart required?
No
END
Yes
Manual Restart
END
Is a restart required?
No
END
Yes
Manual Restart
END

12

Moving to OSM Cloud Native from a Traditional Deployment

You can move to an OSM cloud native deployment from your existing OSM traditional deployment. This chapter describes tasks that are necessary for moving from a traditional OSM deployment to an OSM cloud native deployment.

Supported Releases

You can move to OSM cloud native from all supported traditional OSM releases. In addition, you can move to OSM cloud native within the same release, starting with OSM release 7.4.1.0.1.

Performing Pre-move and Post-move Tasks

Some OSM releases require running some tasks before and after moving to OSM cloud native. These tasks are described in the documentation of the target version of the OSM cloud native release. As mentioned in the documentation, before and after moving to OSM cloud native, perform these tasks.

Some of the patch readme files describe potential error conditions and workarounds listed in the **Known Issues with Workaround** section. Monitor and apply these too if required. An example of this (documented in the 7.3.5.1.x Patch Readmes) is the error condition where OM_DB_STATS_PKG remains in an invalid state. If you encounter this issue, apply the appropriate workaround to grant the required permissions and rebuild the package.

About the Move Process

The move to OSM cloud native involves offline preparation as well as maintenance outage. This section outlines the general process as well as the details of the steps involved in the move to OSM cloud native. However, there are various places where choices have to be made. It is recommended that a specific procedure be put together after taking into account these choices in your deployment context.

The OSM cloud native application layer runs on different hardware locations (within a Kubernetes cluster) than the OSM traditional application layer.

The process of moving to OSM cloud native involves the following sets of activities:

- Pre-move development activities, which include the following tasks:
  - Building OSM cloud native images (cloud native task)
  - Creating project specification OSM cloud native (cloud native and solution task)
  - Creating instance specification OSM cloud native (cloud native and solution task)

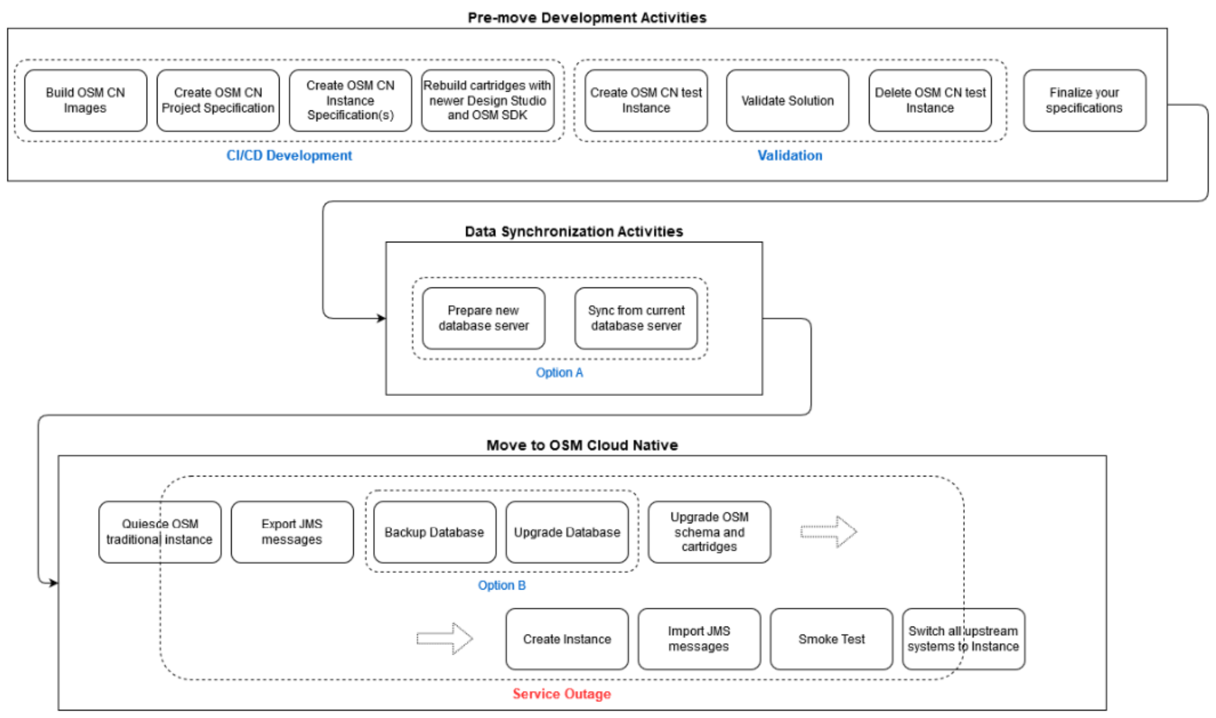
  - Rebuilding cartridges using Design Studio and OSM SDK (solution task)
  - Creating an OSM cloud native instance for testing (cloud native task)
  - Validating your solution cartridges (solution task)
  - Deleting the test OSM cloud native instance (cloud native task)
  - Finalizing your specifications (cloud native and solution task)
- Data synchronization activities, which include the following tasks:
  - Preparing a new database server (database task)
  - Synchronizing the current database server (database task)
- Tasks for moving to OSM cloud native, which include the following:
  - Quiescing the OSM traditional instance (solution task)
  - Exporting JMS messages (WebLogic Server administration task)
  - Backing up the database (database task)
  - Upgrading the database (database task)
  - Upgrading the OSM schema and cartridges (database task)
  - Creating an OSM cloud native instance (cloud native task)
  - Importing JMS messages (WebLogic Server administration task)
  - Performing a smoke test (solution task)
  - Switching all upstream systems (solution task)

The following diagram illustrates these activities.

Note:

In the diagram, the short form of "OSM CN" stands for "OSM cloud native".

Figure 12-1 Move to OSM Cloud Native Process


Pre-move Development Activities
Build OSM CN Images
Create OSM CN Project Specification
Create OSM CN Instance Specification(s)
Rebuild cartridges with newer Design Studio and OSM SDK
CI/CD Development
Create OSM CN test Instance
Validate Solution
Delete OSM CN test Instance
Validation
Finalize your specifications
Data Synchronization Activities
Prepare new database server
Sync from current database server
Option A
Move to OSM Cloud Native
Quiesce OSM traditional instance
Export JMS messages
Backup Database
Upgrade Database
Option B
Upgrade OSM schema and cartridges
Create Instance
Import JMS messages
Smoke Test
Switch all upstream systems to Instance
Service Outage

Pre-move Development Activities

In preparation to move your traditional OSM instance into an OSM cloud native environment, you must do the following activities:

1. Build OSM cloud native images. This task includes creating the OSM Docker image and the DB Installer Docker image by using the OSM cloud native download packages. See "Creating OSM Cloud Native Images" for details.
2. Analyze your solution and create a project specification for your OSM cloud native instance. This specification includes details of JMS queues and topics, as well as SAF connections. See "Configuring the Project Specification" for details. If your solution requires model extensions or custom files, create the additional YAML files for those as well.
3. Create an instance specification for your OSM cloud native instance. Preferably, create a test instance, pointing to a test PDB. You can later change this specification to point to the migrated database. Similarly, any SAF endpoint details should be pointing to the test components or emulators. When creating the specification, choose your cloud native production shape - prodsmall, prod, prodlarge. Alternatively, create a custom production shape by copying and modifying one of these. See "Creating Custom Shapes" for details about custom shapes.
4. If your OSM cartridges were built against an OSM deployment that is older than 7.3.5, use Design Studio to rebuild them with the OSM SDK of the target release. Select the Design Studio version based on its compatibility matrix.

5. Create an OSM cloud native test instance and test your specifications. To do this, create your cartridge users document and follow the process (create instance secrets, install the RCU schema, install the OSM schema, deploy your cartridges, bring up OSM, create ingress, and run test orders) to bring up the OSM cloud native instance as described in "Creating a Basic OSM Instance".
6. Validate the solution.
7. Shut down your test instance and remove the associated secrets, PDB, and ingress.
8. Finalize your specifications for the move by picking up any changes from your test activity and re-create instance secrets to use the migrated database. Change the instance specification to:
    a. Point to the migrated database location once it is known.
    b. Switch SAF endpoints to the actual components, instead of emulators.
    c. Arrange for the same number of managed servers in your OSM cloud native instance.

        OSM cloud native requires the use of standard sizing for managed servers. This is represented as a set of "shapes". As a result, it is possible that your OSM cloud native instance needs a different number of managed servers to handle your workload as compared to your OSM traditional instance. For the purpose of this migration activity, it is recommended to start with the same number of managed servers, perform the import and smoke tests, and then scale (scale-up or scale-down) the OSM cloud native instance to the desired size.

        If it is not possible to arrange for the same number of managed servers in your OSM cloud native instance as there are in your OSM traditional instance, it is recommended that you get as close as you can. You can then import the JMS messages from the leftover managed servers into one of the OSM cloud native managed servers. For example, consider an OSM traditional instance with four managed servers (**ms1**, **ms2**, **ms3**, and **ms4**). The analysis may show that you only need two managed servers (**cn-ms1** and **cn-ms2**) of prod shape in your OSM cloud native instance. You can import all JMS messages from **ms1** into **cn-ms1**, and from **ms2** into **cn-ms2**. And then, import the remaining messages from **ms3** to **cn-ms1** and from **ms4** to **cn-ms2**. Try to spread the load as evenly as possible.

Moving to an OSM Cloud Native Deployment

Moving to an OSM cloud native deployment from an OSM traditional deployment requires performing the following tasks:

1. Quiesce the OSM traditional instance. See "Quiescing the Traditional Instance of OSM".
2. Export JMS messages. See "Exporting and Importing JMS Messages".
3. Take a back up and upgrade the database. See "Upgrading the Database".
4. Upgrade the OSM schema and cartridges. See "Upgrading the OSM Schema and Cartridges".
5. Create the OSM cloud native instance. See "Creating Your Own OSM Cloud Native Instance".

6. Import JMS messages. See "Importing JMS Messages".
7. Perform a smoke test. See "Performing a Smoke Test". Once the OSM cloud native instance passes smoke test and is optionally resized to the desired target value, shut down the OSM traditional instance fully.
8. Switch all upstream systems to the OSM cloud native instance. See "Switching Integration with Upstream Systems".

Quiescing the Traditional Instance of OSM

At the start of the maintenance window, the OSM traditional instance must be quiesced. This involves stopping database jobs, stopping all upstream and peer systems from sending messages (for example, http/s, JMS, and SAF) to OSM, and ensuring all human users are logged out. It also involves pausing the JMS queues so that no messages get queued or dequeued. The result is that OSM is up and running, but completely idle.

Exporting and Importing JMS Messages

Irrespective of the persistence mechanism you use (file-based or JDBC) in your OSM traditional instance, you must still export and import outstanding messages as described in this section. If file-based persistence is used, this procedure accomplishes a switch to JDBC-based persistence. On the other hand, if JDBC-based persistence is already in use, this procedure brings the setup (in WebLogic and in the database) in line with OSM cloud native requirements.

Overall, this procedure consists of exporting the JMS messages to disk, switching over to the OSM cloud native instance, and importing the JMS messages from disk. Due to the configuration in the OSM cloud native instance, the imported messages will get populated into the appropriate database tables of the OSM cloud native instance rather than their original location. The time taken for the export and import depends on the number of messages that are in the persistent store to begin with.

See the following topics for further details:

- Exporting JMS Messages
- Importing JMS Messages

Exporting JMS Messages

Before exporting JMS messages, validate that your OSM traditional instance has the WebLogic patch 31169032 (or its equivalent for your WebLogic version) installed. This patch is required to properly export OSM JMS messages. If it is not installed, follow the standard WebLogic patch procedures to procure and install the patch.

To export JMS messages:

1. Login to the WebLogic Console and navigate to the list of OSM queues.
2. For each queue, open its Monitoring tab to get the list of current destinations for the queue. The Monitoring tab shows as many destinations as the number of managed servers.
3. Select each destination and click **Show Messages**. If there are any messages pending in this destination of this queue, click the **Export** button to export all the

messages to a file. Use the queue name and destination in the filename for ease of tracing.
Future-dated orders result in pending messages in `OrchestrationDependencyQueue`.

4. If you have defined other JMS Modules as part of your solution, repeat steps 2 and 3 for each of those modules.

Importing JMS Messages

Before importing JMS messages, ensure that your OSM cloud native instance is up and running, but quiesced (queues paused and database jobs stopped). It is recommended that your OSM cloud native instance has the same number of managed servers as your OSM traditional instance.

To import JMS messages:

1. Transfer all the exported files into the Admin Server pod using the **kubectl cp** command.
2. Log in to the WebLogic Console and navigate to JMS Modules where the OSM queues are listed.
3. For each queue for which you have an export file, open its Monitoring tab.
4. For each destination on this queue for which you have an export file, find the same destination in the list
5. Select the destination and click **Show Messages**. Click **Import** to specify the filename and import the messages.
6. If your export contains files that came from a custom JMS module in your OSM traditional instance, you should still see those queues in `osm_jms_module` in your OSM cloud native instance. If you do not, check that your project specification is up to date.

Upgrading the Database

To upgrade the database, you perform the following tasks:

- Upgrading the Database Server
- Preparing the Required Database Entities for OSM Cloud Native

Upgrading the Database Server

You may need to upgrade the database server itself to the version supported by the OSM cloud native release you are moving to. To identify the required version of the database server and to determine if you need a database server upgrade, see *OSM Compatibility Matrix*.

If you do need a database server upgrade, choose one of the following options:

- **Option A: Create an additional database server**: Create a second database server of the target database version (with required patches), seeded with an RMAN backup of the OSM traditional database. Enable Oracle Active DataGuard to continuously synchronize data from the OSM traditional database to this new database. Use this new database for the OSM cloud native instance. For further details, see *Mixed Oracle Version support with Data Guard Redo Transport*

Services (Doc ID 785347.1) knowledge article on My Oracle Support. The exact mechanisms to be used are subject to circumstances such as resource availability, size of data, timing, and so on but the goal is to have a second database server running the target database version but always containing the data from the OSM traditional database.
This option has the following advantages:

  - Allows switching from a standalone database to a Container DB and Pluggable DB model that is required for OSM cloud native, without impacting other users of the existing database.
  - Reduces the duration of a service outage since you can avoid having to backup the database and upgrade it as part of the maintenance window.
  - Preserves the OSM traditional database unchanged reducing the risk and cost associated with reverting to OSM traditional instance, if that becomes necessary.

- **Option B: Retain the existing DB server**: You can retain the existing database server, upgrading it in-place to the target database version and patches.

If you choose option A, the upgrade process must pause after the export of JMS messages, and ensure the Active DataGuard sync is complete (if there are pending redo logs). Then, before proceeding, the sync must be turned off and the new database must be brought online fully.

Preparing the Required Database Entities for OSM Cloud Native

To meet the OSM cloud native pre-requisites, you will have to create an RCU schema in the database using the DB Installer, with command 7.

To ensure a clean start for OSM cloud native managed servers, delete the leftover LLR tables. When OSM cloud native managed servers start, these tables are recreated with the required data automatically.

To delete the LLR tables:

1. Connect to the database using the OSM cloud native user credentials
2. Get the list of tables to delete:

```
select 'drop table '||tname||' cascade constraints PURGE;' from tab
where tname like ('WL_LLR_%');
```

3. For the tables listed, run the commands for dropping a table.

Upgrading the OSM Schema and Cartridges

To upgrade the OSM schema and cartridges, do the following:

- **Migrate the OSM schema**: To migrate the schema of your OSM traditional instance into a schema that is compatible with OSM cloud native, run the OSM cloud native DB Installer with command 12.
- **Upgrade the OSM Schema to the target version**: If you are running a version of OSM traditional instance that is older than the target OSM cloud native version, use the OSM cloud native DB Installer with command 1 to upgrade the OSM schema to the correct version.

- **Rebuild solution cartridges**: Depending on the version of your current OSM traditional deployment, you may have to rebuild your solution cartridges using the latest release of Design Studio and the target OSM SDK. This is a preparatory step, and the new cartridge lineup would be reflected in the project specification that is also created as part of the preparatory step. All cartridges built targeting OSM versions prior to release version 7.3.5 require rebuilding. This rebuild is the same requirement that exists for OSM traditional deployments as well.

Switching Integration with Upstream Systems

After you shut down the OSM traditional instance fully, do the following:

- Ensure that the OSM cloud native instance has its JMS and SAF objects unpaused and its DB jobs restarted.
- Configure the upstream and peer systems to resume sending messages. See "Integrating OSM" for more details.

Reverting to Your OSM Traditional Deployment

During the move to OSM cloud native, if there is a need to revert to your OSM traditional deployment, the exact sequence of steps that you need to perform depend on the options you have chosen while moving to OSM cloud native.

In general, the OSM traditional deployment application layer should be undisturbed through the upgrade process. If Option A was followed for upgrading the database, the OSM traditional instance can simply be started up again, still pointing to its database.

If however, Option B was followed for upgrading the database, the following steps are required before the OSM traditional instance can be spun up:

- Revert the database server version to the earlier version (if a database server upgrade was performed as part of the switch to OSM cloud native)
- Restore the database contents from the backup taken as part of Option B for upgrading the database.

Cleaning Up

Once the OSM cloud native instance is deemed operational, you can release the resources used for the OSM traditional application layer.

If Option A was adopted for the database, then you can delete the database used for OSM traditional instance and release its resources as well. If Option B was followed and your OSM traditional instance was using JDBC persistent stores, the tables corresponding to these are now defunct and you can delete these as well.

13

Debugging and Troubleshooting

This chapter provides information about debugging and troubleshooting issues that you may face while setting up OSM cloud native environment and creating OSM cloud native instances.

This chapter describes information about the following:

- Setting Up Java Flight Recorder (JFR)
- Troubleshooting Issues with Traefik, OSM UI, and WebLogic Administration Console
- Common Error Scenarios
- Known Issues

Setting Up Java Flight Recorder (JFR)

The Java Flight Recorder (JFR) is a tool that collects diagnostic data about running Java applications. OSM cloud native leverages JFR. See *Java Platform, Standard Edition Java Flight Recorder Runtime Guide* for details about JFR.

You can change the JFR settings provided with the toolkit by updating the appropriate values in the instance specification.

To analyze the output produced by the JFR, use Java Mission Control. See *Java Platform, Standard Edition Java Mission Control User's Guide* for details about Java Mission Control.

JFR is turned on by default in all managed servers. You can disable this feature by setting the `enabled` flag to **false**.

You can customize how much data is maintained, by changing the `max_age` parameter in the instance specification:

```
# Java Flight Recorder (JFR) Settings
jfr:
 enabled: true
 max_age: 4h
```

Data that is generated by the JFR is saved in the container in **/pvMount/***project-instance*/$*server_name*/**repository**.

Persisting JFR Data

JFR data can be persisted outside of the container by re-directing it to persistent storage through the use of a PV-PVC. See "Setting Up Persistent Storage" for details.

Once the storage has been set up, enable `storageVolume` and set the PVC name. Once enabled, JFR data is re-directed automatically.

```
# The storage volume must specify the PVC to be used for persistent
storage.

storageVolume:
  enabled: true
  pvc: storage-pvc
```

Troubleshooting Issues with Traefik, OSM UI, and WebLogic Administration Console

This section describes how to troubleshoot issues with access to the OSM UI clients, WLST, and WebLogic Administration Console.

It is assumed that Traefik is the Ingress controller being used and the domain name suffix is `osm.org`. You can modify the instructions to suit any other domain name suffix that you may have chosen.

The following table lists the URLs for accessing the OSM UI clients and the WebLogic Administration Console, when the Oracle Cloud Infrastructure load balancer is used and not used:

Table 13-1 URLs for Accessing OSM Clients

Client	If Not Using Oracle Cloud Infrastructure Load Balancer	If Using Oracle Cloud Infrastructure Load Balancer
OSM Task Web Client	http://*instance*.*project*.osm.org:30305/OrderManagement	http://*instance*.*project*.osm.org:80/OrderManagement
WLST	http://t3.*instance*.*project*.osm.org:30305	http://t3.*instance*.*project*.osm.org:80
WebLogic Admin Console	http://admin.*instance*.*project*.osm.org:30305/console	http://admin.*instance*.*project*.osm.org:80/console

Error: Http 503 Service Unavailable (for OSM UI Clients)

This error occurs if the managed servers are not running.

To resolve this issue:

1. Check the status of the managed servers and ensure that at least one managed server is up and running:

   ```
   kubectl -n project get pods
   ```

2. Log into WebLogic Admin Console and navigate to the Deployments section and check if the State column for **oms** shows **Active**. The value in the Targets column indicates the name of the cluster.

If the application is not Active, check the managed server logs and see if there are any errors. For example, it is likely that the OSM DB Connection pool could not be created. The following could be the reasons for this:

- DB connectivity could not be established due to reasons such as password expired, account locked, and so on.
- DB Schema heath check policy failed.

There could be other reasons for the application not becoming Active.

Resolution: To resolve this issue, address the errors that prevent the application from becoming Active. Depending on the nature of the corrective action you take, you may have to perform the following procedures as required:

- Upgrade the instance, by running **upgrade-instance.sh**
- Upgrade the domain, by running **upgrade-domain.sh**
- Delete and create a new instance, by running **delete-instance.sh** followed by **create-instance.sh**

Security Warning in Mozilla Firefox

If you use Mozilla Firefox to connect to an OSM cloud native instance via HTTP, your connection may fail with a security warning. You may notice that the URL you entered automatically change to `https://`. This can happen even if HTTPS is disabled for the OSM instance. If HTTPS is enabled, it only happens if you are using a self-signed (or otherwise untrusted) certificate.

If you wish to continue with the connection to the OSM instance using HTTP, in the configuration settings for your Firefox browser (URL: "about:config"), set the **network.stricttransportsecurity.preloadlist** parameter to FALSE.

Error: Http 404 Page not found

This is the most common problem that you may encounter.

To resolve this issue:

1. Check the Domain Name System (DNS) configuration.

   Note:

   These steps apply for local DNS resolution via the **hosts** file. For any other DNS resolution, such as corporate DNS, follow the corresponding steps.

   The hosts configuration file is located at:

   - On Windows: **C:\Windows\System32\drivers\etc\hosts**
   - On Linux: **/etc/hosts**

   Check if the following entry exists in the hosts configuration file of the client machine from where you are trying to connect to OSM:

- Local installation of Kubernetes without Oracle Cloud Infrastructure load balancer:

  ```
  Kubernetes_Cluster_Master_IP  instance.project.osm.org 
  t3.instance.project.osm.org  admin.instance.project.osm.org
  ```

- If Oracle Cloud Infrastructure load balancer is used:

  ```
  Load_balancer_IP  instance.project.osm.org 
  t3.instance.project.osm.org admin.instance.project.osm.org
  ```

Resolve the DNS configuration.

2. Check the browser settings and ensure that ***.osm.org** is added to the No proxy list, if your proxy cannot route to it.
3. Check if the Traefik pod is running and install or update the Traefik Helm chart:

   ```
   kubectl -n traefik get pod
   NAME                                READY   STATUS    RESTARTS   AGE
   traefik-operator-657b5b6d59-njxwg   1/1     Running   0          
   128m
   ```

4. Check if Traefik service is running:

   ```
   kubectl -n traefik get svc
   NAME                         TYPE           CLUSTER-IP      
   EXTERNAL-IP     PORT(S)                       AGE
   oci-lb-service-traefik       LoadBalancer   10.96.136.31    
   100.77.18.141   80:31115/TCP                   20d    <---- Is 
   expected in OCI environment only --
   traefik-operator             NodePort       10.98.176.16    
   <none>          443:30443/TCP,80:30305/TCP    141m
   traefik-operator-dashboard   ClusterIP      10.103.29.101   
   <none>          80/TCP                         141m
   ```

   If the Traefik service is not running, install or update the Traefik Helm chart.
5. Check if Ingress is configured, by running the following command:

   ```
   kubectl -n project get ing
   NAME                    
   HOSTS                                                    
                     ADDRESS   PORTS   AGE
   project-instance-traefik   
   instance.project.osm.org,t3.instance.project.osm.org,admin.instance.
   project.osm.org              80      89m
   ```

   If Ingress is not running, install Ingress by running the following commands:

   ```
   $OSM_CNTK/scripts/create-ingress.sh -p project -i instance -s 
   specPath
   ```

6. Check if the Traefik back-end systems are registered, by using one of the following options:

- Run the following commands to check if your project namespace is being monitored by Traefik. Absence of your project namespace means that your managed server back-end systems are not registered with Traefik.

  ```
  $ cd $OSM_CNTK
  $ source scripts/common-utils.sh
  $ find_namespace_list 'namespaces' traefik traefik-operator
  "traefik","project_1", "project_2"
  ```

- Check the Traefik Dashboard and add the following DNS entry in your hosts configuration file:

  ```
  Kubernetes_Access_IP traefik.osm.org
  ```

  Add the same entry regardless of whether you are using Oracle Cloud Infrastructure load balancer or not. Navigate to: `http://traefik.osm.org:30305/dashboard/` and check the back-end systems that are registered. If you cannot find your project namespace, install or upgrade the Traefik Helm chart. See "Installing the Traefik Container Image".

Reloading Instance Backend Systems

If your instance's ingress is present, yet Traefik does not recognize the URLs of your instance, try to unregister and register your project namespace again. You can do this by using the **unregister-namespace.sh** and **register-namespace.sh** scripts in the toolkit.

Note:

Unregistering a project namespace will stop access to any existing instances in that namespace that were working prior to the unregistration.

Debugging Traefik Access Logs

To increase the log level and debug Traefik access logs:

1. Run the following command:

   ```
   $ helm upgrade traefik-operator traefik/traefik   --version
   9.11.0  --namespace traefik   --reuse-values   --set
   logs.access.enabled=true
   ```

   A new instance of the Traefik pod is created automatically.

2. Look for the pod that is created most recently:

   ```
   $ kubectl get po -n traefik
   NAME                                  READY     STATUS
   RESTARTS   AGE
   traefik-operator-pod_name   1/1       Running   0
   0          5s

   $ kubectl -n traefik logs -f traefik-operator-pod_name
   ```

3. Enabling access logs generates large amounts of information in the logs. After debugging is complete, disable access logging by running the following command:

   ```
   $ helm upgrade traefik-operator traefik/traefik   --version
   9.11.0  --namespace traefik   --reuse-values   --set
   logs.access.enabled=false
   ```

Cleaning Up Traefik

> **Note:**
>
> Clean up is not usually required. It should be performed as a desperate measure only. Before cleaning up, make a note of the monitoring project namespaces. Once Traefik is re-installed, run **$OSM_CNTK/scripts/register-namespace.sh** for each of the previously monitored project namespaces.
>
> **Warning**: Uninstalling Traefik in this manner will interrupt access to all OSM instances in the monitored project namespaces.

To clean up the Traefik Helm chart, run the following command:

```
helm uninstall traefik-operator -n traefik
```

Cleaning up of Traefik does not impact actively running OSM instances. However, they cannot be accessed during that time. Once the Traefik chart is re-installed with all the monitored namespaces and registered as Traefik back-end systems successfully, OSM instances can be accessed again.

Setting up Logs

As described earlier in this guide, OSM and WebLogic logs can be stored in the individual pods or in a location provided via a Kubernetes Persistent Volume. The PV approach is strongly recommended, both to allow for proper preservation of logs (as pods are ephemeral) and to avoid straining the in-pod storage in Kubernetes.

Within the pod, logs are available at: **/u01/oracle/user_projects/domains/domain/servers/ms1/logs**.

> **Note:**
>
> Replace **ms1** with the appropriate managed server or with "admin".

When a PV is configured, logs are available at the following path starting from the root of the PV storage:

project-instance/**logs**.

The following logs are available in the location (within the pod or in PV) based on the specification:

- admin.log - Main log file of the admin server

- admin.out - stdout from admin server
- admin_nodemanager.log: Main log from nodemanager on admin server
- admin_nodemanager.out: stdout from nodemanager on admin server
- admin_access.log: Log of http/s access to admin server
- ms1.log - Main log file of the ms1 managed server
- ms1.out - stdout from ms1 managed server
- ms1_nodemanager.log: Main log from nodemanager on ms1 managed server
- ms1_nodemanager.out: stdout from nodemanager on ms1 managed server
- ms1_access.log: Log of http/s access to ms1 managed server

All the logs in the above list for "ms1" are repeated for each running managed server, with the logs being named for their originating managed server in each case.

In addition to these logs:

- Each JMS Server configured will have its log file with the name **<server>_ms<x>-jms_messages.log** (for example: **osm_jms_server_ms2-jms_messages.log**).
- Each SAF agent configured will have its log file with the name **<server>_ms<x>-jms_messages.log** (for example: **osm_saf_agent_ms1-jms_messages.log**).

OSM Cloud Native and Oracle Enterprise Manager

OSM cloud native instances contain a deployment of the Oracle Enterprise Manager application, reachable at the admin server URL with the path "/em". However, the use of Enterprise Manager in this Kubernetes context is not supported. Do not use the Enterprise Manager to monitor OSM. Use standard Kubernetes pod-based monitoring and OSM cloud native logs and metrics to monitor OSM.

Recovering an OSM Cloud Native Database Schema

When the OSM DB Installer fails during an installation, it exits with an error message. You must then find and resolve the issue that caused the failure. You can re-run the DB Installer after the issue (for example, space issue or permissions issue) is rectified. You do not have to rollback the DB.

Note:

Remember to uninstall the failed DB Installer helm chart before rerunning it. Contact Oracle Support for further assistance.

It is recommended that you always run the DB Installer with the logs directed to a Persistent Volume so that you can examine the log for errors. The log file is located at: *filestore*/*project-instance*/**db-installer**/{*yyyy-mm-dd*}-**osm-db-installer.log**.

In addition, to identify the operation that failed, you can look in the *filestore*/*project-instance*/**db-installer**/**InstallPlan-OMS-CORE.csv** CSV file. This file shows the progress of the DB Installer.

When you install the Oracle Database schema for the first time and if the database schema installation fails, do the following:

1. Delete the new schema or use a new schema user name for the subsequent installation.
2. Restart the installation of the database schema from the beginning.

To recover a schema upgrade failure, do the following:

1. Find the issue that caused the upgrade failure. See "Finding the Issue that Caused the OSM Cloud Native Database Schema Upgrade Failure" for details.
2. Fix the issue. Use the information in the log or error messages to fix the issue before you restart the upgrade process. For information about troubleshooting log or error messages, see *OSM Cloud Native System Administrator's Guide*.
3. Restart the schema upgrade procedure from the point of failure. See "Restarting the OSM Database Schema Upgrade from the Point of Failure" for details.

Finding the Issue that Caused the OSM Cloud Native Database Schema Upgrade Failure

There are several files where you can look to find information about the issue. By default, these files are generated in the managed server pod, but can be re-directed to a Persistent Volume Claim (PVC) supported by the underlying technology that you choose. See "Setting Up Persistent Storage" for details.

To access these files after the DB installer pod is deleted, re-direct all logs to the PVC.

See the following files for details about the issue:

- The database installation plan action spreadsheet file: This file contains a summary of all the installation actions that are part of this OSM database schema installation or upgrade. The actions are listed in the order that they are performed. The spreadsheet includes actions that have not yet been completed. To find the action that caused the failure, check the following files and review the Status column:
  - **filestore/***project-instance***/db-installer/InstallPlan-OMS-CORE.csv**
  - **filestore/***project-instance***/db-installer/InstallPlan-OMS_CLOUD-CORE.csv**

  The failed action is the first action with a status that is FAILED. The **error_message** column of that row contains the reason for the failure.
- The database installation log file: This file provides a more detailed description of all the installation actions that have been run for this installation. The issue that caused the failure is located in the **filestore/***project-instance***/db-installer/{*yyyy-mm-dd*}-osm-db-installer.log** file. The failed action, which is the last action that was performed, is typically listed at the end of log file.

The following database tables also contain information about the database installation:

- **semele$plan_actions**: This contains the same information as the database plan action spreadsheet. Compare this table to the spreadsheet in cases of a database connection failure.
- **semele$plan**: This contains a summary of the installation that has been performed on this OSM database schema.

Restarting the OSM Database Schema Upgrade from the Point of Failure

In most cases, restarting the OSM database schema upgrade consists of pointing the installer to the schema that was partially upgraded, and then re-running the installer.

> **Note:**
>
> This task requires a user with DBA role.

Consider the following when preparing to restart an upgrade:

- Most migration actions are part of a single transaction, which is rolled back in the event of a failure. However, some migration actions involve multiple transactions. In this case, it is possible that some changes were committed.
- Most migration actions are repeatable, which means that they can safely be re-run even if they were committed. However, if a failed action is not repeatable and it committed some changes, either reverse all the changes that were committed and set the status to FAILED, or complete the remaining changes and set the status to COMPLETE.

To restart the upgrade after a failure:

1. Determine which action has failed and the reason for the failure.
2. If the status of the failed action is STARTED, check the database to see whether the action is completed or still running. If it is still running, either end the session or wait for the action to finish.

   > **Note:**
   >
   > The transaction might not finish immediately after the connection is lost, depending on how fast the database detects that the connection is lost and how long it takes to roll back.

3. Fix the issue that caused the failure.

   > **Note:**
   >
   > If the failure is caused by a software issue, contact Oracle Support. With the help of Oracle Support, determine whether the failed action modified the schema and whether you must undo any of those changes. If you decide to undo any changes, leave the action status set to FAILED or set it to NOT STARTED. When you retry the upgrade, the installer starts from this action. If you manually complete the action, set the status to COMPLETE, so that the installer starts with the next action. Do not leave the status set to STARTED because the next attempt to upgrade will not be successful.

4. Restart the upgrade by running the installer.
   The installer restarts the upgrade from the point of failure.

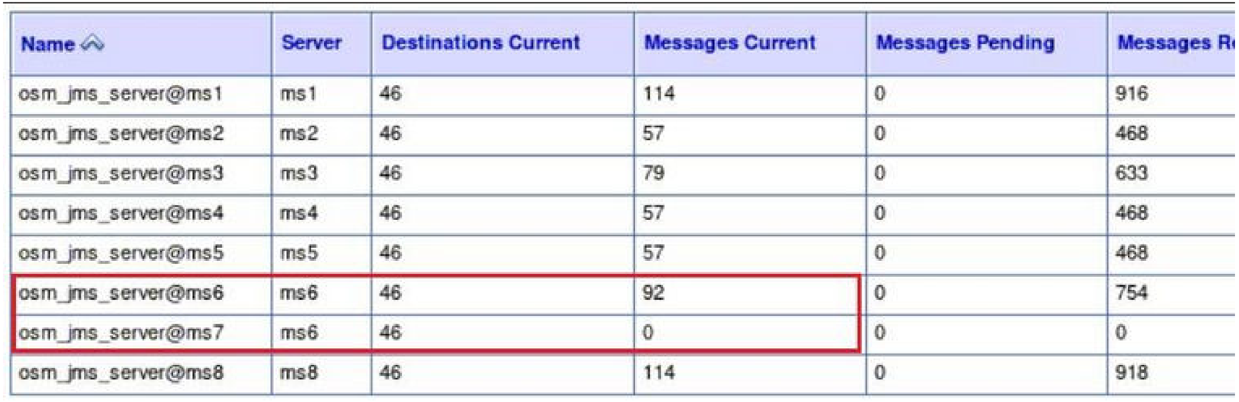
Resolving Improper JMS Assignment

While running OSM cloud native with more than one managed server, sometimes, the incoming orders and the resulting workload may not get distributed evenly across all managed servers.

While there are multiple causes for improper distribution (including the use of an incorrect JMS connection factory to inject order creation messages), one possible cause is the improper assignment of JMS servers to managed servers. For even distribution of workload, each managed server that is running must host its corresponding JMS server.

The following figure shows an example of improper JMS assignment.

Figure 13-1 Example of Improper JMS Assignment



Name	Server	Destinations Current	Messages Current	Messages Pending	Messages R
osm_jms_server@ms1	ms1	46	114	0	916
osm_jms_server@ms2	ms2	46	57	0	468
osm_jms_server@ms3	ms3	46	79	0	633
osm_jms_server@ms4	ms4	46	57	0	468
osm_jms_server@ms5	ms5	46	57	0	468
osm_jms_server@ms6	ms6	46	92	0	754
osm_jms_server@ms7	ms6	46	0	0	0
osm_jms_server@ms8	ms8	46	114	0	918

In the figure, **osm_jms_server@ms7** is incorrectly running on **ms6** even though its native host **ms7** is running. It can be normal for more than one JMS server to be running on a managed server as long as the additional JMS servers do not have a native managed server that is online.

Workaround

As a workaround, terminate the Kubernetes pod for the managed server that has been left underutilized. In the above example, the pod for **ms7** should be terminated. The WebLogic Operator recreates the managed server pod, and that should trigger the migration of **osm_jms_server@ms7** back to **ms7**.

Resolution

To resolve this issue, tune the time setting for **InitialBootDelaySeconds** and **PartialClusterStabilityDelaySeconds**. See the WebLoigic Server documentation for more details.

To tune the time setting:

1. Add the following Clustering fragment to the instance specification:

```
Clustering:
  InitialBootDelaySeconds: 10
  PartialClusterStabilityDelaySeconds: 30
```

2. Increase the value for the following parameters from the base WDT model:
   - **InitialBootDelaySeconds**. The default value in base WDT is 2.
   - **PartialClusterStabilityDelaySeconds**. The default value in base WDT is 5.

Note:

The default values for these parameters in WebLogic Server are **60** and **240** respectively. The actual values required depend on the environmental factors and must be arrived at by tuning. Higher values can result in slower placement of JMS servers. While this is not a factor during OSM startup, it will mean more time could be taken when a managed server shuts down before its JMS server migrates and comes up on a surviving managed server. Orders with messages pending delivery in that JMS server will be impacted by this, but the rest of the system is unaffected.

Common Problems and Solutions

This section describes some common problems that you may experience because you have run a script or a command erroneously or you have not properly followed the recommended procedures and guidelines regarding setting up your cloud environment, components, tools, and services in your environment. This section provides possible solutions for such problems.

Domain Introspection Pod Does Not Start

Upon running **create-instance.sh** or **upgrade-instance.sh**, no change is observed. Running `kubectl get pods -n project --watch` shows that the introspector pod did not start at all.

The following are the potential causes and mitigations for this issue:

- WebLogic Kubernetes Operator (WKO) is not up or not healthy: Confirm if WKO is up by running `kubectl get pods -n $WLSKO_NS`. There should be one WKO pod in the `RUNNING` state. If there is a pod, check its logs. If a pod is not there, check if WKO is uninstalled. You may need to terminate an unhealthy pod or reinstall WKO.
- Project is not registered with WKO.
  Run the following command:

  ```
  kubectl get cm -n $WLSKO_NS -o yaml weblogic-operator-cm | grep
  targetNamespaces
  ```

  Your project namespace must be listed in the list that the command returns. If it is not listed, run `$OSM_CNTK/scripts/register-namespace.sh`.

Other causes are infrastructure related issues such as worker capacity and user RBAC.

Domain Introspection Pod Status

While the introspection is running, you can check the status of the introspection pod by running the following command:

```
kubectl get pods -n namespace

## healthy status looks like this

NAME                                       READY   STATUS    RESTARTS
AGE
project-instance-introspect-domain-job-hzh9t   1/1     Running
0          3s
```

The READY field is showing 1/1, which indicates that the pod status is healthy.

If there is an issue accessing the image specified in the instance specification, then it shows the following:

```
NAME                                           READY   STATUS
RESTARTS   AGE
project-instance-introspect-domain-job-r2d6j   0/1     ErrImagePull
0          5s

### OR

NAME                                           READY
STATUS             RESTARTS   AGE
project-instance-introspect-domain-job-r2d6j   0/1
ImagePullBackOff   0          45s
```

This shows that the introspection pod status is not healthy. If the image can be pulled, it is possible that it took a long time to pull the image.

To resolve this issue, verify that the image name and the tag and that it is accessible from the repository by the pod.

You can also try the following:

- Increase the value of `podStartupDeadlineSeconds` in the instance specification.
  Start with a very high timeout value and then monitor the average time it takes, because it depends on the speed with which the images are downloaded and how busy your cluster is. Once you have a good idea of the average time, you can reduce the timeout values accordingly to a value that includes the average time and some buffer.
- Pull the container image manually on all Kubernetes nodes where the OSM cloud native pods can be started up.

Domain Introspection Errors Out

Some times, the domain introspector pod runs, but ends with an error.

To resolve this issue, run the following command and look for the causes:

```
kubectl logs introspector_pod -n project
```

The following are the possible causes for this issue:

- RCU Schema is pre-existing: If the logs shows the following, then RCU schema could be pre-existing:

  ```
  WLSDPLY-12409: createDomain failed to create the domain:
  Failed to write domain to /u01/oracle/user_projects/domains/domain:
  wlst.writeDomain(/u01/oracle/user_projects/domains/domain) failed :
  Error writing domain:
  64254: Error occurred in "OPSS Processing" phase execution
  64254: Encountered error:
  oracle.security.opss.tools.lifecycle.LifecycleException: Error
  during configuring DB security store. Exception
  oracle.security.opss.tools.lifecycle.LifecycleException: The schema
  FMW1_OPSS is already in use for security store(s).  Please create a
  new schema..
  64254: Check log for more detail.
  ```

  This could happen because the database was reused or cloned from an OSM cloud native instance. If this is so, and you wish to reuse the RCU schema as well, provide the required secrets. For details, see "Reusing the Database State".
  If you do not have the secrets required to reuse the RCU instance, you must use the OSM cloud native DB Installer to create a new RCU schema in the DB. Use this new schema in your `rcudb` secret. If you have login details for the old RCU users in your `rcudb` secret, you can use the OSM cloud native DB Installer to delete and re-create the RCU schema in place. Either of these options gives you a clean slate for your next attempt.

  Finally, it is possible that this was a clean RCU schema but the introspector ran into an issue after RCU data population but before it could generate the wallet secret (opssWF). If this is the case, debug the introspector failure and then use the OSM cloud native DB Installer to delete and re-create the RCU schema in place before the next attempt.

- Fusion MiddleWare cannot access the RCU: If the introspector logs show the following error, then it means that Fusion MiddleWare could not access the schema inside the RCU DB.

  ```
  WLSDPLY-12409: createDomain failed to create the domain: Failed to
  get FMW infrastructure database defaults from the service table:
  Failed to get the database defaults: Got exception when auto
  configuring the schema component(s) with data obtained from shadow
  table:
  Failed to build JDBC Connection object:
  ```

  Typically, this happens when wrong values are entered while creating secrets for this deployment. Less often, the cause is a corrupted RCU DB or an invalid one. Re-create your secrets, verifying the credentials and drop and re-create the RCU DB.

Recovery After Introspection Error

If the introspection fails during instance creation, once you have gathered the required information and have a solution, delete the instance and then re-run the instance creation script with the fixed specification, model extension, or other environmental failure cause.

If the introspection fails while upgrading a running instance, then do the following:

1. Make the change to fix the introspection failure. Trigger an instance upgrade. If this results in successful introspection, the recovery process stops here.
2. If the instance upgrade in step 1 fails to trigger a fresh introspection, then do the following:
   a. Rollback to the last good Helm release by first running the `helm history -n` *`project project-instance`* command to identify the version in the output that matches the running instance (that is, before the upgrade that led to introspection failure). The timestamp on each version helps you identify the version. Once you know the "good" version, rollback to that version by running: `helm rollback -n` *`project project-instance version`*. Monitor the pods in the instance to ensure only the Admin server and the appropriate number of Managed Server pods are running.
   b. Upgrade the instance with the fixed specification.

Instance Deletion Errors with Timeout

You use the **delete-instance.sh** script to delete an instance that is no longer required. The script attempts to do this in a graceful manner and is configured to wait up to 10 minutes for any running pods to shut down. If the pods remain after this time, the script times out and exits with an error after showing the details of the leftover pods.

The leftover pods can be OSM pods (Admin Server, Managed Server) or the DBInstaller pod.

For the leftover OSM pods, see the WKO logs to identify why cleanup has not run. Delete the pods manually if necessary, using the **kubectl delete** commands.

For the leftover DBInstaller pod, this happens only if **install-osmdb.sh** is interrupted or if it failed in its last run. This should have been identified and handled at that time itself. However, to complete the cleanup, run `helm ls -n` *project* to find the failed DBInstaller release, and then invoke `helm uninstall -n` *project release*. Monitor the pods in the project namespace until the DBInstaller pod disappears.

OSM Cloud Native Toolkit Instance Create/Update Scripts Timeout; Pods Show Readiness "0/1"

If your **create-instance.sh** or **upgrade-instance.sh** scripts timeout, and you see that the desired managed server pods are present, but one or more of them show "0/1" in the "READY" column, this could be because OSM hit a fatal problem while starting up. The following could be the causes for this issue:

- A mismatch in the OSM schema found and the expected version: If this is the case, the OSM managed server log shows the following issue:

```
Error: The OSM application is not compatible with the schema code
detected in the OSM database.
Expected version[7.4.0.0.68], found version[7.4.0.0.70]
This likely means that a recent installation or upgrade was not
successful.
```

```
Please check your install/upgrade error log and take steps to
ensure the schema is at the correct version.
```

  To resolve this issue, check the container image used for the DB installer and the OSM domain instances. They should match.

- OSM internal users are missing: This can happen if there are issues with the configuration of the external authentication provider and the standard OSM users (for example, **oms-internal**) and the group association is not loaded. The managed server log shows something like the following:

  ```
  <Error> <Deployer> <BEA-149205> <Failed to initialize the
  application "oms" due to error
  weblogic.management.DeploymentException: The
  ApplicationLifecycleListener "com.mslv.oms.j2ee.LifecycleListener"
  of application "oms"
  has a run-as user configured with the principal name "oms-internal"
  but a principal of that name
  could not be found. Ensure that a user with that name exists.
  ```

  To resolve this issue, review your external authentication system to validate users and groups. Review your configuration to ensure that the instance is configured for the correct external authenticator.

OSM Cloud Native Pods Do Not Distribute Evenly Across Worker Nodes

In some occasions, OSM cloud native pods do not distribute evenly across the worker nodes.

To resolve this issue, prime all the worker nodes with the image using the OSM cloud native sample utility script:

```
$ $OSM_CNTK/samples/image-primer.sh -p project image-name:image-tag
```

This should be done only once for a given image `name+tag` combination, regardless of which project uses that image or how many instances are created with it.

This script is offered as a sample and may need to be customized for your environment. If you are using an image from a repository that requires pull credentials, edit the **image-primer.sh** script to uncomment these lines and add your pull secret:

```
#imagePullSecrets:
  #- name: secret-name
```

If you are planning to target OSM cloud native to specific worker nodes, edit the sample to ensure only those nodes are selected (typically by using a specific label value) as per standard Kubernetes configuration. See the Kubernetes documentation for DaemonSet objects.

User Workgroup Association Lost

During cartridge deployment, if users are not present in LDAP or if LDAP is not accessible, the user workgroup associations could get deleted.

To resolve this issue, restore the connectivity to LDAP and the users. You may need to redo the workgroup associations.

Changing the WebLogic Kubernetes Operator Log Level

Some situations may require analysis of the WKO logs. These logs can be certain kinds of introspection failures or unexpected behavior from the operator. The default log level for the Operator is **INFO**.

You can change this level if required in two ways:

- Upgrade the Helm chart:

  ```
  helm upgrade --namespace $WLSKO_NS --reuse-values --set
  "javaLoggingLevel=FINE" --wait $WLSKO_NS ${WLSKO_HOME}/kubernetes/
  charts/weblogic-operator
  ```

- Remove the operator and reinstall it. See the Deleting and Re-creating the WLS Operator section that follows. Use the **-l** option when you run **install-operator.sh**.

To see the possible values for the log level, consult the WLS operator documentation or run **`install-operator.sh -h`**.

Deleting and Re-creating the WLS Operator

You may need to delete a WLS operator and re-create it. You do this when you want to use a new version of the operator where upgrade is not possible, or when the installation is corrupted.

When the controlling operator is removed, the existing OSM cloud native instances continue to function. However, they cannot process any updates (when you run **upgrade-instance.sh**) or respond to the Kubernetes events such as the termination of a pod.

Normally, the **remove-operator.sh** script fails if there are OSM cloud native projects registered with the operator. But you can use the **-f** flag to force the removal. When this is done, the script prints out the list of registered projects and reminds you to manually re-register them (by running **register-namespace.sh**) after reinstalling the operator.

You can install the operator as usual and then register all the projects again, one by one by running **register-namespace.sh -p *project* -t wlsko**.

Lost or Missing opssWF and opssWP Contents

For an OSM instance to successfully connect to a previously initialized set of DB schemas, it needs to have the opssWF (OPSS Wallet File) and opssWP (OPSS Wallet-file Password) secrets in place. The **$OSM_CNTK/scripts/manage-instance-credentials.sh** script can be used to set these up if they are not already present.

If these secrets or their contents are lost, you can delete and recreate the RCU schemas (using **$OSM_CNTK/scripts/install-osmdb.sh** with command code 5). This deletes data (such as some user preferences and so on) stored in the RCU schemas. On the other hand, if there is a WebLogic domain currently running against that DB (or its clone), the "exportEncryptionKey" offline WLST command can be run to dump out the "ewallet.p12" file. This command also takes a new encryption password. See *Oracle Fusion MiddleWare WLST Command Reference for Infrastructure Security* for details. The contents of the resulting ewallet.p12 file can be used to recreate the opssWF secret, and the encryption password can be used to recreate the opssWP secret. This method is also suitable when a DB (or the clone of a DB) from a traditional OSM installation needs to be brought into OSM cloud native.

Clock Skew or Delay

When submitting JMS message over the Web Service queue, you might see the following in the SOAP response:

```
Security token failed to validate.
weblogic.xml.crypto.wss.SecurityTokenValidateResult@5f1aec15[status:
false][msg UNT Error:Message older than allowed MessageAge]
```

Oracle recommends synchronizing the time across all machines that are involved in communication. See "Synchronizing Time Across Servers" for more details. Implement Network Time Protocol (NTP) across the hosts involved, including the Kubernetes cluster hosts.

It is also possible to temporarily fix this through configuration by adding the following properties to **java_options** in the project specification for each managed **server.managedServers**: project:

```

#JAVA_OPTIONS for all managed servers at project level java_options:
-Dweblogic.wsee.security.clock.skew=72000000
-Dweblogic.wsee.security.delay.max=72000000
```

Known Issues

This section describes known issues that you may come across, their causes, and the resolutions.

Email Plugin
The OSM Email plugin is currently not supported. Users who require this capability can create their own plugin for this purpose.

SituationalConfig NullPointerException

In the managed server logs, you might notice a stacktrace that indicates a NullPointerException in situational config.

This exception can be safely ignored.

Connectivity Issues During Cluster Re-size

When the cluster size changes, whether from the termination and re-creation of a pod, through an explicit upgrade to the cluster size, or due to a rolling restart, transient errors are logged as the system adjusts.

These transient errors can usually be ignored and stop after the cluster has stabilized with the correct number of Managed Servers in the Ready state.

If the error messages were to persist after a Ready state is achieved, then looking for secondary symptoms of a real problem would be appropriate. Such connectivity errors could result in orders that were inexplicably stuck or were otherwise processing abnormally.

While not an exhaustive list, some examples of these transient errors you may see in a managed server log are:

- An MDB is unable to connect to a JMS destination. The specific MDB and JMS destination can vary, such as:
  - `<The Message-Driven EJB OSMInternalEventsMDB is unable to connect to the JMS destination mslv/oms/oms1/internal/jms/events.`
  - `<The Message-Driven EJB DeployCartridgeMDB is unable to connect to the JMS destination mslv/provisioning/internal/ejb/deployCartridgeQueue.`
- Failed to Initialize JNDI context. Connection refused; No available router to destination. This type of error is seen in an instance where SAF is configured.
- Failed to process events for event type[AutomationEvents].
- Consumer destination was closed.

Upgrade Instance failed with spec.persistentvolumesource: Forbidden: is immutable after creation.

You may come across the following error when you run the commands for upgrading the OSM Helm chart:

```
Error: UPGRADE FAILED: cannot patch "<project>-<instance>-nfs-pv" with
kind
PersistentVolume: PersistentVolume "<project>-<instance>-nfs-pv" is
invalid: spec.persistentvolumesource:
Forbidden: is immutable after creation
Error in upgrading osm helm chart
```

Once created, the Persistent Volume Claim cannot be changed.

To resolve this issue:

1. Disable NFS by setting the `nfs.enabled` parameter to **false** and run the upgrade-instance script. This removes the PV from the instance.
2. Enable it again by changing `nfs.enabled:` to **true** with the new values of NFS and then run upgrade-instance.

JMS Servers for Managed Servers are Reassigned to Remaining Managed Servers

When scaling down, the JMS servers for managed servers that do not exist are getting reassigned to remaining managed servers.

For example, for a SimpleResponseQueue when there is only 1 managed server running, you can notice something like the following in the logs:

```
Jun 15, 2020 11:01:32,821 AM UTC> <Info>
<oracle.communications.ordermanagement.automation.plugin.JMSDestinationL
istener> <BEA-000000> <
All local JMS destinations: ms1
JNDI
     JMS Server         WLS Server Migratable Target Local    Member
Type                   Partition
------------------------------------------------------------------------
---- ------------------ ---------- ---------------- --------
```

```
---------------------------- ---------
osm_jms_server@ms1@mslv/oms/oms1/internal/jms/
events                         osm_jms_server@ms1
ms1                          true     MEMBER_TYPE_CLUSTERED_DYNAMIC
DOMAIN
osm_jms_server@ms1@oracle.communications.ordermanagement.SimpleResponseQ
ueue osm_jms_server@ms1 ms1                          true
MEMBER_TYPE_CLUSTERED_DYNAMIC DOMAIN
>
```

Notice that `osm_jms_server@ms1` is targeting `ms1`.

When scaled to 2 Managed Servers, the log shows the following:

```
<Jun 15, 2020 11:02:20,461 AM UTC> <Info>
<oracle.communications.ordermanagement.automation.plugin.JMSDestinationL
istener> <BEA-000000> <
All local JMS destinations: ms1
JNDI
     JMS Server         WLS Server Migratable Target Local    Member
Type                   Partition
------------------------------------------------------------------------
---- ----------------- ---------- ---------------- --------
---------------------------- ---------
osm_jms_server@ms1@mslv/oms/oms1/internal/jms/
events                         osm_jms_server@ms1
ms1                          true     MEMBER_TYPE_CLUSTERED_DYNAMIC
DOMAIN
osm_jms_server@ms1@oracle.communications.ordermanagement.SimpleResponseQ
ueue osm_jms_server@ms1 ms1                          true
MEMBER_TYPE_CLUSTERED_DYNAMIC DOMAIN
osm_jms_server@ms2@mslv/oms/oms1/internal/jms/
events                         osm_jms_server@ms2
ms2                          true     MEMBER_TYPE_CLUSTERED_DYNAMIC
DOMAIN
osm_jms_server@ms2@oracle.communications.ordermanagement.SimpleResponseQ
ueue osm_jms_server@ms2 ms2                          true
MEMBER_TYPE_CLUSTERED_DYNAMIC DOMAIN
>
```

Notice that `osm_jms_server@ms1` is targeting `ms1` and `osm_jms_server@ms2` is targeting `ms2`.

After scaling back to 1 managed server, the log shows the following:

```
<Jun 15, 2020 11:02:20,461 AM UTC> <Info>
<oracle.communications.ordermanagement.automation.plugin.JMSDestinationL
istener> <BEA-000000> <
All local JMS destinations: ms1
JNDI
     JMS Server         WLS Server Migratable Target Local    Member
Type                   Partition
------------------------------------------------------------------------
---- ----------------- ---------- ---------------- --------
```

```
---------------------------- ---------
osm_jms_server@ms1@mslv/oms/oms1/internal/jms/
events                        osm_jms_server@ms1
ms1                          true     MEMBER_TYPE_CLUSTERED_DYNAMIC
DOMAIN
osm_jms_server@ms1@oracle.communications.ordermanagement.SimpleResponseQ
ueue osm_jms_server@ms1 ms1                          true
MEMBER_TYPE_CLUSTERED_DYNAMIC DOMAIN
osm_jms_server@ms2@mslv/oms/oms1/internal/jms/
events                        osm_jms_server@ms2
ms1                          true     MEMBER_TYPE_CLUSTERED_DYNAMIC
DOMAIN
osm_jms_server@ms2@oracle.communications.ordermanagement.SimpleResponseQ
ueue osm_jms_server@ms2 ms1                          true
MEMBER_TYPE_CLUSTERED_DYNAMIC DOMAIN
>
```

Notice that the JMS Server `osm_jms_server@ms2` is not deleted and is targeting ms1.

This is completely expected behavior. This is a WebLogic feature and not to be mistaken for any inconsistency in the functionality.

A

Differences Between OSM Cloud Native and OSM Traditional Deployments

If you are moving from a traditional deployment of OSM to a cloud native deployment, this section describes the differences between OSM cloud native and OSM traditional.

- **Embedded LDAP**

  You no longer need to create human users using the embedded LDAP capabilities of WebLogic Server.

  By default, OSM uses the WebLogic embedded LDAP as the authentication provider and all OSM system users are created in embedded LDAP during the creation of the instance. The OSM cloud native toolkit provides a sample configuration that uses OpenLDAP to demonstrate how to integrate with external LDAP server for human users.

  A sample script for populating users to OpenLDAP can be found at: **$OSM_CNTK/samples/credentials/manage-cartridge-credentials.sh**. See "Provisioning Cartridge User Accounts" for more details.

- **Credential Store for Automation Code**

  The existing Fusion MiddleWare Credential Store framework has been replaced with Kubernetes Secrets in OSM cloud native. See "Provisioning Cartridge User Accounts" for more details on configuration differences. However, the automation plugin code in your cartridges that accesses this information using the automation framework APIs continues to receive the credentials transparently.

- **Credential Store for Custom Code**

  If you use custom code that relies on the OPSS Keystore Service, then port the code. This mechanism is no longer supported. The recommended replacement is Kubernetes Secrets. Kubernetes Secrets can be specified as custom secrets in OSM cloud native and are mounted into the instance's pods for your code to use and access.

- **XMLIE Operations**

  The following operations are still available using XMLIE. However, these should not be used in OSM cloud native. See the following table that describes the replacement mechanism for using these operations.

Table A-1 Replacement Mechanisms for XMLIE Operations

Operation	Replacement Mechanism
credStoreAdmin	Sample script in **$OSM_CNTK/samples/credentials/manage-cartridge-credentials.sh**. Use the `secret` option in the user text file. See "Provisioning Cartridge User Accounts" for more details.
userAdmin	Sample script in **$OSM_CNTK/samples/credentials/manage-cartridge-credentials.sh**. Use the `ldap` option in the user text file. See "Provisioning Cartridge User Accounts" for more details.

Table A-1 (Cont.) Replacement Mechanisms for XMLIE Operations

Operation	Replacement Mechanism
import	OSM DB Installer. Additionally, this operation relies on a pre-built par file, instead of an XML model file. It can use a local file or pull it from a remote repository. See "Working with Cartridges" for more details.
fastUndeploy	OSM DB Installer. See "Working with Cartridges" for more details.

- **WebLogic Domain Configuration**

  In a traditional deployment of OSM, the WebLogic domain configuration is done using WLST or the WebLogic Admin Console. In OSM cloud native, domain configuration is done by providing WDT metadata in the instance creation process. See "Extending the WebLogic Server Deploy Tooling (WDT) Model" for details.

  Do not perform WebLogic administrative activities such as changing the configuration, shutting down and restarting the server directly on the WebLogic Server cluster of the OSM cloud native instance. The same applies to the activities done using WebLogic Server Admin Console, WLST invocation, or any mechanism, other than those supplied by the specification files for updating and upgrading the OSM cloud native instance.

- **Incoming SAF and Outgoing SAF**

  For incoming SAF agents, the originator must use T3 over HTTP tunneling.

  Outgoing SAF mechanism has not changed.

- **OSM Solution Cartridges**

  Your OSM cartridges work normally in OSM cloud native.

  For cartridges that might access a custom property file, this can be done by injecting custom files into the specifications. See "Injecting Custom Configuration Files" for details. An alternative is to use a database table instead. This has the advantage of becoming part of backups and replication automatically.

  Using custom tables or datasources needs to be declared by providing the necessary WDT extensions. See "Extending the WebLogic Server Deploy Tooling (WDT) Model" for details.

- **OSM Workgroups**: OSM Workgroups, including user and workgroup associations, are still managed through the orchestration UI.

- **OSM User Interfaces**: All OSM user interfaces are still available with both OSM traditional and OSM cloud native deployments. The UIs can be accessed using the default hostname: *instance*.*project*.**osm.org** and port 30305, which is the default but configurable and the path that is necessary for the specific UI. For example, to access the Order Management UI, use:

  ```
  http://instance.project.osm.org:30305/OrderManagement/Login.jsp
  ```

- **OSM API**: Accessing OSM through the traditional APIs such as the Web Services API, the REST API, and the XML API has not changed.

- **Order Partitioning Realm Configuration**: Runtime configuration of order partitioning realms is not supported. In traditional OSM deployments, this is specified in the **oms-config.xml** file as a set of files against the `oracle.communications.ordermanagement.OrderPartitioningRealmConfigFileURLs` parameter.

- **OSM Runtime Parameters**: Some OSM runtime parameters can be controlled using the **oms-config.xml** file. This configuration is still available in OSM cloud native, but is managed differently. See "Configuring OSM Runtime Parameters" for more details.
  - **Operational Order Jeopardies**: Configuration to support operational order jeopardies is specified in the **oms-config.xml** file as a set of files against the `oracle.communications.ordermanagement.order.OperationalOverrideFileURLs` parameter. These configuration files are custom files and must be injected properly. See "Injecting Custom Configuration Files" for details.
  - **OACC Runtime Configuration**: This is specified in the **oms-config.xml** file as a set of files against the `AutomationConcurrencyModels` parameter. These configuration files are custom files and must be injected properly. See "Injecting Custom Configuration Files" for details.