

Oracle® Communications Order and Service Management Cloud Native Deployment Guide



Release 8.0
G38005-01
October 2025

ORACLE®

Copyright © 2020, 2025, Oracle and/or its affiliates.

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

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

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

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

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

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

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

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

Contents

About This Content

1 Overview of the OSM Cloud Native Deployment

About the OSM Cloud Native Deployment	1
OSM Cloud Native Architecture	1
About the WebLogic Domain	3
About Kubernetes Custom Resource Definitions (CRD) and Domain Configuration Config Map	4
About Oracle WebLogic Server Deploy Tooling (WDT)	5
About Projects and Instances	5
About Specification Layers	5
About Helm Overrides	6
About the OSM Cloud Native Toolkit	6

2 Planning and Validating Your Cloud Native Environment

Required Components for OSM Cloud Native	1
Planning Your Cloud Native Environment	2
Setting Up Your Kubernetes Cluster	2
Synchronizing Time Across Servers	4
Provisioning Oracle Multitenant Container Database (CDB)	4
Provisioning an Empty PDB	4
Provisioning a Seed OSM PDB	7
Secondary Database Support for OSM Cloud Native	8
Prerequisites	8
About Container Image Management	9
Installing Helm	9
Setting Up Oracle WebLogic Server Kubernetes Operator	11
About Load Balancing and Ingress Controller	11
Using NGINX as the Ingress Controller	12
Using Domain Name System (DNS)	12
Configuring Kubernetes Persistent Volumes	13
About NFS-based Persistence	13

About Authentication	14
Management of Secrets	14
Using Kubernetes Monitoring Toolchain	15
About Application Logs and Metrics Toolchain	15
Role of Continuous Integration (CI) Pipelines	16
Role of Continuous Delivery (CD) Pipelines	16
Planning Your Container Engine for Kubernetes (OKE) Cloud Environment	17
Compute Disk Space Requirements	17
Connectivity Requirements	18
Using Load Balancer as a Service (LBaaS)	18
About Using Oracle Cloud Infrastructure Domain Name System (DNS) Zones	18
Using Persistent Volumes and File Storage Service (FSS)	18
Leveraging Oracle Cloud Infrastructure Services	19
Validating Your Cloud Environment	19
Performing a Smoke Test	20
Validating Common Building Blocks in the Kubernetes Cluster	21
Running Oracle WebLogic Kubernetes Operator Quickstart	24

3 Creating OSM Cloud Native Images

Downloading the OSM Cloud Native Image Builder	1
Prerequisites for Creating OSM Images	2
Configuring the OSM Cloud Native Images	3
Creating OSM Cloud Native Images	7

4 Creating a Basic OSM Cloud Native Instance

Installing the OSM Cloud Native Artifacts and the Toolkit	1
Using Oracle Autonomous Database Serverless	1
Using RDS or RDS Custom for Oracle	5
Installing WebLogic Kubernetes Operator (WKO) and Ingress Controller	6
Installing the WebLogic Kubernetes Operator	6
Installing the Ingress Controller	7
Creating a Basic OSM Instance	7
Setting Environment Variables	7
Registering the Namespace	7
Assembling the Specifications	9
Configuring OpenID Connect for OSM Microservices	9
Creating Secrets	13
Configuring OSM Cloud Native to Connect with a TCPS Enabled Database	14
Installing the OSM and RCU Schemas	16
Configuring the Project Specification	19

Tuning the Project Specification	20
Configuring the Instance Specification	22
Creating an Ingress	23
Creating an OSM Instance	24
Validating the OSM Instance	25
Scaling the OSM Application Cluster	26
Deploying the Sample Cartridge	26
Submitting Orders	27
Deleting and Recreating Your OSM Instance	28
Cleaning Up the Environment	28
Troubleshooting Issues with the Scripts	29
Next Steps	31

5 Planning Infrastructure

Sizing Considerations	1
Managing Configuration as Code	1
Creating Source Control Repository	2
Managing OSM Instances	2
Deciding on the Scope	2
About the Repository Directory Structure	2
Deployment Considerations	3
Setting the Repository Path During Instance Creation	4
Setting Up Automation	4
Securing Operations in Kubernetes Cluster	10

6 Creating Your Own OSM Cloud Native Instance

Selecting a Deployment Topology	1
Installing OSM Cartridge Assembler (OCA) for Integration with Solution Designer	2
Preparing for OSM Cartridge Assembler	3
Managing an OSM Cartridge Assembler Deployment via Cloud Native Toolkit	6
Configuring OSM Runtime Parameters	7
Configuring Schema Validation	8
Configuring Target Systems for Events and System Interactions	9
Configuring Security Schemes for Target Systems	10
Using the OIDC Security Scheme	11
Using the Basic Authentication Security Scheme	11
Basic Auth with User Credentials	12
JSESSIONID in Authentication Header	12
JSESSIONID in Cookies	12
Configuring OSM Gateway Readiness	13

Configuring the Order Operations User Interface	13
Configuring the Alerts Displayed in the Order Operations Dashboard	14
Configuring Session Timeout	15
Preparing Cartridges	16
Working with Kubernetes Secrets	18
About Mandatory Secrets	19
About Optional Secrets	19
About Custom Secrets	21
Accommodating the Scope of Secrets	22
Mechanism for Creating Custom Secrets	24
Adding JMS Queues and Topics	25
Generating Error Queues for Custom Queues and Topics	26
Creating a JMS Template	27
Provisioning Cartridge User Accounts	28
Working with Cartridges	32
Cartridge Deployment Tool in OSM Cloud Native	32
Single or One-off Cartridge Deployment	33
Specification-driven Cartridge Deployment	33
Offline Cartridge Deployment Using the OSM Cloud Native Toolkit	34
Online Cartridge Deployment Using the OSM Cloud Native Toolkit	35
Deploying Cartridges Using Design Studio	35
Listing Deployed Cartridges Using the OSM Cloud Native Toolkit	36
Cartridge par Sources	36
Local Files	36
Remote File Repository	36
Container Images	37
Selecting Deployment Style and Cartridge Source	38
Deploying Cartridges in Open Environments	39
Deploying Cartridges in Controlled Environments	39

7 Extending the WebLogic Server Deploy Tooling (WDT) Model

About the Custom WDT Extension Mechanism	1
Using the WDT Model Tools	1
WDT Discover Domain Tool	1
WDT Validate Model Tool	2
Common WDT Extension Mechanism	2
Using the Sample Scripts to Extend the WDT Model	5
Adding a JDBC Datasource	5
Adding a JMS System Resource	7
Deploying Entities to an OSM WebLogic Domain	8
Extending the WDT Metadata for an External Authenticator	10

Accessing Kubernetes Secrets from WDT Metadata	13
Troubleshooting WDT Issues	13

8 Exploring Configuration Options

Manage LDAP Providers in WLS via OSM	1
Working with Shapes	3
Init and Sidecar Containers Resourcing	5
Creating Custom Shapes	6
Injecting Custom Configuration Files	6
Choosing Worker Nodes for Running OSM Cloud Native	8
Working with Ingress, Ingress Controller, and External Load Balancer	9
Using an Alternate Ingress Controller	12
Preconfiguration on Primary and Standby Database	14
Configuring the OSM Application for High Availability	14
Data Guard Setup on OCI	15
Reusing the Database State	16
Recreating an Instance	16
Creating a New Instance	17
Setting Up Persistent Storage	18
Setting Up Database Optimizer Statistics	19
Leveraging Oracle WebLogic Server Active GridLink	20
Managing Logs	21
Configuring Fluentd Logging	21
Obfuscating Sensitive Data in Logs	25
Configuring Logging and Log Rotation	28
Managing OSM Cloud Native Metrics	30
Configuring Prometheus for OSM Cloud Native Metrics	31
Viewing Metrics Without Using Prometheus	34
Viewing OSM Cloud Native Metrics in Grafana	34
Exposed OSM Order Metrics	36
Managing WebLogic Monitoring Exporter (WME) Metrics	42
Enabling WebLogic Monitoring Exporter (WME)	43
Configuring the Prometheus Scrape Job for WME Metrics	44
Viewing WebLogic Monitoring Exporter Metrics in Grafana	45
Proxy Configuration in Microservices	45

9 Automating OSM Cloud Native Operations for Continuous Delivery

About Continuous Delivery Mechanism	1
Prerequisites for Creating an OSM Instance Using the Introspector Job	2
Git Strategy for OSM Instances	2

OSM Cloud Native Toolkit Introspector Job Helm Chart	5
Creating an OSM Cloud Native Instance Using Flux-CD	6
Setup Flux-descriptors Configuration	6
Kustomization Resource	7
GitRepository Resource	8
HelmRelease Resource	8
Setup Job-Values Configuration	9
Triggering CD for a New Instance	11
Making Changes to an OSM Instance	12
Adding Custom Content	12
Custom Extensions	12
Custom Files	15
Custom Shapes	16

10 Integrating OSM

Connectivity With Traditional OSM Instances	1
Connectivity With OSM Cloud Native	2
Connectivity Between the Building Blocks	2
Inbound HTTP Connectivity	3
Inbound JMS Connectivity	4
Inbound JMS Connectivity Within the Same Kubernetes Cluster	5
Outbound HTTP Connectivity	6
Outbound JMS Connectivity	6
Configuring SAF	7
Security for Remote SAF and Bridges	9
Configuring the Instance Specification	10
Configuring the Project Specification	11
Configuring WebLogic Messaging Bridges	12
Applying the WebLogic Patch for External Systems	16
Configuring SAF On External Systems	17
Setting Up Secure Communication with SSL	17
Configuring Secure Incoming Access with SSL	17
Generating SSL Certificates for Incoming Access	17
Setting Up OSM Cloud Native for Incoming Access	18
SSL Termination at the Load Balancer	21
Configuring Incoming HTTP and JMS Connectivity for External Clients	22
Configuring Access to External SSL-Enabled Systems	23
Loading Certificates for Outgoing Access	23
Enabling SSL on an External WebLogic Domain	24
Setting Up OSM Cloud Native for Outgoing Access	25
Adding Additional Certificates to an Existing Trust	27

11 Running the SAF Sample for OSM Cloud Native

Preparing the WebLogic System to Run the Emulator	2
Deploying the Emulator on the WebLogic System	4
Deploying the SimpleProvisioning Sample Cartridge	5
Preparing the OSM Cloud Native Instance	5
Validating the SAF Endpoints	8
Submitting Orders	8
Submitting Orders with HTTP	9
Submitting Orders with T3 over HTTP	9

12 Maintaining the OSM Cloud Native Environment

Before You Upgrade	1
About Upgrade Paths and Procedures	1
Rolling Restart	2
Identifying Your Upgrade Path	2
Offline Change Upgrade Paths	8
Online Change Upgrade Paths	9
Exceptions	10
Unsupported Tasks	10
OSM Cloud Native Upgrade Procedures	10
PDB Upgrade Procedure	11
OSM Application Upgrade	11
Offline Cartridge Deployment	11
Online Cartridge Deployment	12
Upgrades to Infrastructure	12
Miscellaneous Upgrade Procedures	13
Running Operational Procedures	14
Triggering Introspection	14
Scaling Down the Cluster	14
Scaling Up the Cluster	15
Restarting the Instance	15
Fast Delete	16
Upgrade Path Flow Chart	17

13 Upgrading your OSM Cloud Native Deployment

Overview of the Upgrade Steps	1
Installing WebLogic Kubernetes Operator	1

WKO Monitoring Mechanism	1
Operator Installation	2
Unregistering and Registering the Namespace with Weblogic Operator	2
Ingress Controller	3
Updating Specification Files	3
Updating the Project Specification	3
Updating the Instance Specification	4
Updating Shape Specification	7
Upgrading to OSM Cloud Native 8.0	7
Prerequisites for the Upgrade	7
Preparation Steps for the Upgrade	7
Updating the Secrets	9
Update Existing Secrets	9
Creating New Secrets	9
Upgrading the OSM DB Schema	11
OSM Application Upgrade	12

14 Moving to OSM Cloud Native from a Traditional Deployment

Supported Releases	1
Performing Pre-move and Post-move Tasks	1
About the Move Process	1
Pre-move Development Activities	3
Moving to an OSM Cloud Native Deployment	4
Quiescing the Traditional Instance of OSM	5
Exporting and Importing JMS Messages	5
Migrating JMS Messages By Using the Cloud Native Toolkit	5
Migrating JMS Messages By Using the WebLogic Remote Console	8
Upgrading the Database	9
Upgrading the Database Server	9
Preparing the Required Database Entities for OSM Cloud Native	10
Upgrading the OSM Schema and Cartridges	10
Switching Integration with Upstream Systems	10
Reverting to Your OSM Traditional Deployment	11
Cleaning Up	11

15 Debugging and Troubleshooting

Setting Up Java Flight Recorder (JFR)	1
Troubleshooting Issues with NGINX, OSM UI, and WebLogic Remote Console	2
Recovering an OSM Cloud Native Database Schema	6

Finding the Issue that Caused the OSM Cloud Native Database Schema Upgrade Failure	6
Restarting the OSM Database Schema Upgrade from the Point of Failure	7
Resolving Improper JMS Assignment	8
Common Problems and Solutions	9
Known Issues	14
Image Build Failure Due to OPatch Error	17

A Differences Between OSM Cloud Native and OSM Traditional Deployments

B Reference of Secrets Created by the Scripts

OSM Gateway OIDC Credentials Secret	B-4
OCA OIDC Credentials Secret	B-5
DB Credentials Secret	B-5
RCU DB Credentials Secret	B-5
TCPS DB Wallet Secret	B-6
WebLogic Credentials Secret	B-6
WebLogic Runtime Encryption Secret	B-6
FMW Wallet Encryption Secret	B-6
FMW Secure Wallet Secret	B-6
OSM Internal User Passwords Secret	B-6
OSM OIDC Credentials Secret	B-7
OSM Fluentd Credentials Secret	B-7
Certificate and Key to Access the Gateway HTTPS Endpoint	B-7
Certificate and Key to Access the OSM HTTPS Endpoint	B-7
Certificate and Key to Access the OSM WebLogic Remote Console HTTPS Endpoint	B-8
Certificate and Key to Access the OSM t3 over HTTPS	B-8
Trusted CA Injection	B-8
Secure Identity	B-8
ADB Wallet Secret	B-8
ADB Admin Secret	B-9
Cartridge Defined Custom User Credentials	B-9
External LDAP Information	B-9
SAF Credentials	B-10
Global Trust Credentials	B-10
Cross Domain Users in Remote Domains	B-10
Xtrust Secret	B-10
Generic Credentials	B-10
Security Scheme Credentials	B-11

C Leveraging OSM Cloud Native SAF Connectivity Patterns for Your Use-Case

About SAF Connectivity Patterns	C-1
Common Integration Patterns	C-2
OSM Cloud Native Colocated SAF	C-3
OSM Remote SAF	C-3
DNS Considerations	C-5
Cloud Native to Cloud Native Remote SAF	C-5

About This Content

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 OSM Cloud Native Deployment.

Documentation Accessibility

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

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Conventions

The following text conventions are used in this document.

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

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:

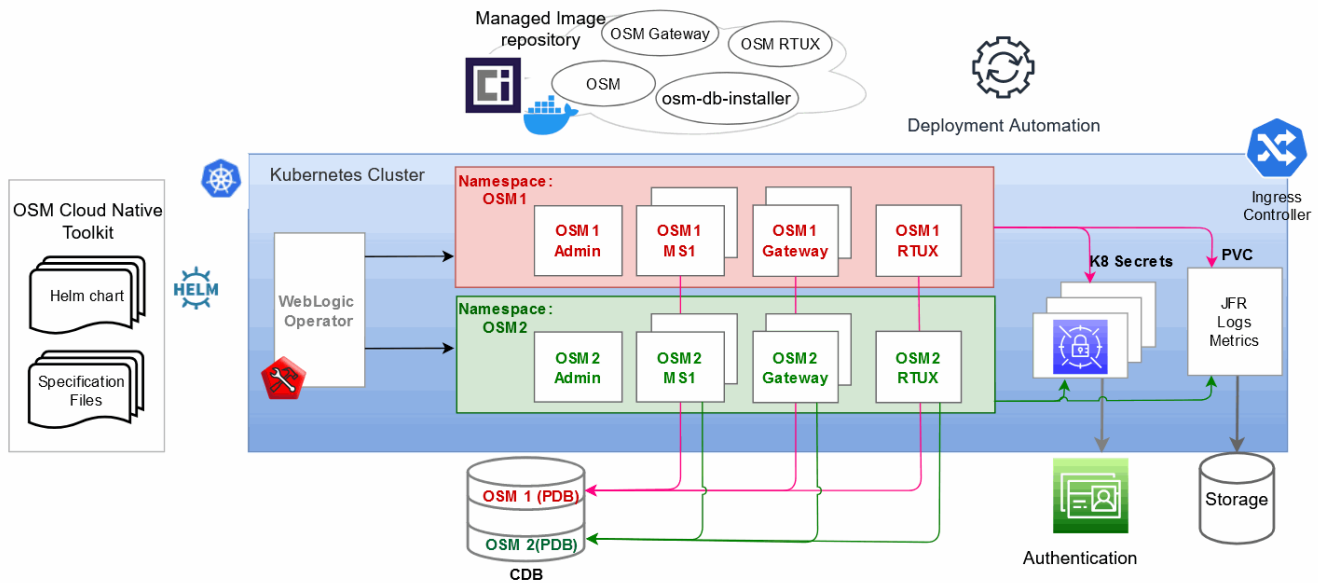
- **On Private Kubernetes Cluster:** OSM cloud native is certified for a general deployment of Kubernetes.
- **On Oracle Cloud Infrastructure Container Engine for Kubernetes (OKE):** OSM cloud native is certified to run on Oracle's hosted Kubernetes OKE service.

OSM Cloud Native Architecture

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

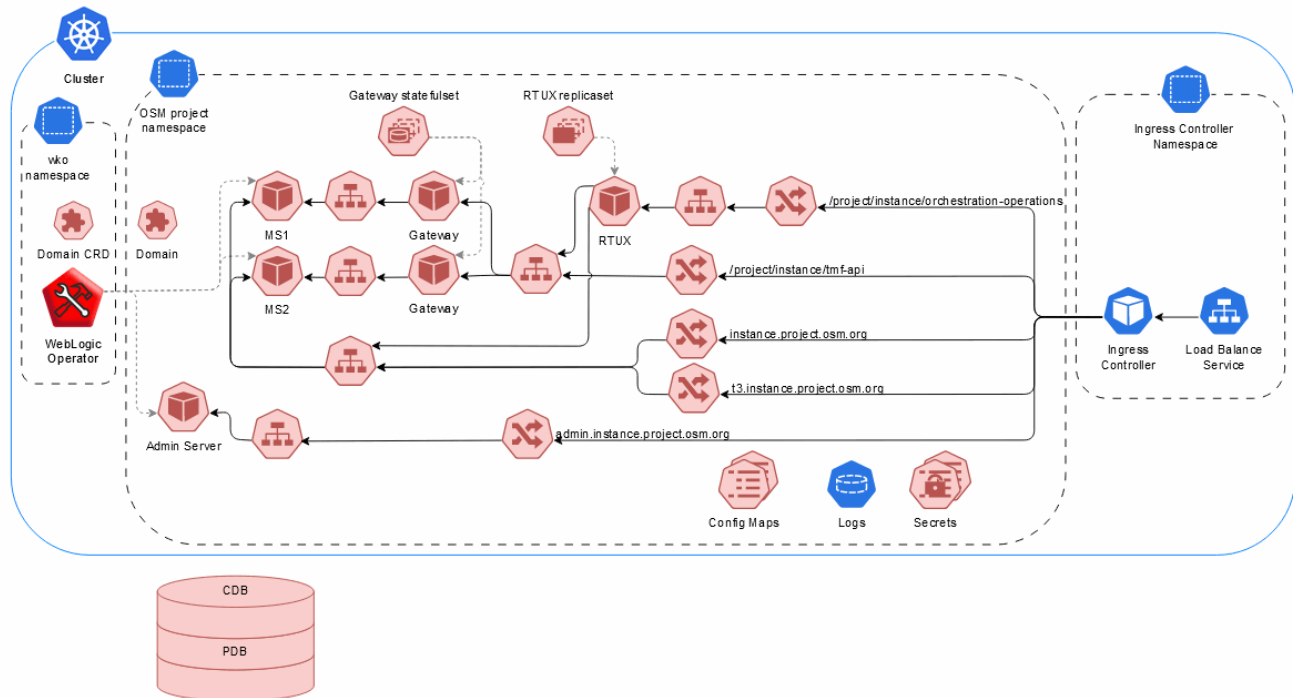
The following diagram illustrates the OSM cloud native architecture.

Figure 1-1 OSM Cloud Native Architecture

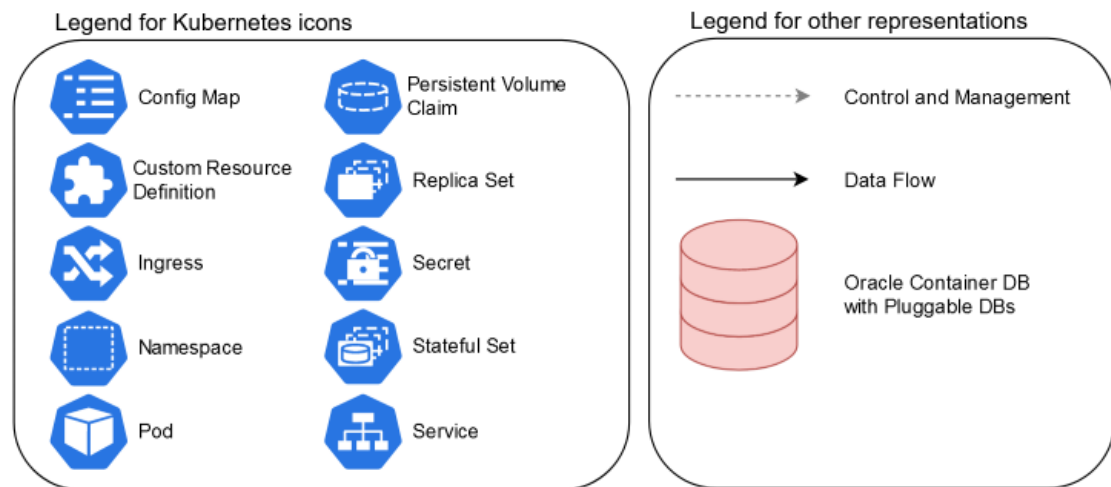


The following diagram illustrates the runtime deployment in Kubernetes.

Figure 1-2 OSM Cloud native Runtime Deployment in Kubernetes



The following diagram provides legend for the runtime deployment in Kubernetes diagram.

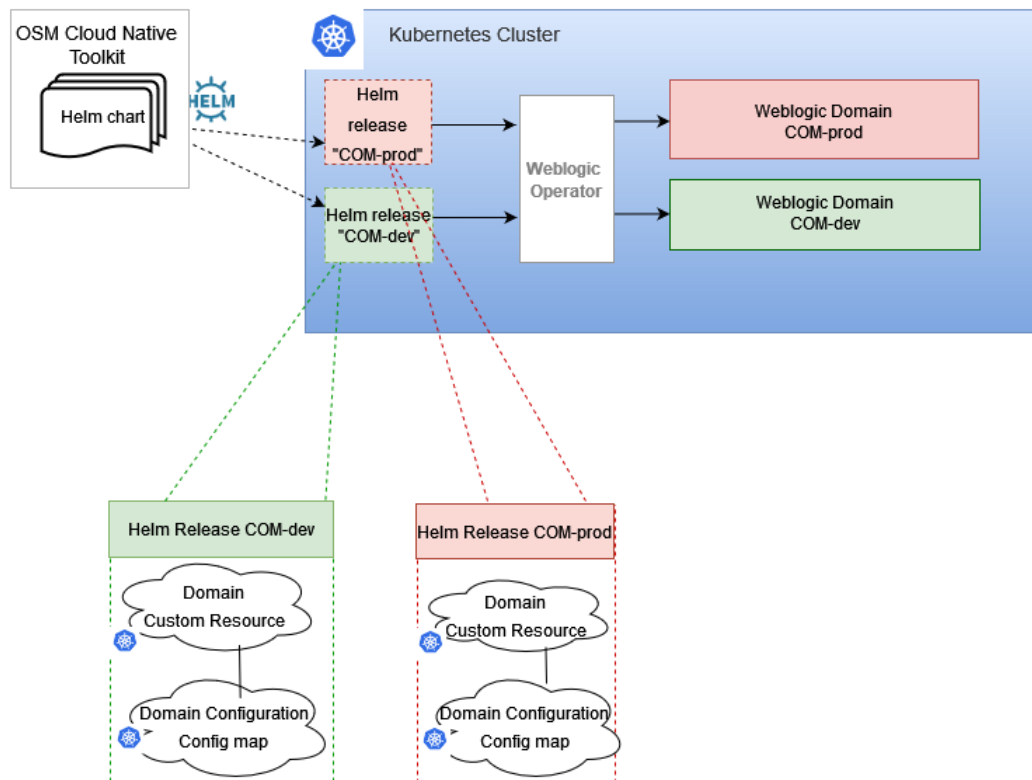
Figure 1-3 Legend for the run-time deployment in Kubernetes diagram

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 1-4 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 "[WKO Quickstart](#)" 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 Native 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

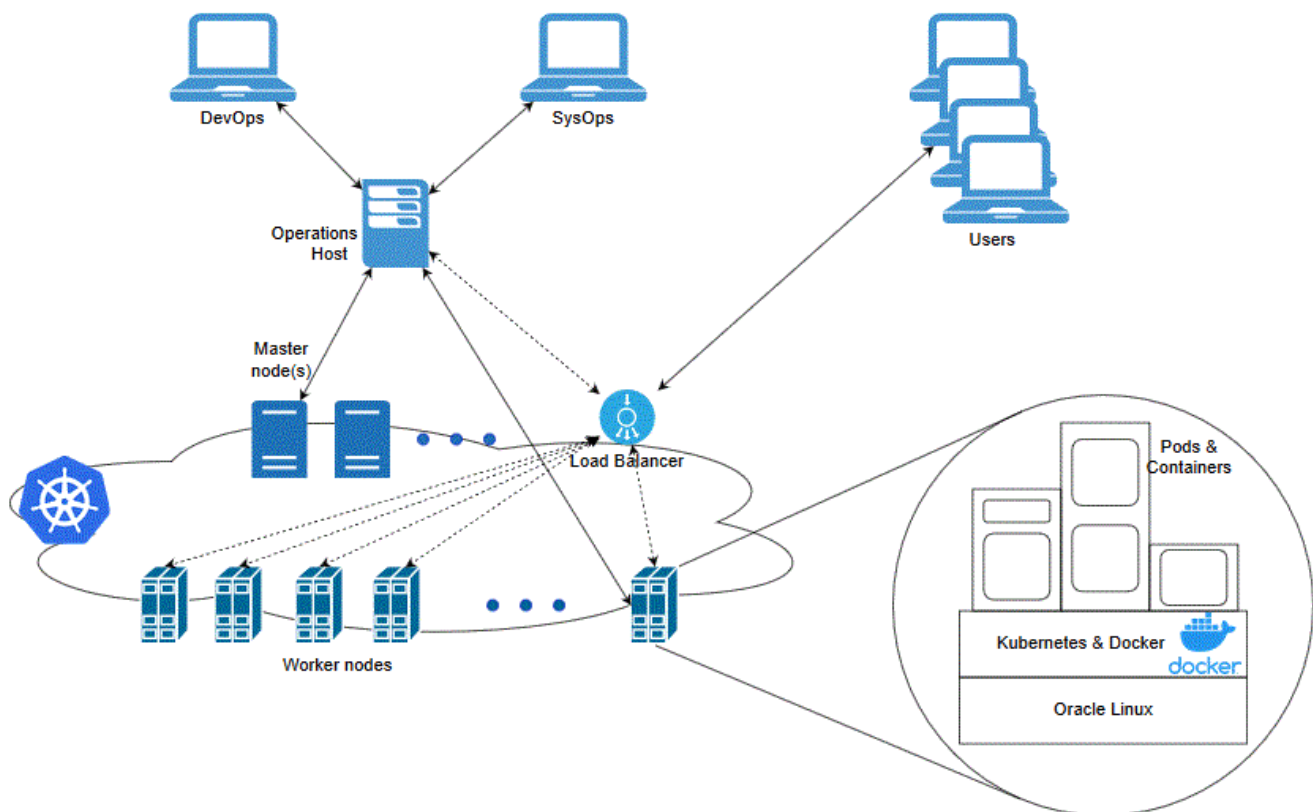
This section provides information about planning and setting up OSM cloud native environment. As part of preparing your environment for OSM cloud native, you choose, install, and set up various components and services in ways that are best suited for your cloud native environment. The following sections 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:

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

builder will be Docker or Podman.

- **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 16 GB of RAM and 3 CPUs, in terms of Kubernetes worker node capacity. A small increment is needed for WebLogic Kubernetes Operator and NGINX. 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 19c 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 the section about 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_;
ADMINISTER KEY MANAGEMENT SET KEY USING TAG 'tag' FORCE KEYSTORE
IDENTIFIED BY "sys_password" WITH BACKUP USING 'db_service_name_backup';
```

2. Provide the required GRANTS to the user by running the following SQL commands:

```
GRANT CREATE ANY CONTEXT TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_ WITH  
ADMIN OPTION;  
GRANT QUERY REWRITE TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_ WITH ADMIN  
OPTION;  
GRANT CREATE TABLE TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_ WITH ADMIN  
OPTION;  
GRANT GRANT ANY PRIVILEGE TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_ WITH  
ADMIN OPTION;  
GRANT CREATE USER TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_ WITH ADMIN  
OPTION;  
GRANT CREATE ANY VIEW TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_ WITH ADMIN  
OPTION;  
GRANT UNLIMITED TABLESPACE TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_ WITH  
ADMIN OPTION;  
GRANT CREATE MATERIALIZED VIEW TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_  
WITH ADMIN OPTION;  
GRANT CREATE SYNONYM TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_ WITH ADMIN  
OPTION;  
GRANT SELECT ON SYS.V_$PARAMETER TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_  
WITH GRANT OPTION;  
GRANT SELECT ON SYS.DBA_TABLESPACES TO  
_REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_ WITH GRANT OPTION;  
GRANT SELECT ON SYS.DBA_JOBS TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_  
WITH GRANT OPTION;  
GRANT SELECT ON SYS.DBA_AUTOTASK_CLIENT_JOB TO  
_REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_ WITH GRANT OPTION;  
GRANT EXECUTE ON SYS.DBMS_LOB TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_  
WITH GRANT OPTION;  
GRANT EXECUTE ON SYS.DBMS_LOCK TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_  
WITH GRANT OPTION;  
GRANT EXECUTE ON SYS.UTL_FILE TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_  
WITH GRANT OPTION;  
GRANT RESOURCE TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_ WITH ADMIN OPTION;  
GRANT IMP_FULL_DATABASE TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_ WITH  
ADMIN OPTION;  
GRANT CONNECT TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_ WITH ADMIN OPTION;  
GRANT DBA TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_ WITH ADMIN OPTION;  
GRANT EXP_FULL_DATABASE TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_ WITH  
ADMIN OPTION;  
GRANT EXECUTE ON DBMS_RANDOM TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_  
WITH GRANT OPTION;  
GRANT EXECUTE ON SYS.DBMS_SCHEDULER TO  
_REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_ WITH GRANT OPTION;  
GRANT EXECUTE ON UTL_HTTP TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_ WITH  
GRANT OPTION;  
GRANT EXECUTE ON DBMS_SQL TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_ WITH  
GRANT OPTION;  
GRANT EXECUTE ON UTL_TCP TO _REPLACE_THIS_TEXT_WITH_DB_ADMIN_NAME_ WITH  
GRANT OPTION;
```

3. Log into the PDB as the sys dba account for the PDB (defined by the
_REPLACE_THIS_TEXT_WITH_DB_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;
```

Using RDS or RDS Custom for Oracle

If you are using RDS for Oracle or RDS Custom for Oracle, you need to use the following SQL commands:

- Create a custom user:

```
CREATE USER _replace_this_text_with_user_name_ PROFILE DEFAULT IDENTIFIED  
BY <password> ACCOUNT UNLOCK;
```

- Provide all the required GRANTS for OSM to the above created user by running the commands listed [here](#). You can ignore the following commands while configuring the user:

```
GRANT GRANT ANY PRIVILEGE TO _replace_this_text_with_admin_name_ WITH  
ADMIN OPTION;  
GRANT SELECT ON SYS.DBA_AUTOTASK_CLIENT_JOB TO  
_replace_this_text_with_admin_name_ WITH GRANT OPTION;
```

- While configuring a user, instead of the above two GRANTS, run the following commands:

```
EXECUTE rdsadmin.rdsadmin_util.grant_sys_object( p_obj_name =>  
'DBA_AUTOTASK_CLIENT_JOB', p_grantee =>  
'_replace_this_text_with_admin_name_', p_privilege => 'SELECT',  
p_grant_option => true);
```

Note

Use uppercase to define all parameter values, for example DBA_AUTOTASK_CLIENT_JOB even if you are creating them with lower case.

```
GRANT SELECT_CATALOG_ROLE TO _replace_this_text_with_admin_name_ WITH  
ADMIN OPTION;
```

- To create the tablespace with RDS, you cannot specify the filenames for tablespaces as it only supports Oracle Managed Files (OMF). Run the following SQL from the osm user <pdadmin> to create a tablespace named 'OSM' with 100MB space, as an example.

```
create tablespace OSM datafile size 100m autoextend on;
```

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, 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.

Secondary Database Support for OSM Cloud Native

This topic provides detailed guidelines for adding a secondary database with the Oracle Data Guard for users who are currently operating a standalone or a single-instance Oracle Database. Adding a secondary database enhances high availability and provides data protection for critical business operations in OSM.

OSM provides high availability through the use of pod clustering and anti-affinity rules. Pod clustering involves deploying multiple instances of an application within a Kubernetes cluster, allowing for load balancing and failover in case of pod failure. Anti-affinity rules are used to distribute pods across different nodes to minimize the impact of node failures, ensuring that if one node goes down, the remaining nodes can still handle the application's traffic. If you are using a standalone or single-instance database, adding a Data Guard enabled secondary database ensures that OSM benefits from high database availability.

Before setting up the secondary database, it is essential to understand the following:

- This configuration places the primary and standby databases in separate Availability Domains within the same region.
- This setup increases system resilience by ensuring that if one Availability Domain encounters a failure, the other can continue to provide uninterrupted service.

Prerequisites

Before starting the setup process for the secondary database ensure that you have the following:

- Standalone Oracle Database.
- Oracle Data Guard setup and configuration files.
- Administrative access to both the primary and secondary database servers.

In Oracle Data Guard, the primary and secondary databases are key components in maintaining high availability and data protection.

Table 2-2 Role and Function of Primary and Secondary Database in OSM

Database Type	Role	Function
Primary	The primary database is the main database where all the write operations (DML statements) and updates occur. It handles all the user transactions and serves as the source of truth for the data.	It processes all read and write requests from applications and continuously sends changes to the secondary databases to keep them synchronized.

Table 2-2 (Cont.) Role and Function of Primary and Secondary Database in OSM

Database Type	Role	Function
Secondary	The secondary database (also known as the standby database) is a replica of the primary database. It is used for backup, disaster recovery, and can take over if the primary database fails.	It receives and applies the changes from the primary database to maintain data consistency. In the event of a primary database failure, the secondary database can be promoted to become the new primary database.

Active and Standby Roles

- **Active (Primary) Role:** In this role, the database is actively used for all read and write operations. It is the operational database where transactions and updates happen.
- **Standby Role:** The standby database is in a passive state. It continuously receives and applies changes from the primary database. It remains synchronized with the primary database and can be activated if needed, such as in the event of a primary database failure.

With a primary and secondary database setup, Oracle Data Guard ensures that there is minimal downtime and data loss in case of unexpected failures or disasters. The standby database can be promoted to primary if necessary, promoting business continuity.

In order to know more about how to configuring Primary and Secondary Database in OSM, refer to [Preconfiguration on Primary and Standby Database](#).

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.

OSM requires a helm version compliant with the information in the Compatibility Matrix. This should be installed and available as "helm" in the PATH.

Note

The Helm version mentioned in the commands is used as an example only. See *OSM Compatibility Matrix* for the recommended versions.

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

```
$ cd some-tmp-dir
$ wget https://get.helm.sh/helm-v3.9.3-linux-amd64.tar.gz
$ tar -zxvf helm-v3.9.3-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:"v3.9.3",
GitCommit:"c4e74854886b2efe3321e185578e6db9be0a6e29", GitTreeState:"clean",
GoVersion:"go1.20.3"}
```

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

Note

Oracle recommends that if you use any of the recommended components listed in the compatibility matrix, then consider upgrading all other components in the Kubernetes technology stack to the recommended versions.

For more details about 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 Nginx. The OSM cloud native toolkit provides samples and documentation that use Nginx as the ingress controller.

An external load balancer serves to provide a highly reliable single-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 NGINX as the Ingress Controller

Oracle recommends that you use the Ingress-NGINX Controller that supports the generic Kubernetes ingress API.

Weblogic Kubernetes Operator describes the installation and the usage of the Ingress-NGINX controller. Refer to [Install and Configure NGINX](#) for more information about installing and configuring NGINX.

Using Domain Name System (DNS)

A Kubernetes cluster can have many routable endpoints. 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 endpoint for your Kubernetes cluster.

OSM cloud native requires hostnames to be mapped to routable endpoints into the Kubernetes cluster. Regardless of the actual endpoints (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 cloud 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. For details about OSM system users in LDAP, see About Authentication and Authorization in the *OSM Security Guide*. 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 to integrate with external LDAP server for human users. For details on setting up the external LDAP server and the layout of the data within it, see "[Manage LDAP Providers in WLS via OSM](#)."

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 ingress-NGINX (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 meantime, 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.

You must create a Kubernetes service as per OCI LBaaS documentation to expose your Ingress controller via Load Balancer. Once this is done, you can describe the resulting service and note down the "EXTERNAL-IP" and "PORT(S)". The EXTERNAL-IP must be used for DNS mapping and in places where an access hostname-or-IP is required. PORT(S) provide the access port - the number before the colon ":" for each port set.

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.

Troubleshooting File Storage Service Provisioning of PVCs as Native FSS

If you see the "Pod cannot access file system due to insufficient permissions" issue, refer to the "[Troubleshooting File Storage Service Provisioning of PVCs](#)" section in Oracle Cloud Infrastructure documentation.

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 2-3 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.

① 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 image 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 image 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-g7zvz  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 image 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 Ingress-NGINX to perform load balancing activities for your Kubernetes cluster.

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 Generic NGINX Ingress controller:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: nginx
  name: smoke-nginx-ingress
  namespace: default
spec:
  rules:
  - host: smoke.nginx.osmtest.org
    http:
      paths:
      - backend:
          service:
            name: smoke-internal-nginx
            port:
              number: 80
```

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:Ingress_Port/
```

where `Ingress_Port` is the external port that Ingress 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 image 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 minimum resource requirements for building OSM cloud native images are:

- 2 CPUs
- 4GB RAM
- 40GB Disk Space

The following versions of Podman and Oracle Linux Server were used to obtain the requirements listed above:

- Oracle Linux Server 8.8
- Podman 4.4.1

See *OSM Compatibility Matrix* for more details about the required and supported versions of Podman, Oracle Linux Server, and the following components.

Note

It is not recommended to use podman-docker or similar rpm packages where a Docker executable is created only as a front for Podman. You must either use only Podman or a real Docker.

If Podman is used, the image build process might intermittently fail due to Podman's low open files default for image building. These build failures appear as Oracle OPatch failures in the build output. Image builder overcomes this issue by setting Podman's hard limit `ulimit -n -H` as the value for the open files limit, if it is defined. If the hard limit is undefined or unlimited, it uses the value 1048576.

See [Image Build Failure Due to OPatch Error](#) for more information.

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

- 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/>
- (Optional) OSM-CNTK zip. This is required only if you are building the CNTK image. Download this from My Oracle Support:
<https://support.oracle.com/>
- (Optional) Kubectl and Helm binaries. This is required only if you are building the CNTK image. See the [OSM Compatibility Matrix](#) for details about required and supported versions of Kubectl and Helm.
- Java, installed with JAVA_HOME set in the environment.
- Bash, to enable the `<tab>` command complete feature.

Configuring the OSM Cloud Native Images

The dependency manifest file describes the input that goes into OSM images. It is consumed by the image build process. The default configuration in the latest manifest file provides all the necessary components and required patches for creating OSM cloud native images. Refer to the Compatibility Matrix to know more about the supported versions of various components.

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.

```
# Specify the details of the Linux base image for OSM.
# Refer to the OSM documentation for certification statement on supported
# types and versions. This information is coded into the OSM image as a
# LABEL, for tracking purposes.
linux:
  vendor: Oracle
# uncomment below two lines when selecting linux 9
# version: 9.5
# image: container-registry.oracle.com/os/oraclelinux:9-slim
# comment below two lines when selecting linux 9
version: 8.10
image: container-registry.oracle.com/os/oraclelinux:8-slim
```

You can uncomment and modify the specification to choose which Linux version can be used as base image for OSM. The `linux` parameter specifies the base Linux image to be used as the base container 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: 21.0.7
  path: $CN_BUILDER_STAGING/java/jdk-21.0.7_linux-x64_bin.tar.gz
```

The `jdk` parameter specifies the JDK vendor, version, and the staging path. This parameter is referenced by the microservices for the JDK version that needs to be used.

```
fmw:
  version: 14.1.2.0.0
  path: $CN_BUILDER_STAGING/fmw/install/fmw_14.1.2.0.0_infrastructure.jar
```

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

```
oPatch:
  description: Weblogic Opatch
  patchNumber: 28186730
  patchId: 28186730_13.9.4.2.20
  path: $CN_BUILDER_STAGING/fmw/patch/p28186730_1394220_Generic.zip
```

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

```
fmwPatch:
  - description: PSU for WLS (JUL 2025 CPU)
    patchNumber: 38130086
    patchId: 38130086_14.1.2.0.0
    path: $CN_BUILDER_STAGING/fmw/patch/p38130086_141200_Generic.zip
  - description: PSU Coherence 14.1.2.0.3 (JUL 2025 CPU)
    patchNumber: 38018960
    patchId: 38018960_14.1.2.0.0
    path: $CN_BUILDER_STAGING/fmw/patch/p38018960_141200_Generic.zip
  - description: ADF Bundle Patch (JUL 2025 CPU)
    patchNumber: 38015961
    patchId: 38015961_14.1.2.0.0
    path: $CN_BUILDER_STAGING/fmw/patch/p38015961_141200_Generic.zip
```

```

- description: FMW THIRD PARTY BUNDLE PATCH (JUL 2025 CPU)
  patchNumber: 38101364
  patchId: 38101364_14.1.2.0.0
  path: $CN_BUILDER_STAGING/fmw/patch/p38101364_141200_Generic.zip
- description: JDBC BUNDLE PATCH (JUL 2025 CPU)
  patchNumber: 37925693
  patchId: 37925693_14.1.2.0.0
  path: $CN_BUILDER_STAGING/fmw/patch/p37925693_141200_Generic.zip
- description: Oracle Web Services Manager (OWSM) Bundle Patches (JUL
2025 CPU)
  patchNumber: 38184117
  patchId: 38184117_14.1.2.0.0
  path: $CN_BUILDER_STAGING/fmw/patch/p38184117_141200_Generic.zip
- description: RDA Release for FMW 14.1.2.0.0 (JUL 2025 CPU)
  patchNumber: 37887265
  patchId: 37887265_14.1.2.0.0
  path: $CN_BUILDER_STAGING/fmw/patch/p37887265_141200_Generic.zip
- description: UMS Bundle Patch (JUL 2025 CPU)
  patchNumber: 38032126
  patchId: 38032126_14.1.2.0.0
  path: $CN_BUILDER_STAGING/fmw/patch/p38032126_141200_Generic.zip
- description: CROSS DOMAIN SAF CONNECTIVITY ISSUE
  patchNumber: 37940403
  patchId: 37940403_14.1.2.0.250629
  path: $CN_BUILDER_STAGING/fmw/patch/p37940403_14120250629_Linux-
x86-64.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: 8.0.0
  wdt:
    version: 4.3.6
    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-app-archive.zip
  uxwarapplication: $CN_BUILDER_STAGING/cnsdk/osm-model/osm-fallout-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: 8.0.0
```

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

```
oracleInstantClient:
  version: 19.21.0.0.0
  basic:
    path: $CN_BUILDER_STAGING/instant-client/oracle-instantclient19.21-
    basic-19.21.0.0.0-1.x86_64.rpm
  sqlplus:
    path: $CN_BUILDER_STAGING/instant-client/oracle-instantclient19.21-
    sqlplus-19.21.0.0.0-1.x86_64.rpm
  tools:
    path: $CN_BUILDER_STAGING/instant-client/oracle-instantclient19.21-
    tools-19.21.0.0.0-1.x86_64.rpm
```

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

```
osmGwImage:
  name: osm-gateway
  tag: 8.0.0
rtuxMsImage:
  name: osm-runtime-ux-server
  tag: 8.0.0
ocaMsImage:
  name: "osm-cartridge-assembler"
  tag: 8.0.0
```

The `osmGwImage`, `rtuxMsImage` and `ocaMsImage` parameters specify details about the OSM Gateway, RTUX and OSM Cartridge Assembler microservices respectively.

- **OSM Cloud Native Toolkit Image**

This image is required only if OSM cloud native is being managed using the FluxCD-based gitops mechanism. The `osmCntk` parameter specifies the details about the OSM cloud native toolkit and the staging path for the OSM cloud native toolkit zip file, kubectl and helm binary.

```
osmCntk:
  name: "osm-cntk"
  tag: 8.0.0
  path: $CN_BUILDER_STAGING/osm-cntk/osm-cntk-*.zip
  baseImage: container-registry.oracle.com/os/oraclelinux:8
  # The binary provided for kubectl must conform to the guidance in the
  # compatibility matrix,
  # and must be for the same OS as the base image.
  kubectl:
    path: $CN_BUILDER_STAGING/tools/kubectl
```

```
# The binary provided for helm must conform to the guidance in the
compatibility matrix,
# and must be for the same OS as the base image.
helm:
  path: $CN_BUILDER_STAGING/tools/helm
```

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:8-slim** or **oraclelinux:9-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:8-slim** or **oraclelinux:9-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

In addition to OSM, OSM DB installer, OSM Gateway, and RTUX microservices images, OSM cloud native now enables you to create OSM-CNTK images as well.

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 unzip the OSM image builder file, **osm-image-builder.zip** to the workspace directory:

```
unzip ./osm-image-builder-*.zip -d workspace
```

3. (Optional) Download and copy the version of Oracle Instant Client in the manifest you are using to **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. Also, follow your organization's standard for \$http_proxy.

```
curl -x $http_proxy --output osm-image-builder/staging/instant-client/
oracle-instantclient19.21-basic-19.21.0.0.0-1.x86_64.rpm
https://download.oracle.com/otn_software/linux/instantclient/1921000/
oracle-instantclient19.21-basic-19.21.0.0.0-1.x86_64.rpm
```

```
curl -x $http_proxy --output osm-image-builder/staging/instant-client/
oracle-instantclient19.21-sqlplus-19.21.0.0.0-1.x86_64.rpm
https://download.oracle.com/otn_software/linux/instantclient/1920000/
oracle-instantclient19.21-sqlplus-19.21.0.0.0-1.x86_64.rpm
```

```
curl -x $http_proxy --output osm-image-builder/staging/instant-client/
oracle-instantclient19.21-tools-19.21.0.0.0-1.x86_64.rpm
https://download.oracle.com/otn_software/linux/instantclient/1920000/
oracle-instantclient19.21-tools-19.21.0.0.0-1.x86_64.rpm
```

4. Download JDK to workspace.
 - Determine the JDK versions specified in the manifest - there will be JDK 21.
 - Download each JDK version **tar.gz** into **./workspace/osm-image-builder/staging/java**.
 - Amend the manifest for each JDK section to include the correct path and filename if they differ.

```
#Example
cp jdk-21.0.7_linux-x64_bin.tar.gz ./workspace/osm-image-builder/staging/
java/jdk-21.0.7_linux-x64_bin.tar.gz
```

Note

You can modify the manifest by substituting the default tags with tags that have relevance to your specific work. Specifying tags is not required, but it is recommended.

```
# Information about the OSM Gateway image
osmGwImage:
  name: osm-gateway
  tag: 8.0.0
```

```
# Information about the OSM rtux m-s image
rtuxMsImage:
```

```
name: osm-runtime-ux-server
tag: 8.0.0
```

```
# Information about the OSM oca m-s image
ocaMsImage:
  name: "osm-cartridge-assembler"
  tag: 8.0.0
```

5. (Optional) You need to do this if you want to create an image for OSM cloud native toolkit:
 - a. Download and copy the **osm-cntk** zip file into the **./workspace/osm-image-builder/staging/osm-cntk** directory.
 - b. Download and copy the Kubectl and Helm binaries into the **./workspace/osm-image-builder/staging/tools** directory.

```
cp osm-cntk-*.zip ./workspace/osm-image-builder/staging/osm-cntk/
```

```
cp kubectl ./workspace/osm-image-builder/staging/tools/kubectl
cp helm ./workspace/osm-image-builder/staging/tools/helm
```

6. (Optional) You need to do this if the external system is using the SSL certificate as a self signed certificate. For the OSM Gateway to successfully participate in a handshake with an external system to emit the message, the SSL certificates from the external domain must be made available to the OSM Gateway. This can be achieved adding the external system SSL certificates to the Java keystore. To add the SSL certificates to the Java keystore, do the following:

```
$ cd $CN_BUILDER_STAGING/java
```

```
#untar the java tar file
```

```
$ tar -xvzf jdk-21.0.7_linux-x64_bin.tar.gz
```

```
# navigate to the bin folder
```

```
$ cd jdk-21.0.7/bin
```

```
#import the certificate to java keystore
```

```
keytool -import -trustcacerts -keystore ../lib/security/cacerts -alias
ALIAS_NAME -file /path/to/certificate
```

```
# repeat import for each set of certificates until all affected external
systems are covered
```

```
#tar the java folder again
```

```
tar czf jdk-21.0.7_linux-x64_bin.tar.gz jdk-21.0.7
```

7. 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_14.1.2.0.0_infrastructure.jar ./workspace/osm-image-builder/
staging/fmw/install/fmw_14.1.2.0.0_infrastructure.jar
```

8. 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. If the image build process fails with credential errors from My Oracle Support, retry in some time or switch to the recommended option.

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

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

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

- Select the images to create using the **-c** command-line argument. If you are specifying more than one image to create, provide a comma-separated list. Valid values are:
 - **osm**: OSM image.
 - **dbinstaller**: OSM DB Installer image
 - **gateway**: OSM Gateway microservice image
 - **rtuxms**: OSM Runtime microservice image
 - **oca**: OSM Cartridge Assembler microservice image
 - **cntk**: OSM cloud native toolkit image

- To build all images, for example:

```
./workspace/osm-image-builder/bin/build-osm-images.sh -f $DMANIFEST -
s $STAGING -c osm,dbinstaller,gateway,rtuxms,oac,cntk
```

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 a 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: 8.0.0
- osm-cn-db-installer: 8.0.0
- osm-gateway: 8.0.0
- osm-runtime-ux-server: 8.0.0
- osm-cntk: 8.0.0

Once images are built in a CI pipeline, the pipeline uniquely tags the images and pushes them to an internal image 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
- Install the WKO and Ingress-Nginx 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 image 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
```

Using Oracle Autonomous Database Serverless

OSM cloud native provides experimental capability to use Oracle Autonomous Database Serverless (the transaction-based variant of ADB-S) on a shared infrastructure.

However, this functionality has the following limitations:

- This capability is made available only for exploration and investigation purposes. It must not be used for production or similar environments.
- Online order-based purging is not supported.

- Order purge (online and partition-based) is not supported.
- Performance under high order volume is not quantified.

Both OSM schema and RCU schema can be installed on Autonomous Database.

Note

If you choose to use Autonomous Database, instead of Standard DB (PDB), then both RCU and OSM schemas will be created on the same Autonomous Database.

For more information about Autonomous Database, see the documentation at: <https://docs.oracle.com/en/cloud/paas/autonomous-database/serverless/adbsb/index.html>.

Using Wallet-based Connection

OSM uses wallet-based connection associated with Database Resident Connection Pooling (DRCP), which is a connection pooling mechanism in Oracle Database that allows you to manage database connections efficiently.

For information about using wallet-based connection in Oracle Autonomous Database, see ADB-S documentation at: <https://docs.oracle.com/en/cloud/paas/autonomous-database/serverless/adbsb/getting-started.html>.

While downloading the wallet, provide a password that needs to be provided while creating secrets as well. Ensure that you remember the password used at the time of downloading the wallet.

Note

Do not change the contents of the wallet.

Unzip the wallet and copy it to your local filesystem, which will be used while creating secrets.

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 although Autonomous Database Serverless does.

Note

As a pre-requisite to using the toolkit for either installing the OSM database or creating an OSM instance, you must create OSM database secrets and RCU DB secrets.

The toolkit provides sample scripts for this purpose. However, they are not pipeline-friendly. Use the scripts for creating an instance manually and quickly, but not for any automated process for creating instances. See [Reference of Secrets Created by the Scripts](#) for the full list of secrets that OSM cloud native can process, including naming and content requirements as well as when each secret is required. You must create secrets prior to running the `install-osmdb.sh` or `create-instance.sh` scripts. The `manage-instance-credentials.sh` script is the main tool for manual secrets management. This script uses the configuration settings defined in your spec files to reduce the number of prompts for user input.

At the time of creating secrets, you are prompted to answer a question related to OSM schema followed by questions related to wallet. The following is a sample:

Are you using Autonomous Database Serverless (Experimental OSM feature)?
(select number from menu)

- 1) Yes
- 2) No

After you select 1, questions related to the ADB-S wallet are prompted.

Updating the Instance Specification

In the instance specification, modify the database parameters as follows:

- Set `db.type` to "ADB".
- For `defaultTablespace`, specify the default tablespace name. For Oracle ADB-S, the default is "temp".

Note

Refer to ADB-S documentation and ensure the passwords used conform to the specified requirements.

db:

```
type: "ADB" # Acceptable values are STANDARD and ADB
#serviceName: dbserver-servicename
# This DB protocol is all applicable for all database connections.
# Default value is TCP, Uncomment and change it to TCPS when required.
# If TCPS is selected, the dbwallet "<project>-<instance>-db-ssl-wallet"
secret must exist
#protocol: TCP
# datasourcesPrimary section is applicable only for STANDARD DB. For ADB,
values will be used from Autonomous Database Serverless secrets+configMap.
datasourcesPrimary:
  port: 1521
  # Provide the DB server hostname/IP address
  #host: dbserver-ip
  #
  # If using RAC, provide 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 these out "#vips:" and "#instances:"
  #
  #vips:
  # - vip1-ip
  # - vip2-ip
  # --- OR ---
  #instances:
  # - instance-1
```

```
# - instance-2

# Default log level. Valid value
#
## The levels in descending order are:
## SEVERE (highest value)
## WARNING
## INFO
## CONFIG
## FINE
## FINER
## FINEST (lowest value)
##
logLevel: "WARNING"
#
# The remaining parameters must match the values used when the PDB was
# created. Failure to match will result in dbInstaller errors
#
# The default tablespace name of OSM schema
defaultTablespace: "temp"
# The temporary tablespace name of OSM schema
tempTablespace: "TEMP"
# The time zone offset in seconds
timezoneOffsetSeconds: "-28800"
# The model data tablespace name of OSM schema
modelDataTablespace: "temp"
# The model index tablespace name of OSM schema
modelIndexTablespace: "temp"
# The order data tablespace name of OSM schema
orderDataTablespace: "temp"
# The order index tablespace name of OSM schema
orderIndexTablespace: "temp"
```

Run the following command:

```
$OSM_CNTK/scripts/manage-instance-credentials.sh -p project -i instance -s
specPath create osmdb,rcudb
```

You will be prompted with questions related to OSM schema followed by questions related to the wallet:

```
Do you have pre-existing ADB-S wallet secrets conforming to OSM CNTK ?
(select number from menu)
1) No
2) Yes
#? 1
```

```
Provide Autonomous Database Serverless credentials for 'dev-quick' ...
Supplied values must align with rules dictated by component that owns the
password policy
ADB-S Admin Username: ADB-S admin_username
ADB-S Admin Password: ADB-s password
ADB-S unzipped wallet location: /file/location/to/unzipped/wallet
```

```
ADB-S wallet password: Wallet_password
ADB-S tns_alias: alias_in_wallet
```

TNS_Alias refers to the aliases mentioned in the **tnsnames.ora** wallet.

After creating secrets, you will be able to see the following secrets and configMap:

- `secret/project-instance-db-wallet` created
- `secret/project-instance-db-secret` created
- `configmap/project-instance-db-config` created

Using RDS or RDS Custom for Oracle

OSM cloud native provides the capability to use Relational Database Service (RDS) for Oracle. RDS is a managed database service that helps you set up, operate, and scale relational databases in the Amazon Web Services (AWS) cloud.

Both OSM schema and RCU schema can be installed on an RDS Database for Oracle.

Updating the Instance Specification

In the instance specification, modify the database parameters as follows:

- Set `db.rcuDb.honorOMF` to `true`.
- Provide default and temp tablespace names.

Note

While creating the tablespace with RDS, you cannot specify the filenames for tablespaces as it only supports Oracle Managed Files (OMF).

Run the following SQL as the `<pdadmin>` user set up for OSM to create a tablespace named 'OSM' with 100MB space, as an example.

```
create tablespace OSM datafile size 100m autoextend on;
```

instance.yaml

```
db:
  type: "STANDARD" # Acceptable values are STANDARD and ADB
  ...
  ...
  rcuDb:
    honorOMF: true # Enable this if using AWS RDB; for others, enable if DB
server is setup to use Oracle Managed Files.
```

Installing WebLogic Kubernetes Operator (WKO) and Ingress Controller

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

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

- WebLogic Kubernetes Operator (WKO) container
- Ingress Controller

Note

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

Before installing the WKO and the Ingress Controller, do the following tasks:

- Remove the instances of the WKO and Ingress Controller 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

For information about installation packages and installation instructions, visit the WebLogic Kubernetes Operator (WKO) documentation at: <https://oracle.github.io/weblogic-kubernetes-operator/managing-operators/installation/#install-the-operator>.

For information about the recommended WKO version, see the *OSM Compatibility Matrix*.

For example, if the recommended WKO version is 4.1.2:

- For details about WKO 4.1.2, see the WKO documentation at: <https://github.com/oracle/weblogic-kubernetes-operator/releases/tag/v4.1.2>.
- Choose a namespace for the operator and set the WLSKO_NS environment variable to the Kubernetes namespace in which WKO will be deployed.
- Use **--version=4.1.2** during the installation.
- Set the label to the same as namespace using **--set domainNamespaceLabelSelector=namespace=enabled**. Do not use the default label "weblogic-enabled" because it is not advised to have multiple operators installed with the same label.

After the successful installation of WKO, validate that the operator is installed, by running the following command:

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

Note

Prior to version 3.1.0, the operator supported specifying the namespaces that it manages only through a list. From release 4.0.0 onwards, WKO supports a list of namespaces, a label selector, or a regular expression matching the namespace names. OSM cloud native supports the label selector method.

If you are upgrading from OSM Cloud Native release 7.4.1 and an older version of the WebLogic Kubernetes Operator, see "[Maintaining the OSM Cloud Native Environment](#)" for special considerations.

Installing the Ingress Controller

You can use any Ingress Controller that conforms to the standard Kubernetes ingress API and that supports annotations needed by OSM. Oracle does not certify individual Ingress controllers to confirm this generic compatibility. Refer to "[Working with Ingress, Ingress Controller, and External Load Balancer](#)" for more details on annotations and generic Ingress configurations.

You can find examples in GitHub at: "<https://github.com/kubernetes/ingress-nginx>".

Weblogic Kubernetes Operator describes the installation and the usage of the NGINX Ingress controller. Refer to "[Install and Configure NGINX](#)" for more information.

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 that you set the path to your private specification repository in your environment.

To set the environment variables, 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
```

Registering the Namespace

To register your namespace, complete the following steps:

1. Set the environment variables that you will use throughout the installation and registration steps.

```
export WKO_NS=wkoversion           # Operator namespace
export WLSKO_NS=wlsko             # Logical name for the WebLogic
operator target
export PROJECT=sr                 # Your domain (project) namespace
```

where *version* is the version of the WebLogic Kubernetes Operator that you are using.

2. Add the Operator Helm Chart Repository. Ensure that you are using the latest operator chart information.

```
helm repo add weblogic-operator https://oracle.github.io/weblogic-
kubernetes-operator/charts --force-update
```

3. Install the WebLogic Kubernetes Operator into its dedicated namespace. This installation will configure the operator to monitor all namespaces that have the label you specify.

```
helm install $WKO_NS \
  weblogic-operator/weblogic-operator \
  --namespace $WKO_NS \
  --version compatible-version \
  --set "domainNamespaceLabelSelector=$WKO_NS=enabled"
```

where *compatible-version* is the version of the WKO that is compatible with your OSM instance.

Note

Refer to the *Order and Service Management Compatibility Matrix* to determine the correct version for your OSM instance. For more information about installing WebLogic Kubernetes Operator, refer to the WebLogic Kubernetes Operator documentation.

4. Validate the Operator Installation by checking that the operator and webhook pods are deployed and running.

```
kubectl get pods -n $WKO_NS
```

5. Create and Label the Domain Namespace. If the domain namespace (project) does not already exist, create it and add the necessary label so that the operator can manage it.

```
kubectl create namespace $PROJECT
kubectl label namespace $PROJECT $WKO_NS=enabled
```

6. . Confirm Namespace Label. Ensure the label \$WKO_NS=enabled is present on your domain namespace.

```
kubectl get namespace $PROJECT --show-labels
```

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.

Configuring OpenID Connect for OSM Microservices

This section provides information about configuring OpenID Connect (OIDC) for OSM microservices.

Prerequisites

Create a unified authentication service that supports the standard OIDC protocol such as Keycloak.

Securing REST APIs

OSM microservices such as OSM Gateway are integrated with OIDC. OIDC is an authentication protocol that enhances security and simplifies user identity management in applications. It is an extension of OAuth 2.0, which is primarily focused on authorization.

Note

IDP Identity Provider (IDP): An Identity Provider (IDP) is a system or service that authenticates and provides identity information about users to other applications, services, or systems. It plays a crucial role in identity and access management (IAM). The primary function of an IDP is to verify a user's identity and provide authentication to various relying parties (applications or services) without the need for the user to authenticate separately with each of them. Examples of IDPs: Keycloak and IDCS (Oracle Identity Cloud Service).

OIDC OpenID Connect (OIDC): OpenID Connect (OIDC) is an identity layer built on top of the OAuth 2.0 protocol. It is designed to enable secure and standardized authentication and user information sharing between clients (OSM cloud native) and IDPs. OIDC provides a framework for identity verification, user authentication, and obtaining user profile information.

OSM Gateway authenticates external requests to OSM REST APIs (for TMF and for Fallout Exception Management) to ensure service security. To do this, it is configured as the relying party for an external OpenID provider, which has to be set up as an authentication provider implementing OpenID Connect. It connects to the authentication system based on the OIDC protocol and provides unified authentication for connecting to internal services. To configure OSM Gateway microservice as the relying party for a third-party OpenID provider, set up an authentication provider that implements OpenID Connect. With this configuration, OSM

Gateway REST APIs are accessible from the OpenID provider and authorize users to access protected data.

Creating OSM Secret for OIDC

OSM cloud native uses a secret for the administrator to provide the required OIDC information to secure its REST APIs. This secret can be created using the `manage-instance-credentials.sh` script from the OSM cloud native toolkit, with the `gw0idc` or `oca0idc` option. Make sure you have the required credentials available before running this script.

```
$OSM_CNTK/scripts/manage-instance-credentials.sh -p project -i instance -s
spec_path create gw0idc,oca0idc
```

where `gw0idc` and `oca0idc` specify the details of the OIDC IdP that will be used to authenticate access to the Gateway and OCA APIs.

Note

The `oidc` secret option is now deprecated. You can use `gw0idc` in place of `oidc`. If provided, it behaves the same as `gw0idc` for backward compatibility. The script will ask for OIDC details for both TMF and Fallout API with the `gw0idc` option. You can use the same details for both the APIs. For the `oca0idc` option, the script will additionally ask if want to configure OIDC details for Solution Designer. If you affirm that you do, it will then ask for OIDC details for Solution Designer.

This command creates the secrets `project-instance-gateway-credentials` and `project-instance-oca-credentials`.

Here are the parameters that the script requests in order to create the secret:

Table 4-1 **OIDC Parameters**

OIDC Parameters	Description
Auth URL	This is the URL where the client initiates the authentication process. You are redirected to this URL to log in and grant permissions to the client application.
client_id	Identifier for the client application.
client_secret	Secret key shared between the client and OIDC server.
Token URL	This is the URL where the client exchanges the authorization code for an access token and ID token. This step typically occurs after you have successfully authenticated and granted permissions.

The OCA Microservice is disabled by default. The OIDC secret for OCA is needed only when you enable the microservice. To enable, set the OSM cartridge assembler property to true in the project specification file

```
osm-cartridge-assembler.enabled: true
```

An OIDC secret is needed to use the OSM Gateway microservice. The exception to this requirement is if you have the runtime type property set to WLS in the project specification file.

```
omsConfig.project.osm_runtime_type: WLS
```

Note

Refer to your OpenID Connect authentication provider's documentation for details on these parameters.

You need to configure the audience and scope values as specified below in IdP to access the OSM Microservice APIs. You will not be able to use the authentication token generated for one API (using api specific scope) to access any other API.

Table 4-2 Audience and Scope Configuration for OSM Microservices

API	Scope	Audience
TMF (GW m-s)	tmf	tmf
Fallout (GW m-s)	fallout	fallout
OCA (OCA m-s)	catalog	catalog

To verify the above OIDC parameters, run the following cURL command:

```
curl --noproxy '*' -i -H 'Authorization: Basic base64Encoded_client_id:client_secret' -H
'Content-Type: application/x-www-form-urlencoded' -XPOST token_url -d
'grant_type=client_credentials&scope=scope'
```

Fundamentally, OpenId Connect defines two tokens:

- **IDToken:** The ID Token carries information about the authenticated user. This information is often used for user authentication and identity verification.
- **AccessToken:** This is a short-lived token generated by OIDC provider which is sent to OSM Gateway to gain access to resources defined in the token.

Note

In case of troubleshooting authentication issues using the OIDC token, make sure that the '**aud**' listed in the OIDC token has the desired **client_id** added.

Note

OIDC details are required for accessing OSM TMF REST endpoints, Fallout Exception REST endpoints and OCA REST endpoints. Human users accessing OSM UIs like Task Web client, Order Management UI, Order Operations and Fallout Order Management UI would still be users from External LDAP or SAML.

Keycloak as an example IDP

Keycloak is a widely used open-source identity and access management (IAM) system that can act as an IDP, supporting OIDC among other authentication and authorization protocols. It provides a complete solution for user authentication and user management. For more information on Keycloak, see Keycloak documentation at: <https://www.keycloak.org/>.

IDP Certificate Management

If the OSM Gateway or RTUX microservice cannot establish an SSL connection to the IDP due to the certificate being unknown, you can introduce the IDP's certificate into the OSM Gateway and RTUX microservices while building microservice images.

To introduce the IDP's certificate:

1. Identify the JDK version used in the **osmCnInfrastructureImage.jdk** section from the manifest file used for building OSM cloud native images. You would have obtained this JDK file as part of creating images.
2. Add the certificate to this JDK's truststore and rebuild the OSM Gateway image, and the RTUX image. Refer to the example below for more details.

Note

The example below assumes JDK 21.0.7, but use the JDK as per your manifest file:
cd \$CN_BUILDER_STAGING/java

```
cd $CN_BUILDER_STAGING/java

#untar the java tar file
tar -xvzf jdk-21.0.7_linux-x64_bin.tar.gz

# navigate to the bin folder
cd jdk-21.0.7/bin

#import the certificate to java keystore
keytool -import -trustcacerts -keystore jdk-21.0.7/lib/security/cacerts -
alias ALIAS_NAME -file /path/to/idpcert.pem
#Run this command to add multiple certificates if any (Ex: OIDC )
#Example: keytool -import -trustcacerts -keystore $JAVA_HOME/lib/security/
cacerts -alias foo -file idpcert.pem

#tar the java folder again
tar czf jdk-21.0.7_linux-x64_bin.tar.gz jdk-21.0.7

#build the image again using OSM image builder tool
#To create OSM Gateway image, use "-c gateway" as shown:
./workspace/osm-image-builder/bin/build-osm-images.sh -f $DMANIFEST -
s $STAGING -c gateway

#To create rtux m-s image, use "-c rtuxms" as shown:
./workspace/osm-image-builder/bin/build-osm-images.sh -f $DMANIFEST -
s $STAGING -c rtuxms
```

(Optional) Verify the Certificate in the Keystore:

```
cd $CN_BUILDER_STAGING/java/jdk-21.0.7/bin
keytool -list -keystore $CN_BUILDER_STAGING/java/jdk-21.0.7/lib/security/
cacerts -alias ALIAS_NAME
```

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. For more information on creating secrets, see [Reference of Secrets Created by the Scripts](#).

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

The toolkit provides sample scripts for this purpose. However, they are not pipeline-friendly. Use the scripts for creating an instance manually and quickly, but not for any automated process for creating instances. See [Reference of Secrets Created by the Scripts](#) for the full list of secrets that OSM cloud native can process, including naming and content requirements as well as when each secret is required. 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 -s  
$SPEC_PATH \  
    create \  
    osmdb,rcudb,wlsadmin,osmldap,opssWP,wlsRTE,gwOidc,ocaOidc
```

where:

- `osmdb` specifies the credentials for connecting to the OSM PDB mentioned in the instance specification and credentials required to create 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**)
 - OSM Gateway user (username is **osm-tmf-internal**)
- `rcudb` specifies the credentials required to create 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.
- `gwOidc`, `ocaOidc` authenticate users and provide access for the REST APIs exposed by microservices. For more details about configuring an authentication provider using OpenID Connect, see [Configuring OpenID Connect for OSM Microservices](#).

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
sr-quick-gateway-credentials
sr-quick-oca-credentials
```

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](#).

Configuring OSM Cloud Native to Connect with a TCPS Enabled Database

This section describes what you need to do to configure your OSM cloud native instance to connect with a TCPS enabled database. Follow this procedure to successfully configure your instance:

1. Enabling TCPS from the Instance Specification

Uncomment `protocol` under the `db` section in the instance specification and provide TCPS instead. Populate the `db` section with the `service name` and the `db.datasourcesPrimary` section with the database's host, and `tcps port`. If you are using a RAC database, provide scan IPs and VIPs/Instances as well.

Note

Providing `db.datasourcePrimary.host` is mandatory. If you are using a RAC DB, you need to give a SCAN address instead.

```
db:
  type: "STANDARD" # Acceptable values are STANDARD and ADB
  serviceName: dbserver-servicename
  # This DB protocol is all applicable for all database connections.
  # Default value is TCP, Uncomment and change it to TCPS when required.
  # If TCPS is selected, the dbwallet "<project>-<instance>-db-ssl-wallet"
secret must exist
  protocol: TCPS
  # datasourcesPrimary section is applicable only for STANDARD DB. For
ADB, values will be used from Autonomous Database Serverless
secrets+configMap:
  datasourcesPrimary:
    port: 2484
    # Provide the DB server hostname/IP address
    host: dbserver-ip
```

2. Create Oracle Wallet with Database SSL Certificate

OSM expects a single wallet containing the SSL certificates of the database that needs to be connected using TCPS. You can create a wallet using the **orapki** tool. You can obtain the **orapki** tool from a Fusion Middleware installation. The tool relies on Java. Use the *OSM Compatibility Matrix* to ensure that you are using the right versions of Fusion Middleware and Java.

- a. Check the **orapki** version.

```
$ $FMW_HOME\Oracle_Home\oracle_common\bin>orapki
Oracle PKI Tool : Version 12.2.1.4.0
```

- b. Create an Oracle wallet using **orapki**.

```
$ $FMW_HOME\Oracle_Home\oracle_common\bin>orapki wallet create -wallet
<ssl_wallet_dir> -auto_login -pwd <password>
```

- c. Add certificates to the wallet one at a time.

```
$ $FMW_HOME\Oracle_Home\oracle_common\bin>orapki wallet add -wallet
<ssl_wallet_dir> -pwd <password> -trusted_cert -cert <cert_dir>/<xx.crt>
```

where,

- `ssl_wallet_dir` is the directory where the wallet is to be created.
- `cert_dir` is the directory where the certificate from the database is stored.
- `xx.crt` is the certificate obtained from the database.

Note

You need to repeat this step for each certificate that you need to include.

3. Create DBWallet Secret using Oracle Wallet

Create a secret for the DBWallet using the Oracle wallet you created in the previous step.

```
$ $OSM_CNTK/scripts/manage-instance-credentials.sh -p <project> -i  
<instance> -s <spec_path> create dbwallet
```

Unzipped db wallet location: <ssl_wallet_dir>

secret/<project>-<instance>-db-ssl-wallet configured

4. Create or Update the OSMDB Secret

Create or update the **osmdb** secret to provide the login credentials for the TCPS enabled database.

```
$ $OSM_CNTK/scripts/manage-instance-credentials.sh -p <project> -i  
<instance> -s <spec_path> create osmdb
```

```
$ $OSM_CNTK/scripts/manage-instance-credentials.sh -p <project> -i  
<instance> -s <spec_path> update osmdb
```

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 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 image 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`.

Note

`osmDBInstaller.resources` is used by the pod while creating, upgrading or dropping the OSM and RCU schemas.

Table 4-3 Calculating Sub-partitions

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

- 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.

- Check the console to see if the DB installer is installed successfully.
- 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
```

- Clean up the failed pod by running the following command:

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

- 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 4-4 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:

```
# OSM CN Cluster Image
image: "osm"

# OSM Gateway Image
osm-gateway:
  image: "osm-gateway"

# OSM Runtime UX Server Image
osmRuntimeUXServer:
  image: "osm-runtime-ux-server"
```

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 image 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 Remote Console Online Help for Oracle WebLogic Server 14.1.2.0*. 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 gigabyte (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:
  type: "STANDARD" # Acceptable values are STANDARD and ADB
  #serviceName: dbserver-servicename
  # This DB protocol is all applicable for all database connections.
  # Default value is TCP, Uncomment and change it to TCPS when required.
  # If TCPS is selected, the dbwallet "<project>-<instance>-db-ssl-wallet"
secret must exist
  #protocol: TCP
  # datasourcesPrimary section is applicable only for STANDARD DB. For
ADB, values will be used from Autonomous Database Serverless
secrets+configMap.
  datasourcesPrimary:
    port: 1521
    # Provide the DB server hostname/IP address
    #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. If you are using Oracle Cloud Infrastructure Load Balancer as a Service (LBaaS), run the following command to find the IP address of the load balancer, assuming the relevant service is configured in your namespace:

```
kubectl get svc -n <namespace> <service-name> --
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 `nginx-ingress` 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: GENERIC
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 Native 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 WebLogic Remote Console is deployed on the Admin Server and can be accessed at: `http://admin.<host>:<Port>/rconsole`.
For additional information, see [Oracle WebLogic Remote Console Online Help](#).
- 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.weblogic.oracle sr-quick -n sr
```

The command displays the domain configuration information.

To verify the client UIs:

- Log into the WebLogic Remote console using the URL specified in the output of the **create-instance** script: `http://admin.quick.sr.osm.org:30305/rconsole`
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/ordermanagement/
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. If the Traefik version you are using is different from the version you specified at the time of this release, modify the Traefik chart version in the **unregister-namespace.sh** script.

4. If you wish to unregister a namespace without deleting the namespace and secrets, do one of the following:

- If you have not added domainNamespaceLabelSelector, run the following command for the operator to stop monitoring your namespace:

```
$OSM_CNTK/scripts/unregister-namespace.sh -p project -t wlsko
# For example, $OSM_CNTK/scripts/unregister-namespace.sh -p sr -t wlsko
```

- If you have specified a value for domainNamespaceLabelSelector (for example, wko412=enabled) during the installation of the operator, run the following command for unregistering the namespace:

```
$OSM_CNTK/scripts/unregister-namespace.sh -p project -t wlsko
# For example, $OSM_CNTK/scripts/unregister-namespace.sh -p sr -t wlsko
-l wko412
```

5. Drop the PDB or delete the instance schemas from it. Deleting the schemas is recommended if you are using the PDB to re-create the instance.

Note

If PDB is dropped, all schemas hosted on it will be lost, not just the instance schemas.

```
$(Deletes OSM Schema)
$OSM_CNTK/scripts/install-osmdb.sh -p sr -i quick -s $SPEC_PATH -c 8
$(Deletes RCU Schema)
$OSM_CNTK/scripts/install-osmdb.sh -p sr -i quick -s $SPEC_PATH -c 6
```

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.weblogic.oracle -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 image 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 wireline (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



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, oidc, OPSS, Introspector and the WLS Admin credentials used when creating the domain.

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

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
- toadmin
```

After the configuration and the input are available, the remaining activities are focused on Continuous Delivery, which can be automated. If your project namespace does not have the label for the desired WebLogic Kubernetes Operator (WKO):

1. Remove any existing WKO labels from the namespace. Replace `<old-operator-label>` with any previous WKO label key:

```
kubectl label namespace $PROJECT <old-operator-label>-
```

Note

If you are not sure which labels are present, you can generate a list by using

```
kubectl get namespace $PROJECT --show-labels
```

2. Add the desired WKO label to the namespace:

```
kubectl label namespace $PROJECT $WKO_NS=enabled --overwrite
```

This ensures that only the intended operator will manage the namespace.

3. 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 6
```

Note

Refer to the help content to obtain the list of valid and correct command codes for the `install-osmdb.sh` script. Run

```
$OSM_CNTK/scripts/install-osmdb.sh -h
```

Running the script above will generate the following output:

This script is the entry point to perform all deployment-type activities on the OSM CN database. It manages the OSM schema as well as the FMW RCU schema.

Usage:

```
install-osmdb.sh <parameters>
  -p <projectName> : mandatory
  -i <instanceName> : mandatory
  -s <specPath>    : mandatory; locations of
specification files
                                specPath is a colon(:) delimited
list of directories
  -c <commandCode> : mandatory
  -y               : optional; assume "y" is entered
for any prompt
  -d               : optional; do --dry-run only
  -q               : optional; disable verbose progress
indicators
  -a               : optional; source partition name
where statistic data
                                to be copy from.
                                Only valid for command 11, if not
specified, use the
                                default partition statistic data
as copy source.
  -b               : optional; Comma delimited list of
partition name
                                where statistic data will be
copied to.
                                Only valid for command 11, if not
specified, copy to
                                all partitions.
```

or just `-h` for help

CommandCode allowed values:

```
1 - Create or upgrade OSM schema
2 - Create or upgrade OSM schema, then create RCU
schema
3 - Create or upgrade OSM schema, then drop RCU schema
4 - Create or upgrade OSM schema, then drop and re-
create RCU schema
5 - Drop and re-create RCU schema
6 - Drop RCU schema
7 - Create RCU schema
```

```
8 - Drop OSM schema
9 - Reserved
10 - Reserved
11 - Import partition-level DB optimizer statistics
12 - Migrate OSM Traditional schema
13 - Reserved
14 - Reserved
15 - Reserved
16 - Reserved
17 - Reserved
```

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.

4. Create the Ingress:

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

5. 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 -  
s spec_path\  
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 "[Manage LDAP Providers in WLS via OSM](#)" in this guide.

About Service Accounts

In the WebLogic Kubernetes Operator (WKO), the serviceAccount Helm chart property specifies the name of a Kubernetes ServiceAccount in the namespace where the operator is installed. The operator uses this service account to call the Kubernetes API server, and the operator's Helm chart creates the appropriate access controls for the service account.

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. For more information about how to use this service account and its associated privileges, refer to the WKO documentation here: <https://oracle.github.io/weblogic-kubernetes-operator/managing-operators/using-helm/#serviceaccount>

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 NGINX namespace (if NGINX 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 WebLogic Kubernetes Operator documentation at: <https://oracle.github.io/weblogic-kubernetes-operator/>.

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 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:

- Selecting a Deployment Topology
- Configuring OSM Runtime Parameters
- Preparing Cartridges
- Extending the WDT Model
- Working with Kubernetes Secrets
- Adding JMS Queues and Topics
- Generating Error Queues for Custom Queues and Topics
- Creating a JMS template
- Deploying Cartridges
- Provisioning Cartridge User Accounts

Selecting a Deployment Topology

OSM CNTK can be instructed to deploy only WebLogic-based components either with or without new microservices. This can be achieved with the help of the **omsConfig.osm_runtime_type** element in the project specification. The default value of this specification is MultiService.

project.yaml

```
# Provide values to override the defaults for oms-config.xml
omsConfig:
  # anything here overrides what is in shape spec in case of duplication
  # anything here can be overridden by instance spec
  project:
    osm_runtime_type: MultiService # MultiService(CN with full topology) or WLS (CN
    with only core fulfillment topology)
```

The possible values can be:

- **WLS:** OSM is running with only the WebLogic based components without new microservices.
 - Microservice dependent features such as TMF orders, System Interactions and Fallout Exception, are not available and cartridges that use such features cannot be deployed.
- **MultiService:** OSM is running with new microservices with the OSM cloud native 7.5 features enabled.
 - OSM cloud native 7.5 features which are enabled are TMF orders, System Interactions and Fallout Exception.

While deploying older cartridges, you must rebuild cartridges using Design Studio. For more information, about the Design Studio version you need to use, refer to *OSM Software Compatibility*. If the rebuild fails, you must not deploy the cartridges.

For more information about TMF and non-TMF cartridges, refer to "About TMF Cartridges and Non-TMF Cartridges".

Upgrade Scenarios

An OSM cloud native instance set up as WLS can be upgraded to MultiService. To do this, you must:

1. Scale down your cluster to 0.
2. Upgrade the database using the database installer command code 1.
3. Scale your cluster back to the original clusterSize.

This upgrade will impact microservices and features such as TMF orders, System Interactions and Fallout Exception.

If you create an OSM cloud native instance with **osm_runtime_type** as WLS and then later upgrade to MultiService, you need to recreate Ingress to access the newly created microservices.

Note

An OSM Instance set up as MultiService must not be downgraded to WLS even though such a change is not blocked by the CNTK.

Pre-requisite Configuration For WLS

When installing an OSM cloud native instance with the WLS option, you need to configure empty values for the following properties in the project specification file:

```
osm-gateway.image: ""
osmRuntimeUXServer.image: ""
ingress.osmgw.annotations: {}
ingress.rtux.annotations: {}
```

Installing OSM Cartridge Assembler (OCA) for Integration with Solution Designer

The OSM Cartridge Assembler (OCA) microservice unifies the TMF cartridges created in Design Studio with content modeled in Solution Designer. This process enables dynamic

assembly and streamlined deployment of cartridges to OSM to support more flexible, collaborative, and efficient cartridge design processes.

To learn about the dynamic cartridge assembly and the critical role that OCA plays, see *About Dynamic Cartridge Assembly (Cloud Native Only)* in the *OSM Concepts*.

You need to install the OCA Microservice when OSM is integrating with Solution Designer. OCA is disabled by default and you can enable it with the configuration in the following topics.

Preparing for OSM Cartridge Assembler

To install the OCA microservice, you need to do the following tasks:

1. [Create Secrets](#)
2. [Build the OCA Image](#)
3. [Build the Capabilities Cartridge Image](#)
4. [Configure the Project Specification](#)
5. [Configure the Instance Specification](#)

Creating Secrets

Creating secrets is part of creating a basic OSM cloud native instance. When you are using the OCA microservice you need to ensure that the secret was created properly. For more information on creating the required secrets for OCA, see [About Optional Secrets](#).

Building the OCA Image

Similar to creating secrets, building images is a part of creating an OSM cloud native instance. For more information on building images for OCA, see *Creating the OSM and OSM DB Installer Images* in [Creating OSM Cloud Native Images](#).

Building the Capabilities Cartridge Image

You need to build the Capabilities Cartridge Image that will be used with the OCA Microservice. To do this, you need to first generate the capabilities cartridge. For more information on creating the capabilities cartridge, see *OSM Modeling Guide* and *Design Studio Modeling OSM Orchestration*.

Once you have generated the **.cpar** file in Design Studio, you need to use it to build the image that will be used with OCA. See *Building Cartridge Images* in [Container Images](#) for more information about building cartridge images for OCA.

Configuring the Project Specification

After building the capabilities cartridge image, you need to configure the project specification.

```
osm-cartridge-assembler:
  image: "osm-cartridge-assembler"
  enabled: false ## OCA is disabled by default. Enable it if required.
```

<ingress annotations. Uncomment the section for OCA>

ingress:

```

oca:
  annotations: {} # This empty declaration should be removed if adding
items here.
  # This annotation is required for nginx ingress controller for oca.
  #annotations:
  # # Enables regular expression (regex) support in the path field
  # nginx.ingress.kubernetes.io/use-regex: "true"
  # # To rewrite the URL path before forwarding the request to the backend
service.
  # nginx.ingress.kubernetes.io/rewrite-target: /$1
  # # To configure timeout in NGINX Ingress Controller for reading data
from the the backend service.
  # nginx.ingress.kubernetes.io/proxy-read-timeout: "240"
  # # To configure timeout for sending data from the NGINX Ingress
Controller to the backend service.
  # nginx.ingress.kubernetes.io/proxy-send-timeout: "240"

```

<specify the capabilities cartridge versions that should be loaded into OCA>

```

capabilitiesCartridgeList: []
#capabilitiesCartridgeList:
# - name: fulfillment-capabilities-1-0-2
#   version: "1.0.2"
#   image: "/mypath/fulfillment-capabilities-1-0-2-cpar:latest"
# - name: fulfillment-capabilities-1-0-3
#   version: "1.0.3"
#   image: "/mypath/fulfillment-capabilities-1-0-3-cpar:latest"
#   imagePullPolicy: Always
#   imagePullSecret: capabilitiesCartridge-imagepull

```

where,

- `osm-cartridge-assembler` is the OCA image and a boolean value to enable OCA installation
- `annotations` are the annotations required for the NGINX ingress controller for OCA


```

# Enables regular expression (regex) support in the path field
nginx.ingress.kubernetes.io/use-regex: "true"
# To rewrite the URL path before forwarding the request to the
backend service.
nginx.ingress.kubernetes.io/rewrite-target: /$1
# To configure timeout in NGINX Ingress Controller for reading data
from the the backend service.
nginx.ingress.kubernetes.io/proxy-read-timeout: "240"
# To configure timeout for sending data from the NGINX Ingress
Controller to the backend service.
nginx.ingress.kubernetes.io/proxy-send-timeout: "240"

```
- `capabilitiesCartridgeList` are the image details for the list of capabilities cartridges

Configuring the Instance Specification

After configuring the project specification, you need to complete the following configuration related to the SCD integration in the instance specification file.

```
# OSM Cartridge Assembler
osm-cartridge-assembler: {} # This empty declaration should be removed if
adding items here.
#osm-cartridge-assembler:
# # SCD Base URL
# scdBaseURL: "http://localhost:8080/dummy"
# # Prometheus endpoint is enabled by default.
# prometheus:
#   enabled: true
# # Flag to specify the log level
# # The valid log levels in descending order are:
# #   SEVERE (highest value)
# #   WARNING
# #   INFO
# #   FINE
# #   FINEST (lowest value)
# #
# The default value is "INFO"
# log:
#   level: INFO
# # If disabled, no log files will be created. All logs will go to stdout/
stderr
# fileLogging:
#   enabled: false
# # The storage volume must specify the PVC to be used for persistent
storage.
# # When enabled with PVC value, log data will be directed here
# # With fileLogging.enabled:false, there will be no log file in PVC location
# storageVolume:
#   enabled: false
#   pvc: storage-pvc # dummy value, to be replaced when using pvc
```

where,

- `fileLogging` if enabled creates log files which are used by the FluentD daemonset to collect container logs and import into the log stack. Logs are routed to **stdout/stderr** even when `fileLogging` is enabled.

Note

This parameter is disabled by default.

- `storageVolume` is used to store log data. You can enable storage volume with valid PVC. The PVC which is used to persist OSM cloud native logs when enabled, can also be used here. The logs will be available at `<nfs_path>/<project>-<instance>/osm-cartridge-assembler`.

Note

This parameter is disabled by default.

- prometheus is used for exposing metrics for scraping. You can disable this parameter if you do not want to expose metrics.

Note

This parameter is enabled by default.

- log is the log level. This can be set to different levels of severity in OCA. These levels are: SEVERE, WARNING, INFO, FINE, FINEST
- scdBaseUrl is a URL required to enable a connection with a single Solution Designer instance.

Managing an OSM Cartridge Assembler Deployment via Cloud Native Toolkit

There are no special scripts used to manage the OCA microservice. All operations are available using the standard set of CNTK scripts

Create Ingress

Creating an ingress is part of creating a basic OSM cloud native instance. When the OCA microservice is being used, you need to create a specific ingress for OCA. For more information about creating an ingress for OCA, refer to [Configuring the Project Specification](#).

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

NAME	CLASS
HOSTS	
ADDRESS PORTS AGE	
<project>-<instance>-ingress	<none>
<instance>.<project>.osm.org,t3.<instance>.<project>.osm.org,admin.<instance>.<project>.osm.org	
80 10s	
<project>-<instance>-oca-ingress	<none>
<instance>.<project>.osm.org	
80 10s	
<project>-<instance>-osmgw-ingress	<none>
<instance>.<project>.osm.org,<instance>.<project>.osm.org	
80 10s	
<project>-<instance>-rtux-ingress	<none>
<instance>.<project>.osm.org	

Create OCA Microservice

Run the following script to create the OSM cloud native instance including OCA.

```
$OSM_CNTK/scripts/create-instance.sh -p <project> -i <instance> -s <spec files path>
```

Upgrade OCA Microservice

Run the following script to upgrade the OSM cloud native instance with a new OCA configuration.

```
$OSM_CNTK/scripts/upgrade-instance.sh -p <project> -i <instance> -s <spec  
files path>
```

Restart OCA Microservice

To restart the OCA pod, you need to run the following script. Provide ocams as the input to flag `-r` to restart only the OCA pod. If you want to restart the OCA pod along with the other OSM pods, provide `all` as the input.

```
$OSM_CNTK/scripts/restart-instance.sh -p <project> -i <instance> -s <spec  
files path> -r ocams
```

Delete OCA Microservice

You can delete the OCA microservice in two ways. The first approach is to disable the microservice in the project specification file and run the `upgrade-instance.sh` script.

```
osm-cartridge-assembler:  
  enabled: false  
  image: "osm-cartridge-assembler"
```

The second approach is to delete the entire OSM cloud native instance. This deletes the OCA pod, as well as all other OSM pods.

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

Lint an OCA Chart

To lint OCA charts, use the following command.

```
$OSM_CNTK/scripts/lint-osm-instance-chart.sh -p <project> -i <instance> -s  
<spec files path>
```

OCA Dry Run

To render OCA templates, run the following command to debug the helm chart changes.

```
$OSM_CNTK/scripts/create-instance-dry-run.sh -p <project> -i <instance> -s  
<spec files path>
```

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.Collect
ionCycle.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.

Configuring Schema Validation

OSM Gateway handles incoming and outgoing JSON payload transformation. When a JSON payload contains data that is not registered with OSM, OSM Gateway can either fail validation or silently prune the extra data from the transformation, depending on the settings in the project specification.

Schema dictating the incoming payloads for TMF REST endpoints, are registered via the Hosted Order Specification in TMF cartridges (622 or 641).

This validation is applicable to incoming order payloads only when hosting a TMF cartridge (622 and 641). This validation is also applicable to incoming and outgoing payloads for Freeform or TMF cartridges that use a System Interaction to communicate with external systems.

If incoming or outgoing JSON payloads contain extensions that are
not registered with OSM, use these parameters to specify whether the

```
# payload should be failed (STRICT) or the unrecognized extensions
# simply pruned (PRUNE). Default is STRICT.
#validation:
#  unregisteredSchema:
#    incoming: STRICT
#    outgoing: STRICT
```

Configuring Target Systems for Events and System Interactions

TMF cartridges define an Event Target System name, to identify the recipient of event notifications about the TMF resource.

Freeform cartridges and TMF cartridges that use System Interaction specifications would define a target system name to identify the external system involved in a REST interaction.

In both cases, a logical target system name is provided inside the cartridge. The configuration necessary for OSM to resolve these names at runtime is provided in the CNTK specification files.

Configuring the Project Specification

Whether it is an event target system or a system interaction target system, the cartridge configuration always reflects a logical system name and it is not tied to a specific server instance. Each one must be defined in the project specification.

```
# Define targetSystem info, provide name of the targetSystem like
reverseProxy.
requiredTargetSystems: [] # This empty declaration should be removed if
adding items here.
#requiredTargetSystems:
# - name: BillingSystem
#   description: "Oracle BRM for TMF622 COM cartridge"
# - name: ShippingSystem
#   description: "Unified shipping portal"
```

Configuring the Instance Specification

The logical system name is decoupled from the actual connection details so that cartridge deployment is not impacted by a specific environment. Each logical system name will be resolved against a set of connection details and applicable security scheme, at runtime. To enable this resolution, the connection information must be provided in the CNTK instance specification.

If we take an example where a cartridge and its project specification have a reference to a "BillingSystem" target system. By providing the actual connection details in the instance specification, the same cartridge can be deployed without any change into multiple environments - development instance 1 and 2, QA instance 1, and test instance xyz.

```
# Define targetSystem info, provide server details and security info
targetSystems: {} # This empty declaration should be removed if adding items
here.
#targetSystems:
#systems:
#  #targetSystem_Name:
#    #url: target_system_url
#    #protocol: protocol
```

```

#description: description
#securityScheme: securitySchemeName
# To override default fault tolerance parameters, uncomment this
section and provide values
#fault-tolerance:
#retry:
# The maximum number of retry attempts by pod when emitting a
message failed
# When value is absent, pod will retry 2147483647
(Integer.MAX_VALUE) times.
#maxRetries: 2147483647
# The delay between the retry attempts
#delay: 5000
# The delay unit eg MILLIS
#delayUnit: MILLIS
# Instructs pod on which error code has to retry
#onErrorCodes:
# - 500
# - 503
# - 409
# - 429
#concurrency:
# The maximum number of concurrent connections that can be made to
this target system from the OSM Gateway
# across the OSM cluster (default : 50)
#maxValue: 50

# Define security scheme for target systems enabled with security
# For each security scheme defined kubernetes secret should be created using
# ${CNTK_HOME}/scripts/manage-target-system-credentials.sh script.
#securitySchemes:
#- name: <SecuritySchemeName-1>
# type: "userPassword"
# authorizationUrl: <authorization URL>
# sessionId:
#   type: <"cookie" or "header">
#   name: <Cookie-name or Header-name that carries sessionId>
#- name: <SecuritySchemeName-2>
# type: "OAuth2"
# authorizationUrl: <authorization URL>
# tokenUrl: <token URL>
# scopes:
# - name: scope1
#   description: "Scope 1 Description" #optional
# - name: scope2
#   description: "Scope 2 Description" #optional

```

Configuring Security Schemes for Target Systems

System interactions support OIDC and basic authentication security schemes for target systems that are enabled with security. You can define these security schemes in the Instance Specification under the element `targetSystems` as `securitySchemes`.

When using either of the security schemes, you need to create a Kubernetes secret using the following command:

```
${CNTK_HOME}/scripts/manage-target-system-credentials.sh -p project -i instance -n  
securitySchemeName -t OAuth2/userPassword create
```

Note

In the script above, if you use the userPassword security scheme, you will need to provide a username and password. If you use the OAUTH2 security scheme, then you will need to provide a client and secret.

The following sections provide more information about the security schemes.

Using the OIDC Security Scheme

If the target system is enabled with OAUTH2 security, then you need to configure the OAUTH2 security scheme. The following instance specification sample shows the security scheme configuration for OAUTH2 when the target system is enabled with OAUTH2 security:

```
targetSystems:  
  securitySchemes:  
    - name: SecuritySchemeName  
      type: "OAuth2"  
      tokenUrl: token URL  
      scopes:  
        - name: scope_1  
          description: "Write Scope"  
        - name: scope_2  
          description: "Read Scope"
```

In the security scheme configuration above:

- *token URL* is the access token URL which is used to obtain the access token from the Identity provider such as KeyCloak.
- Scopes provide limitations to the access granted to an access token.
- *SecuritySchemeName* is used to fetch the client id and the client secret from the Kubernetes secret.

Using the Basic Authentication Security Scheme

If the target system is enabled with basic authentication security either utilizing user credentials or JSESSIONID, then you need to configure the userPassword security scheme.

You can configure this security scheme using either of the following three methods:

- Basic auth with user credentials
- JSESSIONID header
- JSESSIONID cookie

Note

The default configuration for the userPassword security scheme is Basic auth with user credentials.

Basic Auth with User Credentials

The following instance specification sample shows the security scheme configuration for `userPassword` when the target system is enabled with basic auth using user credentials security.

```
securitySchemes:  
  - name: SecuritySchemeName  
    type: "userPassword"
```

In the security scheme configuration above, each REST call to the target system includes an authentication header which has a base-64 encoded value of `username:password`. Therefore the authentication header would look like:

Authorization: Basic base-64 encoded value of username:password

JSESSIONID in Authentication Header

In this security scheme configuration, instead of sending user credentials, the OSM gateway sends the user session ID in an authentication header for each request. The OSM gateway exchanges the user credentials only once to obtain the user session ID from the response header.

The following instance specification sample shows the security scheme configuration for `userPassword` when the target system is enabled with authentication using the user session ID in an authentication header.

```
securitySchemes:  
  - name: SecuritySchemeName  
    type: "userPassword"  
    authorizationUrl: authorization URL  
    sessionId:  
      type: "header"  
      name: <Header Name>
```

In the above security scheme configuration:

- The OSM gateway exchanges the user credentials (base-64 encoded value of `username:password`) in an authentication header using the target system's `authorizationUrl` to obtain the user session ID.
- The OSM gateway extracts the user session ID from the response headers using the header name defined in the security scheme.
- The OSM gateway sends the user session ID in an authentication header in each request sent to the target system.
- Therefore, the authentication header would look like:

Authorization: user session ID

JSESSIONID in Cookies

In this security scheme configuration, instead of sending user credentials, OSM gateway sends the user session ID in cookies for each request. OSM gateway exchanges the user credentials only once to obtain the user session ID from the response cookies.

The following instance specification sample shows the security scheme configuration for `userPassword` when the target system is enabled with authentication using user session ID in cookies.

```
securitySchemes:
  - name: SecuritySchemeName
    type: "userPassword"
    authorizationUrl: authorization URL
    sessionId:
      type: "cookie"
      name: Cookie Name
```

In the security scheme configuration above:

- The OSM gateway exchanges the user credentials (base-64 encoded value of `username:password`) in an authentication header using the target system's `authorizationURL` to obtain the user session ID.
- The OSM gateway extracts the user session ID from the response cookies using the `Cookie Name` defined in the security scheme.
- The OSM gateway sends the user session ID in cookies in each request to the target system.
- Therefore, the http header would look like:

```
cookie: user session ID
```

Configuring OSM Gateway Readiness

You can control different aspects of OSM Gateway behavior by editing the instance specification. Additionally, tuning parameters for OSM Gateway are available in the shape specifications.

OSM Gateway establishes a connection with the OSM managed server and waits for it to transition into the "ready" state. However, if the managed server fails to become ready within the specified timeout duration, the gateway declares it as "not ready" and proceeds to initiate Kubernetes cleanup and retry procedures. This parameter is enabled by default.

```
# OSM Gateway
osm-gateway: {} # This empty declaration should be removed if adding items
here.
#osm-gateway:
  # When enabled, in order to start, OSM Gateway app waits until the timeout
  is elapsed for OSM-CN to be up.
  #waitForOSMReadinessBeforeStart:
    #enabled : true
    # Based on the ISO-8601 duration format PnDTnHnMn.nS.
    # For additional information, refer https://docs.oracle.com/en/java/
    javase/21/docs/api/java.base/java/time/Duration.html
    #timeout: "PT300S"

.....
```

Configuring the Order Operations User Interface

Use the instance specification to configure how and when various details are displayed in the Order Operations user interface.

The following block shows the instance specification where you configure settings for the data displayed in the Order Operations user interface:

```
osmRuntimeUX: {} # This empty declaration should be removed if adding items
here.
#osmRuntimeUX:
  #alertThresholds:
    # The percentage of orders that must succeed in a given time interval
    otherwise an alert is displayed to the user.
    # Order Success Rate = OrdersCompletedCount / (OrdersCompletedCount +
    OrdersFailedCount) * 100
    #minOrderSuccess: "98"
    # The maximum number of orders that can be submitted in an hour before an
    alert is displayed to the user.
    #maxHourlySubmittedOrder: "5000"
  #configuration:
    # An ISO 8601 duration period string that represents the oldest
    fromDatetime supported by the operations backend and UX.
    # The default value "P6M" indicates the oldest fromDate allowed by the
    API and the UX is 6 months ago.
    #minFromDatePeriod: "P6M"
    # The maximum percentage of orders that can fail otherwise above KPI
    badge is displayed to the user on Order Operations KPI dashboard.
    # Order Failure Rate = (OrdersFailedCount + OrdersRejectedCount) /
    TotalOrdersCount * 100
    #maxOrderFailure: "2"
    # The number seconds between automatic dashboard refresh in the
    Operations UX
    #refreshTimeInterval: 180
    # The number seconds for which landing page or iframe applications
    timeout when left idle
    # This should always be less than default OSM core session timeout value
    of 1500s.
    #sessionTimeOut: 1400
    # The number of retries UX applications will perform on errors reaching
    server APIs
    #onErrorMaxRetryCount: 3
    # The number of seconds UX applications will wait between such retries
    #onErrorRetryIntervalSecs : 30
```

Configuring the Alerts Displayed in the Order Operations Dashboard

You can control different aspects of the Order Operations user interface behavior by editing the instance specification. Additionally, tuning parameters for the Order Operations user interface are available in the shape specifications.

The Operations Dashboard displays a bell icon, which upon clicking opens the Alerts panel. The Alerts panel displays alerts when a threshold set on the orders is reached.

An alert is triggered and displayed when the percentage of orders drops below the number of orders that must succeed in a given time interval.

$\text{Order Success Rate} = \text{OrdersCompletedCount} / (\text{OrdersCompletedCount} + \text{OrdersFailedCount}) * 100$

To configure the alerts displayed in the Operations Dashboard, you configure the following parameters in the instance specification:

- `alertThresholds.minOrderSuccess`: The order completion success threshold that must be met by every order group. Otherwise, an alert is triggered and displayed. The default value is 98.
- `alertThresholds.maxHourlySubmittedOrder`: The number of orders that can be submitted in an hour before an alert is shown. The default value is 5000.

The following block shows the instance specification where you configure parameters for alerts:

```
# osmRuntimeUX
osmRuntimeUX: {} # This empty declaration should be removed if adding items
here.
#osmRuntimeUX:
  #alertThresholds:
    # The percentage of orders that must succeed in a given time interval
    otherwise an alert is displayed to the user.
    # Order Success Rate = OrdersCompletedCount / (OrdersCompletedCount +
    OrdersFailedCount) * 100
    #minOrderSuccess: "98"
    # The maximum number of orders that can be submitted in an hour before an
    alert is displayed to the user.
    #maxHourlySubmittedOrder: "5000"

.....
```

Configuring Session Timeout

When logged into the application, if a user leaves the user interface idle in a tab, a pop-up window appears after 80% of the configured session timeout duration is reached. The pop-up window indicates how much time the user has before the session times out. If the user clicks **OK** or refreshes the page, the timer is reset.

However, if the user does not take any action, they will be logged out after the remaining time is run out. If the user opens the application in two separate browser windows or tabs, and one of them is kept idle, a pop-up window appears on the one that is left idle after 80% of the configured session timeout duration is reached. If the user does not click the **OK** button, or refreshes the page on the idle browser window, when the time runs out, the user will be logged out of the application on both the windows.

To configure session timeout, set the `configuration.sessionTimeout` parameter in the instance specification. This parameter defines the number of seconds after which the landing page or an application will timeout when left idle. By default, this is set to 1400 seconds.

The following block shows the instance specification where you configure session timeout:

```
# osmRuntimeUX
osmRuntimeUX: {} # This empty declaration should be removed if adding items
here.
#osmRuntimeUX:
```

.....

```
# The number seconds for which landing page or iframe applications
timeout when left idle
# This should always be less than default OSM core session timeout value
of 1500s.
#sessionTimeOut: 1400
```

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 Remote 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.

Warning

Cartridges with Java plugins or emulators relying on JDK 8 (or earlier) or Java EE 7 (or earlier) need to be upgraded to use Java 21 and Java EE 8 libraries to ensure compatibility with FMW 14.1.2 and Java 21. To deploy such cartridges on OSM 8.0, you need to rebuild the cartridges using the OSM SDK provided with OSM 8.0 and set Design Studio's target OSM version to 8.0.

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 n f s 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

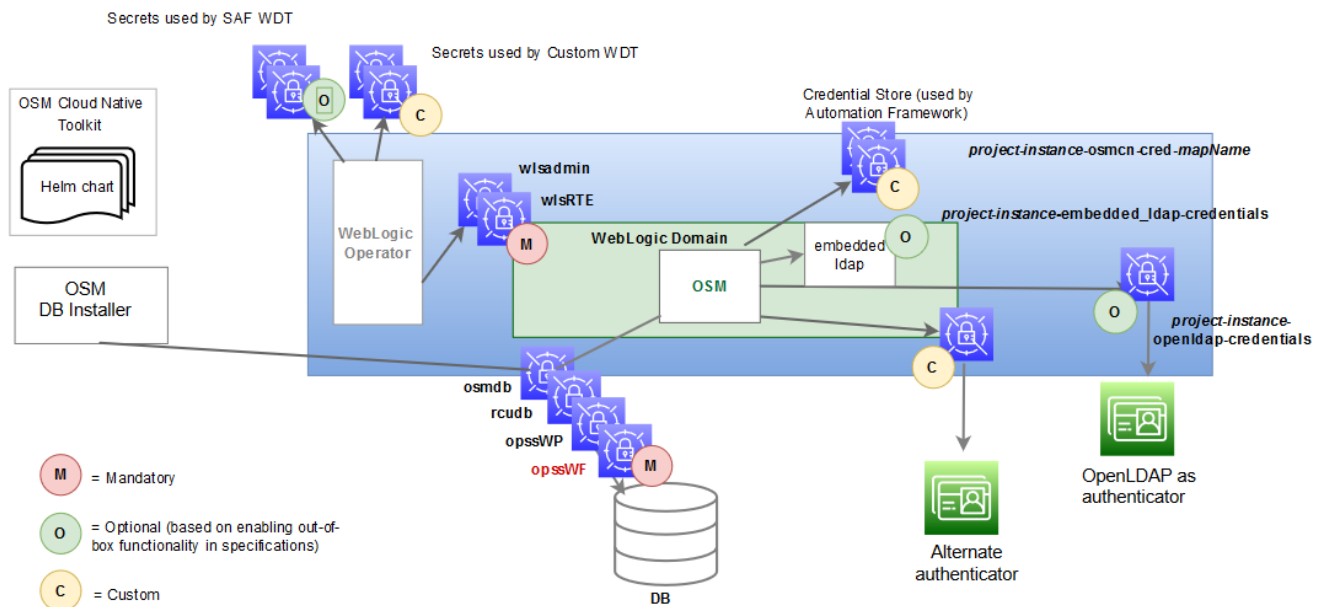
Cartridges may also define users who need access to OSM APIs. These user credentials need to be available in the right locations as described in "[Provisioning Cartridge User Accounts](#)". These credentials must then be made available through the configuration to OSM.

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

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. For more information on secrets created by the script, refer to the script help (-h flag) -h, and to [Reference of Secrets Created by the Scripts](#).

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 any external LDAP for authentication configured via project specification, OSM cloud native relies on the following secret to have been created:

project-instance-ldap-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
```

- If you have installed OSM Cartridge Assembler (OCA), then the following secret needs to be created:

project-instance-oca-credentials

The toolkit provides a sample script to create this secret for you

```
$ {OSM_CNTK}/scripts/manage-instance-credentials.sh -p project -i instance
-s spec-path create ocaOidc
```

The script prompts you for the following details for the OCA microservice:

Note

The OIDC configuration should point to a keycloak instance in an LDAP machine.

- OIDC Client ID
- OIDC Client Secret
- OIDC Access Base URL
- OIDC AccessToken URL

This secret is required to access the OCA endpoint.

Additionally, the script prompts you for information regarding the optional OIDC configuration needed for connectivity with SCD - Solution Designer. The OIDC details (clientId, clientSecret, authTokenUrl, scope) for this can be obtained from the team that installed Solution Designer for you.

This secret is required to enable a Solution Designer-OCA integration.

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.

OSM CNTK provides a utility script which generates the list of secrets required for an OSM cloud native instance based on the provided specification files.

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

This will provide the list of secrets categorized into OSM core and custom secrets as shown below:

secrets:

coreSecrets:

- <project>-<instance>-database-credentials
- <project>-<instance>-rcudb-credentials
- <project>-<instance>-embedded-ldap-credentials
- <project>-<instance>-weblogic-credentials
- <project>-<instance>-opss-walletfile-secret
- <project>-<instance>-opss-wallet-password-secret
- <project>-<instance>-runtime-encryption-secret
- <project>-<instance>-gateway-credentials
- <project>-<instance>-oca-credentials
- <project>-<instance>-fluentd-credentials

customSecrets:

- partitionStatisticSecret
- projectCustomSecret
- instanceCustomSecret

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
```

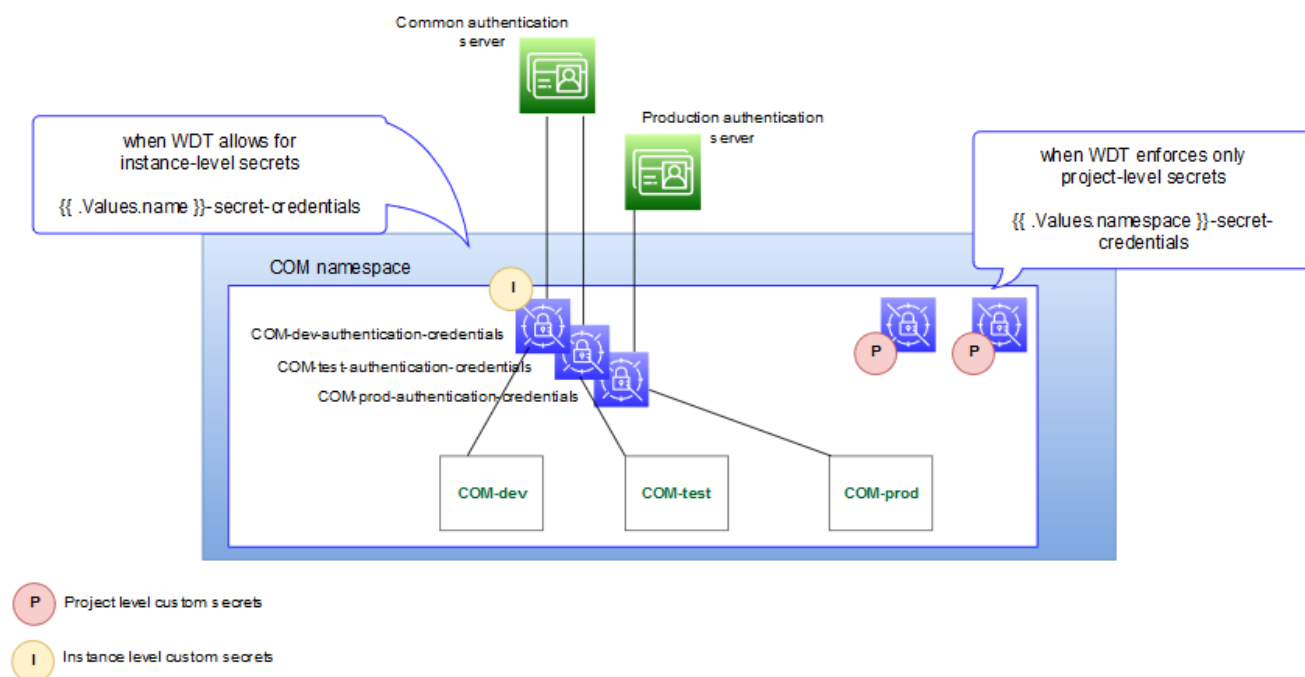
```
kubectl create secret generic COM-test-authentication-credentials \
-n COM \
--from-literal=principal=<value1> \
--from-literal=credential=<value2> \
--from-literal=host=<value3> \
--from-literal=port=<value4>

##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



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 LDAP, 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](#)".

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.

This section covers the following topics:

- Creating Credential Store Secret
- Declaring the Secret

You manage the following types of users, based on the host system:

- **Users "hosted" by this OSM instance:** These are non-human user accounts that systems use to login from outside OSM (via a User Interface, OSM XML API or OSM Web Service API) or are logged in internally (as a Run As user for an automation plugin).

These users:

- Require mappings to OSM groups.
- Use the osm mapname, with _sysgen_ keyname.
- Must be added to the project specification "cartridgeUsers" list.

- **Users "hosted" by external systems:** The external systems could be UIM, ASAP or another instance of OSM. These users are used by cartridge automation while interacting with external systems to authenticate themselves to the external system.

These users:

- Require Kubernetes secret entries. OSM group mapping is not required.
- Use the mapname and keyname the cartridge developer has decided upon in the code.
- Must be added to the project specification "externalCredStore.secrets.mapNames" list.

It is possible for the same credentials to be required in two ways - as a non-human cartridge user and as an access credential for an "external" system. An example would be an instance that hosts both SOM and TOM cartridges. The TOM cartridge may require a non-human user called "tom", while the SOM cartridge needs an access credential to send the TOM order. For flexibility, the instances that send the TOM order would not assume co-location and would fetch access credentials for the user "tom". When the same credentials need to exist for both categories, that user ID must be duplicated - once for each category, as per the syntax of that category. Ensure that the passwords are in sync, as OSM cloud native views these as independent entries.

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 serves as a template for user credentials that need to be created for both human and automation users.

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, as this value is used as the prefix of the secret name to be created.

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 set to:

- **secret:** Creates the human user or automation user against the Kubernetes Secret.

The **osm-groups** parameter represents a list of OSM groups to associate the user to the embedded 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 about OSM user group mapping.

The following text shows a sample user information text file:

```
osm:_sysgen_:osmlf:secret:OMS_xml_api,OSM_automation,OMS_ws_api
uim:uim:uim:secret
tom:osm:tomadmin:secret
```

In the above example:

- The first line creates a Kubernetes secret entry for "osmlf" user in the "osm" credential secret. The entry contains a username, password, and the group associations. Use the

```
context:getOsmCredentialPassword
```

API to retrieve the password. Add "osmlf" to the project specification's "cartridgeUsers" list.

- The second line creates a Kubernetes secret entry for the user "uim" with the access key name "uim". The entry contains a username and a password. Use the `context:getCredential` API or the `context:getCredentialAsXML` API to retrieve both username and password from map "uim" with key "uim". Add "uim" to the project specification's "externalCredStore.secrets.mapNames" list.
- The third line creates a Kubernetes secret entry for the user "tomadmin" with the access key name "osm". The entry contains a username and a password. Use the `context:getCredential` API or the `context:getCredentialAsXML` API to retrieve both username and password from map "tom" with key "osm". Add "tom" to the project specification's "externalCredStore.secrets.mapNames" list.

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.

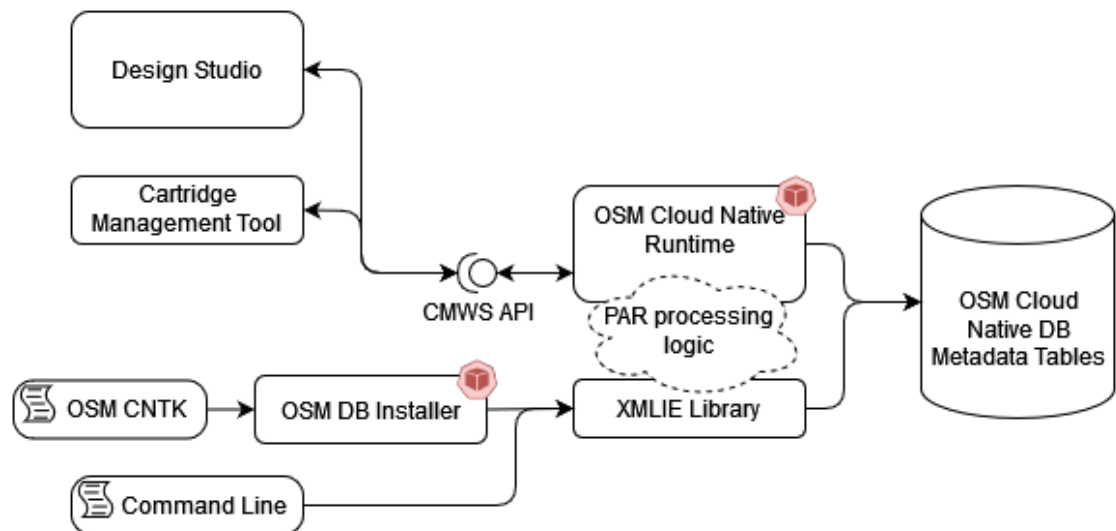
Working with Cartridges

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. There are multiple ways to deploy cartridges, but they all result in cartridge information extracted from the par files and stored in various OSM metadata tables in the OSM DB.

The following diagram illustrates cartridge deployment paths.

Figure 6-3 Cartridge Deployment Paths



Cartridge Deployment Tool in OSM Cloud Native

To deploy cartridge par files, OSM cloud native employs a mechanism using the OSM cloud native toolkit's **manage-cartridges.sh** script.

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 file 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.

Single or One-off Cartridge Deployment

To deploy a single cartridge par file, use the **parDeploy** command to the **manage-cartridges.sh** script along with the **-f *parPath*** parameter. 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 in the project namespace.

Specification-driven Cartridge Deployment

For more control and traceability in the OSM cartridge loadout, use the **sync** command to the **manage-cartridges.sh** script. You must first describe the desired list of cartridges in your project specification. The **sync** command performs the required deploy, redeploy and fast-undeploy changes to modify the in-database set of cartridges to match the list given in the specification. This command also sets the default cartridge as per the specification.

The list in the project specification must depict the desired or target state.

Note

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

- If a cartridge is listed as **deployed in the specification**, but is not deployed in the database: it is deployed.
- If a cartridge is listed as **deployed** in the specification and the same version exists in the database, the two cartridge's contents are compared; if there is a difference, the new par file is redeployed.
- If a cartridge is listed in the specification with a **default** setting that does not match with what is in the database, the **default** setting in the database is updated to match the specification; no change is done to this setting if they already match.
- If a cartridge is listed as **fastundeployed** in the specification 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 encoded in the cartridge 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**.

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. Alternatively, it can specify a container image that carries the cartridge par. The URL would be pointing to a remote file repository that may require authentication or other parameters. The cartridge entry's param fields 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.

Refer to the "[Cartridge par Sources](#)" section for details on the different options possible.

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* - Provide the URL or the image details.

- secret: *Kubernetes secret in the project namespace* - Optional. Required only if remote URL server requires authentication.

- image: *image built with par file*- Provide either the image or the URL, but not both.

- imagePullSecret: *The secret required to pull the image built for cartridge deployment via image.*

- params: *Commandline parameters will be passed to curl* - Optional. User can provide additional parameters such as 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. Specify if this cartridge is the default cartridge.

- deploymentState: deployed|fastundeployed - Mandatory. Indicate the desired target state of the cartridge.

Use the **manage-cartridges.sh** script from the CNTK with the command option **-c** of **parDeploy**

Offline Cartridge Deployment Using the OSM Cloud Native Toolkit

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.

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 **manage-cartridges.sh** script with the desired command – **parDeploy** or **sync**.

Note

cartridgeManagement.resources (from the shape specification) is used by the pod while deploying the cartridges.

Additionally, when you are using the **sync** command and the cartridges are listed in the project specification using container images, **cartridgeInitContainer.resources** is used to pull the cartridge images.

Edit the instance spec to restore the original number for the cluster size and run the same **upgrade-instance.sh** command as before to bring up all the managed servers.

Online Cartridge Deployment Using the OSM Cloud Native Toolkit

This deployment mode only 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 with zero down time. During the deployment process, OSM CN remains reachable by external HTTP clients and JMS/SAF endpoints – this means OSM will continue to accept orders and will continue to work on existing orders.

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. This restart loads the new cartridge+version into memory and when all managed servers have the changes loaded, OSM switches to using the new cartridge+version in sync across all managed servers.

Note

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

Run the **manage-cartridges.sh** command with the additional **-o** command-line option to deploy cartridges in online mode, with either **parDeploy** or **sync**

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.

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 Native 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`.

Listing Deployed Cartridges Using the OSM Cloud Native Toolkit

This command provides a report on all the cartridges present in the OSM cloud native instance. The report contains the cartridge name, cartridge version, cartridge ID, whether it is the default version or not, number of orders still open for it and number of orders completed by it.

To invoke it, run the following command:

```
$OSM_CNTK/scripts/manage-cartridges.sh -p project -i instance -s spec_Path -c  
list
```

The output includes fast-undeployed cartridges.

Cartridge par Sources

This topic outlines the sources from where you can deploy the Cartridge par files. For OSM, Cartridge par files can be deployed from the following sources:

- Local Files
- Remote File Repository
- Container Images

Local Files

Use this method for only development environments. You can use the **parDeploy** command with the **manage-cartridges.sh script** to deploy a cartridge par file from your system.

Remote File Repository

Using a remote file repository is more preferable than local files. However, choose between this option and the container images as per your convenience. Use this method with the **sync** command with the **manage-cartridges.sh script**.

There are two approaches to using the Remote File Repository – Secured and Unsecured. The Unsecured approach could be suitable for test environments. You can also disable host verification. However, it is recommended that you do not opt for this as it can be a security risk.

Details of the cartridge, its version and the remote repository must be specified in the project specification as entries in the “cartridges” list. See the comments against this element in the sample specification file for full details:

```
cartridges:
- name: cartridge-name
  version: cartridge-version
  default: true-or-false
  deploymentState: deployed-or-fastundeployed
  url: url-to-cartridge-par-file-in-remote-repository
  secret: name-of-kubernetes-secret-holding-login-credentials-if-secured-remote-repository
  params: any-parameters-that-must-be-passed-to-curl-to-connect-to-above-url
```

secret is only required if the remote repository is secured. The Kubernetes secret must be in the same project namespace and must contain two fields, **username** and **password**, with the appropriate values.

params is optional and depends solely on the networking and nature of the remote repository.

Container Images

Having a repository for cartridges is not always feasible or sustainable. However, there is one kind of repository that is already a mandatory requirement for OSM cloud native (and in general, for any work on Kubernetes). It is the container image repository. Allowing OSM to pull cartridges as images from such an image repository allows for reuse of existing infrastructure and security.

Use this method with the **sync** command with the **manage-cartridges.sh** script. Details of the cartridge, its version and the image location must be specified in the project specification as entries in the **cartridges** list (see the comments against this element in the sample specification file for full details):

```
cartridges:
- name: cartridge-name
  version: cartridge-version
  default: true-or-false
  deploymentState: deployed-or-fastundeployed
  image: image-name-and-tag
  imagepullsecret: credentials-for-image-repository
```

imagepullsecret is only required if the image repository demands authenticated access. It is the standard Kubernetes image pull secret.

Building Cartridge Images

OSM Cloud Native Image Builder provides a sample utility to create a container image for a cartridge par file or capability cartridge cpar file for use in deployment activities. See **osm-image-builder/samples/cartridgeAsImage**.

```
#Script that builds container image using the par/cpar file that is provided
in -f parameter
./buildCartridgeImage.sh -n <Cartridge_Name> -v <Cartridge_Version> -f /
path/to/par/file
or
./buildcartridgeImage.sh -f <cpar_file_path>
```

Example:

```
./buildCartridgeImage.sh -n SimpleRabbits -v 1.7.0.1.0 -f /home/user/  
Cartridge/SimpleRabbits.par
```

Cartridge_Name and Cartridge_Version are extracted from the .par/.cpar archive file, which are specified in Design Studio during the build process.

This creates an image with tag name as <Cartridge_Name>:<Cartridge_Version> in lowercase for the .par archive file and <Cartridge_Name>-cpar:<Cartridge_Version> in lowercase for the .cpar archive file. For the example above, the tag would be `simplerabbits:1.7.0.1.0`.

Note

Cartridge_Name and Cartridge_Version must match what is encoded within the par file.

Example :

```
./buildCartridgeImage.sh -f /path/to/parfile/SimpleRabbits.par  
- This results in the image "simplerabbit:1.0.1", which can be retagged and  
  pushed as required.
```

OR

```
./buildCartridgeImage.sh -f /path/to/cparfile/COMCapabilities.cpar  
- This results in the image "comcapabilities-cpar:1.0.2", which can be  
  retagged and  
  pushed as required.
```

A Dockerfile is present at **osm-image-builder/samples/cartridgeAsImage**, and is responsible for creating a Docker image when this script is invoked. Modifications can be made to the Dockerfile as follows:

- Base Image: You can update the base image as needed, as long as it is a compatible Linux image.
- Custom configuration: You can add any OS level or network commands in the Dockerfile if required, such as applying patches, updating **/etc/hosts** , or certificates.

Note

Do not modify any other lines in the Dockerfile

Selecting Deployment Style and Cartridge Source

This topic outlines the deployment styles and methods you can use to deploy cartridge par files. We can consider there to be two categories of environments where cartridges need to be managed:

- 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 combinations:

- Design Studio deploy (during active cartridge development activity).
- Online or offline pardeploy using local file.
- Online or offline sync deploy using remote file repository or container images.

Deploying Cartridges in Controlled Environments

To install cartridges in controlled environments such as UAT, pre-production, and production, use only the declarative approach. Such environments require careful control of content as well as strong auditing of changes. Using the **sync** approach with an online or offline deployment using **manage-cartridges.sh** will ensure fitment to pipelines and strong validation and traceability. Choose remote file repositories or container image repositories to serve the cartridge par files in a secure, versioned and auditable way. Use Design Studio's headless build to automate building the par file itself from source as a precursor to putting it into the desired secured location.

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 the 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 }}
{{- 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-connection-pool>':
    JdbcResource:
      JDBCDriverParams:
        URL: 'jdbc:oracle:thin:@<database_host>:<database_port>/<database-
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
builder build -t "image_name:tag" --build-arg base_image=osm_base_image --
build-arg archive=archive_file_name.zip .
```

builder will be `docker` or `podman`.

3. Upload the generated image to your private image 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 supports configuration for any generic external LDAP provider. However, if the chosen LDAP provider has limitations with the generic LDAP provider, you would need 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 Fusion MiddleWare 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`).

For OSM cloud native, a custom authentication provider offers integration with an additional identity store, for example, Active Directory. This is in addition to the default authentication providers offered by WebLogic, `DefaultAuthenticator` and `DefaultIdentityAsserter`, which are required by OSM.

The following WDT model fragment is automatically combined with the `DefaultAuthenticator` and `DefaultIdentityAsserter` defined in OSM's default WDT model. Because of this, the custom authentication provider appears last among all authentication providers. Since the order of the authentication providers determines the order that they are queried, this means that users are authenticated against WebLogic's embedded LDAP first, and if no such user exists then the custom authentication provider is queried. This means that when a human user logs in, they are first checked against the machine users in embedded LDAP (which will not find a match) and then checked against the custom authentication provider.

```
{{- define "osm.custom-provider-support" -}}
topology:
  SecurityConfiguration:
    Realm:
      myrealm:
        AuthenticationProvider:
          YouLDAPProviderStartHere:
            .... <specific details here>

{{- end }}
```

Oracle recommends that you keep this order. If required, you can query the custom provider first by using the following pattern:

```
{{- 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 }}
```

This is WDT syntax that deletes the default providers specified in OSM's default WDT model, adds the custom provider, and recreates the default providers.

Note

In this configuration, the `ControlFlag` for the custom provider must be `SUFFICIENT`.

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.

① 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:

```
kubect1 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 Configuration Options

The OSM cloud native toolkit gives you options to set up your configuration based on your requirements. This chapter describes the configurations you can explore, allowing you to decide how best to configure your OSM cloud native environment to suit your needs.

You can choose configuration options for the following:

- [Manage LDAP Providers in WLS via OSM](#)
- [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](#)
- [Managing WebLogic Monitoring Exporter \(WME\) Metrics](#)
- [Proxy Configuration in Microservices](#)

The sections that follow provide instructions for working with these configuration options.

Manage LDAP Providers in WLS via OSM

This section explains how to manage your LDAP providers in WLS using OSM.

- OSM no longer supports the management of LDAP server via OSM cloud native toolkit scripts (such as managing accounts, users and groups in the LDAP server).
- Support for Generic LDAP to adapt all supported directory services by weblogic which uses LDAP including OpenLDAP has been provided.
- OSM will be providing functionality to users for other directory services via model extensions.

Integrating LDAP service with Weblogic via OSM

OSM provides two ways to integrate the LDAP service:

- **Generic LDAP:** It supports many of the LDAP vendors.
- **Model Extensions:** If a specific LDAP vendor is not compatible with Generic LDAP provider, a custom model extension can be used to configure a vendor-specific LDAP provider.

Using Generic LDAP

To use generic LDAP, you need to do the following:

Secret Creation

- `ldap-secret-creation`
`$OSM_CNTK/samples/credentials/manage-osm-ldap-credentials.sh -p project -i instance -c create -l ldap`
- After executing the above command, the script asks the host and domain node to get users and groups, user domain node and password to access the LDAP server.
- After providing all the credentials, the script creates `project-instance-ldap-credentials` secret which is used while creating an instance to import users from the LDAP server and assign respective groups.

Enabling LDAP in Project Specification

`<project>.yaml`

```
#External authentication
authentication:
# When enabled, kubernetes secret "project-instance-ldap-credentials"
# must exist
ldap:
  enabled: true
```

After above configurations, create the OSM cloud native instance. If it is already running, delete it and create it again.

Using LDAP Providers via Model Extensions

Other LDAP Providers as listed by [Weblogic](#) can be used via model extensions.

- Add the custom model tpl file which has configurations for the LDAP provider to the `$OSM_CNTK/samples/_custom-domain-model.tpl`.

`custom-ldaprovider.tpl`

```
{{- if .Values.custom.jms }}
custom-jms-support.74.yaml: |+
  {{- include "osm.custom-jms-support" $root | nindent 2 }}
{{- end }}
custom-<ldap-provider>-support.76.yaml: |+
  {{- include "osm.custom-ldap-provider-support" $root | nindent 2 }}
```

- Enable custom flag and add both `custom-domain-model.tpl` and `custom-ldap-provider-support.tpl` to `wdtFiles` in the project specification file.

`project.yaml`

```
# Sample WDT extensions can be enabled here. When enabled is true, then
# _custom-domain-model.tpl needs to be un-commented. Custom template files can
# also be added.
custom:
  enabled: true
  application: false
  jdbc: false
  jms: false
  #wdtFiles: [] # This empty declaration should be removed if adding items here.
  wdtFiles:
    # - _myWDTFile1.tpl
    # - _myWDTFile2.tpl
```

- `_custom-domain-model.tpl`
- `_custom-ldap-provider-support.tpl`

- Create the OSM cloud native instance. If it is already running, delete it and create it again. Here *custom-models-path* refers to the directory where the wdtFiles exist.

create-instance

```
$OSM_CNTK/scripts/create-instance.sh -p project -i instance -s specpath -m custom-models-path
```

Sample LDAP Provider Model File

If the decision is made to use a vendor specific LDAP provider instead of Generic LDAP provider, to illustrate the model extension required a sample model extension for Active Directory is provided at `$OSM_CNTK/samples/customExtensions/_custom-active_directory-support.tpl`. Such extensions can be populated with direct values for the fields or by referencing a custom secret. If referencing a custom secret, the secret should contain referenced fields and the secret must be specified in the section for custom secrets in project specification.

<project>.yaml

```
#Custom secrets at Project level
# Any custom secrets that can be used within WDT metadata fragments specified
# above. Secrets used in instance specific WDT should be listed in the instance
# specification. More than one secret name can be provided
project:
  customSecrets:
    #secretNames: {} # This empty declaration should be removed if adding items here.
    secretNames:
      - ldap-provider-secret
```

To configure a vendor specific LDAP Provider, refer to the "WebLogic Server MBean Reference Document" for the WebLogic version in your image manifest file, under **Configuration MBeans** → **Security MBeans**. Alternatively, use the method described in the section "[Displaying Valid Attributes and Child Attributes of a WDT Model](#)".

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.

- **charts/osm/shapes/prodxsmall.yaml**. This shape is similar to **prodsmall** with the exception that the CPU resources are scaled down to accommodate instances with lower order workloads.

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 enables you to plan your Kubernetes cluster resource requirement and are available under directory **charts/osm/shapes/** inside the toolkit.

The pre-defined shapes will have sizing requirements for the WebLogic pods (admin server and managed server), the microservice pods (OSM Gateway and RTUX) and for init (such as osm-gateway-init) as well as sidecar containers (wme, fluentd).

Based on the values (resource "request" and "limit") defined in the pre-defined shapes (or your custom shape), the Kubernetes scheduler attempts 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 OSM gateway microservices across all the development instances: Md (The number of gateway microservices will be always equal to the number of managed servers since we have 1:1 relationship)
- Number of RTUX microservices across all the development instances: D (RTUX microservice is always one per instance)
- Number of Fluentd containers across all the development instances (if enabled): D + Md
- Number of WME containers across all the development instances (unless disabled): D + Md
- Number of production (and production-like) instances required to be running in parallel: P
- Number of managed servers expected across all production instances: Mp
- Number of OSM gateway microservices across all the production instances: Mp
- Number of RTUX microservices across all the production instances: P
- Number of Fluentd containers across all the production instances (if enabled): P + Mp
- Number of WME containers across all the production instances (unless disabled): P + Mp
- Assume use of "dev" and "prod" shapes. The tables shown below are for example only. The real values will always come from the toolkit pre-defined shapes (or your custom shape).
- CPU requirement (CPUs) = $D * 1 + Md * 2 + Md * 1 + D * 0.5 + D * 0.5 + Mp * 0.5 + D * 0.5 + Mp * 0.5 + P * 2 + Mp * 15 + Mp * 6 + P * 2 + P * 0.5 + Md * 0.5 + P * 0.5 + Md * 0.5 = D * 2.5 + Md * 4 + P * 5 + Mp * 22$
- Memory requirement (GB) = $D * 4 + Md * 8 + Md * 2 + D * 1 + D * 1 + Md * 1 + D * 1 + Md * 1 + P * 8 + Mp * 48 + Mp * 12 + P * 4 + P * 1 + Mp * 1 + P * 1 + Mp * 1 = D * 7 + Md * 12 + P * 14 + Mp * 62$

The following table lists sample sizing requirements for "dev" shapes:

Table 8-1 Sample Sizing Requirements of "dev" Shapes

Component	Kube Request	Kube Limit
Admin Server	3 GB RAM, 1 CPU	4 GB RAM, 1 CPU
Managed Server	8 GB RAM, 2 CPU	8 GB RAM, 2 CPU
OSM Gateway	0.5 GB RAM, 0.5 CPU	2 GB RAM, 1 CPU
OSM RTUX	1 GB RAM, 0.5 CPU	1 GB RAM, 0.5 CPU
Fluentd	1 GB RAM, 0.5 CPU	1 GB RAM, 0.5 CPU
WME	1 GB RAM, 0.5 CPU	1 GB RAM, 0.5 CPU

The following table lists sample sizing requirements for "prod" shapes:

Table 8-2 Sample Sizing Requirements of "prod" Shapes

Component	Kube Request	Kube Limit
Admin Server	8 GB RAM, 2 CPU	8 GB RAM, 2 CPU
Managed Server	48 GB RAM, 15 CPU	48 GB RAM, 15 CPU
OSM Gateway	12 GB RAM, 6 CPU	12 GB RAM, 6 CPU
OSM RTUX	4 GB RAM, 2 CPU	4 GB RAM, 2 CPU
Fluentd	1 GB RAM, 0.5 CPU	1 GB RAM, 0.5 CPU
WME	1 GB RAM, 0.5 CPU	1 GB RAM, 0.5 CPU

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 nativeshapes are devsmall, dev, prodsmall, prod, and prodlarge.
Alternatively, custom shape name can be specified (as the filename without the extension)

Init and Sidecar Containers Resourcing

The standard shapes mentioned above have sections respect to resources for all containers including init (OSM Gateway and OSM DB Installer) and sidecar containers (WME and FluentD associated with Weblogic pods). The default values in the shapes are already tested for the respective shapes' corresponding workloads. While creating custom shapes, if you want to modify resource allocation for a particular instance, you can uptake the particular section from the standard shapes and adjust the resources accordingly. For example, if you want to adjust the resource allocation for the WME container, you can uptake the following section from any of the standard shape and adjust the resources. For any custom sidecar containers configured in the project specification, you have to take account for configuring your resource allocation. For more information about working with custom shapes, refer to "[Working with Shapes](#)."

```
# Sets resources to the wme sidecar container when enabled.
wme:
  resources:
    requests:
      cpu: "500m"
      memory: "1G"
    limits:
      cpu: "500m"
      memory: "1G"
```

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 recommended approach to tune these parameters, see the section about "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 colon-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 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 overridden in instance specification if required.
# osmWLSTargetNodes, if defined, restricts all OSM cloud native WebLogic pods
# and DB
# Installer pods to worker nodes that match the label conditions and for these
# pods, will take precedence over osmcnTargetNodes.
osmWLSTargetNodes: {} # This empty declaration should be removed if adding
items here.
#osmWLSTargetNodes:
#  nodeLabel:
#    example.com/licensed-for-coherence is just an indicative example; any
#    label and its values can be used for choosing nodes.
#    key: example.com/licensed-for-coherence
#    values:
#      - true

# osmcnTargetNodes, if defined, restricts all OSM cloud native pods to worker
# nodes
# that match the label conditions. This value will be ignored for OSM cloud
# native
# WebLogic pods and DB Installer pods if osmWLSTargetNodes is also specified.
osmcnTargetNodes: {} # This empty declaration should be removed if adding
items here.
#osmcnTargetNodes:
#  nodeLabel:
#    # example.com/use-for-osm is just an indicative example; any label and
#    its values can be used for choosing nodes.
#    key: oracle.com/use-for-osm
```

```
# 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 allows for the configurable exposure of back-end services to clients outside the Kubernetes cluster, via edge objects like NodePort services, Load Balancers, and so on. In OSM cloud native, an ingress controller can be selected and configured in the project specification.

OSM cloud native supports annotation-based "generic ingress" creation, which means the use of the standard Kubernetes Ingress API (as opposed to a proprietary ingress Custom Resource Definition), as verified by Kubernetes Conformance tests. The benefit of this is that it works for any Kubernetes certified ingress controller, provided that the ingress controller offers annotations (which are generally proprietary to the ingress controller) required for proper functioning of OSM.

Annotations applied to an Ingress resource allow you to use advanced features (like connection timeout, URL rewrite, retry, additional headers, redirects, sticky cookie services, and so on) and to fine-tune behavior for that Ingress resource. Different Ingress controllers support different annotations. For information about different Ingress controllers, see Kubernetes documentation at <https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/>. Review this documentation for your Ingress controller to confirm which annotations are supported.

Any Ingress Controller, which conforms to the standard Kubernetes ingress API and supports annotations needed by OSM should work, although Oracle does not certify individual Ingress controllers to confirm this "generic" compatibility.

The configurations required in your project specification are as follows:

Note

The configurations below take NGINX as an example and NGINX specific annotations are provided at `$OSM_CNTK/samples/project.yaml`.

```
# valid values are GENERIC, OTHER  
ingressController: "GENERIC"
```

```
# When ingressController is set to GENERIC, the actual ingress controller  
# might require some annotations to be added to the Kubernetes  
# Ingress object. Place annotations that are must-have for such a controller  
# and/or common to all instances here. Instance specific  
# annotations can be placed in the instance spec file.
```

```

ingress:
  ingressClassName: nginx
  # This annotation is required for nginx ingress controller.
  annotations:
    nginx.ingress.kubernetes.io/proxy-body-size: "50m"
    nginx.ingress.kubernetes.io/affinity: 'cookie'
    nginx.ingress.kubernetes.io/session-cookie-name: 'sticky'
    nginx.ingress.kubernetes.io/affinity-mode: 'persistent'
  osmgw:
    # This annotation is required for nginx ingress controller for osm
gateway:
  annotations:
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /$1
rtux:
  # This annotation is required for nginx ingress controller for rtux.
  annotations:
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /orchestration-operations/$1
oca:
  # This annotation is required for nginx ingress controller for oca.
  annotations:
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /$1
    nginx.ingress.kubernetes.io/proxy-read-timeout: "240"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "240"

```

Note

You need to ensure that Ingress has annotation to handle large body size of client request, like large order payloads (during regular processing) or large cartridge par files (while deploying from Design Studio). For example, `nginx.ingress.kubernetes.io/proxy-body-size: "50m"`. See the documentation about Ingress NGINX Controller at <https://github.com/kubernetes/ingress-nginx/blob/main/README.md#readme>.

When **ssl.incoming** is set to **true** in the instance specification file, weblogic custom request headers `X-Forwarded-Proto: https` and `WL-Proxy-SSL: true` need to be added as annotations. You also need to ensure that you remove any incoming `WL-Proxy-SSL` header. This protects you from a malicious user sending in a request to appear secure to WebLogic when it isn't.

The configurations required in your instance specification are:

Note

The configurations below take NGINX as an example and NGINX specific annotations are provided at `$OSM_CNTK/samples/instance.yaml`.

```

# SSL Configuration
ssl:

```

```

incoming: true

ingress:
  # These annotations are required if project spec ingressController is
  "GENERIC" and SSL enabled.
  # Different Ingress controller can have implementation specific
  annotations and can be added here.
  # Provided annotations below for nginx.
  # These annotations are required if project spec ingressController is
  "GENERIC"
  # and the actual ingress controller is nginx with wls custom request
  headers.
  annotations:
    nginx.ingress.kubernetes.io/configuration-snippet: |
      more_clear_input_headers "WL-Proxy-Client-IP" "WL-Proxy-SSL";
      more_set_input_headers "X-Forwarded-Proto: https";
      more_set_input_headers "WL-Proxy-SSL: true";
    nginx.ingress.kubernetes.io/ingress.allow-http: "false"

```

The NGINX controller works by creating an operator in its own "nginx" (or other specified) namespace, and exposing this as a service outside of the Kubernetes cluster (NodePort, LoadBalancer, and so on).

Using NGINX Ingress Controller

OSM can use an NGINX-ingress common to all applications in the Kubernetes cluster or it can use an NGINX-ingress that is dedicated to all OSM instances in that cluster.

Common NGINX-Ingress to All Applications

OSM requires snippet annotations for NGINX (annotations are provided at **\$OSM_CNTK/samples/project.yaml** and **\$OSM_CNTK/samples/instance.yaml**) which is disabled by default. Enabling this is recommended only for trusted ingress objects. If you want to use common NGINX, you need to deem OSM Cloud Native ingresses as trusted. To do so, you need to edit the the NGINX controller configmap to include following lines under the data field:

```

data:
  allow-snippet-annotations: "true"
  annotations-risk-level: Critical

```

Dedicated NGINX-Ingress Specific to All OSM Instances

If you do not want the **annotations-risk-level** to be **Critical**, you can configure a dedicated NGINX-Ingress for OSM. NGINX-Ingress is still shared but only among OSM instances that have SSL access. To do so, you need to include a custom configmap with the following content for you dedicated NGINX-ingress:

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: custom-headers
  namespace: <nginx-namespace>
data:
  WL-Proxy-SSL: "true"
  X-Forwarded-Proto: "https"

```

You also need to add the following field in the data section of the NGINX-controller configmap:

```
data:
  proxy-set-headers: "<nginx-namespace>/custom-headers"
```

To make OSM instances use the dedicated NGINX-ingress, add the IngressClass name of the NGINX-ingress controller in the project specification.

```
ingress:
  ingressClassName: <nginx-ingressClassName>
```

To accommodate all types of ingress controllers and exposure options, you need to specify the **inboundGateway.host** and **inboundGateway.port** in the **instance.yaml** file. Populate the values in the **instance.yaml** file before invoking the `create-instance.sh` command to create an instance:

```
inboundGateway:
  accessStyle: host
  # Provide FQDN (recommended) or IP address of the actual ingress point/
loadbalancer
  host:
  # Provide Port of the actual ingress point/loadbalancer
  port:
  # Provide if protocol is different from ssl.incoming setting
  # Accepted values (http, https)
  #protocolOverride: # Optional
```

To create an ingress, run:

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

To delete an ingress, run:

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

Using an Alternate Ingress Controller

By default, OSM cloud native supports standard Kubernetes ingress API and provides sample files for integration. If your desired ingress controller does not support one or more configurations via annotations on generic ingress, or you wish to use your ingress controller's CRD instead, 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 tables that follow. The toolkit uses an ingress Helm chart (**`$OSM_CNTK/samples/charts/ingress-per-domain/templates/generic-ingress.yaml`**) and scripts for creating the ingress objects. These samples can be used as a reference to make copies and customize as necessary.

The actual ingress controller might require some annotations to be added to the Kubernetes ingress object. The toolkit has provided all those annotations in the sample project and instance specification files in the cloud native toolkit. These can be used as a reference to make copies and customize as necessary.

The host-based rules and the corresponding back-end Kubernetes service mapping are provided using the `clusterName` definition, which is the name of the cluster in lowercase. Replace any hyphens with underscores. The default, unless overridden, is `c1`.

The host-based rules and the corresponding back-end Kubernetes service mapping are provided using the `clusterName` definition, which is the name of the cluster in lowercase. Replace any hyphens with underscores. The default, unless overridden, is **`c1`**.

The following table lists the service name and service ports for Ingress rules. All services require ingress session stickiness to be turned on so that once authenticated, all subsequent requests reach the same endpoint. All these services can be addressed within the Kubernetes cluster using the standard Kubernetes DNS for services.

Table 8-3 Service Name and Service Ports for Host-based Ingress Rules

Rule (host)	Service Name	Service Port	Purpose
<code>instance.project.loadBalancerDomainName</code>	<code>project-instance-cluster-clusterName</code>	8001	For access to OSM through UI, XMLAPI, Web Services, and so on.
<code>t3.instance.project.loadBalancerDomainName</code>	<code>project-instance-cluster-clusterName</code>	30303	OSM T3 Channel access for WLST, JMS, and SAF clients.
<code>admin.instance.project.loadBalancerDomainName</code>	<code>project-instance-admin</code>	7001	For access to OSM WebLogic Admin and Remote Console UI.

The path-based rules and the corresponding back-end Kubernetes service mapping are provided using the following definitions. All these services can be addressed within the Kubernetes cluster using the standard Kubernetes DNS for services.

The following table lists the service name and service ports for Ingress rules:

Table 8-4 Service Name and Service Ports for Path-based Ingress Rules

Rule (path)	rewrite-target	Service Name	Service Port	Purpose
<code>/orchestration/project/instance/tmf-api/(.*)</code>	<code>/s1</code>	<code>project-instance-osm-gateway</code>	8080	For access to OSM TMF REST APIs.
<code>/orchestration/project/instance/fallout/(.*)</code>	<code>/s1</code>	<code>project-instance-osm-gateway</code>	8080	For access to OSM Fallout Exception REST APIs.
<code>/orchestration/project/instance/orchestration-operations/(.*)</code>	<code>/orchestration-operations/s1</code>	<code>project-instance-osm-runtime-ux-server</code>	8080	For user interface access to instance data.
<code>/orchestration/project/instance/(cartridgeAssembler/(.*)</code>	<code>/s1</code>	<code>project-instance-oca</code>	8080	For access to OSM Cartridge Assembler APIs.

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.

- The reference sample for creation is: **`$OSM_CNTK/scripts/create-ingress.sh`**

- The reference sample for deletion is: `$OSM_CNTK/scripts/delete-ingress.sh`

Preconfiguration on Primary and Standby Database

To achieve a highly available database service with automatic failover and role interchange you must adopt a role based approach utilizing both primary and standby databases. This configuration ensures that the service always connects to the active database. In the event of a failover or switchover, roles are automatically interchanged and the connection is rerouted to the new active database.

Preconfigure the service at the PDB level using the command below. The default service created during PDB creation cannot be used.

```
srvctl add service -d database_Uniquename -s servicename -pdb pdb_name -l  
PRIMARY -e SESSION  
-m BASIC -w 10 -z 10
```

```
Example: srvctl add service -d database_name -s service_name -pdb pdbname -l  
PRIMARY -e  
SESSION -m BASIC -w failover_delay -z failback_delay
```

In order to know more, refer to *Appendix F Oracle Database Clusterware Administration and Deployment Guide*.

Note

The service name must be identical on both servers.

Configuring the OSM Application for High Availability

You can configure OSM to recognize both the primary and standby databases, so all OSM components and microservices interact with these databases and benefit from high availability and data protection.

In instance specification add the secondary database details as shown below.

In order to know more about adding a secondary database in case of a standalone setup, refer to [Planning and Validating Your Cloud Native Environment](#).

```
db:  
  type: "STANDARD"  
  # datasourcesPrimary section is applicable only for STANDARD DB. For ADB,  
  values will be used from Autonomous Database Serverless secrets+configMap.  
  datasourcesPrimary:  
    port: 1521  
    host: dbserverPrimary-ip  
  datasourcesSecondary:  
    port: 1521  
    host: dbserverSecondary-ip
```

Ensure that you make a note of the following things while configuring OSM:

- Always ensure that there are at least two managed servers when configuring the secondary database in OSM. For example, when the `clusterSize >= 2`.
- Standby database configuration in OSM is applicable only for a single instance database and not RAC database.
- `db.type` should be standard in `instance.yaml`.

Data Guard Setup on OCI

Oracle Data Guard setup on OCI allows for the configuration of a standby database that automatically syncs with the primary Oracle database. OCI can handle both the set up and monitoring of Data Guard as a service. To set up Oracle Data Guard on OCI, do the following:

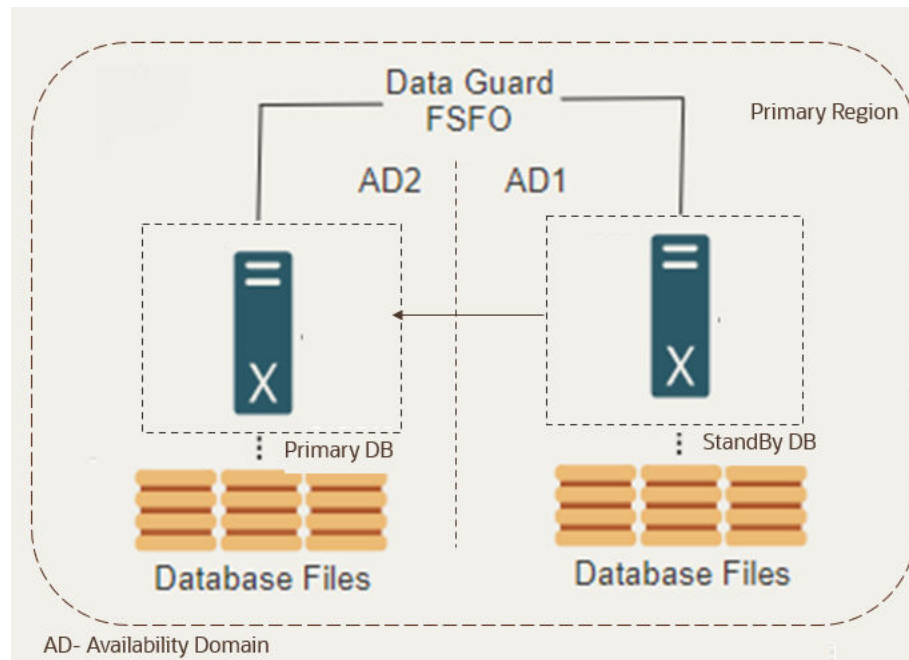
1. **Provision Primary and Standby Databases:** Use the OCI console to provision the primary and standby databases, ensuring they are in the same region.
2. **Enable Data Guard Configuration:** In the OCI console, navigate to the database service and configure Data Guard by creating a standby database from the existing primary database.

In order to know more, you can refer to the *Oracle Cloud Infrastructure* documentation and *Oracle Database Data Guard Broker* documentation.

Note

It's recommended that you select the Protection Mode as high availability.

Figure 8-1 Data Guard Setup on OCI



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

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 sr -i quick -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.

- **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 opssWP and opssWF 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", "prodsml", 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:
  agLlincensed: 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*
- The OSM Gateway logs can be found at: *pv_directory/project-instance/osm-gateway*
- The OSM Runtime UX logs can be found at: *pv_directory/project-instance/osm-runtime-ux-server*

The following applies to OSM Gateway logs:

- Each log file gets rolled over daily and will be retained for 30 days.
- The size of each log file is limited to 10 MB.

Configuring Fluentd Logging

OSM logs can be processed via Fluentd using the following mechanisms:

- The WebLogic pods (admin server and managed server) and DB Installer pods log to Fluentd via a sidecar. For the pods covered by a sidecar, you configure Fluentd using Helm charts.
- The microservice pods (OSM Gateway and RTUX) write to container logs. Fluentd has to be configured externally to process these logs.

Configuring Fluentd Logging for OSM Core Pods

OSM supports integration of Fluentd as a sidecar container to read the log entries from WebLogic pods and DB Installer pods. Fluentd can be integrated with Elasticsearch, OpenSearch or other equivalent upstream components and Kibana, OpenSearch Dashboard or other equivalent visualization components.

OSM provides samples and configuration for the following combinations

- Fluentd, Elasticsearch, and Kibana
- Fluentd, OpenSearch and OpenSearch Dashboards

Using a log processing stack such as Elastic Stack (ElasticSearch, Fluentd, and Kibana) or OpenSearch Stack (OpenSearch, OpenSearch Dashboards and Fluentd) in OSM cluster can make it much easier to collect, store, and analyze log data from all the Weblogic pods and OSM DB installer pods in the OSM cluster, making it more manageable and more accessible for different users. You can use the fields defined in Fluentd as filters in Kibana to filter and search for particular log events.

To enable Fluentd with built-in log parsing configuration:

1. In the **instance.yaml** file, enable fluentdLogging.

```
fluentdLogging:
  enabled: true
  outputType: elasticsearch # Acceptable values are elasticsearch and
  opensearch, Default is elasticsearch
  outputProtocol: http # Acceptable values are http and https, Default is
  http
# image: fluent/fluentd-kubernetes-daemonset:v1.14.5-debian-
  elasticsearch7-1.1 # default if none specified
# imagePullPolicy: IfNotPresent
```

- Example for enabling Fluentd logging with ElasticSearch

```
fluentdLogging:
  enabled: true
  outputType: elasticsearch
  outputProtocol: http # Protocol of elasticsearch
# image: fluent/fluentd-kubernetes-daemonset:v1.14.5-debian-elasticsearch7-1.1
# default if none specified
# imagePullPolicy: IfNotPresent
```

- Example for enabling Fluentd logging with OpenSearch

```
fluentdLogging:
  enabled: true
  outputType: opensearch
  outputProtocol: http # Protocol of opensearch
  image: fluent/fluentd-kubernetes-daemonset:v1-debian-opensearch
# imagePullPolicy: IfNotPresent
```

Note

If you are using OpenSearch, then you must provide a Fluentd-OpenSearch image as the default image is for Fluentd-ElasticSearch. You can use any Fluentd image which is mandated by your organization or else use the Fluentd image from Docker Hub or an equivalent.

2. Get the IP address or host name and port details of ElasticSearch, OpenSearch or other equivalent upstream component for Fluentd.
3. Create a secret for Fluentd credentials:

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

```
Provide 'elasticsearch' credentials for 'sr-quick' ...
Host: host_IP_address
Port: port
```

Username: *username*
Password: *password*

secret/sr-quick-fluentd-credentials configured

4. Create or upgrade the OSM instance. Use the Kibana user interface, OpenSearch Dashboard or other equivalent visualization tools to view the resulting logs. By default, OSM populates some default fields, which you can use to filter the log messages in a visualization tool such as Kibana or OpenSearch Dashboards.

The following table lists some fields for filtering logs.

Table 8-5 Fields for Filtering Logs

Field	Description	Example
tag	Retrieves the log events related to a particular log file. The value should be in the following pattern: {{ProjectName}}- {{InstanceName}}_{{LogFileName}}	DB Installer logs → sr-quick_dbInstaller.log Weblogic introspector logs → sr- quick_introspector_script.out Weblogic server logs → Adminserver → sr- quick_admin.log, sr- quick_admin.out, sr- quick_admin_nodemanager.log, sr- quick_admin_nodemanager.out MS1 → sr- quick_ms1.log, sr- quick_ms1.out, sr- quick_ms1_nodemanager.log, sr- quick_ms1_nodemanager.out

Table 8-5 (Cont.) Fields for Filtering Logs

Field	Description	Example
-index	Retrieves the log messages related to a particular instance. The value should be in the following pattern: {{ProjectName}}-{{InstanceName}}	sr-quick
servername	Retrieves the log messages related to a particular server. The value should be in the following pattern: {{ProjectName}}-{{InstanceName}}_{{ServerName}}	sr-quick-admin, sr-quick-ms1
level	Retrieves the log messages of specific log level. The value should be in the following pattern: Info/Warning/Debug/Error ...	Info/Warning/Debug/Error ...
logger	Retrieves the log messages generated by a class. The value should be the logger name. For example, if you want to see the status of automation plugins, enter oracle.communications.ordermanagement.automation.plugin.AutomationPluginManager	oracle.communications.ordermanagement.automation.plugin.AutomationPluginManager

Configuring Fluentd Logging for Microservices

For the microservice pods, you must configure Fluentd externally to process the logs. The Fluentd configuration to interpret OSM logs is provided in the cloud native toolkit's **samples/ fluentd/ fluentd.conf** sample file. Refer to Fluentd documentation for information about setting up Fluentd and importing this configuration into it.

All logs of OSM microservices logs are written to **stdout/stderr** and then appear in container logs. You can parse the OSM microservices logs using the tool of your choice. You can use Fluentd deployed as a daemonset within your Kubernetes cluster. To interpret the OSM microservices logs, utilize the log format provided.

All OSM microservices logs are written in the following format:

```
%date{yyyy-MM-dd HH:mm:ss.SSS} [%t] [%-5level] [%logger] - %msg%n
```

The log messages start with a date in the yyyy-mm-dd hh:mm:ss.sss format. This is followed by the thread id and severity, after which, the logger and the log message can be seen. Parse all OSM microservices logs using this pattern.

For parsing OSM microservices logs, use Fluentd deployed as a daemonset within your Kubernetes cluster.

The following is an example of the OSM Gateway microservice log:

```
2023-07-14 14:13:02.842 [pool-3-thread-32] [INFO ]
[oracle.comms.ordermanagement.noa.cloudevent.HttpNOAEventProcessor] -
path:productOrderingManagement/v4.0.0.1.0/listener/
productOrderStateChangeEvent\n
```

The following is an example of the RTUX microservice log:

```
2023-07-14 14:13:02.842 [pool-3-thread-32] [INFO ]
[oracle.comms.ordermanagement.noa.cloudevent.HttpNOAEventProcessor] -
path:productOrderingManagement/v4.0.0.1.0/listener/
productOrderStateChangeEvent\n
```

Note

Ensure that all OSM microservices logs originating in the same Kubernetes cluster are either in JSON or Text format. Do not generate them in both the formats.

Obfuscating Sensitive Data in Logs

You can mask sensitive data (personal information) that is logged to files, the terminal (stdout and stderr), or sent to a log monitor. Sensitive data includes details such as names of persons, addresses, and account numbers.

The masking of sensitive data applies only to the DEBUG log level. Cartridge developers are not expected to expose potentially sensitive data unless they do it via DEBUG logs. OSM itself does not expose potentially sensitive data except in DEBUG logs. OSM masks such exposed sensitive data only in DEBUG logs.

To leverage OSM's capability to mask sensitive data in logs:

- Cartridge developers have to provide a draft of the Personal Information (PI) regex configuration required to identify sensitive data, as part of the cartridge development process.
- Testers have to monitor log outputs and adjust the PI regex configuration as required to ensure all sensitive information is masked.
- Administrators have to ensure the tested PI regex configuration is added to the instance specification when creating or upgrading an OSM cloud native instance.

Prerequisite Configuration

OSM cloud native turns on its log masking capability if the instance has the following effective values in its instance specification:

```
...
log:
  # handlerLevel filters the logs lower than its level.
  # Here log level TRACE takes a numeric value between 1(highest severity)
  and 32(lowest severity) e.g. TRACE:1
  handlerLevel: "TRACE:1"
  # This is to optionally control logging level for specific classes.
  Uncomment to add the entries.
  # 'class' will have full ClassName e.g. com.mslv.oms.poller.EventPoller
  # 'level' should have same possible values as above e.g. TRACE:1
  # Give class as "root" to set level for all classes.
  #loggers:
  # - class:
  #   level:
  loggers:
    - class: "root"
      level: "TRACE:1"
...
```

The OSM cloud native loggers.level property TRACE is equivalent to DEBUG.

The OSM Gateway and RTUX microservices turn on their log masking capability if the instance has the following effective values in the instance specification:

```
...
osm-gateway:
  log:
    level: FINE
osmRuntimeUX:
  log:
    level: FINE
...
```

The osm-gateway and osmRuntimeUX log.level properties determine the logging level of the logs and must be set to FINE to enable log masking. FINE is equivalent to DEBUG.

Configuring Log Masking

Sensitive data is identified and masked based on Java regular expression (regex) patterns defined in the instance specification as a list of entries under "logMaskingCustomRegexes". Each entry describes one item of PI data and how to recognize it using a regex. Any entries provided here are added to predefined entries. To see the full list of predefined entries for your version of the toolkit, review the contents of the **charts/osm/templates/osm-gdpr-regex-json.yaml** file in the toolkit. Predefined entries include patterns for email addresses and for phone numbers contained in an element called "phoneNumber".

Once PI data is identified in a log message using one of these regexes, the masking is done by substituting the sensitive data with a string of 4 stars "****".

Custom Regex Patterns

Depending on the logs emitted by the cartridge code, additional regex patterns can be configured to mask PI data that may become exposed. This is done by adding entries to the "logMaskingCustomRegexes" element in the instance specification:

```
...
logMaskingCustomRegexes:
  - description: "account number"
    type: partial
    regex: "\"(?i)accountNumber\"\\s*:\\s*\"(.*?)\""
```

where:

- description is a human readable description of the field targeted by this regex entry.
- type is the type of the regex pattern. Possible values are either **partial** or **exact**.
- regex is the Java regex pattern for the value to be recognized

Note that the regex patterns should be a valid string as per Java standards described at: <https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/regex/Pattern.html> or similar source.

Partial Regex Patterns

Partial regexes provide a pattern that matches only some part of the target field's values. These regex patterns are applied to XML and JSON documents or fragments in log messages. The regex pattern should begin with the field name in it and encompass the rest of the value, using wildcards as necessary. If this regex matches an XML or JSON line in a log message, the value part of the matching field is masked.

For example, to mask the account number contained in a JSON field called "accountNumber".

The following block shows the partial regex pattern in the instance specification:

```
...
logMaskingCustomRegexes:
  - description: "account number"
    type: partial
    regex: "\"(?i)accountNumber\"\\s*:\\s*\"(.*?)\""
```

The following block shows a sample log message as generated by the cartridge:

```
...
"ownerAccount": {
  "accountNumber": "1234321",
  "id": "0CX-1XYHGQ",
  "@type": "AccountRef",
}
```

The following block shows a sample output log message:

```
...
"ownerAccount": {
  "accountNumber": "*****",
  "id": "0CX-1XYHGQ",
  "@type": "AccountRef",
}
...
```

Exact Regex Patterns

Exact regex patterns provide a complete match mechanism to identify the PI data to mask. OSM looks through all log messages (not just XML and JSON portions) for such regex matches.

Everything in the log message that matches an exact regex pattern will be masked. Field names or other situating strings cannot be part of an exact regex pattern as otherwise, they too will get masked.

For example, given the below exact regex pattern, OSM will look for any string composed of digits in the format xxx-xx-xxxx in any log message. If found, that string is masked. To illustrate the scope of application of exact regex patterns, in the example below, even though the intention was to mask US Social Security Numbers, the masking feature will apply suppression to any string or sub-string that has digits in the format xxx-xx-xxxx.

The following block shows the Exact regex pattern in the instance specification:

```
...
logMaskingCustomRegexes:
- description: "US Social Security Number"
  type: exact
  regex: "\\d{3}-\\d{2}-\\d{4}"
...
```

The following block shows a sample log as generated by a cartridge:

```
...
"US Social Security Number": "232-45-3434",
"orderReference": "987-87-8765"
...
```

The following block shows a sample output log message:

```
...
"US Social Security Number": "*****",
"orderReference": "*****"
...
```

Configuring Logging and Log Rotation

OSM cloud native provides a way to configure Oracle Diagnostic Logging (ODL) logging level to debug logs in an efficient manner.

This configuration is defined via the instance specification as follows:

```
# The valid log levels in descending order are:
# INCIDENT_ERROR (highest value)
# ERROR
# WARNING
# NOTIFICATION
# TRACE (lowest value)
# Each log level also takes a numeric value between 1(highest severity) and
32(lowest severity) e.g. ERROR:1
log: [] # This empty declaration should be removed if adding items here.
#log:
#   handlerLevel filters the logs lower than its level.
#   Set the handlerLevel lower or equal to class level.
#   handlerLevel: ""
#   This is to optionally control logging level for specific classes.
Uncomment to add the entries.
#   'class' will have full ClassName e.g. com.mslv.oms.poller.EventPoller
#   'level' will have same possible values as above e.g. ERROR:1
#   Give class as "root" to set level for all classes.
#   loggers:
#     - class:
#       level:
```

Valid ODL Log Levels

For ODL log levels, refer to the "About Log Severity Levels" section in *OSM Cloud Native System Administrator's Guide*. Each message type can also take a numeric value between 1 (highest severity) and 32 (lowest severity) that you can use to further restrict log output (for example ERROR:1).

When you specify a level, ODL returns all log messages of that type, as well as the messages that have a higher severity. For example, if you set the level to WARNING, ODL also returns log messages of type INCIDENT_ERROR and ERROR.

Configure ODL Handlers Logging Level

To configure Logging level for the ODL handlers (odl-handler, console-handler and wls-domain), set **log.handlerLevel** with appropriate value (for example, ERROR:1). An empty value would have the default setting (WARNING) for the handlers.

Configure Logging Level for Specific Class

To enable logging for a specific class or package, log.loggers[] can also be configured as follows. It can have multiple entries for different classes.

The log level for **class** should be of equal or higher level compared to log **handlerLevel**. Logs of lower level than the handlerLevel do not appear in the logs.

To set log level for all the classes, provide class as "root". The class specific logger level overrides the root log level for that class.

```
log:
  handlerLevel: ""
  loggers:
    - class: "root"
      level: "NOTIFICATION:1"
```

```
- class: "com.mslv.oms.poller.EventPoller"
  level: "ERROR:1"
```

Log Files Rotation

OSM pods use ephemeral storage to store log files, GC logs, and JFR data. All of these have to be managed so that worker nodes do not fail because they run out of ephemeral/container storage. For all the logs, GC logs, and JFR data, OSM cloud native provides log rotation and retention mechanisms to put an upper limit on the space they take. These are defined via specifications as described in the following table:

Table 8-6 Log Files Rotation in Specification Files

Data	Specification	Example	Log Location: PVC Enabled	Log Location: PVC Disabled
OSM logs	Shape specification weblogic.log.FileMinSize weblogic.log.FileCount	Dev shape has file count 7 and size 500k	Admin server: / logMount/\$ {DOMAIN_ID}/logs/ admin.log Managed server: / logMount/\$ {DOMAIN_ID}/logs/ ms1.log	Admin server: /u01/ oracle/user_projects/ domains/domain/ servers/admin/logs/ admin.log Managed server: /u01/ oracle/user_projects/ domains/domain/ servers/ms1/logs/ ms1.log
GC logs	Shape specification NumberOfGCLogFiles GCLogFileSize	Dev shape has file count 7 and size 500k	Admin server: / logMount/\$ (DOMAIN_UID)/logs/ admin-gc-%t.log Managed server: / logMount/\$ (DOMAIN_UID)/logs / gc-\$ (SERVER_NAME)- %t.log	Admin server: /u01/ oracle/user_projects/ domains/domain/gc-\$ (SERVER_NAME)- %t.log Managed server: /u01/ oracle/user_projects/ domains/domain/gc-\$ (SERVER_NAME)- %t.log
JFR data	Instance specification jfr: enabled: max_age: max_size:	Default maximum age is 4 hours , and the maximum size is 100 MB	/logMount/\$ (DOMAIN_UID)/ performance/\$ (SERVER_NAME)	/logMount/\$ (DOMAIN_UID)/ performance/\$ (SERVER_NAME)/

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
```

By default, the OSM Gateway pod and the RTUX pod expose metrics to Prometheus (or any other compliant tool) scrape. This is controlled by the instance specification as follows:

```
# # Prometheus monitoring is enabled by default.
# prometheus:
#   enabled: true
```

To disable this behavior, set `prometheus.enabled` to `false`.

Configuring Prometheus for OSM Cloud Native Metrics

For OSM cloud native metrics, configure the scrape job in Prometheus as follows:

Note

During the installation, the OSM installer creates a user who is authorized to view OSM and Weblogic server metrics.

```
- job_name: 'osmcn'
  # HTTP basic authentication information
  basic_auth:
    username: oms-metrics
    password: password
  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

- job_name: osmgateway
```

```

oauth2:
  client_id: client-id
  client_secret: client-secret
  scopes:
    - scope
  token_url: OIDC_token_URL
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_app
    action: keep
    regex: (^.+osm-gateway$)
  - source_labels:
      - __meta_kubernetes_pod_container_port_number
    action: keep
    regex: (8080)
  - 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_namespace
    action: replace
    target_label: kubernetes_namespace
  - source_labels:
      - __meta_kubernetes_pod_name
    action: replace
    target_label: kubernetes_pod_name

- job_name: 'rtux'
  oauth2:
    client_id: client-id
    client_secret: client-secret
    scopes:
      - scope
    token_url: OIDC_token_URL
  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_app]
    action: keep
    regex: (^.*-osm-runtime-ux-server$)
  - source_labels: [__meta_kubernetes_pod_container_port_number]
    action: keep
    regex: (8080)
  - 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_namespace]
    action: replace
    target_label: kubernetes_namespace
  - source_labels: [__meta_kubernetes_pod_name]
    action: replace
    target_label: kubernetes_pod_name

- job_name: 'oca'
  metrics_path: /metrics
  oauth2:
    client_id: client-id
    client_secret: client-secret
    scopes:
      - scope
    token_url: OIDC_token_URL
  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_app]
    action: keep
    regex: (^.*-oca$)
  - source_labels: [__meta_kubernetes_pod_container_port_number]
    action: keep
    regex: (8080)
  - 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_namespace]
 action: replace
 target_label: kubernetes_namespace
- source_labels: [__meta_kubernetes_pod_name]
 action: replace
 target_label: kubernetes_pod_name

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>. See the *OSM Compatibility Matrix* for the compatible versions.

The endpoint for the OSM Gateway, RTUX and OCA microservices metrics is secured using OAUTH2 credential. Hence, it is required to configure scrape jobs for the pods with the same OAUTH2 credential.

The same job name is used in the sample grafana dashboards to filter the metrics of the microservice.

Viewing Metrics Without Using Prometheus

To view metrics without using Prometheus:

1. Find the internal IP address of the pod by running the following command:

```
kubectl get pod pod-name -n namespace -o yaml
```

2. Log in to any of the worker nodes of the Kubernetes cluster.
3. Do the following:
 - For microservices, curl to `http://pod_ip:port/metrics` using the oath2 token.
 - For OSM, curl to `http://pod_ip:port/OrderManagement/metrics` using basic authorization. OSM metrics should be accessed either with **oms-metrics** as a user or, if you are using another user, then make sure to have the role **omsMetricsGroup** assigned to that user.

For more details, see Kubernetes documentation at: <https://kubernetes.io/docs/tutorials/services/connect-applications-service/#exposing-pods-to-the-cluster>.

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 SDK comes with sample Grafana dashboards to get you started with visualizations.

Import the dashboard JSON files from the location **SDK/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.
- **OSM and Weblogic by Server:** Provides a view of OSM cloud native metrics and WebLogic Monitoring Exporter metrics for one or more managed servers for a given instance in the selected project namespace.
- **TMF and SI Messaging by Instance:** Provides a view of the TMF API and System Interaction messaging metrics for one or more instances in the selected project namespace.
- **OCA Publish Operations:** Provides a view of the OCA Publish Operations metrics for one or more instances in the selected project namespace.

These are provided as samples. You can import them as-is into a Grafana environment. They can also be used as a pattern to create a set of dashboards for specific requirements.

About TMF and SI Messaging by Instance

You can use the following global filters in the sample dashboard:

- **Project:** Displays the available projects and the default. This does not have multi-selection.
- **Instance:** Displays the filter value based on the selected project.
- **Pod:** Displays the filter value based on the selected instance.
- **API Category:** Displays the filter value based on the selected pod.
- **API:** Displays the API names such as serviceOrdering.
- **API Version:** Displays the API version based on the selected API.
- **Target System:** Displays the target system based on the selected API and version.
- **Message Status:** Displays the status of request or events.

You can use the following detail panels in the sample dashboard:

- **Incoming REST Requests per Hour:** Displays the rate at which the incoming REST calls are processed per hour. This reflects the incoming traffic to the OSM Gateway. This detail panel displays the consolidated view of successful and failed requests. It also displays the failed request rate per hour. You can use this to check for bottlenecks during the processing of a REST request.
- **Outgoing REST Requests per Hour:** Displays the rate at which the outgoing messages are processed per hour. This reflects the outgoing traffic from the OSM Gateway. This detail panel displays the consolidated view of successful and failed outgoing REST requests. It also displays the failed outgoing REST request rate per hour. Any spike in failed REST request rate indicates an issue in delivering the events to the target system.
- **Average Incoming REST Request Duration:** Displays the average time taken for processing the incoming REST request to OSM. It reflects the efficiency of request processing of OSM.
- **CPU Load:** Displays the current usage of CPU.
- **Heap Usage:** Displays the amount of heap space used out of the available heap space.

About OCA Publish Operations

You can use the following global filters in the sample dashboard:

- **Project:** Displays the available projects and the default project. Multi-selection is not supported.
- **Instance:** Displays filter values based on the selected project.
- **Pod:** Displays filter values based on the selected instance.
- **Initiative ID:** Displays the initiative ID of the publish operation.
- **Operation State:** Displays the publish operation state (for example, *Completed* or *Failed*) for the selected initiative.
- **Publish Message:** Displays the publish operation success or failure message for the selected initiative.
- **Commit Type:** Displays the database commit type (for example, *Deploy* or *Activate*) for the publish operation on the selected initiative.

You can use the following detail panels in the sample dashboard:

- **Total Publish Initiatives:** Displays the total number of publish requests processed. This panel provides a consolidated view of successful and failed requests. Use this panel to identify potential bottlenecks during REST request processing.
- **Average Publish Duration:** Displays the average time (in seconds) taken to process publish requests. This panel also shows the average time taken by individual OCA pods to process requests, indicating the efficiency of request processing across pods.
- **Average Commit Duration:** Displays the average time (in seconds) taken for database commits against publish requests. This panel also shows the average time taken by individual OCA pods to perform database commits, indicating the efficiency of database operations across pods.
- **CPU Load:** Displays the current CPU usage.
- **Heap Usage:** Displays the amount of heap space used compared to the available heap space.

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 8-7 Order Metrics Exposed via Prometheus APIs

Name	Type	Help Text	Notes
osm_orders_total	Counter	Counter for the number of orders in the Created state.	N/A
osm_orders_completed_total	Counter	Counter for the number of orders in the Completed state.	N/A
osm_orders_failed_total	Counter	Counter for the number of orders in the Failed state.	N/A
osm_orders_cancelled_total	Counter	Counter for the number of orders in the Canceled state.	N/A
osm_orders_aborted_total	Counter	Counter for the number of orders in the Aborted state.	N/A
osm_orders_in_progress	Gauge	Gauge for the number of orders currently in the In Progress state.	N/A
osm_order_items	Histogram	Histogram that tracks the number of order items in an order with buckets for 0, 10, 25, 50, 100, 250, 1000, and 5000 order items.	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.
osm_order_cache_entries	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	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 8-8 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_createedByOperator	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_modelImageDomainZipHash	md5.3d1b561138f3ae3238d67a023771cf45.m d5	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 8-9 Task Metrics Captured for Manual or Automated Task Types Only

Name	Type	Help Text
osm_tasks_total	Counter	Counter for the number of Tasks Created
osm_tasks_completed_total	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 8-10 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

TMF and System Interaction Messaging Metrics

The following table lists the OSM Gateway metrics in additions to Helidon standard metrics that are exposed via Prometheus APIs.

Table 8-11 TMF and System Interaction Messaging Metrics

Name	Type	Label Name	Description
osmgw_outgoing_messages_pending	Gauge	targetSystem, cn_project, cn_instance, pod_name, api_name, api_version, and spec_usage_type.	Count of outgoing messages waiting to be sent.
osmgw_incoming_messages_total	Counter	cn_project, cn_instance, pod_name, api_name, api_version, spec_usage_type, managed_server_names, and status. "status" has the following values: • successful • failed	Count of incoming messages processed.
osmgw_outgoing_messages_processed_total	Counter	targetSystem, cn_project, cn_instance, pod_name, api_name, api_version, spec_usage_type, and status. "status" has the following values: • successful.message.without.retry • expired.message.with.retry • expired.message.without.retry	Count of outgoing messages processed.
osmgw_incoming_rest_duration	SimpleTimer	cn_project, cn_instance, pod_name, api_name, api_version, spec_usage_type, managed_server_names, and status. "status" has the following values: • successful • failed	Cumulative time taken to process incoming REST messages.

Table 8-11 (Cont.) TMF and System Interaction Messaging Metrics

Name	Type	Label Name	Description
cpu_processCpuLoad_percent	Gauge	cn_project, cn_instance, pod_name, and managed_server_name.	Displays the recent CPU usage for the Java Virtual Machine process.

Labels for TMF and System Interaction Messaging Metrics

The following table lists the labels for TMF and System Interaction Messaging Metrics

Table 8-12 Labels for TMF and System Interaction Messaging Metrics

Label Name	Sample Value	Notes	Source of the Label
cn_project	quick	Project Namespace	Gateway Metric Label Name/Value
cn_instance	sr	Instance name within the project	Gateway Metric Label Name/Value
pod_name	quick-sr-gateway-0	Name of the OSM Gateway Pod	Gateway Metric Label Name/Value
api_name	serviceOrdering	TMF API name	Gateway Metric Label Name/Value
api_version	4.1.0	TMF API version	Gateway Metric Label Name/Value
spec_usage_type	Hosted	Roles of TMF API in OSM: either hosted or SI	Gateway Metric Label Name/Value
managed_server_names	ms1	Name of the Managed Server pinned to Gateway pod	Gateway Metric Label Name/Value
status	successful	Status of REST request or SI message	Gateway Metric Label Name/Value
targetSystem	Billing	Downstream or upstream system	Gateway Metric Label Name/Value

Helidon provides the standard Helidon base and vendor metrics for OSM Gateway and RTUX microservices.

The response for the metrics endpoint contains the standard Helidon application and vendor metrics. The following sample shows some of the metrics in the response:

```
# TYPE classloader_loadedClasses_count gauge
# HELP classloader_loadedClasses_count Displays the number of classes that are currently
loaded in the Java virtual machine.
classloader_loadedClasses_count 9667
# TYPE classloader_loadedClasses_total counter
# HELP classloader_loadedClasses_total Displays the total number of classes that have
been loaded since the Java virtual machine has started execution.
classloader_loadedClasses_total 9672
# TYPE classloader_unloadedClasses_total counter
# HELP classloader_unloadedClasses_total Displays the total number of classes unloaded
since the Java virtual machine has started execution.
classloader_unloadedClasses_total 5
# TYPE cpu_availableProcessors gauge
# HELP cpu_availableProcessors Displays the number of processors available to the Java
virtual machine. This value may change during a particular invocation of the virtual
machine.
```

```

cpu_availableProcessors 1
# TYPE cpu_systemLoadAverage gauge
# HELP cpu_systemLoadAverage Displays the system load average for the last minute. The
system load average is the sum of the number of
runnable entities queued to the available processors and the number of runnable entities
running on the available processors averaged over a
period of time. The way in which the load average is calculated is operating system
specific but is typically a damped timedependent average. If the load average is not
available, a negative value is
displayed. This attribute is designed to provide a hint about the system load and may be
queried frequently. The load average may be unavailable on some platforms where it is
expensive to implement this method.
cpu_systemLoadAverage 0.92
# TYPE gc_time_seconds_total gauge
# HELP gc_time_seconds_total Displays the approximate accumulated collection elapsed
time in milliseconds. This attribute displays -1 if the collection elapsed time is
undefined for this collector. The Java
virtual machine implementation may use a high resolution timer to measure the elapsed
time. This attribute may display the same value even if the collection count has been
incremented if the collection elapsed time is very short.
gc_time_seconds_total{name="Copy"} 0.009
gc_time_seconds_total{name="MarkSweepCompact"} 0.212
# TYPE gc_total counter
# HELP gc_total Displays the total number of collections that have occurred. This
attribute lists -1 if the collection count is undefined for this collector.
gc_total{name="Copy"} 1
gc_total{name="MarkSweepCompact"} 2

```

For more details about metrics and about Helidon monitoring, see the following:

- Helidon MP Metrics Guide in the Helidon MP documentation at: <https://helidon.io/docs/v3/##mp/metrics/metrics>
- MicroProfile Metrics specification on the GitHub web site: <https://github.com/eclipse/microprofile-metrics/releases/3.0>

OCA Publish Operations Metrics

The following table lists the OCA metrics in addition to Helidon standard metrics that are exposed via Prometheus APIs.

Table 8-13 OCA Metrics Exposed via Prometheus APIs

Name	Type	Label Name	Description
publish_initative_total	Counter	cn_instance, cn_project, pod_name, initiativeId, cartridge, operation_state, operation_message <i>operation_state has values: completed/failed</i> <i>operation_message will have success message for 'completed' operation_state, and failed message with type of failure for 'failed' operation_state.</i>	Counts the number of HTTP requests in each operation state category (completed, failed)

Table 8-13 (Cont.) OCA Metrics Exposed via Prometheus APIs

Name	Type	Label Name	Description
publish_initiative_duration	Summary	cn_instance, cn_project, pod_name, initiativeId, cartridge, operation_state, operation_message <i>operation_state has values: completed/failed</i> <i>operation_message will have success message for 'completed' operation_state, and failed message with type of failure for 'failed' operation_state.</i>	Computes the REST request execution duration in each operation state category (completed, failed)
commit_initiative	Summary	cn_instance, cn_project, pod_name, initiativeId, cartridge, operation_type, status <i>operation_type has values: Deploy/Activate/Undeploy</i> <i>status has the values: successful/failed</i>	Computes the commit duration in each operation category (Deploy, Activate, Undeploy). Here, the metrics summary will provide other details like commit count, sum of all commit duration. This can be helpful in tracking the number of commits.

Labels for OCA Publish Operations Metrics

The following table lists the labels for OCA Publish Operations Metrics:

Table 8-14 Labels for OCA Publish Operations Metrics

Label Name	Sample Value	Notes	Source of the Label
cn_project	sr	Project Namespace	OCA Metric Label Name/ Value
cn_instance	quick	Instance name within the project	OCA Metric Label Name/ Value
pod_name	sr-quick-oca-8b955fb68-cxszc	Name of the OCA Pod	OCA Metric Label Name/ Value
initiativeId	103001	Initiative ID of the Publish Operation	OCA Metric Label Name/ Value
cartridge	TMF_PO_B2C_Solution_1.0.0.0	CPAR Name and Version	OCA Metric Label Name/ Value
operation_state	completed	State of Publish REST request	OCA Metric Label Name/ Value
operation_message	Publish Operation Successful	Publish request success or fail message	OCA Metric Label Name/ Value
operation_type	Activate	Database commit type	OCA Metric Label Name/ Value

Managing WebLogic Monitoring Exporter (WME) Metrics

OSM cloud native deployment provides an integrated Prometheus-compatible exporter of metrics from all WebLogic Server pods using WebLogic Monitoring Exporter (WME). WME runs as a sidecar container within each of the WebLogic Server pods (admin server and

managed servers). This section describes a sample integration for OSM pods that run WebLogic Server.

OSM cloud native also provides a Grafana sample dashboard that can be used to visualize OSM and WebLogic metrics from a Prometheus data source. See *OSM Compatibility Matrix* for the supported versions of WME.

Note

If you have configured WME as a custom sidecar in 7.4.1 and wish to upgrade to this release, remove your custom sidecar configuration and perform the tasks described in this section. If you wish to retain your custom WME sidecar, while this is not recommended, you may do so provided you disable the integrated WME. The integrated WME is enabled by default.

The following topics describe a sample integration:

- [Enabling WebLogic Monitoring Exporter \(WME\)](#)
- [Configuring the Prometheus Scrape Job for WME Metrics](#)
- [Viewing WebLogic Monitoring Exporter Metrics in Grafana](#)

Enabling WebLogic Monitoring Exporter (WME)

To enable WebLogic Monitoring Exporter:

1. In the instance specification, set `weblogicMonitoringExporter` to **true**.

```
# Set to true if weblogic monitoring exporter is required
weblogicMonitoringExporter:
  enabled: true #or false to disable
```

By default, this parameter is set to **true**, which creates a sidecar container for WME using the default image for the version of WebLogic Operator in use.

2. (Optional) Specify the WebLogic Operator image yourself by providing the `image` name and `imagePullPolicy` in the project specification:

Note

If you do not provide an image name, the default image pulled is `ghcr.io/oracle/weblogic-monitoring-exporter:tag`, where *tag* depends on the version of WebLogic Operator monitoring this project namespace and is selected by the Operator automatically.

```
# WebLogic Monitoring Exporter
wme:
  image: URL_for_weblogic-monitoring-exporter
  imagePullPolicy: policy
```

For `imagePullPolicy`, specify any one of the following values:

- `IfNotPresent`

- Always
- Never

The following snippet shows an example:

```
# WebLogic Monitoring Exporter
wme:
  image: ghcr.io/oracle/weblogic-monitoring-exporter:2.0.5
  imagePullPolicy: IfNotPresent
```

3. Create or upgrade your instance as usual, using **create-instance.sh** or **upgrade-instance.sh** respectively.

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.

Proxy Configuration in Microservices

OSM communicates with specific external systems for authentication and authorization via OIDC or for System Interactions as target systems via REST. This traffic can go over http or https. It is possible that some or all of these destinations require the use of a proxy server.

A single proxy server can be defined for this purpose, targeting either http or https. You can exclude specific target IPs or hostnames from the proxy. The proxy server configuration does not support credentials to authenticate with the proxy itself. You can configure the proxy in the project specification file:

```
# Proxy configuration
proxy: {} # This empty declaration should be removed if adding items here.
#proxy:
# host: "" # Mandatory Proxy server hostname or IP
# port: "80" # Proxy server port
# protocol: "http" # Protocol to use for the proxy (http/https) default http
# no_proxy: # List of hosts/IPs that should bypass the proxy
#   - "190.190.*.*"
#   - "localhost"
#   - "127.0.0.1"
#   - ".oracle.com"
```

Automating OSM Cloud Native Operations for Continuous Delivery

OSM cloud native provides flexible options to automate its operations for continuous delivery. You can choose to build custom pipelines by invoking OSM cloud native toolkit scripts directly, using your preferred tools and a backing store for configuration management. Alternatively, for organizations adopting GitOps practices, a streamlined integration is available by leveraging Flux-CD, enabling declarative management of deployments through Git as the source of truth.

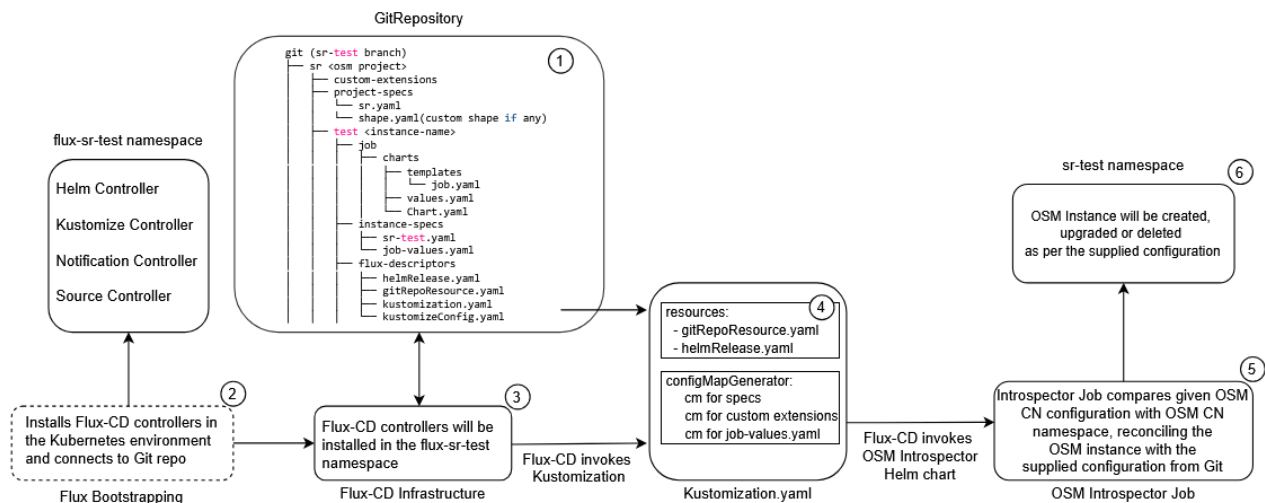
GitOps allows you to manage and deploy applications by keeping all configuration files and settings in a Git repository. For more details on GitOps, see <https://about.gitlab.com/topics/gitops/>.

Flux-CD is a tool that helps keep your Kubernetes environment in sync with the files you store in Git. It constantly compares your running environment with your Git repository, and if something does not match, it automatically makes the necessary updates to keep them aligned. For more details on Flux-CD, see <https://fluxcd.io/flux/get-started/>.

About Continuous Delivery Mechanism

Continuous Delivery (CD) is a process that automatically gets applications ready for release every time a change is made. With CD, new updates are automatically tested and prepared, so that new features or fixes can be delivered safely and automatically. For more details on Continuous Delivery, see <https://about.gitlab.com/topics/continuous-delivery/>.

Figure 9-1 Continuous Delivery Mechanism



In the image above:

- 1. Git Repository:** All configuration for Flux-CD, Kustomize and OSM CN is stored in a git repository.

2. **Flux Bootstrapping:** Flux-CD is instructed to monitor and process this using a bootstrap command.
3. **Flux-CD Infrastructure:** This reads the configuration and installs Flux-CD controllers in a flux-CD namespace for this OSM instance.
4. **Kustomization.yaml:** Using one of these controllers - the Kustomization controller, Flux-CD generates resources in the OSM CN namespace to hold configuration from git that will be required at runtime.
5. **OSM Introspector Job:** Finally, the Flux-CD Helm controller invokes the OSM CN Introspector Helm chart. The OSM CN Introspector Job then compares the desired target state configuration (as relayed from git) with the actual state of the instance. It calculates the operations required to reconcile the two states and runs them.
6. **sr-test namespace:** This results in the OSM cloud native instance being brought in line with the configuration.

Prerequisites for Creating an OSM Instance Using the Introspector Job

Before creating an OSM cloud native instance using the Introspector Job, ensure the following requirements are met:

1. **Create Required Secrets and pdb (and pvc if you are using it):**
All Kubernetes secrets, pdb, and pvc (if you are using it) required for the OSM cloud native instance need to be created before you run the Introspector Job.
2. **Namespace Registration:**
Register the required namespace with one or both of the following as needed:
 - WebLogic Operator
 - Ingress Controller
3. **Role-Based Access Control (RBAC):**
Create a Service Account for OSM Introspector with the RBAC specified in the OSM cloud native toolkit samples. If you are using an existing Service Account, ensure it has the required RBAC.

Git Strategy for OSM Instances

A structured Git strategy is recommended to ensure a robust and reliable deployment process for OSM instances (such as prod and test). This section describes a sample branching approach and directory structure. OSM cloud native has been validated against this strategy. The goal is to use the main branch as the “source of truth,” which holds only the latest, production-ready code and shared configurations. You can manage instance specific deployment and customizations in dedicated branches.

The key elements of this strategy are:

- It organizes the directory structure which separates common content (such as project specifications, custom model extensions, and custom shapes) from instance specific configurations (such as instance specifications and job-values.yaml).
- The main branch serves as a reference and shows the complete set of content for all instances, but does not drive any changes to the Kubernetes environment.

- For each OSM instance, a dedicated branch is created from `main`. Changes to these branches are directly reflected in the associated Kubernetes environment for that instance.
- You can develop and deliver changes on the instance branch, and merge them back into `main`, while keeping the instance branch active for ongoing updates.
- To incorporate updates from common configurations, you can rebase the instance branch from the `main` branch.
- When an OSM instance is deleted, its related branch can also be removed. However, instance details continue to live inert in the `main` branch for reference, future resurrection or future cleanup.

To setup a dedicated branch for an OSM Instance:

1. Create a new instance branch from the `main` branch:

```
# Pull the latest changes to the main branch
git pull origin main

# Checkout a dedicated branch for 'test' instance from main branch
git checkout -b <project>-<instance> origin/main

eg: git checkout -b sr-quick origin/main (where "sr" and "quick"
represents OSM project and instance name respectively)
```

This ensures that the new branch includes the latest common code and configuration.

2. Switch to the instance branch: Set up directory structure and instance specific configurations in the instance branch.
3. Push your changes and Bootstrap:
 - a. Commit and push updates to your remote instance branch. It will trigger the OSM cloud native instance creation/upgradation for that instance environment.
 - b. Bootstrap your instance branch (eg: `sr-quick`). This needs to be done only once for a given branch, because after that Flux-CD will continue to monitor for ongoing changes.
4. Deliver to main: Deliver changes back into `main` branch, while keeping the instance branch active for ongoing updates.

The following example shows how to create a test environment, for an OSM cloud native project `sr`.

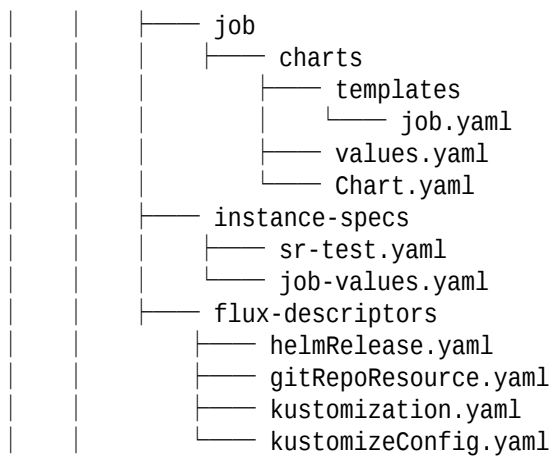
To set up a dedicated branch for the test instance:

1. Create a branch: **sr-test**

- `git pull origin main`
- `git checkout -b sr-test origin/main`

2. Switch to the **sr-test** branch, setup up directory structure and instance specific changes in the **sr-test** branch as required.

```
git (in sr-test branch)
├── sr <project-name>
│   ├── custom-extensions
│   ├── project-specs
│   │   ├── sr.yaml
│   │   └── shape.yaml(custom shape if any)
└── test <instance-name>
```



3. Push your changes and Bootstrap:

- Commit and push changes to the **sr-test** branch, this will trigger the OSM cloud native instance creation/upgradation for the test environment.
- Bootstrap the **sr-test** instance.

4. Deliver changes back into main branch, and keep the **sr-test** instance branch active for ongoing updates.

The following table lists the directory names and their contents:

Table 9-1 Directory Structure and Contents

Directory	Contents
osm-project	Top-level directory named after your OSM project (for example, sr)
osm-project/custom-extensions	Provide all OSM custom extension files
osm-project/project-specs	Project specification file and custom shape file (if required)
osm-project/instance-name	Subdirectory for each OSM instance
osm-project/instance-name/job	OSM cloud native toolkit Introspector Job Helm Chart. Copy all contents from the job directory present in OSM cloud native toolkit.
osm-project/instance-name/instance-specs	Instance specification file and job-values.yaml , from the OSM cloud native toolkit samples directory.
osm-project/instance-name/flux-descriptors	Flux-CD resources: HelmRelease , GitRepository , Kustomization , kustomizeConfig.yaml (with nameReference). Copy all files from the OSM cloud native toolkit samples/flux-descriptors directory

The following example shows how to create a production instance, for an OSM cloud native project **sr**.

To create a production instance:

1. Create a branch: **sr-prod**

- `git pull origin main`

- `git checkout -b sr-prod origin/main`
2. Switch to the **sr-prod** branch, setup up the directory structure and the instance specific changes in the **sr-prod** branch as required.

```
git (in sr-prod branch)
├── sr
│   ├── custom-extensions
│   ├── project-specs
│   │   ├── sr.yaml
│   │   └── shape.yaml (custom shape if any)
│   └── prod (newly added in this step)
│       ├── job
│       │   ├── charts
│       │   │   ├── templates
│       │   │   │   └── job.yaml
│       │   │   ├── values.yaml
│       │   │   └── Chart.yaml
│       ├── instance-specs
│       │   ├── sr-prod.yaml
│       │   └── job-values.yaml
│       ├── flux-descriptors
│       │   ├── helmRelease.yaml
│       │   ├── gitRepoResource.yaml
│       │   ├── kustomization.yaml
│       │   └── kustomizeConfig.yaml
│       └── test
│           ├── job
│           │   ├── charts
│           │   │   ├── templates
│           │   │   │   └── job.yaml
│           │   │   ├── values.yaml
│           │   │   └── Chart.yaml
│           ├── instance-specs
│           │   ├── sr-test.yaml
│           │   └── job-values.yaml
│           ├── flux-descriptors
│           │   ├── helmRelease.yaml
│           │   ├── gitRepoResource.yaml
│           │   ├── kustomization.yaml
│           │   └── kustomizeConfig.yaml
```

3. Push changes and Bootstrap:
 - Commit and push changes to the **sr-prod** branch, this will trigger the OSM cloud native instance creation/upgradation for the production environment.
 - Bootstrap the **sr-prod** instance.
4. Deliver changes back into main branch, and keep the **sr-prod** instance branch active for ongoing updates.

OSM Cloud Native Toolkit Introspector Job Helm Chart

The Introspector Job Helm chart provisions a Kubernetes Job that runs the OSM cloud native toolkit Introspector script.

The introspector script automates and manages the lifecycle of OSM instance. It performs the following tasks sequentially:

- Delete or restart the OSM Instance. This is done only if it is specifically requested.
- Determine if the operation will trigger an outage.
- Validate if the outage is allowed.
- Shutdown pods if the outage is required and permitted.
- Install or skip the RCU schema as per the requirements.
- Install or upgrade the OSM schema as per the requirements.
- Deploy or sync OSM cartridges.
- Create or upgrade the OSM cloud native instance. This depends on the configuration.
- Create or upgrade ingress.

Note

All customization and operational parameters for the Introspector Job are defined externally in the **job-values.yaml** configuration file. The Helm chart itself must not be modified.

Creating an OSM Cloud Native Instance Using Flux-CD

This section describes how to create an OSM cloud native instance, expanding on the configuration changes required in git and the setup of Flux-CD. This can only be done after you have set up the basic file system for the instance on the instance's branch.

Setup Flux-descriptors Configuration

The following artifacts are configurations of Flux-CD and contain resources that will create and manage the OSM cloud native instance. OSM cloud native uses three Flux-CD resources and a nameReference:

- Kustomization
- HelmRelease
- GitRepository

Kustomization is the entry point that triggers the creation of the other two resources (helmRelease and gitRepository) along with configMaps. The nameReference is used to override the helm chart values.

The samples that OSM cloud native ships have placeholders that need to be updated to your specific details. The following table lists the placeholders that you need to update in all the files you copy from the OSM cloud native toolkit.

Table 9-2 Placeholders in the Cloud Native Toolkit

Placeholder	Value
<project>	Name of your OSM project, for example <code>sr</code> .
<instance>	Name of your OSM instance, for example <code>quick</code> .

Table 9-2 (Cont.) Placeholders in the Cloud Native Toolkit

Placeholder	Value
<git-repository-url>	URL of the Git repository that Flux-CD will watch.
<branch-name>	Branch that Flux-CD will watch.
<git-repo-secret>	Secret to access the Git repository referenced via <git-repository-url>. For more information, see Git Access Secret .
<PATH-TO-JOB>	Location of the Job directory in the provided Git repository source. In the branching structure described previously, this would be <project>/<instance>

Kustomization Resource

The Kustomization resource acts as the primary entry point for Flux-CD. When Flux-CD detects a Kustomization resource inside the **flux-descriptors** directory, it orchestrates the creation and management of related resources defined within it. The Kustomization resource creates two resources, gitRepository and helmRelease. It also creates configuration maps to hold the configuration derived from Git to make it available to the OSM Introspector Job.

If you are using custom extensions, uncomment the **configMap** specific to Custom Extensions. This will be disabled by default. For example, **sr-quick-customs-cm** and you will need to mention the files provided in the **custom-extensions** directory in this **configMap**. For more information on adding custom extensions, refer to [Custom Extensions](#).

The following is a sample kustomization resource:

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: sr
resources:
  - gitRepoResource.yaml
  - helmRelease.yaml
configMapGenerator:
  ## configMap for specs
  - name: sr-quick-specs-cm
    ## add all specs and shape files here. This will be mounted in /osm-specs
    folder in job container
    files:
      - ../../project-specs/sr.yaml
      - ../instance-specs/sr-quick.yaml
      - ../../project-specs/shape.yaml
    ## configMap for job-values.yaml
  - name: sr-quick-helm-values-cm
    files:
      - values.yaml=../instance-specs/job-values.yaml
generatorOptions:
  disableNameSuffixHash: true
configurations:
  - kustomizeConfig.yaml
```

GitRepository Resource

The GitRepository resource defines the Git project that Flux-CD continuously monitors. It specifies the repository URL, the branch, and the directory path that Flux-CD will watch to monitor configuration changes.

The following is a sample gitRepository resource:

```
apiVersion: source.toolkit.fluxcd.io/v1beta2
kind: GitRepository
metadata:
  name: sr-quick-flux-cntk-gr
  namespace: sr
spec:
  interval: 1m
  # Git-Repository URL for Flux-cd to watch
  url: https://<git-repo-url>
  ref:
    # Specify the branch to watch
    branch: sr-quick
  secretRef:
    # Secret to access the above mention Git Repository
    name: gitlab-creds
  ignore: |-
    #Exclude all
    /*
    #Include specific directory to watch
    !/sr/custom-extensions/
    !/sr/project-specs/
    !/sr/quick/
```

HelmRelease Resource

The HelmRelease resource defines the deployment of Helm charts within the cluster and ensures that they remain synchronized with the desired state stored in Git. It leverages the GitRepository resource as the source for pulling Helm charts and applies updates automatically when it detects changes.

The following is a sample HelmRelease resource:

```
apiVersion: helm.toolkit.fluxcd.io/v2beta1
kind: HelmRelease
metadata:
  name: cntk-flux-hr
  namespace: sr
spec:
  interval: 1m
  timeout: 60m
  releaseName: sr-quick-osmcn-release
  chart:
    spec:
      chart: sr/quick/job/charts
      reconcileStrategy: Revision
      sourceRef:
```

```
kind: GitRepository
name: sr-quick-flux-cntk-gr
valuesFrom:
- kind: ConfigMap
  name: sr-quick-helm-values-cm
```

Setup Job-Values Configuration

The Job-Values configuration provides a mechanism to define and customize parameters required by the OSM cloud native Introspector Job. The following properties can be defined in the **job-values.yaml** file to configure the Job:

- **introspectorJobVersion:** This version is a suffix appended to the job name to distinguish between different iterations of the job. Every time there is a change to the specification or configuration, this version must be incremented or changed.
- **project:** This specifies the name of the project.
- **instance:** This specifies the name of the instance.
- **image:** This specifies the container image that needs to be used for the Introspector Job.
- **imagePullPolicy:** This defines how Kubernetes pulls the container image for the Introspector Job's pods.
- **imagePullSecret: (Optional)** This provides the credentials to pull container images from secured registries.
- **customModel:** This field is used to enable or disable the Custom Model Extensions. By default it is set to **False** (disable).
- **customFiles:** This field is used to enable or disable the Custom Files. By default it is set to **False** (disable).
- **serviceAccountName: (Optional)** This specifies a custom service account. If you do not mention a custom service account, a default service account will be used.

Note

Oracle recommends that you do not use the default service account but that you create a custom service account.

- **targetState:** This specifies the desired end state of the OSM instance as **running** or **notRunning**. If you set it to **notRunning**, the instance will not be created if it is a new instance, or it will get deleted if it is an existing instance.
- **action:** This applies operational actions on the OSM instance. These actions are only performed if the job version listed under the action matches the introspectorJobVersion. At present, the **restart** operation is controlled using **action**.
- **tolerance:** This sets the deployment tolerance for various runtime situations. Tolerances are applied if their jobVersion matches the introspectorJobVersion. The tolerance for OSM outage is controlled here. By default, any change that will result in an OSM outage will not be allowed to proceed, and will return a failure from the Introspector Job. If it is acceptable for a change to cause an outage, the fullOutage tolerance can be set to **true**. OSM production systems have minimum cluster size of 2 or more. For development systems, which tend to have cluster size of 1, you can either setup outage tolerance on each change that requires pod restart, or simply set the jobVersion value to **all**.

 Warning

Setting the jobVersion to **all** is not recommended for production systems as that bypasses the tolerance safety net.

The following is a sample **job-values.yaml** file:

```
# This version need to be changed to rerun Introspector Job
introspectorJobVersion: v1

# Provide project name in place of <project-name>
project: sr

# Provide instance name in place of <project-name>
instance: quick

# Provide the OSM CNTK image
image: osm-cntk:8.0.0

# Provide when the container image should be pulled from the container
registry
# Valid values (Always, IfNotPresent, Never)
imagePullPolicy: IfNotPresent

# Provide authentication credentials for pulling container images,
# Optional field, uncomment to specify the imagePullSecret
#imagePullSecret: <image-pull-secret-name>

# Enable if there are any OSM CN model extensions, by default it will be
disabled.
customModel: false

# Enable if there are customFiles provided in the project specification.
customFiles: false

# Provide service account name,
# Optional field, if not provided 'default' serviceAccount will be used,
# Uncomment to specify custom serviceAccount.
#serviceAccountName: <service-account-name>

# To manage OSM CN instance create/upgrade OR delete.
# running : OSM Introspector Job will 'create' or 'upgrade' the instance.
# notRunning : OSM Introspector Job will 'delete' the instance.
targetState: running

# Actions that can be applied based on jobVersion.
action:
# OSM Introspector Job will restart or skip the restart, based on the
jobVersion value,
# if jobVersion is same as introspectorJobVersion, restart will be applied,
# otherwise if jobVersion is different from introspectorJobVersion, restart
will be skipped.
# valid values for Restart.Type are: full, admin, ms
# * full : Restarts the whole instance (rolling restart)
# * admin : Restarts the WebLogic Admin Server only
```

```
# * ms      : Restarts all the Managed Servers (rolling restart)
restart:
  type: full
  appliesTo:
    jobVersion: v0

# Deployment tolerance that can be applied based on jobVersion.
tolerance:
# OSM Introspector Job will upgrade or skip the upgrade based on fullOutage
allowed,
# This will applies to provided jobVersion only,
# if jobVersion is different from introspectorJobVersion, fullOutage will not
be allowed.
# if jobVersion value is "all", this tolerance will be applicable for all
introspectorJobVersion.
# if fullOutage is allowed : Upgrade will be done (with or without full
outage).
# if fullOutage is not allowed : Upgrade will fail if full outage is
required, else upgrade will be done.
fullOutage:
  allowed: false
  appliesTo:
    jobVersion: v0
```

Note

A sample **job-values.yaml** file is provided in the OSM cloud native toolkit package under **samples/job-values.yaml**.

Triggering CD for a New Instance

The Flux-CD bootstrap command links Flux-CD to the namespace, the branch and path in your Git repository. This allows automated deployment and lifecycle management of your OSM cloud native instances. Flux-CD also requires its bootstrap command to be run by a user who has cluster administration rights on the target Kubernetes cluster.

Oracle recommends using the main branch as the source of truth for maintaining the latest code. Do not use this branch directly to create instances. For each environment (for example, prod, dev, or test), create a separate branch cloned from the main branch. Each branch will contain the required **flux-descriptor** path to bootstrap and create its own OSM cloud native instance. **token-auth** is required for Flux-CD to interact with the Git repository, and tokens need be generated from an appropriate Git repository account.

The following are the minimum parameters needed for OSM cloud native use:

- **-n flux-<project>-<instance>**: Assigns a unique namespace for the Flux-CD components.
- **--url**: The Git repository URL.
- **--branch**: The branch associated with the environment/instance.
- **--path**: The directory path (within the branch) containing the **flux-descriptor** manifests.

- **--token-auth**: Enables authentication using a git access token.

```
bash-4.4$ flux bootstrap -n flux-<project>-<instance> git \
  --url=<Provide your Git repository URL> \
  --branch=<Provide your branch here> \
  --path=<project>/<instance>/flux-descriptors \
  --token-auth
```

After bootstrapping, Flux-CD will process the **kustomization.yaml** file located in the specified **flux-descriptor** directory. Flux-CD will create various housekeeping resources in the **flux-<project>-<instance>** namespace, as well as operational pods.

Making Changes to an OSM Instance

When modifying the configuration of an instance managed by Flux-CD, it is important to ensure that Flux-CD detects and reconciles the changes correctly.

To update the configuration of an instance:

1. **Modify Configuration**: Make the required changes in the configuration files (for example, the **job-values.yaml**, **project**, and **instance** specification). This can also include normal git operations like rebasing from main or cherry-picking a commit from elsewhere. If it is okay for this instance to experience an outage, then set the outage tolerance as well.
2. **Increment the Job Version**: Update the **introspectorJobVersion** in the **job-values.yaml** file. For example,

```
introspectorJobVersion: v2 # previously v1
```

3. **Commit and Push as One Transaction**: All the configuration changes and the **introspectorJobVersion** updates must be committed and pushed in a single Git transaction. This ensures that the Flux-CD identifies the update as a new version.

Once the changes are pushed, Flux-CD will detect them. A new Job will be created with the updated version. The Job's name will automatically include a suffix corresponding to the specified **introspectorJobVersion**, for example, **sr-quick-osmops-v2**. The versioned job name provides clear traceability of configuration changes applied over time.

Adding Custom Content

This section explains how you can configure the custom content supported by OSM cloud native.

Custom Extensions

If you need to include custom extensions as part of your OSM cloud native deployment, follow these steps in the **<instance>** branch:

1. Add all the custom extensions files under the **custom-extensions** directory present under **<project>/custom-extensions**. Refer to [Git Strategy for OSM Instances](#) for more information about the directory structure.

```
git (dev branch)
|—— project
|   |—— custom-extensions # Specify all the .tpl files under this
```

```

directory.
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
...

```

2. Update the Kustomization resource with all the custom extension files:

- a. Open the Kustomization resource present in the **flux-descriptors** directory present under **<project>/<instance>/flux-descriptors**. Refer to [Git Strategy for OSM Instances](#) for more information about the directory structure.
- b. In Kustomization resource under the **configMapGenerator** tag, a configMap is available that has been configured for custom extensions. Uncomment that section.

```

apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
...
configMapGenerator:
...
  ## configMap for custom extensions
  ## Please uncomment this section and add all custom extension tpl
  files.
  #- name: <project>-<instance>-customs-cm
  # ## add all custom extensions files here. This will be mounted in /
  # osm-customs folder in job container
  # files:
  #   - ../../custom-extensions/_custom-jms-support.tpl
  #   - ../../custom-extensions/_custom-domain-model.tpl
...

```

- c. Replace the placeholders **<project>** and **<instance>** in the uncommented section with real values.

```

configMapGenerator:
...
  # configMap for custom extensions
  # Please uncomment this section and add all custom extension tpl
  files.
  - name: sr-quick-customs-cm
    ## add all custom extensions files here. This will be mounted in /
    osm-customs folder in job container
    files:
...

```

- d. After setting up the name, under the files tag, you need to specify each custom extension tpl file path. The path will always follow the format **../../custom-extensions/<name-of-custom-extension>.tpl**. For example, if the custom extension name is **_custom-domain-model.tpl** then you need to add an entry **../../custom-extensions/_custom-domain-model.tpl** under the files tag.

```

configMapGenerator:
  # configMap for custom extensions
  # Please uncomment this section and add all custom extension tpl
  files.
  - name: sr-quick-customs-cm
    ## add all custom extensions files here. This will be mounted in /
    osm-customs folder in job container

```

```
files:
  - ../../custom-extensions/_custom-domain-model.tpl
  - ../../custom-extensions/_custom-jms-support.tpl
```

- e. The following is a sample Kustomization resource with custom model extensions:

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: sr
resources:
  - gitRepoResource.yaml
  - helmRelease.yaml
configMapGenerator:
  ## configMap for specs
  - name: sr-quick-specs-cm
    ## add all specs and shape files here. This will be mounted in /osm-
    specs folder in job container
    files:
      - ../../project-specs/sr.yaml
      - ../instance-specs/sr-quick.yaml
    # configMap for custom extensions
    # Please uncomment this section and add all custom extension tpl
    files.
  - name: sr-quick-customs-cm
    ## add all custom extensions files here. This will be mounted in /
    osm-customs folder in job container
    files:
      - ../../custom-extensions/_custom-domain-model.tpl
      - ../../custom-extensions/_custom-jms-support.tpl
    ## configMap for job-values.yaml
  - name: sr-quick-helm-values-cm
    files:
      - values.yaml=../instance-specs/job-values.yaml
generatorOptions:
  disableNameSuffixHash: true
configurations:
  - kustomizeConfig.yaml
```

3. Enable custom extensions in the **job-values.yaml** file present under **<project>/<instance>/instance-specs/job-values.yaml**.

Note

By default custom extensions are disabled and set to **false**. To enable custom extensions, set it to **true**.

```
# Enable if there are any OSM CN model extensions, by default it will be
disabled.
customModel: true
```

4. Enable custom extension in the project specification. To do this, set the **enabled** tag as **true** and specify the model extension files names in the **wdtFiles list** element. Ensure this

list matches the list provided in **<project>/<instance>/flux-descriptors/kustomization.yaml**.

```
# Sample WDT extensions can be enabled here. When enabled is true, then
# _custom-domain-model.tpl needs to be un-commented. Custom template
# files can
# also be added.
custom:
  enabled: true
  application: false
  jdbc: false
  jms: true
  #wdtFiles: [] # This empty declaration should be removed if adding items
  here.
  wdtFiles:
    - _custom-jms-support.tpl
    - _custom-domain-model.tpl
```

Custom Files

If you need to include custom files as part of your OSM cloud native deployment, follow these steps in the **<instance>** branch:

1. Add the configured **custom-file-support.yaml** file under the **custom-extensions** directory present under **<project>/custom-extensions**. A sample file for this is present in the OSM cloud native toolkit. Refer to [Git Strategy for OSM Instances](#) for more information about the directory structure.

```
git (dev branch)
├── project
│   ├── custom-extensions
│   └── custom-file-support.yaml
...
```

2. Update the Kustomization resource with the **custom-file-support.yaml** file.
 - a. Open the Kustomization resource present in the **flux-descriptors** directory present under **<project>/<instance>/flux-descriptors**.
 - b. In Kustomization resource under the **configMapGenerator** tag, a configMap is available that has been configured for custom extensions. Uncomment that section and replace the placeholders **<project>** and **<instance>** in the uncommented section with real values.

```
configMapGenerator:
...
  # configMap for custom extensions
  # Please uncomment this section and add all custom extension tpl
  files.
  - name: sr-quick-customs-cm
    ## add all custom extensions files here. This will be mounted in /
    osm-customs folder in job container
    files:
...
```

- c. After setting up the name, you need to specify custom file path under the files tag as `../../custom-extensions/custom-file-support.yaml`.

```
configMapGenerator:
  # configMap for custom extensions
  - name: sr-quick-customs-cm
    ## add all custom extensions files here. This will be mounted in /
    osm-customs folder in job container
    files:
      - ../../custom-extensions/custom-file-support.yaml
```

3. Enable custom files in the `job-values.yaml` file present under `<project>/<instance>/instance-specs/job-values.yaml`.

Note

By default custom files are disabled and set to **false**. To enable custom file, set it to **true**.

```
# Enable if there are customFiles provided in the project specification.
customFiles: true
```

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"
```

Custom Shapes

You can use custom shapes as part of your OSM cloud native deployment. For more information on creating custom shapes, refer to [Creating Custom Shapes](#).

Once you have created the custom shape, follow these steps in the `<instance>` branch:

1. Under the `<project>/project-specs` directory add the file of the custom shape.
2. In the instance specification specify the name of the custom shape.

```
shape: <custom-shape-filename-without-extension>
```

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 workforce 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.

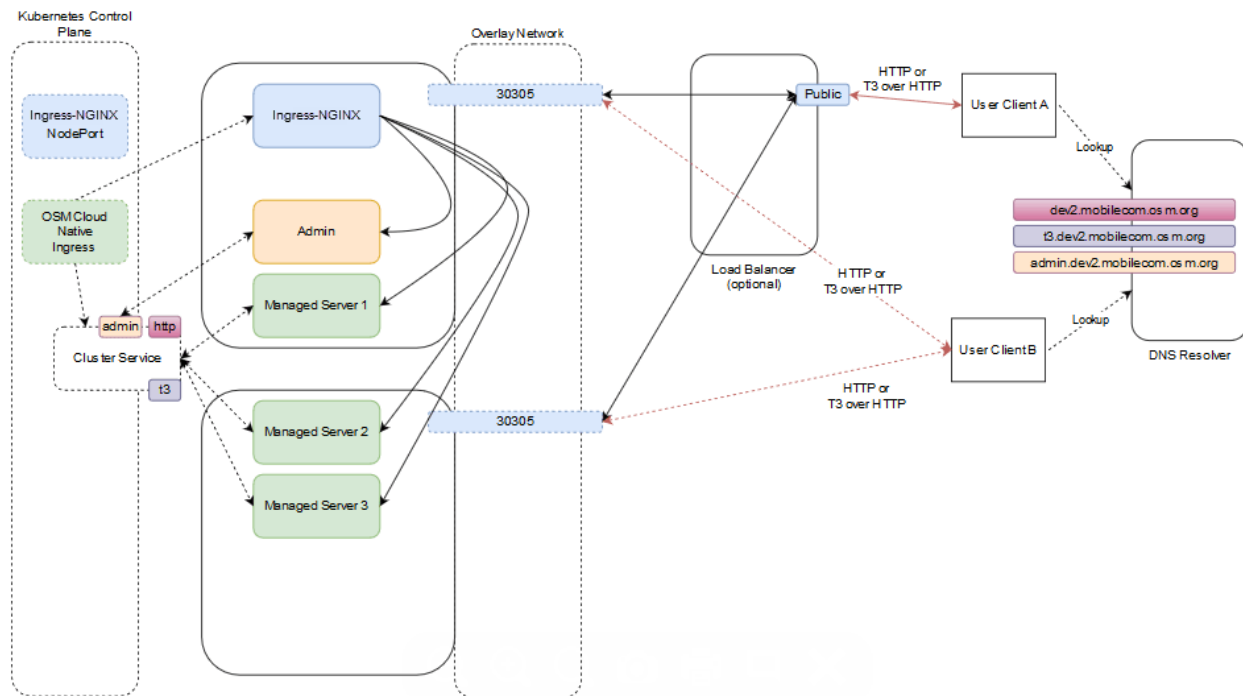
Connectivity With OSM Cloud Native

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 10-1 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 ingress-NGINX controller as a sample. ingress-NGINX 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 ingress-NGINX has already exposed ingress-NGINX 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 ingress-NGINX port (30305) is common to all OSM cloud native instances in the cluster, ingress-NGINX 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 ingress-NGINX 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 ingress-NGINX, 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 Remote console via **http://admin.dev2.mobilecom.osm.org/rconsole** and the credentials specified while setting up the "wlsadmin" secret for this instance.

Note

Access to the WebLogic Remote 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. ingress-NGINX 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`**.

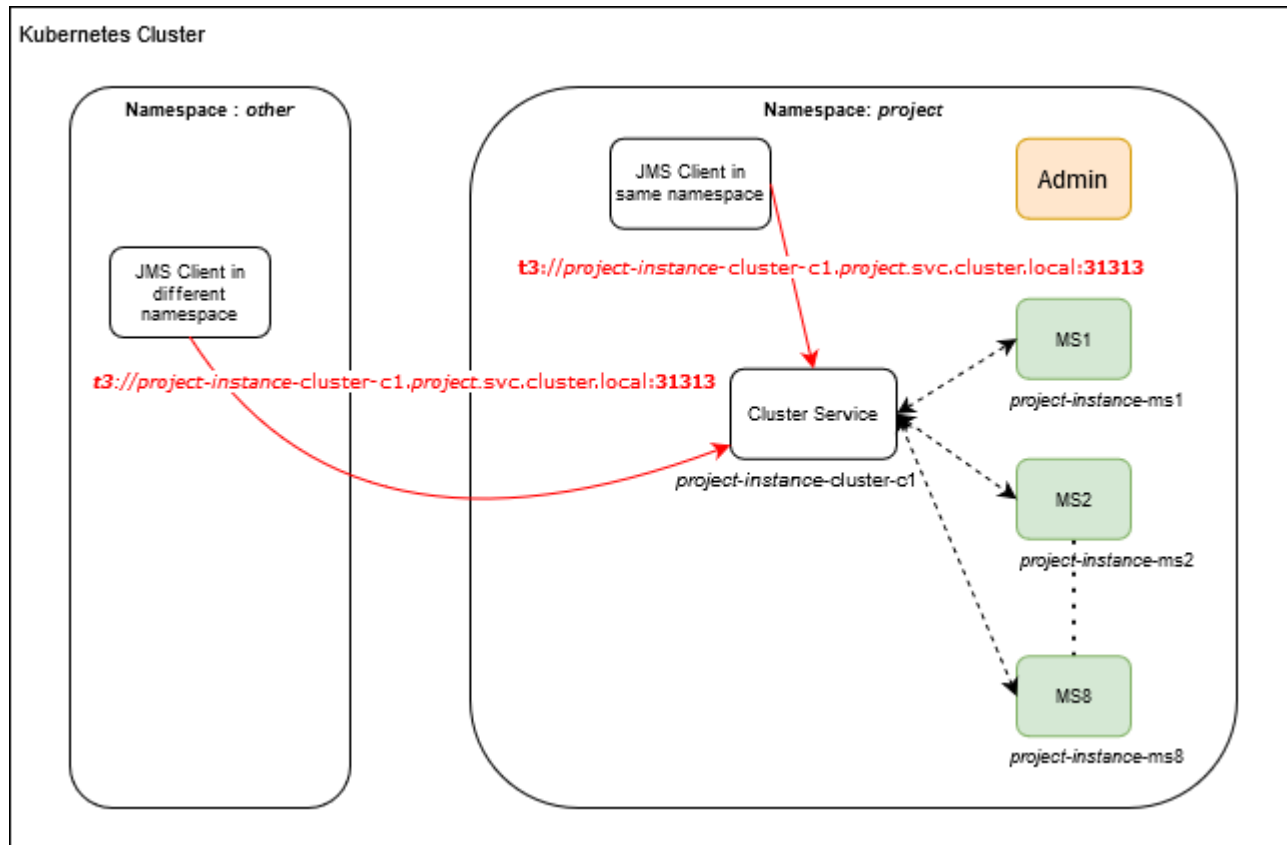
This saves the various network hops typically involved in routing a request from an external client to OSM cloud native deployed in a Kubernetes cluster.

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 10-2 Inbound JMS Connectivity in a Kubernetes Cluster

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 WebLogic 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 Remote 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 that have permission to inject JMS messages into the remote system.

```
safConnectionConfig:
- name: external_system_identifier
  t3Url: t3_url
  secretName: secret_t3_user_password
```

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](#)".

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`).

Security for Remote SAF and Bridges

WebLogic Server supports cross-domain security that establishes trust between domains, and this can be achieved in two ways. These are:

- Global Trust
- Cross-domain Security

Refer to Oracle Fusion Middleware WebLogic Server Documentation for more details. While OSM cloud native supports both the types of domain trust, cross-domain security is preferable due to its finer-grained security.

Configuring Global Trust

For details about global trust, see Oracle Fusion Middleware WebLogic Server documentation.

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 global trust, in your instance specification file, for `domainTrust.globalEnabled`, change the default value to **true**:

```
domainTrust:  
  globalEnabled: true
```

Note

In previous versions, the configuration in the instance specification file was `domainTrust.enabled`. While this is still supported for backward compatibility, it is now deprecated in favor of the configuration mentioned above, `domainTrust.globalEnabled`. If you are using specification files from an older version, consider updating it with the new configuration.

Configuring Cross-Domain Security

Weblogic Server supports cross-domain security, which establishes trust between two domains using specific credentials. Whereas, global trust uses the same credentials to communicate with all domains. Hence, in most cases, using cross-domain security is better as opposed to using global trust. To enable cross-domain security in OSM cloud native, refer to [Configuring the Instance Specification](#) and [Configuring the Project Specification](#).

Note

When you are using **JTA with XA-enabled messaging** with either SAF (Store-and-Forward) messaging or JMS bridge messaging (with *Exactly-once* QoS), it is recommended to enable Cross Domain Trust instead of Global Trust.

Enabling Global Trust introduces security risks, especially for anonymous users and has been observed to cause JMS message rollbacks with errors. To ensure secure and reliable message delivery, always configure Cross Domain Trust.

Configuring the Instance Specification

The instance specification lists out the cross-domain security components that are to be configured on the OSM domain. These are listed under the `domainTrust.crossDomain` element of the instance specification.

To enable cross-domain security, for `domainTrust.crossDomain.enabled`, set the default value to true. You can choose to enable cross-domain security for some, but not all of the remote domains that OSM communicates with. In that case you need to add the names of the domains for which cross-domain security is not enabled to the list of excluded domains in the `domainTrust.crossDomain.excludeDomainNames` element. For those remote domains for which you have enabled cross-domain security, you need to specify a user on the remote domain whose credentials are to be trusted by OSM cloud native. To map each credential you need the remote domain name and a secret with the credentials of the remote user who is authorized to interact with the OSM cloud native instance. These are set using the elements `domainTrust.remoteDomains.name` and `domainTrust.remoteDomains.secretName` respectively.

The following sample shows the cross-domain security configuration for multiple remote domains:

```
domainTrust:
  globalEnabled: false
  crossDomain:
    enabled: true
    excludeDomains:
      - remote-domain-1
      - remote-domain-2
```

```
remoteDomains:
  - name: remote-domain-3
    secretName: remote-domain-3-secret
  - name: remote-domain-4
    secretName: remote-domain-4-secret
```

Note

If you enable cross-domain security to communicate between two domains, then you must not enable global trust. Therefore, for `globalEnabled` (or the deprecated `enabled`) set the value to `false` when enabling cross-domain security.

You need to create a Kubernetes secret to store the credentials of the remote users.

```
kubectl create secret generic -n project <remote-domain-secret> --from-literal=username=<username> --from-literal=password=<password>
```

Configuring the Project Specification

The project specification lists out all the cross-domain users that are to be configured on the OSM cloud native instance. These users must be used while configuring credential mapping on remote domains that are communicating with OSM cloud native with the cross-domain security enabled. These are listed under the `crossDomainTrustUsers` element of the project specification. There is no restriction on the number of cross-domain users configured on each domain. There can be one or 'n' such users tied to specific remote domains in either a '1:1' or a '1:Many' orientation.

The following sample shows the cross-domain users that are to be configured in the OSM domain:

```
crossDomainTrustUsers:
  - user-1
  - user-2
```

You must create the **xtrust** secret for configuring the user credentials for the cross-domain trust users in the OSM domain. Run the following script to configure the secret:

```
$ $OSM_CNTK/scripts/manage-instance-credentials.sh -p $PROJECT -i $INSTANCE -s $SPEC_PATH create xtrust
```

Verify that the following secret is created:

```
secret/<project>-<instance>-crossdomain-users configured
```

Note

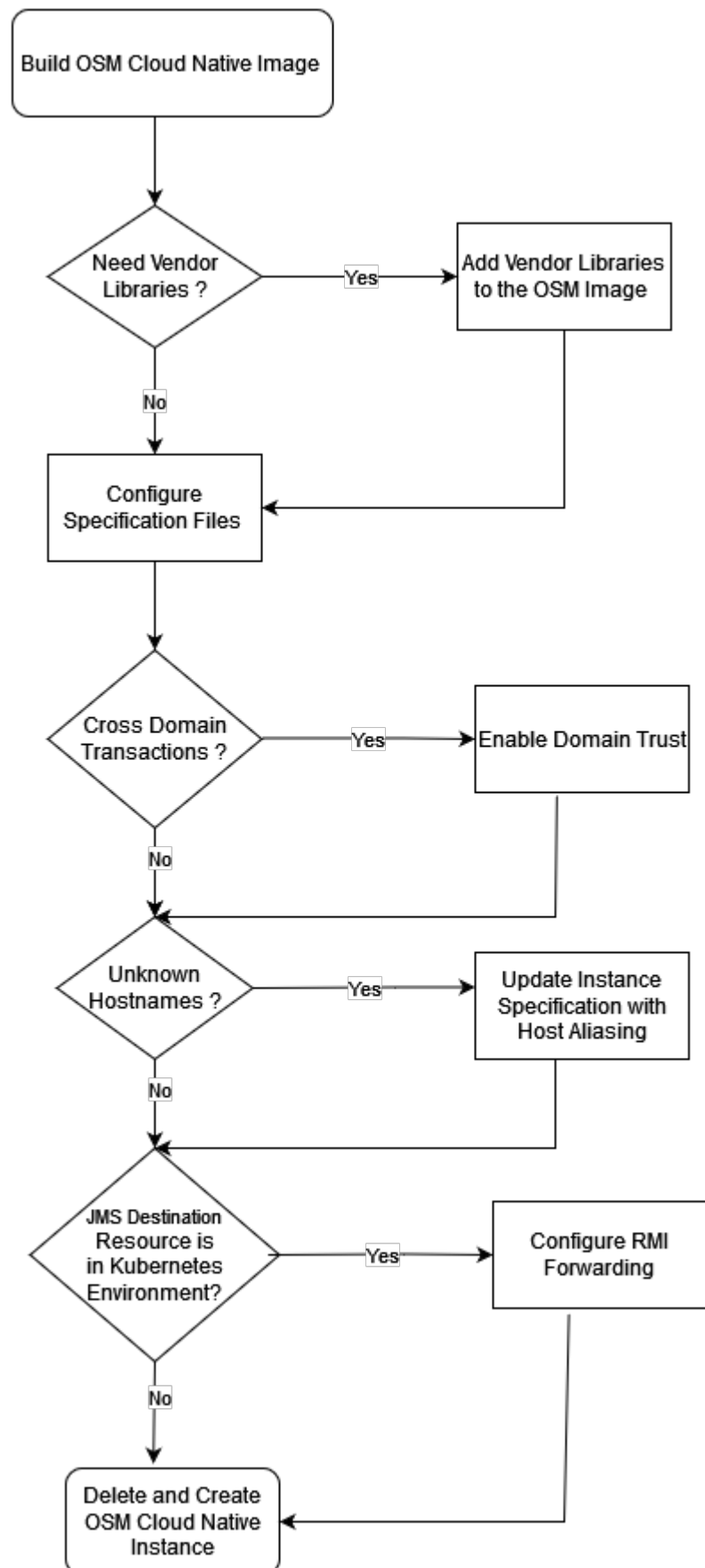
The cross-domain trust users are specifically configured for the purpose of cross-domain security and must not be assigned other roles apart from the default `CrossDomainConnector`.

Configuring WebLogic Messaging Bridges

This section explains the configuration steps required when integrating OSM cloud native with other systems via Messaging Bridges hosted on OSM cloud native.

Use the process described below to set up and configure JMS Bridges in OSM cloud native.

Figure 10-3 Configuring JMS Bridges for OSM Cloud Native



1. Building the OSM Image: Use the OSM cloud native image builder to build the required images. Refer to [Creating OSM Cloud Native Images](#) for more information about building images. Ensure that you are using the latest manifest and patch list as per the OSM Compatibility Matrix. Note down the image name and tag.
2. Adding Vendor Libraries: If your JMS Bridges involve destinations (either source or target) hosted on non-WebLogic platforms, you may need to make vendor-specific libraries available to WebLogic. Consult the appropriate vendor documentation to see if this is needed to obtain the required libraries. Any such libraries need to be included in the OSM image.
 - a. You need a container image builder, either Docker or Podman for this process. Refer to the Compatibility Matrix for more information about which version to use. You can reuse the main cloud native image build host and environment.
 - b. Copy the vendor libraries to the build host.
 - c. Create a Dockerfile to augment the standard OSM image with the libraries. You can use the following as an example to modify to your requirements:

```
ARG base_image
FROM $base_image
ARG username=username
ARG groupname=groupname
ARG vendor_jar1
COPY --chown=$username:$groupname $vendor_jar1 /u01/oracle/wlserver/
server/lib/
```

If you have more than one vendor jar, you can add more ARG and COPY commands using vendor_jar2, vendor_jar3, and so on.

- d. Build an updated image in the same directory as the above Dockerfile:

```
builder build -tag output-image-name:output-image-tag . --build-arg
vendor_jar1=vendor-jar
```

Note

builder will be Docker or Podman.

You can repeat the --build-arg for each vendor jar you defined the way you did above for vendor_jar. If your image build manifest uses hostname or groupname other than the default oracle, provide these values as well:

```
--build-arg username=username --build-arg groupname=groupname
```

- e. Use the output image name and tag in your project specification as the OSM image.
3. Configuring Specifications: You can configure the specifications using the following process:
 - a. Use your project specification to provide a list of JMS Bridges to be hosted on each OSM cloud native instance for this project. This takes the form of a YAML list called jmsBridges. Refer to the sample project specification in the OSM cloud native Toolkit for details on the format and parameters of each list entry.

- b. Use your instance specification to provide the actual endpoints for each JMS Bridge to be hosted for this project. This takes the form of a YAML list called `jmsBridgeDestinations`, where each item describes one of the JMS destinations referenced in the `jmsBridges` list in the project specification. Refer to the sample instance specification in the OSM cloud native ToolKit for details on the format and parameters of each list entry.

Note

Each destination will need credentials for access. The OSM destination definition in the instance specification takes the name of secret where these credentials can be found. This secret must be present in the project namespace before the instance is created or upgraded. Each such secret should have the keys "username" and "password" with the appropriate value. Refer to the example given below:

```
kubectl -n project create secret generic project-instance-othersystem \
  --from-literal=username=username \
  --from-literal=password=
```

4. Configuring Optional Specifications: You can use the following optional configurations for the specifications.
 - a. Domain Trust: If one or more of the JMS destinations are on another WebLogic domain, configure domain trust in the instance specification. For more information, refer to [Configuring SAF](#).
 - b. Host Aliasing: If one or more of the JMS destinations use URLs that include hostnames that cannot be resolved by DNS, provide the hostname-to-IP mappings using the `hostAliases` section of the instance specification.
 - c. RMI Forwarding: If one or more of the JMS destinations are on another WebLogic domain running in a Kubernetes environment, configure a RMI forwarding proxy in the project specification:

```
managedServers:
  project:
    java_options:
```

In the value of the above `java_options`, for each foreign domain, add the option `-Dweblogic.rjvm.domain.proxy.domainname=url-of-load-balancer`, where, `domainname` is the name of the domain configured in the remote WebLogic system and `url-of-load-balancer` is the URL to reach the remote WebLogic domain. If this is a different Kubernetes cluster from where OSM cloud native is running, it will use protocol `http` or `https` and the IP, hostname and port of the load balancer configured for that cluster. If you are using the same Kubernetes cluster as that of OSM cloud native, it will use protocol `t3` and the hostname or port of the `T3ClustChannel` service (`<project>-<instance>-cluster-c1.com.svc.cluster.local:31313`)

5. Applying the Changes: To apply the changes made above, complete the following steps:
 - a. Create the OSM cloud native instance using the above specification for configuration and secrets. This will automatically create the JMS bridges as specified.

- b. If you have an existing OSM cloud native instance for this project and it has been created using an OSM cloud native version older than 7.4.1.0.17, do the following:
 - Delete the instance using the CNTK script **delete-instance.sh**
 - Create the instance using the CNTK script **create-instance.sh**
- c. If you are not using an OSM cloud native version older than 7.4.1.0.17, then invoke the CNTK script **upgrade-instance.sh**.

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. You need to do this 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 adhere to the following guidelines:

- **For the external system configured with FMW 14.1.x:** Apply July 2025 (or later) CPU patches as listed in the *OSM Compatibility Matrix*. Ensure that you have applied WebLogic patch 37940403 related to SAF.
- **For the external system configured with FMW 12.2.1.x:** Apply July 2025 (or later) CPU patches as listed in the *OSM Compatibility Matrix*. Ensure that you have applied WebLogic patch 37940403 related to SAF.

Note

SAF configuration will not work with any patches, even if you are using the SAF workaround if you are using either the October 2024 or the January 2025 CPU patches. Oracle recommends that you move to July 2025 (or later) CPU patches as listed in the *OSM Compatibility Matrix*. Ensure that you have applied WebLogic patch 37940403 related to SAF.

If you are using CPU patches from prior to October 2024 CPU patches, then each 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.

If you cannot apply any patches to your environment follow the workaround given below. 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>.project
0.0.0.0 project-instance-<ms2>.project
...
0.0.0.0 project-instance-<ms17>.project
0.0.0.0 project-instance-<ms18>.project
```

You need to 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. You need to list all eighteen managed servers possible in your OSM cloud native instance, regardless of how many you have 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:

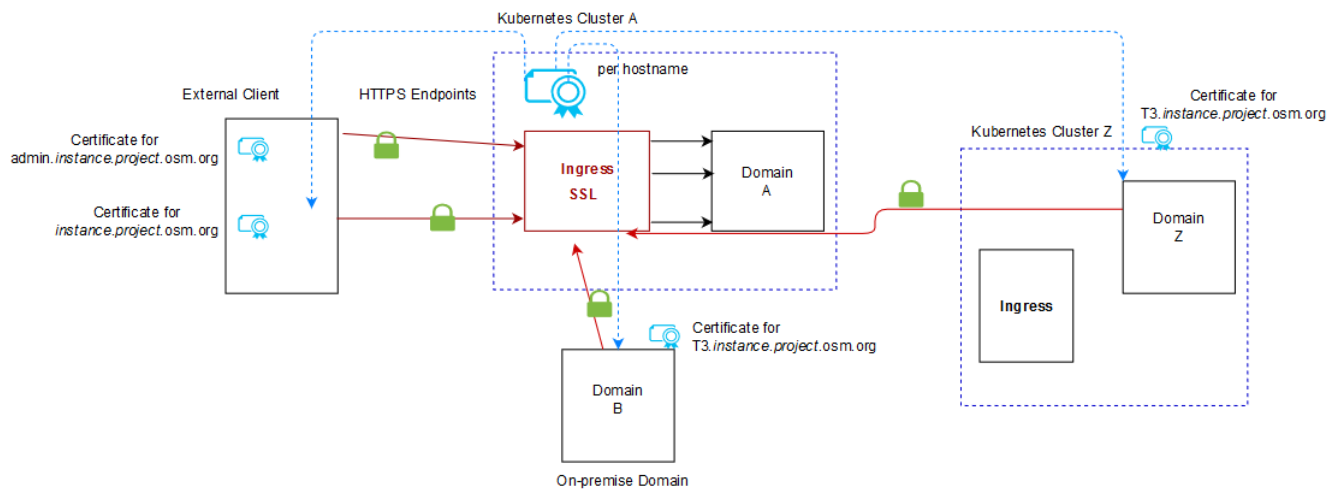
- Configuring Secure Incoming Access with SSL
- Configuring Access to External SSL-enabled Systems

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.

Generating SSL Certificates for Incoming Access

The following illustration shows when certificates are generated.

Figure 10-4 Generating SSL Certificates

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](#)".

TLS Secrets

If you have SSL turned on for incoming connections (by setting `ssl.incoming` to `true` in the specification files), then you must create `wlstls` and `gatewaytls` secrets to provide the required certificate information. This information is provided to the ingress controller securely.

This secret carries the application or ingress TLS credentials for OSM.

```
$OSM_CNTK/scripts/manage-instance-credentials.sh -p project -i instance -s
spec_path create wlstls
```

You get the following three options after you run the above command:

- **WLSStore:** This is for creating secret for truststore when OSM needs to interact with JMS and SAF securely to external system and needs that system's certificates.
- **WLSIngress:** This is for creating secret for Ingress when SSL is enabled.
- **Both:** This is for WLSIngress and WLSStore.

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`):

For more information on the Service name details, see [Using an Alternate Ingress Controller](#).

To set up OSM cloud native for incoming access:

1. Edit the instance specification and set `incoming` to **true** and provide Ingress specific annotations:

Note

Make sure that you remove any incoming WL-Proxy-SSL and WL-Proxy-Client-IP headers. Also, set X-Forwarded-Proto: https and WL-Proxy-SSL: true WebLogic HTTP headers, to notify WebLogic that SSL terminated at Ingress and that the request came in over SSL. See [Oracle Cloud Infrastructure Documentation](#) for more details.

The instance specification contains NGINX annotations as an example. The Ingress resource eliminates the WL-Proxy-Client-IP and WL-Proxy-SSL client headers and adds theX-Forwarded-Proto: https and WL-Proxy-SSL: true input headers.

For any other Ingress controller, identify the corresponding annotations to achieve the same behavior described here:

```
# SSL Configuration
ssl:
  incoming: true
  ingress:
    # These annotations are required if project spec ingressController is
    "GENERIC" and SSL enabled.
    # Different Ingress controller can have implementation specific
    annotations and can be added here.
    # Provided annotations below for nginx and openshift.
    # These annotations are required if project spec ingressController is
    "GENERIC"
    # and the actual ingress controller is nginx with wls custom request
    headers.
    annotations:
      nginx.ingress.kubernetes.io/configuration-snippet: |
        more_clear_input_headers "WL-Proxy-Client-IP" "WL-Proxy-SSL";
        more_set_input_headers "X-Forwarded-Proto: https";
        more_set_input_headers "WL-Proxy-SSL: true";
      nginx.ingress.kubernetes.io/ingress.allow-http: "false"
```

To configure NGINX-Ingress for SSL, see [Using NGINX Ingress Controller](#)

2. Generate key pairs for each hostname corresponding to an endpoint that OSM cloud native exposes to the outside world:

Create TLS Secret For Ingress

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"
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout $SPEC_PATH/ssl/
osm-gateway.key -out $SPEC_PATH/ssl/osm-gateway.crt -subj "/"
CN=instance.project.osm.org"
```

```
# Create secrets to hold each of the certificates.
# Run the manage-instance-credentials.sh to create the TLS secrets.
```

```
$OSM_CNTK/scripts/manage-instance-credentials.sh -p project -i instance -s
spec_path create gatewaytls,wlstls
Provide Gateway Ingress TLS Credentials for 'project-instance' ...
```

```
Ingress TLS Certificate Path: /home/path/gateway.crt
Ingress TLS Key file Path : /home/path/gateway.key
```

```
Please select the option to provide TLS Credentials for 'project-
instance' ...
```

```
1) WLSStore
2) WLSIngress
3) BOTH
```

```
#? 2
```

```
Provide WLS Ingress Keys and Certificates for 'project-instance' ...
```

```
Do you wish to use one common certificate for Admin Server, Managed Server
and T3
```

```
(select number from menu)
```

```
1) Yes
2) No
```

```
#? 1
```

```
Certificate File Path: /home/path/osm.crt
Key File Path : /home/path/osm.key
```

```
secret/project-instance-app-tls-cert configured
secret/project-instance-osm-tls-cert configured
secret/project-instance-admin-tls-cert configured
secret/project-instance-t3-tls-cert configured
```

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
project-instance-app-tls-cert terminates
```

4. Create your instance as usual.

SSL Termination at the Load Balancer

In this configuration, SSL termination happens at the external Load Balancer which handles all incoming HTTPS traffic and forwards decrypted HTTP traffic to the OSM Cloud Native Ingress. The Ingress resource does not expose TLS certificates in this case, as SSL is terminated at the Load Balancer level before the traffic reaches the cluster.

Prerequisites for SSL termination at the external Load Balancer are:

- Ensure that SSL certificates are installed and available at the Load Balancer.
- Confirm that the Load Balancer is configured to forward decrypted traffic to the Ingress service.

Configuration Steps in OSM Cloud Native Specification Files

You must follow the standard steps to configure secure incoming access with SSL in OSM Cloud Native which includes key pair generation, secrets creation, instance specification updates, ingress creation and instance creation. See, [Setting Up Secure Communication with SSL](#) for more information about the configuration steps in OSM Cloud Native specification files.

For SSL pre-termination at the load balancer, you must apply the following changes:

1. Enable SSL Pre-Termination in Instance Specification

In the instance specification file, set:

```
ssl:
  incoming: true
  preTermination: true
```

This ensures that TLS certificates are not mounted in the Ingress resource, since SSL termination is expected at the Load Balancer.

2. Set SSL Redirect Annotation

In the instance specification file, configure the following annotation:

```
nginx.ingress.kubernetes.io/ssl-redirect: "false"
```

Note

This annotation is provided as an example for NGINX Ingress ([ingress-nginx](#)). Other Ingress controllers may use different annotations or configurations.

This prevents NGINX Ingress from forcing an HTTPS redirect, as SSL is already terminated at the Load Balancer.

3. Final Combined Configuration

The ssl section in your instance specification should now look as follows, assuming use of ingress-NGINX:

```
# SSL Configuration
ssl:
  incoming: true
  preTermination: true # This needs to be enabled for ssl termination at
loadbalancer before ingress
ingress:
```

```
# These annotations are required if project spec ingressController is
"GENERIC" and SSL enabled.
# Different Ingress controller can have implementation specific
annotations and can be added here.
# Provided annotations below for nginx and openshift.
# These annotations are required if project spec ingressController is
"GENERIC"
# and the actual ingress controller is nginx with wls custom request
headers.
annotations:
  nginx.ingress.kubernetes.io/configuration-snippet: |
    more_clear_input_headers "WL-Proxy-Client-IP" "WL-Proxy-SSL";
    more_set_input_headers "X-Forwarded-Proto: https";
    more_set_input_headers "WL-Proxy-SSL: true";
# This annotation is required if SSL enabled with pretermination
nginx.ingress.kubernetes.io/ssl-redirect: "false"
```

4. Create the Ingress using `create-ingress.sh`.
5. Create the OSM 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/jdk-21.0.7/lib/security/cacerts -
storepass default_password
```

For example

```
./keytool -importcert -v -trustcacerts -alias osmcn -file /scratch/t3.crt -
keystore /jdk-21.0.7/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.example.osm.org:30443/
oracle.communications.ordermanagement.SimpleResponseQueue
```

Configuring Access to External SSL-Enabled Systems

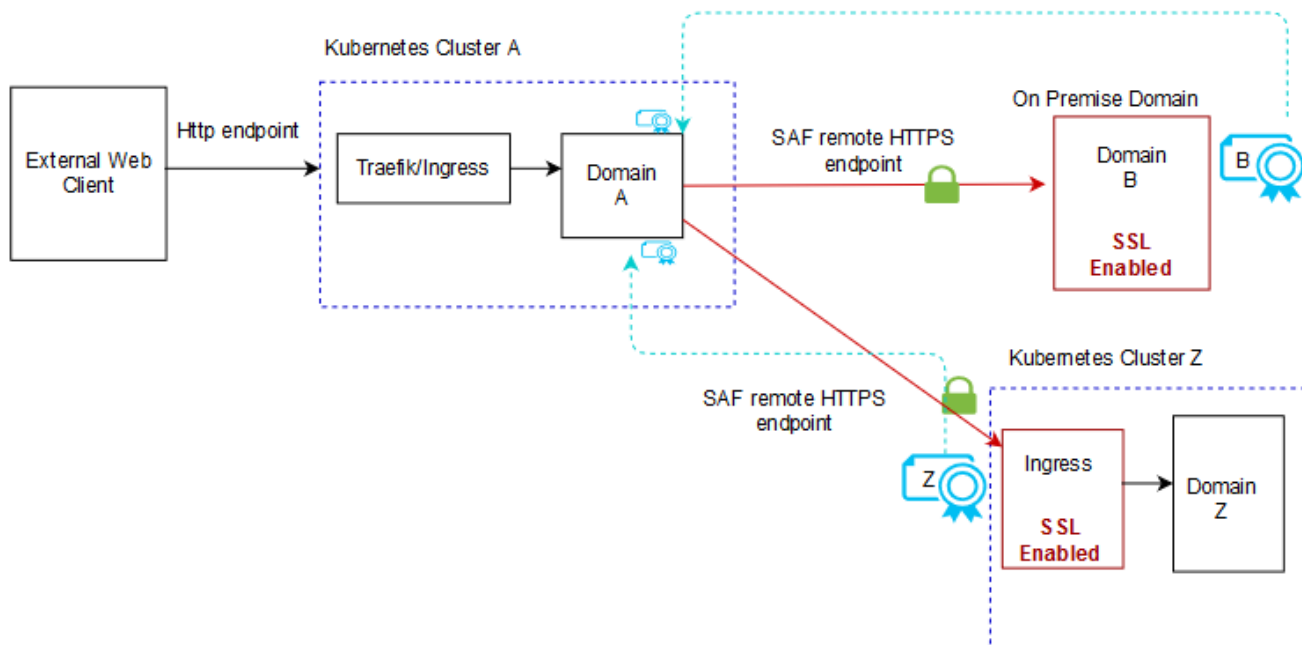
In order for OSM cloud native to participate successfully in a handshake with an external server for SAF connectivity, the SSL certificates from the external domain must be made available to the OSM cloud native setup. See ["Enabling SSL on an External WebLogic Domain"](#) for details about how you could 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 10-5 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.

```
jdk_path/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.

```
jdk_path/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.

```
jdk_path/bin/keytool -importcert -trustcacerts -alias alias -file
certificate -keystore path-to-jdk/jdk-21.0.7/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_password
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. (Optional) Set up custom trust using the following steps:

Note

This step is required when HTTPS/T3S-based SAF connections are configured.

Note

You need to configure a custom identity keystore `customIdentity` whenever you use a custom trust keystore `customTrust`.

- 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. Edit the instance specification, setting the trust name.

```
# SSL trust and identity
ssl:
  trust:
    name: truststore # truststore filename without extension
    (truststore.jks). This must be commented out or a fixed value of
    "truststore" as this is deprecated.
  identity:
    name: keystore # Identity store filename without extension
    (keystore.jks). A valid Custom Identity keystore is required when a
    truststore is used.
```

- c. Create a Kubernetes secret to hold the truststore file and the passphrase. The secret name should match the truststore name.

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

Please select the option to provide TLS Credentials for 'project-

```
instance' ...
1) WLSStore
2) WLSIngress
3) BOTH
#? 1
```

Please select the option to create WLS Store for 'project-instance' ...

```
1) TrustStore
2) KeyStore
3) Both
#? 1
```

Provide WLS Store for 'project-instance' ...

The WLS Truststore File Path+Name. This JKS file will be renamed to truststore.jks, and be attached to the --truststoresecret, and be used by the OSM managed servers and Admin server as the truststore.
WLS Truststore file Path+Name: /truststore.jks
WLS Truststore Passphrase:

Note

If **ssl.trust.name** isn't populated in the instance specification, a warning will be displayed that the truststore isn't required.

2. Set up custom identity using the following steps:

Note

The demo identity keystore is not recommended for production use. If you have not configured a custom identity keystore and you have not enabled secure production mode, the system uses the demo identity keystore by default for non-SSL configurations.

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. Edit the specification file:

```
identity:
  name: keystore # Identity store filename without extension
                 (keystore.jks).
```

c. Create the secret:

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

Please select the option to provide TLS Credentials for 'project-instance' ...

```
1) WLSStore
2) WLSIngress
3) BOTH
#? 1
```

Please select the option to create WLS Store for 'project-instance' ...

```
1) TrustStore
2) KeyStore
3) Both
#? 2
```

Provide WLS Store for 'project-instance' ...

The WLS keystore File Path+Name. This JKS file will be renamed to keystore.jks, and be attached to the --keystore secret, and be used by the OSM managed servers and Admin server as the keystore.

WLS Keystore file Path+Name: /keystore.jks

WLS Keystore Passphrase:

SSL Key Alias: ssl_key

SSL Key Passphrase:

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=password=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"
```

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 14.1.2 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 Native 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 14.1.2 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.
 9. When enabling cross-domain security, configure its components using Weblogic Scripting Tool (WLST) in online mode by following the steps below:

- a. Once the simple domain is up and running, start the WLST at the location **\$FMW_HOME/oracle_common/common/bin**

```
$ ./wlst.sh
```

```
Initializing WebLogic Scripting Tool (WLST) ...
```

```
Welcome to WebLogic Server Administration Scripting Shell
```

```
Type help() for help on available commands
```

```
wls:/offline>
```

- b. Connect to the admin server by setting values corresponding to simple domain Weblogic server.

```
wls:/offline> connect('<username>','<password>','t3://<admin-server-host>:<admin-server-port>')
```

Upon connecting successfully, you will see a message like the one below:

```
Connecting to t3://localhost:7001 with userid weblogic ...
Successfully connected to Admin Server "AdminServer" that belongs to
domain "simple".
```

- c. Since this attribute is set as read-only, start an edit session and navigate to the domain for setting the cross-domain security to true. Save and activate the changes.

```
wls:/simple/serverConfig/> edit()
Location changed to edit tree.
This is a writable tree with DomainMBean as the root.
To make changes you will need to start an edit session via startEdit().
For more help, use help('edit').
You already have an edit session in progress and hence WLST will
continue with your edit session.
```

```
wls:/simple/edit/ !> startEdit()
Starting an edit session ...
Started edit session, be sure to save and activate your changes once
you are done.
wls:/simple/edit/ !> cd('SecurityConfiguration/simple')
wls:/simple/edit/SecurityConfiguration/simple !>
cmo.setCrossDomainSecurityEnabled(true)
wls:/simple/edit/SecurityConfiguration/simple !> save()
Saving all your changes ...
Saved all your changes successfully.
wls:/simple/edit/SecurityConfiguration/simple !> activate()
Activating all your changes, this may take a while ...
The edit lock associated with this edit session is released once the
activation is completed.
Activation completed
```

- d. To configure a cross-domain user, navigate back to the root and then to the authentication provider to create a user **simple-cross-user** by setting an appropriate password. Assign the user to the **CrossDomainConnectors** group. This credential will

be used in step 4 of [Preparing the OSM Cloud Native Instance](#), while creating the secret **simple-cross-user-credentials** in the OSM cloud native instance.

```
wls:/simple/edit/SecurityConfiguration/simple> serverConfig()
wls:/simple/serverConfig/> cd('SecurityConfiguration/simple/Realms/
myrealm/AuthenticationProviders/DefaultAuthenticator')
wls:/simple/serverConfig/SecurityConfiguration/simple/Realms/myrealm/
AuthenticationProviders/DefaultAuthenticator> cmo.createUser('simple-
cross-user', '<password>', 'for cross-domain security')
wls:/simple/serverConfig/SecurityConfiguration/simple/Realms/myrealm/
AuthenticationProviders/DefaultAuthenticator>
cmo.addMemberToGroup('CrossDomainConnectors', 'simple-cross-user')
```

- e. To configure credential mapping for cross-domain security, navigate to the credential mapper directory. Create a credential for **osm-cross-user** as configured in OSM cloud native. Refer to step 4 of [Preparing the OSM Cloud Native Instance](#) for more information. Replace **remoteHost** with the OSM domain name.

```
wls:/simple/serverConfig/SecurityConfiguration/simple/Realms/myrealm/
AuthenticationProviders/DefaultAuthenticator> cd('/
SecurityConfiguration/simple/Realms/myrealm/CredentialMappers/
DefaultCredentialMapper')
wls:/simple/serverConfig/SecurityConfiguration/simple/Realms/myrealm/
CredentialMappers/DefaultCredentialMapper>
cmo.setUserPasswordCredential('type=<remote>, protocol=cross-domain-
protocol, remoteHost=<project>-<instance>', 'osm-cross-user',
'<password>')
wls:/simple/serverConfig/SecurityConfiguration/simple/Realms/myrealm/
CredentialMappers/DefaultCredentialMapper>
cmo.setUserPasswordCredentialMapping('type=<remote>, protocol=cross-
domain-protocol, remoteHost=<project>-<instance>', 'cross-domain', 'osm-
cross-user')
```

Note

While setting the credential and mapping it, do not change the value for **type**. It must be **<remote>** as specified above.

- f. Once all the cross-domain security components are configured successfully, disconnect from the admin server and exit the script.

```
wls:/simple/serverConfig/SecurityConfiguration/simple/Realms/myrealm/
CredentialMappers/DefaultCredentialMapper> disconnect()
Disconnected from weblogic server: AdminServer
wls:/offline> exit()
```

Exiting WebLogic Scripting Tool.

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 Remote 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. When enabling cross-domain security, the components of the instance must be configured via the project and instance specification:
 - a. To configure your instance specification, replace the existing **domainTrust** configuration with the following:

```
domainTrust:
  globalEnabled: false
  crossDomain:
    enabled: true
    remoteDomains:
      - name: simple
        secretName: simple-cross-user-credentials
```

Create the secret **simple-cross-user-credentials** to contain the credentials of the cross-domain user configured for the simple WebLogic domain by running the following command. Enter the project name and password that you are using.

```
kubectl create secret generic -n project simple-cross-user-credentials
--from-literal=username=simple-cross-user --from-
literal=password=<password>
```

- b. To configure your project specification, provide username for the user to be created and added to **CrossDomainConnectors** group in the OSM domain by replacing the existing **crossDomainTrustUsers** configuration with the following:

```
crossDomainTrustUsers:
  - osm-cross-user
```

Create the **xtrust** secret for configuring the user credentials for the above cross-domain trust user by running the following CNTK script:

```
$ $OSM_CNTK/scripts/manage-instance-credentials.sh -p $PROJECT -
i $INSTANCE -s $SPEC_PATH create xtrust
```

This user's credentials must be same as the one supplied in step 9 of [Preparing the WebLogic System to Run the Emulator](#).

5. 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.

6. 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.
7. 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:

On the `simple` WebLogic domain, log in to the WebLogic Remote console and do the following:

1. Navigate to **Remote Endpoints**.
 - a. Navigate from the **Monitoring Tree** to the **Services**. From the **Services**, expand **Messaging**.
 - b. From **Messaging**, select **SAF Runtime**. From **SAF Runtime**, select **Agents**. From the **Agents**, select **<SAF-Agent>**. Here you can access the **Remote Endpoints**.
 - c. You should see a remote endpoint called `simple_osm_saf_agent` with the URL pointing to your OSM cloud native instance.
2. Navigate to **Deployments**.
 - a. Navigate from the **Monitoring Tree** to the **Deployments** section. Here, select **Application Management**.
 - b. You should see the emulator MDB shown with **State** as **Active** and **Health** as **OK**.

On the OSM cloud native instance, log in to the WebLogic Remote console and do the following:

1. Navigate from **Monitoring Tree** to **Services**. From **Services**, expand **Messaging**.
2. From **Messaging**, select **SAF Runtime**. From **SAF Runtime**, select **Agents**. From the **Agents**, select **<SAF-Agent>**. Here you can access the **Remote Endpoints**.
3. Here, 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.SAXParser
-
Djavax.xml.parsers.DocumentBuilderFactory=com.sun.org.apache.xerces.interna
l.jaxp.DocumentBuilderFactoryImpl
-
Djavax.xml.parsers.SAXParserFactory=com.sun.org.apache.xerces.internal.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 11-1 Connection Properties for HermesJMS Session

Property	Value
providerURL	<code>http://t3.instance.project.osm.org:access-port</code> <i>access-port</i> is the Traefik (ingress controller) NodePort or the Load Balancer port.
binding	<code>oracle/communications/ordermanagement/osm/ExternalClientConnectionFactory</code>
initialContextFactory	<code>weblogic.jndi.WLInitialContextFactory</code>
securityPrincipal	<i>osm-user-name</i>
securityCredentials	<i>password-for-osm-user-name</i>

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.

Maintaining the OSM Cloud Native Environment

This chapter describes the tasks you perform in order to apply a change or upgrade a component in your OSM cloud native environment.

Before You Upgrade

Before you upgrade your OSM cloud native environment, you must compare the new samples with the current samples in your cloud native toolkit and migrate any customizations you have made. These include the following:

- Custom shape specifications: Review the shape specifications in the toolkit and identify any changes required to your custom shape files.
- Project specification: Compare the sample project specification of the new toolkit with the sample from the current one and migrate your customizations to the new specification.
- Instance specification: Compare the sample instance specification of the new toolkit with the sample from the current one and migrate your customizations to the new specification.
- Model extensions and custom files.

Also, see the OSM patch readme for the latest patch and *OSM Release Notes* for additional information related to changes in the cloud native toolkit.

About Upgrade Paths and Procedures

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 12-1 Upgrade Paths Using CNTK Scripting

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
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 (some exceptions needing offline change)	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

Table 12-1 (Cont.) Upgrade Paths Using CNTK Scripting

Upgrade Type	Component	Upgrade Path	Requires Changing Image?
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
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
Shared infrastructure	Kubernetes Cluster Software Infrastructure	Online change, external procedure	Not Applicable

Note

The upgrade paths in the following table assume that the **introspectorJobVersion** property in the **job-values.yaml** file is currently set to a value such as **v1**.

Table 12-2 Upgrade Paths with CD Automation Using Flux-CD

Upgrade Type	Component	Allowed	Outage Required	Details
Cartridge Management	Deploy new cartridge version	Yes	No	Change <code>introspectorJobVersion</code> to <code>v2</code> .
Cartridge Management	Redeploy a cartridge against an existing cartridge version	Yes	Yes	<pre> introspectorJobVersion: v2 targetState: running tolerance: fullOutage: allowed: true appliesTo: jobVersion: v2 </pre>
Cartridge Management	Fast undeploy cartridge version	Yes	No	Change <code>introspectorJobVersion</code> to <code>v2</code> .

Table 12-2 (Cont.) Upgrade Paths with CD Automation Using Flux-CD

Upgrade Type	Component	Allowed	Outage Required	Details
Cartridge Management	Purge cartridge version	No	Yes	1. Purge cartridge manually 2. Restart the OSM cloud native instance by the job: <pre> introspectorJobVersion: v2 targetState: running action: restart: type: all appliesTo: jobVersion: v2 </pre>
Configuration and Tuning	OSM cluster size (scaling up or down)	Yes	No	Change introspectorJobVersion to v2.
Configuration and Tuning	Java parameters (memory, GC, and so on)	Yes	No	Change introspectorJobVersion to v2.
Configuration and Tuning	WebLogic domain configuration (WDT such as JMS Queue configuration)	Yes	No	Change introspectorJobVersion to v2.
Configuration and Tuning	OSM configuration parameters (traditionally, oms-config.xml)	Yes	No	Change introspectorJobVersion to v2.
Database Storage Management	Create partition and clone database statistics	No	Yes	1. Shut down the OSM cloud native instance by the job: <pre> introspectorJobVersion: v2 targetState: notRunning </pre> 2. PDB upgrade: Manually create partitions and clone database statistics. 3. Bring up the OSM cloud native instance by the job: <pre> introspectorJobVersion: v3 targetState: running </pre>
Database Storage Management	Online row-based order purge	No	Not applicable	Nothing to do in the job. External procedure.
Database Storage Management	Purge partition	No	Not applicable	Nothing to do in the job. External procedure.

Table 12-2 (Cont.) Upgrade Paths with CD Automation Using Flux-CD

Upgrade Type	Component	Allowed	Outage Required	Details
Security parameters	New, renamed or deleted secrets passed to cartridges	Yes	No	1. New, renamed or deleted secrets passed to cartridges. 2. Restart the OSM cloud native instance using the job: <pre> introspectorJobVersion: v2 targetState: running action: restart: type: full appliesTo: jobVersion: v2 </pre>
Security parameters	Secrets value (For example, changing password)	No	Not applicable	1. Secrets value (For example, changing password) 2. Restart your OSM cloud native instance using the job: <pre> introspectorJobVersion: v2 targetState: running action: restart: type: full appliesTo: jobVersion: v2 </pre>
Software Upgrade and Patching	OSM release or patch upgrade with Database change	Yes	Yes	1. Upgrade the image. 2. Create/Upgrade your OSM cloud native instance: <pre> introspectorJobVersion: v2 targetState: running tolerance: fullOutage: allowed: true appliesTo: jobVersion: v2 </pre>
Software Upgrade and Patching	Fusion MiddleWare upgrade	Yes	No	1. Upgrade the image. 2. Create/Upgrade your OSM cloud native instance: <pre> introspectorJobVersion: v2 targetState: running </pre>

Table 12-2 (Cont.) Upgrade Paths with CD Automation Using Flux-CD

Upgrade Type	Component	Allowed	Outage Required	Details
Software Upgrade and Patching	OSM patch upgrade without Database change	Yes	Yes	1. Upgrade the image. 2. Create/Upgrade your OSM cloud native instance: <pre>introspectorJobVersion: v2 targetState: running tolerance: fullOutage: allowed: true appliesTo: jobVersion: v2</pre>
Software Upgrade and Patching	Fusion MiddleWare overlay patches (for example, PSU or one-off patch)	Yes	No	1. Upgrade the image. 2. Create/Upgrade your OSM cloud native instance: <pre>introspectorJobVersion: v2 targetState: running</pre>
Software Upgrade and Patching	Java upgrade	Yes	No	1. Upgrade the image. 2. Create/Upgrade your OSM cloud native instance: <pre>introspectorJobVersion: v2 targetState: running</pre>
Software Upgrade and Patching	Linux	Yes	No	1. Upgrade the image. 2. Create/Upgrade your OSM cloud native instance: <pre>introspectorJobVersion: v2 targetState: running</pre>
Software Upgrade and Patching	Custom code or third-party tool (custom image)	Yes	No	1. Upgrade the image. 2. Create/Upgrade your OSM cloud native instance: <pre>introspectorJobVersion: v2 targetState: running</pre>
Software Upgrade and Patching	OSM cloud native toolkit	Yes	Not applicable	1. Upgrade the image. 2. Create/Upgrade your OSM cloud native instance: <pre>introspectorJobVersion: v2 targetState: running</pre>

Table 12-2 (Cont.) Upgrade Paths with CD Automation Using Flux-CD

Upgrade Type	Component	Allowed	Outage Required	Details
Shared infrastructure	Operating system or hardware on worker node	Not applicable	Not applicable	Not applicable
Shared infrastructure	WebLogic Operator minor upgrade (backward compatible)	Not applicable	Not applicable	Not applicable
Shared infrastructure	WebLogic Operator major upgrade (non-backward compatible)	Not applicable	Not applicable	Not applicable
Shared infrastructure	Kubernetes Cluster software infrastructure	Not applicable	Not applicable	Not applicable

Once you understand the activities in your upgrade path, you can begin to map out the sequence of activities that you need to perform.

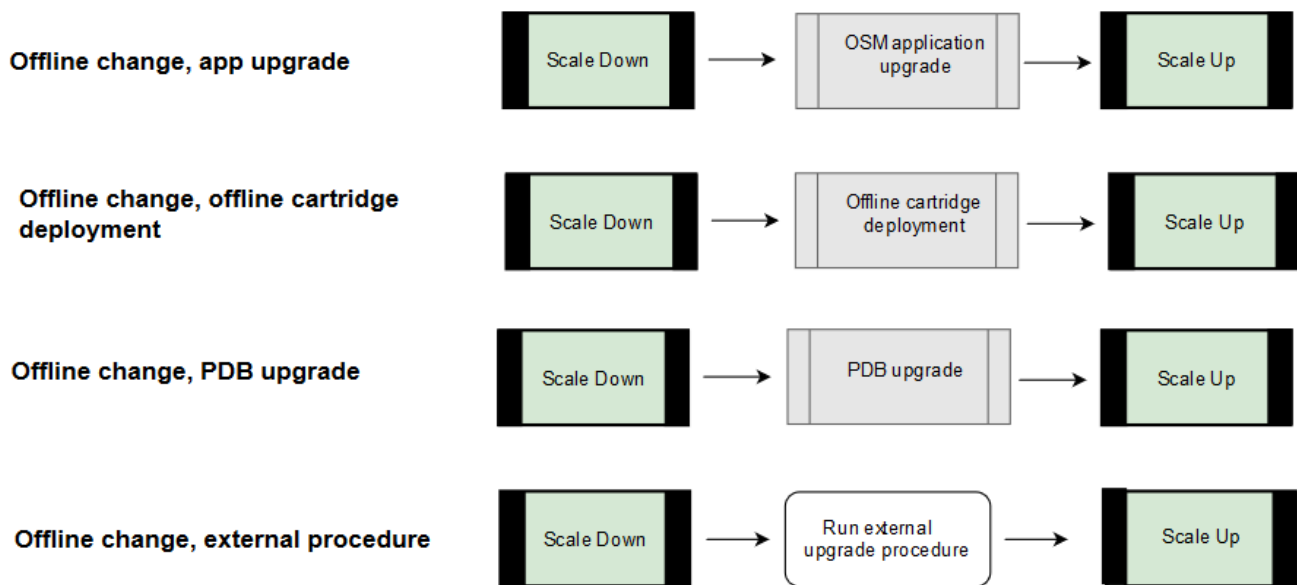
Offline Change Upgrade Paths

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 12-1 Offline Change Upgrade Paths



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**.

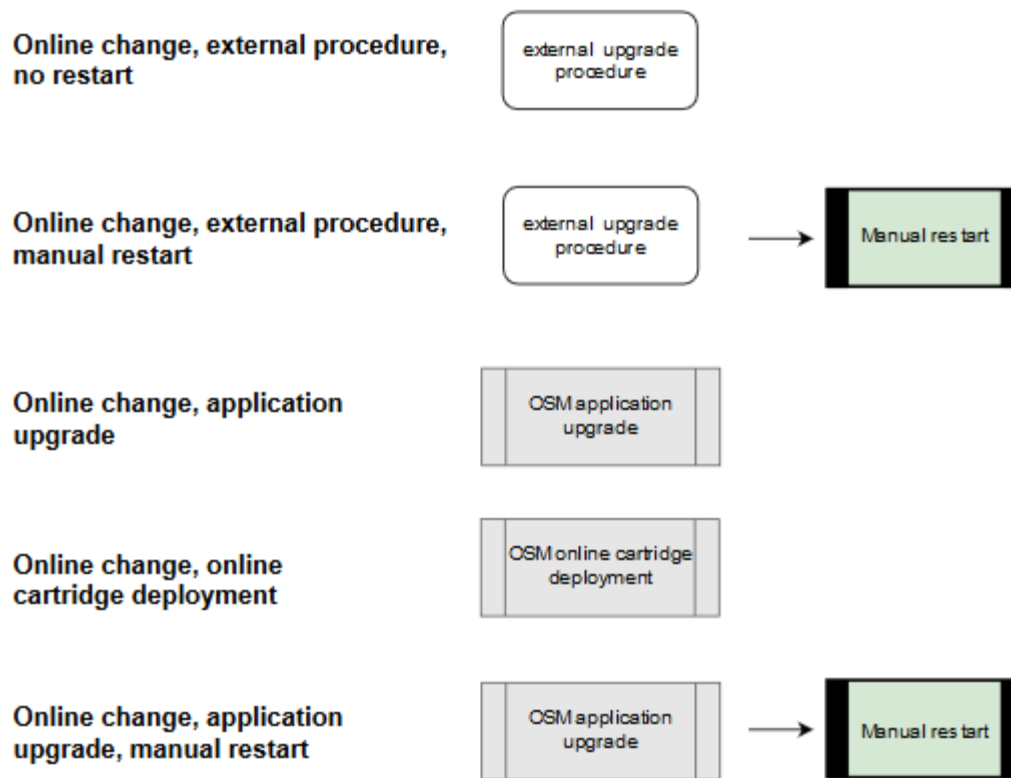
Online Change Upgrade Paths

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 12-2 Online Change Upgrade Paths

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 the 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 assumes 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 Kubernetes Operator Upgrade

To upgrade the WebLogic Kubernetes Operator, you have the following options:

- **Operator Upgrade:** For a standard upgrade process, follow the WKO documentation at: <https://oracle.github.io/weblogic-toolkit-ui/navigate/kubernetes/k8s-wko/#install-operator>. Ensure that the target version is compatible with the current version within your Kubernetes cluster.

Advantages with this option are:

- No additional Kubernetes resources are required.
- No need to re-register namespaces.

Disadvantages with this option are:

- Cannot test with a canary namespace.
- More challenging to roll back to the previous version if needed.

- **Phased Cutover Approach:** To install new WKO, create a new namespace with a fresh label selector. Transition the OSM namespaces by removing the old label and adding the new label to each respective namespace. After all namespaces have successfully transitioned and are stable, proceed to uninstall the old WKO.

Advantages with this option are:

- Ability to test with a canary namespace before full deployment.
- Allows for a phased cutover, accommodating program timelines.
- Easy backout option by reverting the label change on OSM namespaces.

Disadvantages with this option are:

- Requires modification of all OSM namespaces to use the new WKO.
- Additional Kubernetes resources are in use until the old WKO is uninstalled.

For more information on upgrading the operator, refer to the WKO upgrade documentation at: <https://oracle.github.io/weblogic-kubernetes-operator/managing-operators/conversion-webhook/>.

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.

Note

If you scale down the cluster size to **1**, the OSM Gateway microservice will experience downtime.

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
```

Note

If changes are made in secrets while a microservice is up and running, run the **restart-instance.sh** script for that microservice for the changes to take effect. For example, if secrets specific to OSM Gateway are changed, run the following command:

```
$OSM_CNTK/scripts/restart-instance.sh -p project -i instance -  
s $SPEC_PATH -r osmgw
```

Fast Delete

When the entire WebLogic 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 uncomment and add or modify the value of the `serverStartPolicy` parameter to `Never`. This terminates all admin and managed servers including microservices pods.

```
serverStartPolicy: "Never"
```

Apply the changes to the running instance using `upgrade-instance.sh` script.

For example:

```
$OSM_CNTK/scripts/upgrade-instance.sh -p <projectName> -i <instanceName> -s  
<specPath>
```

Starting Up the Cluster

To start up the cluster, edit the instance specification uncomment and modify the value of the `serverStartPolicy` to `IfNeeded`. This starts all admin and managed server including microservices pods.

```
serverStartPolicy: "IfNeeded"
```

Apply the changes to the running instance using `upgrade-instance.sh` script.

For example:

```
$OSM_CNTK/scripts/upgrade-instance.sh -p <projectName> -i <instanceName> -s  
<specPath>
```

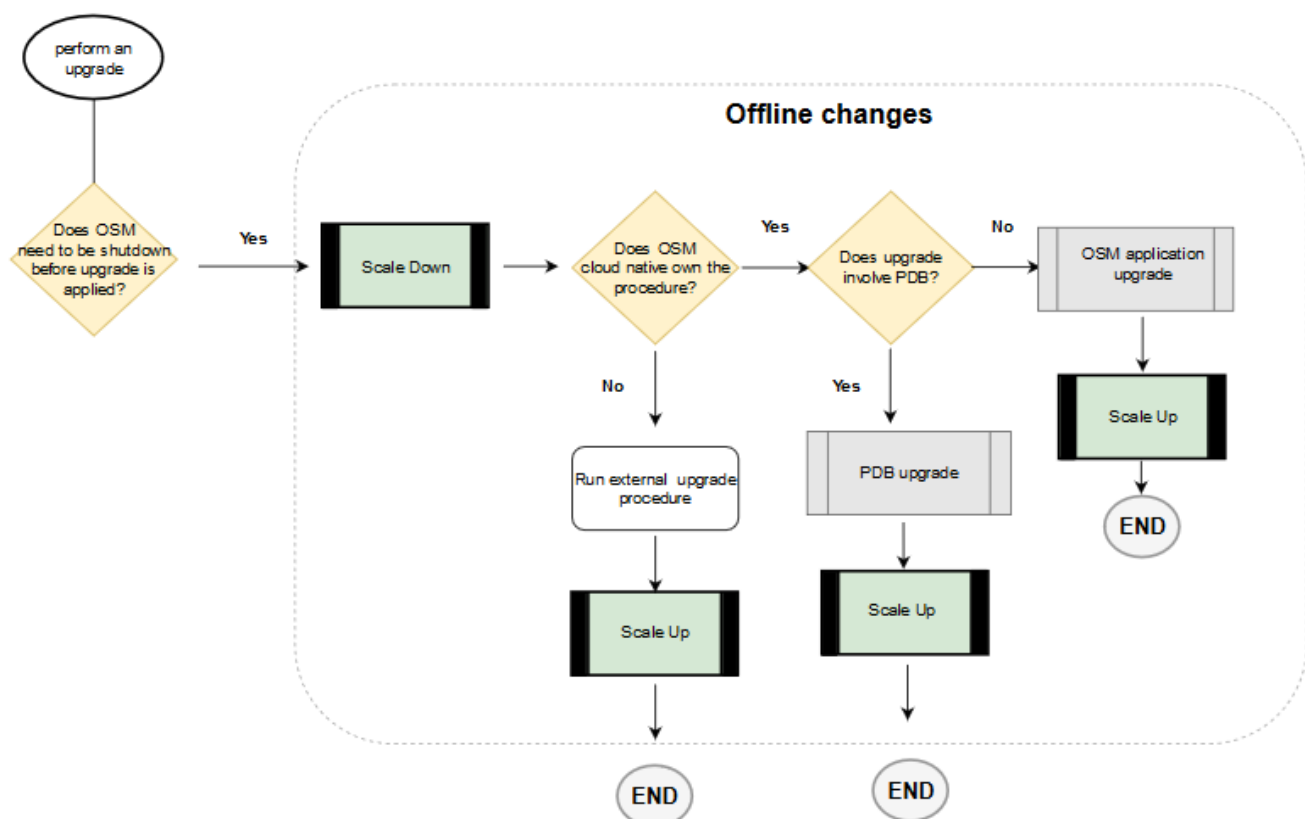
Upgrade Path Flow Chart

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.

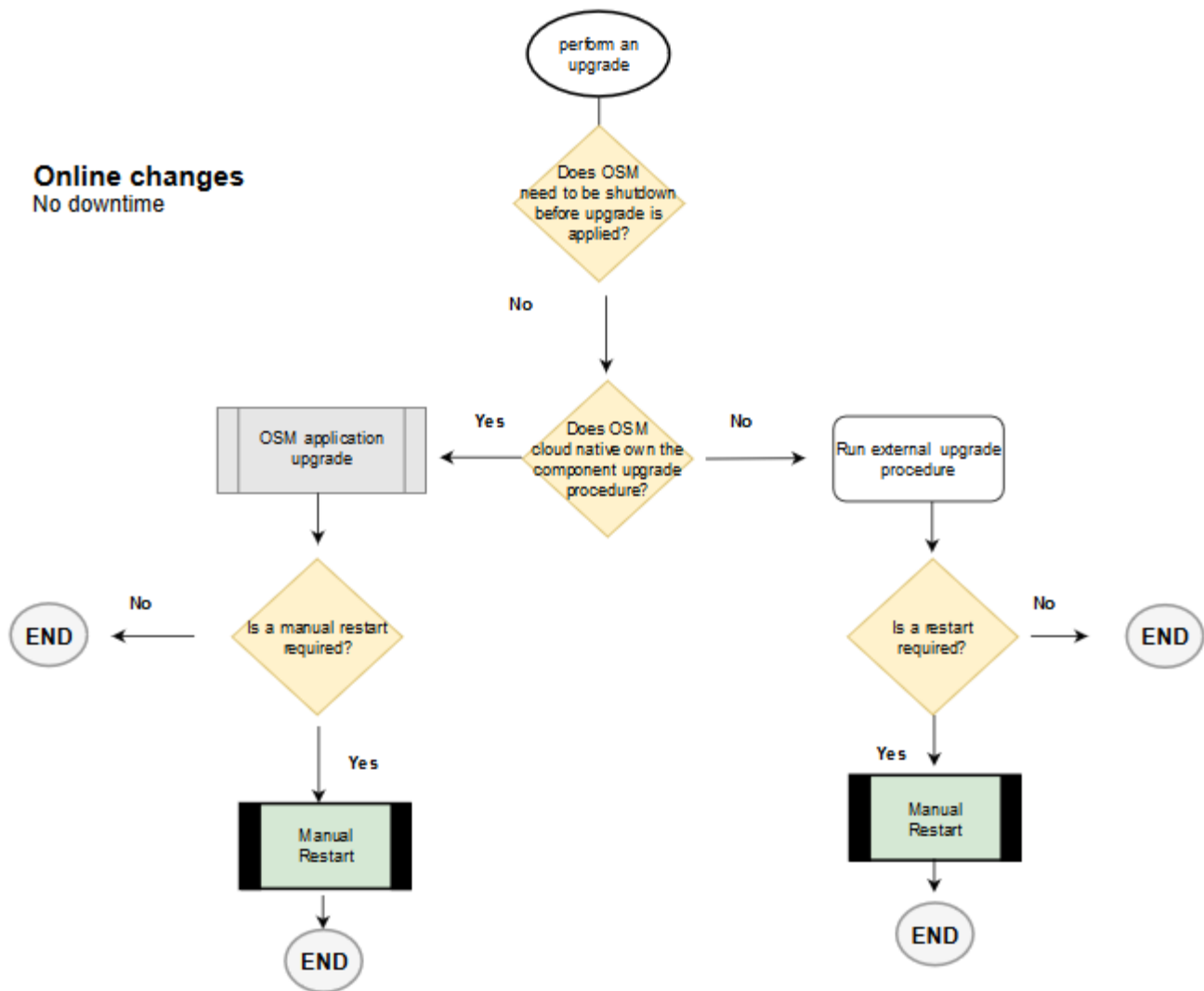
The following flowchart illustrates the flow for offline upgrades and various scenarios.

Figure 12-3 Upgrade Path Flow for Offline Changes



The following flowchart illustrates the flow for online upgrades and various scenarios.

Figure 12-4 Upgrade Path Flow for Online Changes



Upgrading your OSM Cloud Native Deployment

This chapter provides instructions for upgrading OSM cloud native deployment release 7.4.1 or later to OSM cloud native 8.0.

Overview of the Upgrade Steps

This section outlines the upgrade steps:

- Download the OSM 8.0 CNTK version.
- Install or upgrade WebLogic Kubernetes Operator (WKO). See "[Installing WebLogic Kubernetes Operator](#)".
- Decide on the Ingress Controller. See "[Ingress Controller](#)".
- Build the OSM, DB Installer, OSMGW, RTUX and OCA images, using the same OSM cloud native version as the toolkit. See "[Creating OSM Cloud Native Images](#)" for building the images.
- Updating the specification files. See "[Updating Specification Files](#)".
- Upgrading to OSM Cloud Native 8.0. See "[Upgrading to OSM Cloud Native 8.0](#)".

Installing WebLogic Kubernetes Operator

This section provides information about installing WebLogic Kubernetes Operator.

WKO Monitoring Mechanism

Once WKO is installed in a Kubernetes cluster, it needs to know which namespaces will have OSM instances. Of interest are two mechanisms:

- **List:** WKO is given an explicit list of namespaces. This is done by upgrading the WKO helm chart to put in a new value for the list (one that takes the existing value and either adds a new namespace or removes one of the namespaces). This mechanism is supported in WKO 3.x and 4.x and is the mechanism in OSM 7.4.1 CNTK as well (upto 7.4.1.0.13). With this, the user needs RBAC access to the WKO namespace, which is likely administratively problematic in a large shared environment.
- **Label:** When WKO is installed, it is given a label to look for in namespaces. Any namespace in the cluster that has that label is monitored. Adding or removing that label from a namespace has the effect of adding or removing that namespace from WKO's monitoring. This mechanism is supported in WKO 4.x. With this, the user needs RBAC access only to the OSM project namespace and not to WKO namespace.

For OSM cloud native, the recommendation is to use the Label mechanism for WKO 4.x and especially for WKO 4.0.8 and newer.

Operator Installation

Before starting with the procedure in this section, refer to *OSM Compatibility Matrix* for release 8.0 for the list of supported WKO versions. It is mandatory to have WKO installed with label-based monitoring mechanism for OSM 8.0.

If your existing WebLogic Kubernetes Operator satisfies the OSM 8.0 version requirements and uses the label-based monitoring mechanism, no action is required. Skip to the "[Ingress Controller](#)" section.

If you need to upgrade, it is recommended to install the new WKO into a new namespace. As individual namespaces are deregistered from old WKO and registered to new WKO, eventually, the old WKO will no longer be managing any namespaces. It can then be safely uninstalled.

- Refer to Weblogic Kubernetes Operator documentation for operator installation at: <https://oracle.github.io/weblogic-kubernetes-operator/managing-operators/>
- During installation, it is strongly recommended to choose a specific label that this version of WKO must look for, rather than relying on the default label. Relying on the default label causes problems when multiple WKOs need to be installed for any reason (such as during a phased WKO upgrade).
- To provide the custom label, use the `domainNamespaceLabelSelector` parameter as shown below.
- It is recommended that the custom label you select is descriptive and unique to this WKO. In the example below, this is achieved by specifying the full WKO version.

```
# Eg: For installing WKO version 4.1.2 with a custom label
wlsko412=enabled, use below after configuring the right helm repo
$ helm install $WLSKO_NS \
  weblogic-operator/weblogic-operator \
  --namespace $WLSKO_NS \
  --version 4.1.2 \
  --set "domainNamespaceLabelSelector=wlsko412\=enabled"
```

Unregistering and Registering the Namespace with Weblogic Operator

Using your older CNTK, unregister your namespace forcefully with operator using "unregister-namespace.sh" script

```
# Set required variables with earlier CN values (before upgrade)
$ export OSM_CNTK=<osm_7.4.1/7.5.0_cntk>
$ export WLSKO_NS=<operator_namespace_before_upgrade>
$ export WLSKO_HOME=<path_for_operator_home>
$ $OSM_CNTK/scripts/unregister-namespace.sh -p $PROJECT -t wlsko -f
```

Install/upgrade your operator by following weblogic operator documentation.

Register OSM namespace with recent weblogic operator using "register-namespace.sh" script.

```
# Set required variables with OSM CN 8.0 values
$ export OSM_CNTK=<osm_8.0_cntk>
$ export WLSKO_NS=<operator_namespace_before_upgrade>
```

```
$ $OSM_CNTK/scripts/register-namespace.sh -p $PROJECT -t wlsko -l <custom-label>
```

Ingress Controller

OSM cloud native supports any annotations based ingress controller.

Support for Traefik (deprecated earlier) has been discontinued in favour of ingress controllers that support the generic Kubernetes ingress API. See "[Working with Ingress, Ingress Controller, and External Load Balancer](#)" for more details. If you were using Traefik earlier, and you wish to transition to such an Ingress Controller, you must delete ingresses, deregister the OSM project namespace from traefik, install the new ingress controller and, if required, register the OSM project namespace with the new controller. This will result in new ingresses being created. Further, it is recommended to uninstall the Traefik chart and delete the Traefik namespace before beginning the upgrade process. As part of upgrading, add the required annotations for the Ingress object to your 8.0 specification files, ensure Ingress objects in the namespace will be monitored by the replacement Ingress Controller, and create the new Ingress with the 8.0 toolkit using the `create-ingress.sh` script.

Updating Specification Files

This section focuses on the delta in the specification files between OSM 7.4.1, OSM 7.5.0 and new OSM 8.0 releases.

It is strongly recommended that you begin by copying over the sample specification files from the OSM 8.0 CNTK. You can then re-make the customizations from your current specification files one by one.

Updating the Project Specification

This section describes the sections that you update in the project specification:

Upgrading from 7.4.1 to 8.0

- OSM cloud native 8.0 adds new capabilities (such as OSMGW, RTUX and OCA microservices) for which there are corresponding sections in the project and instance specification files. See [Creating OSM Cloud Native Images](#) for more information.
- The default value for Ingress Controller has been set to **GENERIC**.
- If you use **GENERIC** ingress controller, the required annotations need to be added to the `ingress` section. In the project specification file, `ingress.annotations` lists the required annotations if you are using **NGINX** ingress controller. Adjust the annotations based on the ingress controller that you use.
- `crossDomainTrustUsers` section has been added which lists users for the **CrossDomainConnectors** group, required for cross-domain security and credential mapping. If you were using global trust before, it is recommended to switch to cross-domain trust for finer grained control.
- `fluentdLogging` section has been added that enables Fluentd sidecar logging by specifying image options for WebLogic and DB Installer pods.
- In OSM 7.4.1 cloud native, there is a separate step to deploy WME WAR file to the instance as a custom archive. In 8.0 cloud native, WebLogic Monitoring Exporter can be enabled and disabled by changing the value of **enabled** in the **instance.yaml**. By default, it is set to **true**, which creates a sidecar container for WME using the default image for the

version of WebLogic Operator in use. If you want to use a different image than the one that WKO uses, you have a provision to do that by updating WME in the project specification file.

- **openldap** authentication is deprecated; use **ldap** authentication instead. Refer to the `authentication` section.
- There are new capabilities added for deploying cartridges. OSM cloud native cartridge deployment supports container image built with par file. Either the **url** or the **Image** field should be populated for a particular cartridge but not both. See [Cartridge par Sources](#) for more details.
- A new section named `capabilitiesCartridgeList` has been added to the list capabilities cartridge. See About Dynamic Cartridge Assembly (Cloud Native Only) in OSM *Concepts*.
- Defining `osmWLSTargetNodes` restricts all OSM cloud native WebLogic pods and DB Installer pods to worker nodes, that match the label conditions and for these pods and it will take precedence over `osmcnTargetNodes`.
- Update the `requiredTargetSystems` section if your cartridge needs to integrate with external systems using REST API.
- Declarations for `secretNames`, `customFiles`, `safDestinationConfig`, `cartridges`, `certificates`, `certificates.secrets`, `uniformDistributedQueues`, and `uniformDistributedTopics` have been changed from `{}` (map) to `[]` (list).

Upgrading from 7.5.0 to 8.0

- OSM cloud native 8.0 adds the OCA microservice for which there are corresponding sections in the project and instance specification files. See [Creating OSM Cloud Native Images](#) for more information.
- You can update ingress annotations related to oca in `ingress.oca.annotations` based on your configuration.
- A new section named `capabilitiesCartridgeList` has been added to the list capabilities cartridge. See About Dynamic Cartridge Assembly (Cloud Native Only) in OSM *Concepts*.
- Double quotes have been removed from numeric values such as `threshold`, `duration`, and `resetDuration`, so they are now specified as integers instead of strings.
- Declarations for `safDestinationConfig`, `cartridgeUsers`, `uniformDistributedTopics`, `uniformDistributedQueues` have been changed from `{}` (map) to `[]` (list).

Updating the Instance Specification

This section describes the sections that you update in the instance specification:

Upgrading from 7.4.1 to 8.0

- The `serverStartPolicy` parameter now requires use of the correct case-sensitive value depending on your deployment version. Refer to the section in 8.0 **instance.yaml** file.
- If you are planning to use log masking functionality, uncomment and use the `logMaskingCustomRegexes` section.

- OSM supports fine grain control of the log level for OSM core using the instance specification file. Uncomment and use the log section for overriding the default log level.
- WebLogic Monitoring Exporter can be enabled and disabled by changing the value of `weblogicMonitoringExporter.enabled` in the **instance.yaml** file. By default, this is set to **true**.
- You can use FluentD as a sidecar container for reading the logs from OSM Servers and DB Installer. Enable the `fluentdLogging` section for making use of this functionality. See "[Configuring Fluentd Logging](#)" for more details.
- The storage volume must specify `pvc` to be used for persistent storage or as `emptydir` to share with a custom sidecar which is enabled via `sidecar.enabled` in instance specification file.

```
storageVolume:
  enabled: true
  type: pvc # Acceptable values are pvc and emptydir
  volumeName: storage-volume
  pvc: storage-pvc # Specify this only in case type is pvc
```

- If you want to enable a sidecar other than FluentD, make use of the section `sidecar.enabled`.
- In earlier versions, to enable global trust, the configuration you needed to use was `domainTrust.enabled`. While this is still supported for backward compatibility, it is deprecated in favor of the configuration `domainTrust.globalEnabled`. Oracle recommends that if you are using an instance specification from older versions, you should update it with this new configuration.
- A valid Custom Identity keystore must be provided. The use of `DemoIdentity` is no longer supported.

```
identity:
  name: keystore # Identity store filename without extension
  (keystore.jks)
```

- For enabling SSL using generic Ingress, add the annotations required for OSM. The `ssl.ingress.annotations` section in the instance specification file lists the sample annotations required for OSM if you are using NGINX ingress controller. Add the annotations based on the ingress controller that you choose.
- The `db` section now adds a required field type (`STANDARD` or `ADB`) to specify your database type. Fields like `serviceName`, `datasourcesSecondary`, and `rcuDb` are now included for expanded configuration. Refer to the `db` section for more details.
- In the 7.4.1 cloud native, the **instance.yaml** file contains the `loadBalancerPort` section. In OSM 8.0 cloud native, it is renamed to `inboundGateway`. As part of the `inboundGateway` section, you need to provide the host and port details of the actual ingress point or the load balancer.
- Using Host Aliases, you can achieve hostname resolution inside the pods. Refer to the `hostAliases` element and its comments in the instance specification for more details.
- Update the `osm-gateway`, `osmRuntimeUX` and `osm-cartridge-assembler` sections as per "[Configuring Target Systems for Events and System Interactions](#)".
- If you have defined `targetSystems` in the project specification file, provide the details of the respective target system in the instance specification file.

- The `serviceName` field in the `db` section and the `host` field under the `db.datasourcesPrimary` section has been made mandatory.

```
db:
  type: "STANDARD" # Acceptable values are STANDARD and ADB
  #serviceName: dbserver-servicename
  # This DB protocol is all applicable for all database connections.
  # Default value is TCP, Uncomment and change it to TCPS when required.
  # If TCPS is selected, the dbwallet "<project>-<instance>-db-ssl-wallet"
secret must exist
  #protocol: TCP
  # datasourcesPrimary section is applicable only for STANDARD DB. For
ADB, values will be used from Autonomous Database Serverless
secrets+configMap.
  datasourcesPrimary:
    port: 1521
    # Provide the DB server hostname/IP address
    #host: dbserver-ip
```

- Declarations for `secretNames` and `safConnectionConfig` has been changed from `{}` (map) to `[]` (list).

Upgrading from 7.5.0 to 8.0

- A valid Custom Identity keystore must be provided. The use of `DemoIdentity` is no longer supported.

```
identity:
  name: keystore # Identity store filename without extension
                (keystore.jks)
```

- The `serviceName` field in the `db` section and the `host` field under the `db.datasourcesPrimary` section has been made mandatory.

```
db:
  type: "STANDARD" # Acceptable values are STANDARD and ADB
  #serviceName: dbserver-servicename
  # This DB protocol is all applicable for all database connections.
  # Default value is TCP, Uncomment and change it to TCPS when required.
  # If TCPS is selected, the dbwallet "<project>-<instance>-db-ssl-wallet"
secret must exist
  #protocol: TCP
  # datasourcesPrimary section is applicable only for STANDARD DB. For
ADB, values will be used from Autonomous Database Serverless
secrets+configMap.
  datasourcesPrimary:
    port: 1521
    # Provide the DB server hostname/IP address
    #host: dbserver-ip
```

- A new `portConfig` section has been introduced under `inboundGateway.port` to enable detailed configuration of individual service ports when using the `port` access style at the ingress point. Refer to the instance specification file for more details.
- A new `osm-cartridge-assembler` section has been introduced. Refer to the instance specification file for more details.

- Declarations for `logMaskingCustomRegexes` have been changed from `{}` (map) to `[]` (list) and `log` has been changed from `[]` (list) to `{}` (map).

Copy the remaining sections for which there are no changes from your existing OSM cloud native version without making any changes to OSM cloud native 8.0 based on your usage of the respective functionalities in OSM cloud native 8.0.

Updating Shape Specification

If you have a custom shape specification, identify the standard shape on which your custom shape is based. Make a copy of this standard shape from the OSM 8.0 cloud native toolkit and make your customization one-by-one to it.

Upgrading to OSM Cloud Native 8.0

Oracle recommends that you upgrade the existing OSM 7.4.1 or 7.5.0 cloud native application to 8.0 cloud native using the same project namespace and the instance name.

Prerequisites for the Upgrade

Check *OSM Compatibility Matrix* for details about the required and supported versions of the pre-requisite software, before proceeding with the upgrade.

Note

Perform the following steps using the existing 7.4.1 or 7.5 Cloud Native toolkit specification files.

Delete the Instance

To delete an instance, run the following:

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

Delete the Ingress

To delete an ingress, run the following:

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

If you are switching from Traefik as part of this upgrade, to stop Traefik from monitoring this namespace, run the following:

```
$OSM_CNTK/scripts/unregister-namespace.sh -t traefik -p project
```

Preparation Steps for the Upgrade

This section lists preparatory steps for the upgrade:

- Take a backup of the working OSM 7.4.1 or 7.5.0 project and instance specification files or rely on your source code management system to maintain versioning.

- Take the OSM 7.4.1 or 7.5.0 DB schema backup before the upgrade.
- Take a backup of the following secrets and also of any custom OSM secrets created for 7.4.1 or 7.5.0 cloud native before the upgrade.

```
<project>-<instance>-database-credentials
<project>-<instance>-embedded-ldap-credentials
<project>-<instance>-weblogic-credentials
<project>-<instance>-rcudb-credentials
<project>-<instance>-opss-wallet-password-secret
<project>-<instance>-runtime-encryption-secret
<project>-<instance>-osmcn-cred-<user> (If you have this secret created.
Ignore, if the secret do not exist.)
<project>-<instance>-saf-<remote-system> (If you have this secret created.
Ignore, if the secret do not exist.)
```

Note

If you are upgrading from an older version of OSM cloud native and have OpenLDAP, note that it was deprecated in 7.5 and is now discontinued. You need to switch to Generic LDAP.

- Run the following command on each secret which would copy the output to the *secret.yaml* file. Maintain a separate file for each secret.

```
kubectl get secret -n project secretName -o yaml > secret.yaml
```

- Set the **\$OSM_CNTK** environment variable to point to the 8.0 cloud native toolkit.
- Copy the specification files updated in your earlier steps to your **\$SPEC_PATH** location and rename them to match the same as existing 7.4.1 or 7.5.0 cloud native specification files (to match the existing project name and the instance name).
- (Optional) If you use a newer version of WKO, install the newer version WKO in a new namespace and register the project namespace with it.
- (Optional) If you replace the existing ingress with another ingress controller, install the new ingress controller.

Note

You need to provide the following GRANT on the dba user which has been used in the secrets created for osmdb and rcudb. This is required for RCU Schema creation.

```
GRANT execute on dbms_lob to _replace_this_text_with_admin_name_ with
grant option;
```

OSM no longer requires the sysdba role for its DB schema owner. This role can be safely rescinded. To rescind the role, run the following command:

```
REVOKE sysdba from _replace_this_text_with_admin_name_;
```

Updating the Secrets

Note

Starting with the below steps, you will use the OSM 8.0 cloud native toolkit and specification files.

Update Existing Secrets

As a pre-requisite to using the toolkit for upgrading the OSM 7.4.1 or 7.5.0 cloud native environment, you must update the below secrets for access to the following:

- OSM database
- RCU DB
- OSM system users

Secrets can be updated using `manage-instance-credentials` as below:

```
$OSM_CNTK/scripts/manage-instance-credentials.sh -p project -i instance -s  
spec_path update osmdb,rcudb,osmldap
```

In OSM 8.0 cloud native:

- Database host, port and service name have been moved from secrets to instance specifications. OSM and RCU schemas will be created on the same database.
- When you choose *osmldap*, the script prompts the password for a new user "osm-gateway-internal", which is used to access the OSM Gateway microservices.

Creating New Secrets

This section describes how to create new secrets and credentials.

Note

For more details about these secrets, see [Reference of Secrets Created by the Scripts](#).

Creating Secrets for an OSM Upgrade

If you are upgrading from a release prior to OSM 7.5.0, you should create the following secrets. If the upgrade is from OSM 7.5.0, you can update these existing secrets.

To create a new secret, run the following command:

```
$OSM_CNTK/scripts/manage-instance-credentials.sh -p project -i instance -s  
spec_path create secret
```

To update an existing secret, run the following command:

```
$OSM_CNTK/scripts/manage-instance-credentials.sh -p project -i instance -s  
spec_path update secret
```

Clean Up OIDC Secret

If you are upgrading from a release prior to OSM 7.5.0, you need to create OIDC secrets. If you are upgrading from OSM 7.5, you need to delete the existing OIDC secret, `project-instance-oidc-credentials`, using the OSM 7.5 cloud native toolkit or manually. You can use the following command to delete the existing OIDC Secret using the OSM 7.5 cloud native toolkit:

```
$OSM_CNTK/scripts/manage-instance-credentials.sh -p project -i instance -s  
spec_path delete oidc
```

or,

```
kubectl delete secret -n project project-instance-oidc-credentials
```

After deleting the existing secret, you need to re-create the required OIDC secrets using the OSM 8.0 CNTK. You need to create the new OIDC secrets to the APIs exposed by the Gateway, RTUX and OCA microservices. For more information on how to create the new secrets, see *Creating OSM Secrets for OIDC* in [Configuring OpenID Connect for OSM Microservices](#)

Creating TLS Secrets

If you have SSL turned on for incoming connections (by setting `ssl.incoming` to true in the specification files), then you must create `wlstls` and `gatewaytls` secrets as below to provide the required certificate information. This information is provided securely to the ingress controller.

This secret carries the application or ingress TLS credentials for OSM.

Note

If you have the secrets `project-instance-osm-tls-cert`, `project-instance-admin-tls-cert` and `project-instance-t3-tls-cert` already created as part of 7.4.1, skip this step and proceed to `gatewaytls` secret creation. In 7.5.0, if you have already created the `gatewaytls` secret, you do not need to perform the `gatewaytls` secret creation step again.

```
$OSM_CNTK/scripts/manage-instance-credentials.sh -p project -i instance -s  
spec_path create wlstls
```

This carries the application and ingress TLS credentials for microservices.

```
$OSM_CNTK/scripts/manage-instance-credentials.sh -p project -i instance -s  
spec_path create gatewaytls
```

Creating FluentD Credentials

If FluentD is enabled as a sidecar, then create the credentials for its connection to Elastic Search server.

```
$OSM_CNTK/scripts/manage-instance-credentials.sh -p project -i instance -s  
spec_path create fluentd
```

Creating Target Systems Credentials

If security schemes for the target systems are enabled, then create secrets for each target system separately.

```
$OSM_CNTK/scripts/manage-target-system-credentials.sh -p project -i instance -  
n securitySchemeName -t authenticationType create
```

Creating SAML SSO Secret

If SAML SSO is enabled, you should create a secret that carries the archive file that is needed by the OSM Weblogic domain to support SAML2 SSO.

```
$OSM_CNTK/scripts/manage-instance-credentials.sh -p project -i instance -s  
spec_path create samlssso
```

Creating DB Wallet Secret

If the `db.protocol` is TCPS, you need to create a secret to carry the wallet that is needed for OSM to make the connection with the database.

```
$OSM_CNTK/scripts/manage-instance-credentials.sh -p project -i instance -s  
spec_path create dbwallet
```

Upgrading the OSM DB Schema

Make sure you have the updated specification files for OSM 8.0 in your `$SPEC_PATH`.

Run the following command to upgrade the existing OSM schema:

```
$OSM_CNTK/scripts/install-osmdb.sh -p project -i instance -s $SPEC_PATH -c 1
```

Back-out procedure

Perform this procedure if the DB Schema upgrade does not work as expected. Use the 8.0 cloud native toolkit and specification files for this procedure.

To perform back-out procedure:

1. Drop the OSM Schema using DB installer code 8.

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

2. Delete all the secrets that were created while updating the secrets (including fluentd, gatewaytls, target systems).

```
$OSM_CNTK/scripts/manage-instance-credentials.sh -p project -i instance -s
spec_path \ delete \
oidc,osmdb,rcudb,wlsadmin,osmlap,opssWP,wlsRTE,fluentd,gatewaytls
```

```
$OSM_CNTK/scripts/manage-target-system-credentials.sh -p project -i
instance -n security_scheme_name delete (optional)
```

3. Restore the OSM 7.4.1 or 7.5.0 installation schema backup that was taken during the "[Preparation Steps for the Upgrade](#)" task.
4. Copy the 7.4.1 or 7.5.0 cloud native backup specification files to **\$SPEC_PATH** location and set the **\$OSM_CNTK** variable to point to the cloud native toolkit of your 7.4.1 or 7.5.0 OSM cloud native (before upgrade) environment.
5. Import all the secrets from the backup files taken before the upgrade. Run the following command using each secret file to import the secret.

```
kubectl apply -f secret.yaml
```

6. Create the instance.

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

7. Create the ingress.

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

OSM Application Upgrade

Make sure you have the updated specification files for OSM 8.0 in your **\$SPEC_PATH**.

Run `$OSM_CNTK/scripts/create-instance.sh` to create an instance with OSM 8.0 cloud native artifacts.

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

Once the `create-instance.sh` script completes running, admin and managed server pods will be in the running state.

Create the ingress.

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

Back-out procedure (in case OSM application upgrade didn't work as expected):

With 8.0 cloud native toolkit and specification files, follow below steps:

1. Delete the OSM cloud native instance.

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

2. Perform the DB backout procedure as described above.

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)

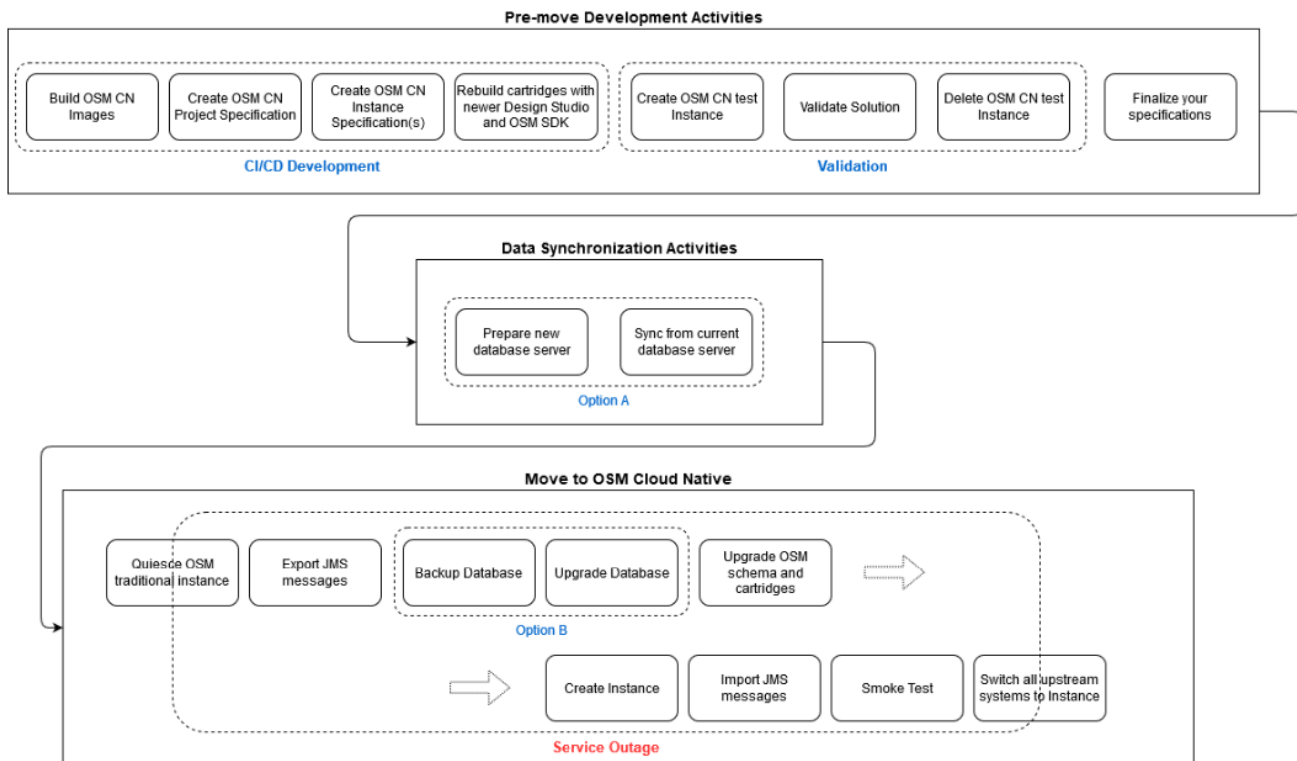
- Deleting the test OSM cloud native instance (cloud native task)
- Finalizing your specifications (cloud native and solution task)
- Data synchronization activities, which include the following tasks:
 - Preparing a new database server (database task)
 - Synchronizing the current database server (database 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 14-1 Move to OSM Cloud Native Process



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 container image and the DB Installer container 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. Upgrade your cartridges only if they were built using an OSM SDK older than version 7.3.5. If you need to build or re-build cartridges for OSM 8.0, use the OSM 8.0 SDK and set the Design Studio target version for OSM as 8.0. In general, use the Design Studio target version for OSM that is closest to the current OSM version.

Note

You need to upgrade or rebuild cartridges whose Java plugins or emulators currently depend on JDK 8 or earlier and Java EE 7 or earlier libraries. This is to ensure that all such components are migrated to use Java 21 and Java EE 8 libraries for compatibility with FMW 14.1.2 and Java 21.

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**. 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.

You can migrate the JMS messages in any of the following ways:

- [Migrating JMS Messages By Using the Cloud Native Toolkit](#)
- [Migrating JMS Messages By Using the WebLogic Remote Console](#)

Migrating JMS Messages By Using the Cloud Native Toolkit

This section describes the steps for migrating JMS messages by running the scripts provided with the OSM cloud native toolkit.

Note

This procedure is applicable if you are moving from OSM traditional releases 7.3.x.x.x and 7.4.x.x.x. to OSM cloud native release 7.4.1.0.8 and later. It is also applicable if you are moving from OSM traditional 7.5.x.x.x to OSM cloud native 7.5.0 or later

The scripts you run and the steps you perform are different for the options (Option A and Option B) illustrated in [Figure 12-1 Move to OSM Cloud Native Process](#) and your choice.

See the following topics for the details:

- [Prerequisites for Running the Migration Scripts](#)
- [Using a New Database Server for the Cloud Native Instance \(Option A\)](#)
- [Upgrading the Database Server for the OSM Traditional Environment and Using it for the Cloud Native Instance \(Option B\)](#)

Prerequisites for Running the Migration Scripts

Before performing the steps for the option you have chosen, do the following:

- Update the `$OSM_CNTK/samples/migration/config.sh` file, provided with the toolkit, with details about the OSM traditional and cloud native environments.
- Ensure that all **In progress** orders are completed to the extent possible. This minimizes the impact on the systems.
- Grant sufficient RBAC permissions, equivalent to an Admin role, for the user who is running the scripts on the cloud native instance (bastion host or equivalent).
- In WebLogic Remote Console, select **Edit Tree**. From **Edit Tree**, select **Environment**. In the **Environment** section, select **Servers**. From **Servers**, select **admin**. Here, open the **Protocols** tab to access the settings for the admin server of your OSM traditional deployment and select **Enable Tunneling**.
- Ensure that the database snapshot of the cloud native system is taken. Alternatively, the cloud native database can be rebuilt if there is no prior data to preserve. This would be useful in case of a failure when importing the messages.
- Verify the specification files to ensure that all user-defined queues (both JMS and SAF) on the traditional OSM instance are present on the cloud native instance. The queue destinations do not have to match.

Using a New Database Server for the Cloud Native Instance (Option A)

If you choose to use a new database server for the cloud native instance, do the following:

1. Ensure that the instances of both traditional and cloud native deployments are up and running.
2. Ensure that the required HTTP and HTTPS ports are enabled in the firewall for connecting to the traditional system from the cloud native instance.
3. Run the following script from the bastion host or an equivalent to migrate the JMS messages:

```
$OSM_CNTK/samples/migration/migrate-jms-messages.sh command
```

The script contains commands that can perform full migration or individual operations such as exporting, importing, validating, and resuming queues.

Run `$OSM_CNTK/samples/migration/migrate-jms-messages.sh -h` for details about the commands to use with the script.

Upgrading the Database Server for the OSM Traditional Environment and Using it for the Cloud Native Instance (Option B)

If you choose to upgrade the database server of your traditional deployment and use it for the cloud native instance, do the following:

1. Ensure that the admin server and the managed servers of the OSM traditional deployment are up and running.
2. Copy the following scripts to a server on which Oracle Fusion Middleware is installed:
 - **\$OSM_CNTK/samples/migration/export-jms-messages.sh**
 - **\$OSM_CNTK/samples/migration/config.sh**
3. Update the **config.sh** file with details about the OSM traditional environment.
4. Run the following script, which exports the JMS messages from the traditional OSM environment:

`$OSM_CNTK/samples/migration/export-jms-messages.sh` *command*

Run `$OSM_CNTK/samples/migration/export-jms-messages.sh -h` for details about the commands to use with the script.

5. If the messages are exported successfully and you want to proceed with the migration, shut down the OSM traditional deployment servers. If there are issues with the export, run the script again.
6. Ensure the server of the OSM cloud native deployment is up and running.
7. After the script finishes exporting the JMS messages, copy the migration scripts from **\$OSM_CNTK/samples/migration** and the output files from the OSM traditional system to the admin server pod of the OSM cloud native instance manually.
8. Update the path to the exported files in the **\$OSM_CNTK/samples/migration/config.sh** file.
9. Stop the database jobs.
10. On the OSM cloud native admin server pod, run the following script, which imports the JMS messages into the OSM cloud native deployment:

`$OSM_CNTK/samples/migration/import-jms-messages.sh` *command*

Run `$OSM_CNTK/samples/migration/import-jms-messages.sh -h` for details about the commands to use with the script.

11. If the messages are imported into the OSM cloud native instance without errors and you want to transition to the cloud native deployment, resume the queues on the OSM cloud native instance and start the database jobs back up.
Resume the queues on the cloud native instance using the appropriate options with the **migrate-jms-messages.sh** script or the **import-jms-messages.sh** script.

If the import fails or if you find any errors while running the script, do the following:

- a. Make sure that copies of the exported files exist outside of the OSM cloud native admin pod.
- b. Shut down the OSM cloud native instance.
- c. Restore the OSM cloud native database from backup. Alternatively, if there is no prior data, use the **install-osmdb.sh** script in the cloud native toolkit to delete the OSM and RCU database schemas and recreate them.
- d. Start the OSM cloud native servers.
- e. Copy the exported files to the admin server pod.
- f. Run the `$OSM_CNTK/samples/migration/import-jms-messages.sh` script using the appropriate commands.

Migrating JMS Messages By Using the WebLogic Remote Console

This section describes the steps for migrating JMS Messages by using the WebLogic Remote Console.

Perform the following tasks to migrate JMS messages:

- [Exporting JMS Messages](#)
- [Importing JMS Messages](#)

Exporting JMS Messages

For a release prior to OSM 8.0, 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.

For more information on exporting JMS messages, for a release prior to OSM 8.0, refer to [Moving to OSM Cloud Native from a Traditional Deployment](#) and *Oracle Fusion Middleware Administration Console Online Help for Oracle WebLogic Server 12.2.1.4.0*.

For OSM 8.0, before exporting JMS messages, validate that your OSM traditional instance has the WebLogic July 2025 CPU patches installed. These patches are required to view the message content. If it is not installed, follow the standard WebLogic patch procedures to procure and install the patch.

To export JMS messages for OSM 8.0:

1. Login to the WebLogic Remote Console and navigate to the **Monitoring Tree**.
2. In the left navigation pane, select **Services**. From **Services**, select the **Messaging** section. In the **Messaging** section, select the **JMS Runtime** section.
3. Find and select the JMS Server that hosts your target queue. For example: `osm_jms_server@ms1`
4. Under the selected server, click on **Destinations**. This displays a list of all destinations on that server.
5. Click the name of the desired destination. Select the queue and then go to the **Messages** tab. If there are any messages pending in the destination of this queue, click the **Export** button to export all the messages to a file. Note the queue name and destination in the file name for ease of tracing.
6. If you have defined other JMS Modules as part of your solution, repeat steps [2](#) to [5](#) 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. Login to the WebLogic Remote Console and navigate to the **Monitoring Tree**.

3. In the left navigation pane, select **Services**. From **Services**, select the **Messaging** section. In the **Messaging** section, select the **JMS Runtime** section.
4. Find and select the JMS Server that hosts your target queue. For example: `osm_jms_server@ms1`.
5. Under the selected server, click on **Destinations**. This will display a list of all destinations on that server.
6. For each destination on this queue for which you have an export file, find the same destination in the list. Select the queue then go to **Messages** tab.
7. Click **Import** to specify the filename and import the messages.
8. 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_LL_%');
```

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.

Note

You need to upgrade or rebuild cartridges whose Java plugins or emulators currently depend on JDK 8 or earlier and Java EE 7 or earlier libraries. This is to ensure that all such components are migrated to use Java 21 and Java EE 8 libraries for compatibility with FMW 14.1.2 and Java 21

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.

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 NGINX, OSM UI, and WebLogic Remote Console
- Recovering an OSM Cloud Native Database Schema
- Resolving Improper JMS Assignment
- 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 **`/logMount/project-instance/performance/$server_name`**.

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 NGINX, OSM UI, and WebLogic Remote Console

This section describes how to troubleshoot issues with access to the OSM UI clients, WLST, and WebLogic Remote 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 Remote Console, when the Oracle Cloud Infrastructure load balancer is used and not used:

Table 15-1 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	<code>http://instance.project.osm.org:30305/OrderManagement</code>	<code>http://instance.project.osm.org:80/OrderManagement</code>
WLST	<code>http://t3.instance.project.osm.org:30305</code>	<code>http://t3.instance.project.osm.org:80</code>
WebLogic Remote Console	<code>http://admin.instance.project.osm.org:30305/rconsole</code>	<code>http://admin.instance.project.osm.org:80/rconsole</code>

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 Remote Console and navigate from the **Monitoring Tree** to **Deployments**. From **Deployments**, navigate to **App Management** and check if the **State** column for **oms** shows **Active**. To validate the name of the cluster in the **Targets** column, navigate from **Edit Tree** to **Deployments**. In the **Deployments** tab, select **App Deployments**. 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 health 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 NGINX is running and its status:

```
kubectl get all --namespace nginx
```

NAME	STATUS	RESTARTS	AGE	READY
pod/nginx-operator-ingress-nginx-controller-546cc856c9-q5rk8	Running	0	56d	1/1

NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	TYPE	AGE
service/nginx-operator-ingress-nginx-controller	10.97.152.128	<pending>	80:30305/TCP,443:30443/TCP	LoadBalancer	56d
service/nginx-operator-ingress-nginx-controller-admission	10.109.25.98	<none>	443/TCP	ClusterIP	56d

NAME	DATE	AVAILABLE	AGE	READY	UP-TO-
deployment.apps/nginx-operator-ingress-nginx-controller	1	1	56d	1/1	

4. Check and Install Ingress:

- a. Check for an Existing Ingress. To see if an Ingress resource already exists for your instance, run:

```
kubectl get ingress -n <project> --no-headers | grep '^<project>-<instance>.*-ingress'
```

Look for Ingress names related to your instance. For more information about the ingresses that are expected to be present, refer to [Reference of Secrets Created by the Scripts](#).

If you see expected Ingress(es) listed, you do not need to create a new one. Continue to the next step.

- b. Check NGINX Annotations. Make sure all required NGINX annotations in your project specification and instance specification files are uncommented. These annotations are needed for the Ingress to work correctly.
- c. Create the Ingress. If the expected Ingress for your instance is missing, or if you had fixed or updated any NGINX annotations, run the following command to (re)create or update your Ingress:

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

The -u flag safely updates or installs the Ingress as needed.

- d. Confirm Ingress Creation. After completing the above steps, refer to Step A and confirm that the expected ingresses are present for your instance.

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, if PV is not configured, logs are available at: **`/u01/oracle/user_projects/domains/domain/servers/ms1/logs`** and **`/u01/oracle/user_projects/domains/domain`**. If PV is configured, logs are available at **`/logMount/project-instance/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 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_msn-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_msn-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 15-1 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 WebLogic 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

There may be a case where introspector doesn't start. This could mean that the operator is not monitoring your namespace or your namespace is not tagged to the correct label which the operator is monitoring.

For more information about operator monitoring, see: <https://oracle.github.io/weblogic-kubernetes-operator/managing-operators/common-mistakes/#forgetting-to-configure-the-operator-to-monitor-a-namespace>

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
project-instance-introspect-domain-job-r2d6j	0/1	ErrImagePull
0	5s	

OR

NAME	READY	STATUS
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:

```
WLSDPY-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.

```
WLSPLY-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 and 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**.

For information about changing the log level for debugging, see the documentation at: <https://oracle.github.io/weblogic-kubernetes-operator/managing-operators/troubleshooting/#operator-and-conversion-webhook-logging-level>.

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.

To avoid common mistakes during the installation of WKO, refer to the WKO troubleshooting information at: <https://oracle.github.io/weblogic-kubernetes-operator/managing-operators/common-mistakes/#forgetting-to-configure-the-operator-to-monitor-a-namespace>.

To uninstall WKO, follow the steps in WKO documentation at: <https://oracle.github.io/weblogic-kubernetes-operator/managing-operators/installation/#uninstall-the-operator>.

Re-register your namespaces using the **register-namespace.sh** and **unregister-namespace.sh** scripts in the cloud native toolkit.

You can install the operator by following the instructions in WKO documentation at: <https://oracle.github.io/weblogic-kubernetes-operator/managing-operators/installation/#install-the->

[operator](#). Then, register all the projects again, one by one. See "[Registering the Namespace](#)" for details.

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. For details about WLST Command Reference for Infrastructure Security, see Oracle Fusion MiddleWare documentation. 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 OSInternalEventsMDB 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.JMSDestinationListene
r> <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.SimpleResponseQueue
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.JMSDestinationListene
r> <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.SimpleResponseQueue
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.SimpleResponseQueue
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.JMSDestinationListene
r> <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.SimpleResponseQueue
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.SimpleResponseQueue
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.

Image Build Failure Due to OPatch Error

You may face the following error while building images using the OSM cloud native builder toolkit:

```
OPatch failed with error code 73
[SEVERE ] Build command failed with error: OPatch failed to restore OH '/u01/
oracle'. Consult OPatch document to restore the home manually before
proceeding.
UtilSession failed: ApplySession failed in system modification phase...
'ApplySession::apply failed:
oracle.glcm.opatch.common.api.install.HomeOperationException: A failure
occurred while processing patch: 31676526'
Error: building at STEP "RUN /u01/oracle/OPatch/opatch apply -silent -oh /u01/
oracle -nonrollbackable /tmp/imagetool/patches/p31676526_122140_Generic.zip":
while running runtime: exit status 73
```

The root cause for this error is that Podman's default value for number of open files is too low for an OPatch invocation.

The way to resolve this error is to configure the build system's Linux to have a higher hard limit for open files. The current hard limit on the number of open files can be known by running `ulimit -n -H` on the host. For more information, refer to [Prerequisites for Creating OSM Images](#).

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 to demonstrate how to integrate with external LDAP server for human users.

- **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 <code>\$OSM_CNTK/samples/credentials/manage-cartridge-credentials.sh</code> . Use the <code>secret</code> option in the user text file. See Provisioning Cartridge User Accounts for more details.
userAdmin	Sample script in <code>\$OSM_CNTK/samples/credentials/manage-cartridge-credentials.sh</code> . Use the <code>ldap</code> option in the user text file. See Provisioning Cartridge User Accounts for more details.
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 Remote 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 Remote 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**

If a cartridge defines and uses model variables, these variables are always set to their default value when the cartridge is deployed into a CN environment. The default value is as specified by the cartridge developer as part of the definition of the model variable. If no default value is specified, the model variable is left as-is, not substituted.

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.

B

Reference of Secrets Created by the Scripts

The secrets created by the OSM cloud native toolkit scripts follow the naming pattern of `<project>-<instance>-<suffix>`, where the "suffix" differentiates between the secrets.

The following table lists the secrets, describes their purpose, and provides other details.

Secret Name	Purpose	Must Have?	Creation	Details
<code><project>-<instance>-database-credentials</code>	Credentials required for OSM DB schemas.	Yes	<code>manage-instance-credentials osmdb</code>	DB Credentials Secret
<code><project>-<instance>-gateway-credentials</code>	Credentials and connection details for the OIDC IdP to secure TMF and Fallout Exception REST APIs.	No	- Required if <code>omsConfig.project.osm_runtime_type</code> is <code>MultiService</code> <code>manage-instance-credentials gw0idc</code>	OSM Gateway OIDC Credentials Secret
<code><project>-<instance>-oca-credentials</code>	Credentials and connection details for the OIDC IdP to secure OSM Cartridge Assembler (OCA) REST APIs.	No	- Required if <code>omsConfig.project.osm_runtime_type</code> is <code>MultiService</code> and <code>osm-cartridge-assembler.enabled</code> is <code>true</code> <code>manage-instance-credentials.sh oca0idc</code>	OCA OIDC Credentials Secret
<code><project>-<instance>-rcudb-credentials</code>	Credentials required for FMW RCU DB schemas.	Yes	<code>manage-instance-credentials rcudb</code>	RCU DB Credentials Secret
<code><project>-<instance>-db-ssl-wallet</code>	Credentials and certificates required to connect with TCPS enabled database.	No	<code>manage-instance-credentials</code> <code>dbwallet</code>	TCPS DB Wallet Secret
<code><project>-<instance>-weblogic-credentials</code>	WebLogic admin credential.	Yes	<code>manage-instance-credentials wlsadmin</code>	WebLogic Credentials Secret
<code><project>-<instance>-runtime-encryption-secret</code>	Password used to secure instance metadata in Kubernetes.	Yes	<code>manage-instance-credentials wlsRTE</code>	WebLogic Runtime Encryption Secret
<code><project>-<instance>-opss-wallet-password-secret</code>	Password used to encrypt the FMW wallet.	Yes	<code>manage-instance-credentials opssWP</code>	FMW Wallet Encryption Secret

Secret Name	Purpose	Must Have?	Creation	Details
<project>-<instance>-opss-walletfile-secret	Secure storage of FMW wallet.	No	- Automatically during create-instance - Manually using manage-instance-credentials opssWF	FMW Secure Wallet Secret
<project>-<instance>-embedded-ldap-credentials	Passwords for OSM's internal users.	Yes	manage-instance-credentials osmlldap	OSM Internal User Passwords Secret
<project>-<instance>-fluentd-credentials	Credentials and connection details to the ElasticSearch service.	No	- Required if fluentdLogging.enabled is true - manage-instance-credentials fluentd	OSM Fluentd Credentials Secret
<project>-<instance>-app-tls-cert	Certificate and key to access OSM TMF REST APIs, Fallout Exception APIs and UX backend APIs.	No	- Required if ssl.incoming is true - manage-instance-credentials gatewaytls	Certificate and Key to Access the Gateway HTTPS Endpoint
<project>-<instance>-osm-tls-cert	Certificate and key to access the OSM HTTPS endpoint.	No	- Required if ssl.incoming is true - manage-instance-credentials wlstls (with option WLSIngress or Both)	Certificate and Key to Access the OSM HTTPS Endpoint
<project>-<instance>-admin-tls-cert	Certificate and key to access the OSM WebLogic Remote Console HTTPS endpoint.	No	- Required if ssl.incoming is true - manage-instance-credentials wlstls (with option WLSIngress or Both)	Certificate and Key to Access the OSM WebLogic Remote Console HTTPS Endpoint
<project>-<instance>-t3-tls-cert	Certificate and key to access the OSM t3 over HTTPS endpoint.	No	- Required if ssl.incoming is true in the specification - manage-instance-credentials wlstls (with option WLSIngress or Both)	Certificate and key to access the OSM t3 over HTTPS
<project>-<instance>-truststore	Providing OSM with trusted CAs for secure outbound JMS/SAF	No	- Required if ssl.trust is populated in the specification - manage-instance-credentials wlstls (with option WLSStore or Both)	Trusted CA Injection

Secret Name	Purpose	Must Have?	Creation	Details
<project>-<instance>-keystore	Providing OSM with private keys for secure outbound JMS/SAF or SAML IdP.	Yes	- Required if <code>ssl.identity.name</code> or SAML SSO is enabled. - Required when SSL is enabled, or a custom trust keystore is configured. <code>manage-instance-credentials wlstls</code> (with option <code>WLSStore</code> or <code>Both</code>)	Secure Identity
<project>-<instance>-db-wallet	Secure storage of details to connect to the ADB database.	No	- Required if <code>adb</code> is used for the OSM instance <code>manage-instance-credentials osmdb</code>	ADB Wallet Secret
<project>-<instance>-db-secret	ADB administrator password.	No	- Required if <code>adb</code> is used for the OSM instance <code>manage-instance-credentials osmdb</code>	ADB Admin Secret
<project>-<instance>-osmcn-cred-<user>	Credentials for custom users defined by the cartridge. Credentials required by the cartridge accessed from the map named "osm"	No	- Required if <code>cartridgeUsers</code> is specified, or if cartridge code uses <code>getOsmCredentialPassword</code> <code>manage-cartridge-credentials</code> with <code>cartridgeUsers:</code> <code>"osm:_sysgen_<username>:secret:<group-list>"</code> <code>getOsmCredentialPassword:</code> <code>"osm:_sysgen_<username>:secret"</code>	Cartridge Defined Custom User Credentials
<project>-<instance>-ldap-credentials	Information required for OSM to use an external LDAP for human user credentials	No	- Required if <code>authentication.ldap.enabled</code> is <code>true</code> <code>manage-osm-ldap-credentials -c create -l ldap</code>	External LDAP Information
<project>-<instance>-saf-<remote-system>	Credentials to establish SAF connectivity to <remote-system>	No	- Required if secret is named in <code>safConnectionConfig.secretName</code> - Create manually	SAF Credentials

Secret Name	Purpose	Must Have?	Creation	Details
<project>-<instance>-global-trust-credentials	Shared password for configuring global trust	No	- Required if <code>domainTrust.enabled</code> (deprecated) or <code>domainTrust.globalEnabled</code> is true - Create manually	Global Trust Credentials
<remote-domain-secret>	User credentials for the cross-domain users in remote domain	No	- Required if <code>domainTrust.crossDomain.enabled</code> is true - Create manually	Cross Domain Users in Remote Domains
<project>-<instance>-crossdomain-users	User credentials for the cross-domain users in OSM cloud native	No	- Required if <code>domainTrust.crossDomain.enabled</code> is true and you have a list of cross-domain users in the specification <code>manage-instance-credentials xtrust</code>	Xtrust Secret
<repository-access-secret>	Credentials to access a repository	No	- Required if secret is named in <code>cartridges.[].secret</code> or <code>partitionStatistic.secret</code> - Create manually	Generic Credentials
<project>-<instance>-<securityScheme>	Secrets for establishing connections to target systems that are defined in the security scheme.	No	- Required for each <code>targetSystems.securitySchemes.[].name</code> <code>manage-target-system-credentials.sh</code>	Security Scheme Credentials
<project>-<instance>-ssoaml-archive	Secure information for OSM to communicate with SAML IdP.	No	- Required if <code>sso.enabled</code> is true in the specification <code>manage-instance-credentials samlso</code>	SAML Archive for IdP
<git-repo-secret>	Credentials to access a repository.	No	- Required if using Continuous Delivery and GitRepository resources in Flux-CD configuration refers to a secured Git repository. - Create manually.	Git Access Secret

OSM Gateway OIDC Credentials Secret

Credentials and connection details for the OIDC IdP to secure TMF and Fallout Exception REST APIs.

<project>-<instance>-gateway-credentials

```
oidc-base-url: <the oidc base url>
oidc-access-token-url: <the token access url>
tmf-oidc-client-id: <the oidc client id>
tmf-oidc-client-secret: <the oidc client secret>
fallout-oidc-client-id: <the oidc client id>
fallout-oidc-client-secret: <the oidc client secret>
```

OCA OIDC Credentials Secret

Credentials and connection details for the OIDC IdP to secure OCA REST APIs.

<project>-<instance>-oca-credentials

```
oidc-base-url: <the oidc base url>
oidc-access-token-url: <the token access url>
oca-oidc-client-id: <the oidc client id>
oca-oidc-client-secret: <the oidc client secret>
scd-oidc-access-token-url: <optional access token url of SCD app>
scd-oidc-client-id: <optional client id of SCD app>
scd-oidc-client-secret: <optional client secret of SCD app>
scd-oidc-scope: <optional scope of SCD app>
```

DB Credentials Secret

Credentials required for OSM DB schemas.

<project>-<instance>-database-credentials

```
db_password: <osmschema-user-password>
db_reports_password: <reportsschema-user-password>
db_reports_user: <reportsschema-user-name>
db_rule_password: <ruleschema-user-password>
db_rule_user: <ruleschema-user-name>
db_service_name: <db-service-name>
db_user: <osmschema-user-name>
dba_password: <dbadmin-password>
dba_user: <dbadmin-user-name>
is_adb: <Y/N> -- Y for yes, N for No.
```

RCU DB Credentials Secret

Credentials required for FMW RCU DB schemas.

<project>-<instance>-rcudb-credentials

```
is_adb: <Y/N> -- Y for yes, N for No.
rcu_prefix: <unique-prefix-for-this-instance>
rcu_schema_password: <password-for-all-rcu-schemas>
```

TCPS DB Wallet Secret

Credentials and certificates required to connect with TCPS enabled database.

<project>-<instance>-db-ssl-wallet

ewallet.p12: <credentials and certificates required to connect with DB>
cwallet.sso: <provides ability to open the wallet without password>

WebLogic Credentials Secret

WebLogic admin credential.

<project>-<instance>-weblogic-credentials

password: <weblogic-admin-password>
username: <weblogic-admin-username>

WebLogic Runtime Encryption Secret

Password used to secure instance metadata in Kubernetes.

<project>-<instance>-runtime-encryption-secret

password: <runtime-encryption-password>

FMW Wallet Encryption Secret

Password used to secure instance metadata in Kubernetes.

<project>-<instance>-opss-wallet-password-secret

walletPassword: <wallet-encryption-password>

FMW Secure Wallet Secret

Secure storage of FMW wallet.

<project>-<instance>-opss-walletfile-secret

walletFile: <encrypted-wallet>

OSM Internal User Passwords Secret

Passwords for OSM's internal users.

<project>-<instance>-embedded-ldap-credentials

```
automation_password: <password for oms-automation user>
gateway_internal_password: <password for gateway internal user>
gateway_internal_user: <username for gateway internal user>
internal_password: <password for oms-internal user>
metrics_password: <password for metrics user>
omsadmin_password: <password for omsadmin user>
sceadmin_password: <password for sceadmin user>
```

OSM OIDC Credentials Secret

Credentials and connection details for the OIDC IdP in order to secure TMF and Fallout Exception REST APIs.

<project>-<instance>-oidc-credentials

```
app-oidc-audience: <the oidc audience>
app-oidc-base-url: <the oidc base url>
app-oidc-client-id: <the oidc client id>
app-oidc-client-secret: <the oidc client secret>
client-oidc-access-token-url: <the token access url>
client-oidc-scope: <the scope>
```

OSM Fluentd Credentials Secret

Credentials and connection details to the ElasticSearch service.

<project>-<instance>-fluentd-credentials

```
elasticsearchhost: <host name of the elastic search server>
elasticsearchpassword: <password to access the elastic search service>
elasticsearchport: <port id of the elastic search service>
elasticsearchuser: <user name to access the elastic search service>
```

Certificate and Key to Access the Gateway HTTPS Endpoint

Certificate and key to access OSM TMF REST APIs, Fallout Exception APIs and UX backend APIs.

<project>-<instance>-app-tls-cert

```
tls.crt: <TLS access certificate>
tls.key: <TLS access key>
```

Certificate and Key to Access the OSM HTTPS Endpoint

Certificate and key to access the OSM HTTPS endpoint.

<project>-<instance>-osm-tls-cert

tls.crt: <TLS access certificate>
tls.key: <TLS access key>

Certificate and Key to Access the OSM WebLogic Remote Console HTTPS Endpoint

Certificate and key to access the OSM WebLogic Remote Console HTTPS endpoint.

<project>-<instance>-admin-tls-cert

tls.crt: <TLS access certificate>
tls.key: <TLS access key>

Certificate and Key to Access the OSM t3 over HTTPS

Certificate and key to access the OSM t3 over HTTPS.

<project>-<instance>-t3-tls-cert

tls.crt: <TLS access certificate>
tls.key: <TLS access key>

Trusted CA Injection

CA trust for secure outbound JMS/SAF connections.

<project>-<instance>-truststore

<cert-name>.crt: <concatenated-CA-certs>
passphrase: <truststore access password>

Secure Identity

Private key to define identity for secure outbound JMS/SAF or SAML IdP connections.

<project>-<instance>-identitystore

<key-name>.key: <private key>
passphrase: <keystore access password>

ADB Wallet Secret

Secure storage of details to connect to the ADB database.

<project>-<instance>-db-wallet

```
wallet-password: <adb wallet password>
ojdbc.properties: <ojdbc.properties>
tnsnames.ora: <tnsnames.ora>
sqlnet.ora: <sqlnet.ora>
cwallet.sso: <cwallet.sso>
ewallet.p12: <ewallet.p12>
keystore.jks: <keystore.jks>
truststore.jks: <truststore.jks>
```

ADB Admin Secret

ADB administrator password.

<project>-<instance>-db-secret

```
admin-password: <Adb administrator password>
```

Cartridge Defined Custom User Credentials

This example is for a custom user named "osmprime" defined by the cartridge. These three lines will repeat for each custom user, with "osmprime" being replaced by each user in turn.

<project>-<instance>-osmcn-cred-<user>

```
osmUser_osmprime_groups: <comma-separated list of OSM groups for this user>
osmUser_osmprime_name: <osmprime>
osmUser_osmprime_password: <password for osmprime>
```

This example is for a cartridge that invokes `getOsmCredentialPassword` with user "osmsom". These two lines will repeat for each user invoked by the cartridge using `getOsmCredentialPassword`.

```
osmUser_osmsom_name: <osmsom>
osmUser_osmsom_password: <password for osmsom>
```

External LDAP Information

Credentials and connection details required to connect with the external LDAP server.

<project>-<instance>-ldap-credentials

```
ldap_credential: <password to access external LDAP>
ldap_groupBaseDn: <base DN on external LDAP to use to look for groups>
ldap_host: <hostname or IP of LDAP server>
ldap_port: <port of LDAP server>
ldap_principal: <LDAP principal to use>
ldap_userBaseDn: <base DN on external LDAP to use to look for users>
```

SAF Credentials

Each SAF credential secret contains exactly one set of credentials.

SAF Credentials

```
username: <SAF destination weblogic user name>  
password: <password for above user>
```

Global Trust Credentials

Shared password for establishing global trust with those domains with which OSM cloud native communicates.

<project>-<instance>-global-trust-credentials

```
password: <shared trust password>
```

Cross Domain Users in Remote Domains

Secret(s) for remote users configured in each remote domain with which OSM cloud native communicates.

<remote-domain-secret>

```
username: <user configured in remote domain>  
password: <password for above user as configured in remote domain>
```

Xtrust Secret

Credential for each of the cross-domain users to be configured in an OSM cloud native instance.

<project>-<instance>-crossdomain-users

```
<cross-domain-user-1>_password: <local password for cross-domain-user-1>  
<cross-domain-user-2>_password: <local password for cross-domain-user-2>...
```

Generic Credentials

Each credential secret contains exactly one set of credentials.

Generic Credentials

```
username: <user name>  
password: <password for above user>
```

Security Scheme Credentials

Secrets for establishing connections to target systems that are defined in the security scheme. It supports two types of authentication: OAuth2 and Username/Password.

- OAuth2: uses OIDC for authentication

<project>-<instance>-<securitySchemeName> (OAuth2)

clientId: <client id>
secret: <secret>

- Username/Password: uses username and password for authentication

<project>-<instance>-<securitySchemeName> (userPassword)

password: <password>
user: <user>

SAML Archive for IdP

Secret to carry the secure information for OSM to be a SAML2 participant for the configured IdP for SSO functionality. Refer to *OSM Security Guide* for more details.

<project>-<instance>--ssosaml-archive

sso-saml2.zip: <archive of secure IdP information>

Git Access Secret

Access credentials for secured git repositories.

<git-repo-secret>

username: <your-git-username>
password: <your-git-access-token>

C

Leveraging OSM Cloud Native SAF Connectivity Patterns for Your Use-Case

This appendix provides details about leveraging OSM cloud native SAF connectivity patterns for your use-case. See the following topics for more information:

- [About SAF Connectivity Patterns](#)
- [Common Integration Patterns](#)
- [OSM Cloud Native Colocated SAF](#)
- [OSM Remote SAF](#)
- [Cloud Native to Cloud Native Remote SAF](#)

About SAF Connectivity Patterns

The integration of OSM with upstream, downstream and peer systems is an essential part of the role it plays at a service provider. These integrations typically use SAF. While the specific integrations are dependent on the service provider's solution, some common patterns can be laid out for OSM cloud native and are described in the rest of this document. These can be used to plan, capture and communicate the connectivity.

When laying out OSM cloud native integration for a project, Oracle recommends that you carefully review the chapter [Integrating OSM](#). You can then use the common patterns below to identify which one best fits each integration. Prepare an overall integration diagram, or set of diagrams, that use the pattern diagrams here, annotated for the specific solution being implemented. Such an artifact is very useful to control the integration, and drive content into the project and instance specifications. Additionally, if an OSM cloud native integration issue is referred to Oracle Support, the service request will require such an integration diagram to describe your use-case.

The diagrams in this document were prepared using draw.io, leveraging its built-in palette of standard Kubernetes icons (as per <https://github.com/kubernetes/community/tree/master/icons>).

Figure C-1 Standard Kubernetes Diagrams



Service



Ingress



Pod

Raw Sources

The draw.io sources for all diagrams are available in the OSM SDK, under the ConnectivityDiagrams folder.

Common Integration Patterns

T3 Routing

JMS and SAF messages use the T3 or T3S protocols. While WebLogic itself can process this protocol and perform any necessary routing within its cluster, the components involved in exposing OSM cloud native to the outside world cannot perform T3 or T3S routing. Such components could be Kubernetes-based Ingress Controllers or Load Balancers, or similar reverse-proxy functions. Kubernetes services themselves do not care about the protocol and can handle T3 or T3S.

This means that if OSM cloud native and the other system are in the same Kubernetes cluster, they must use T3 or T3S protocols. The underlying infrastructure supports it, and direct use of these protocols removes any packaging overhead.

However, if the other system is outside OSM cloud native's Kubernetes cluster, JMS and SAF traffic have to use the tunneling feature of WebLogic. For more information about this feature, see ["Setting Up WebLogic Server for HTTP Tunneling"](#) in *Administering Server Environments for Oracle WebLogic Server*. Here the T3 messages are carried in an http or https envelope and can therefore be routed by Ingress Controllers and Load Balancers. OSM cloud native supports this capability. It is automatically triggered if the JMS or SAF URI begins with "http://" or "https://". The other system needs similar configuration. See WebLogic documentation for more details.

DNS Resolution

There are two aspects to consider regarding hostname resolution in the context of OSM cloud native integration:

1. Accessing an OSM cloud native environment from outside the Kubernetes cluster requires the use of a hostname derived from the instance name and project name. For example, an environment with the project name "com" and instance name "prod" requires the use of the hostname "prod.com.osm.org". Configuration can override the "osm.org" portion as required. Say this results in "prod.com.csp.net". All external systems (browsers trying to access OSM UIs, peer systems trying to invoke OSM web services, peer systems wanting to interact via JMS or SAF, etc) need to use this hostname and have it mapped to the IP address of the Load Balancer or Ingress Controller that exposes OSM cloud native. Ideally, this is done using wildcards in the corporate DNS as described in the *OSM Cloud Native Deployment Guide*. An alternative would be to add this mapping locally to each system (for instance, via /etc/hosts on Linux-based systems).
2. Adding hostname resolution within OSM cloud native itself as the equivalent of /etc/hosts requires such mappings to be injected into the OSM pods if there is no DNS system configured to handle it. This is done by specifying the mapping in the instance specification's "hostAliases" list.

Inbound SAF or JMS

An external system needs to establish SAF or JMS connectivity with OSM cloud native.

Table C-1 Remotes Within the Same Kubernetes Cluster and Outside the Kubernetes Cluster

x	Remote in Same Kubernetes Cluster	Remote Outside Kubernetes Cluster
Protocol	t3	http or https

Table C-1 (Cont.) Remotes Within the Same Kubernetes Cluster and Outside the Kubernetes Cluster

x	Remote in Same Kubernetes Cluster	Remote Outside Kubernetes Cluster
Hostname	<i>project-instance-cluster-c1.project.svc.cluster.local</i>	<i>instance.project.domainname</i>
Port	31313	30303

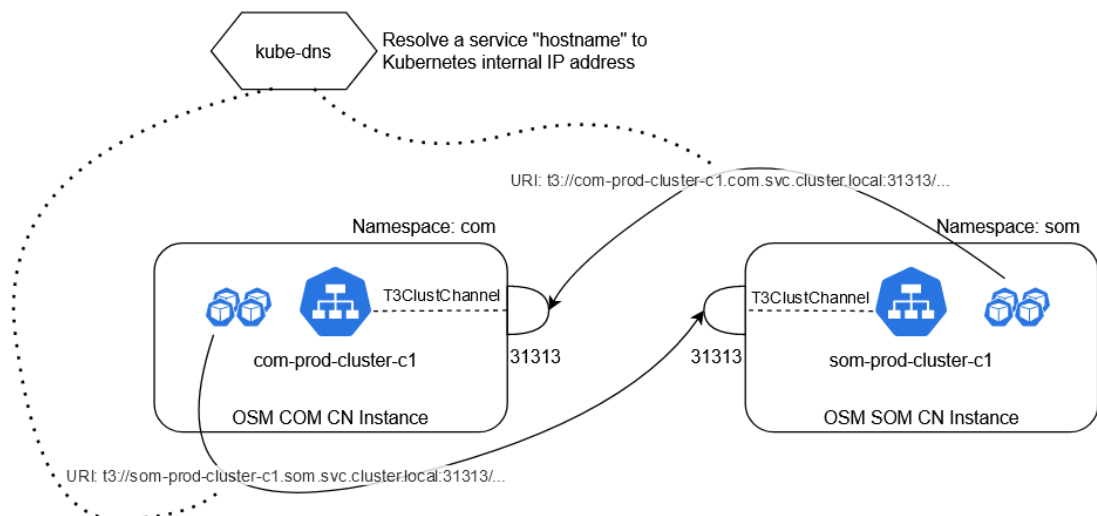
Outbound SAF or JMS

When OSM cloud native needs to talk to an external system, it can do so via T3, http or https, depending on what that external system accepts. Other details, like hostname and port, also depend on that external system.

OSM Cloud Native Colocated SAF

This section describes how an OSM cloud native instance can have bidirectional SAF with another WLS based component (WLS Peer Component) which is also cloud native and running in the same Kubernetes cluster. This other component can be another instance of OSM cloud native or an instance of UIM cloud native, for example.

OSM cloud native has a service *project-instance-cluster-c1* for each instance. This service includes a "T3ClustChannel" - which allows it to ingest t3 messages and route them to the OSM pods - exposed at port 31313 with the "hostname" *project-instance-cluster-c1.namespace.svc.cluster.local*.



The above image, Same Cluster, can be accessed as a draw.io file from the OSM SDK. See the [Raw Sources](#) section for more details.

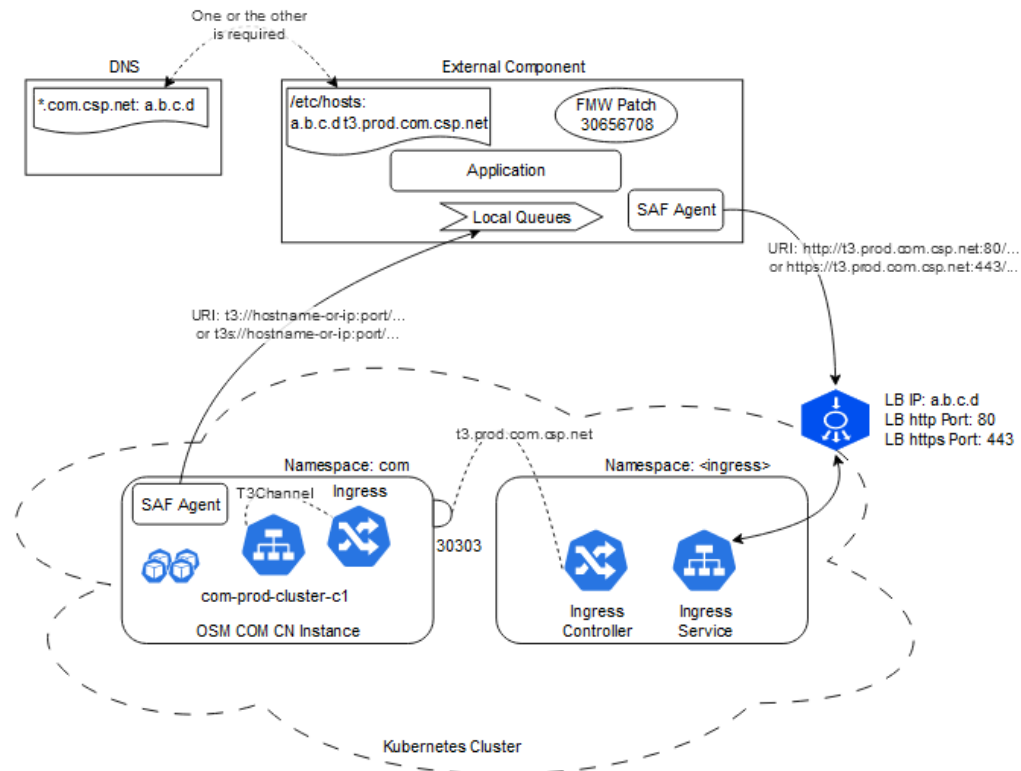
Note that the URIs are both still t3. Since we are in the same Kubernetes cluster, we do not need to involve any ingresses or load balancers. Also note that the URIs take the form *t3://service.namespace.svc.cluster.local:31313/...*

OSM Remote SAF

If the WLS Peer Component is not hosted on the same Kubernetes cluster as OSM cloud native, then the SAF connectivity is done using URIs that reference DNS resolvable

hostnames, externally exposed ports and the http/https protocol, as opposed to Kubernetes resolvable hostnames, internally exposed ports and the t3 protocol.

Internally, OSM cloud native exposes this as a "T3Channel" in its *project-instance-cluster-c1* service at port 30303, with the "hostname" *instance.project.domainname*. This channel is meant to be accessed via an ingress route over http or https. This channel is different from the "T3ClustChannel" described above.



The above image, Remote SAF, can be accessed as a draw.io file from the OSM SDK. See the [Raw Sources](#) section for more details.

Here, an OSM cloud native instance called "prod" is running in the namespace "com", with a configured domain name of "csp.net". It exposes its endpoints to the outside world (outside Kubernetes) using an Ingress. For the purpose of SAF configuration, in this example, the Ingress contains a rule to recognize http or https traffic addressed to "t3.prod.com.csp.net". This rule routes the http or https traffic to the com-prod-cluster-c1 service's t3channel (port 30303). Elsewhere in the cluster, an Ingress Controller monitors Ingress rules and implements them. It exposes its endpoint to the outside world via an Ingress Service, which is typically a LoadBalancer in type. This will result in a load balancer on the cluster boundary. Details of the Ingress Service and the Load Balancer are specific to each Ingress Controller (like Traefik, nginx, etc.) and to each deployment environment (like Oracle OCI, Google GKE, etc.).

The Load Balancer is routable from the outside world in some controlled fashion via its own IP address and hostname. In the above example, the Load Balancer is exposing http ingress rules via port 80 and https ingress rules via port 443.

DNS Considerations

If the external system is addressed via a hostname instead of IP address by OSM cloud native, this hostname must be resolvable by the DNS configured for the Kubernetes cluster. If this is not the case, the OSM cloud native instance specification must have its "hostAliases" list updated to include this hostname. An example of this is in the next section.

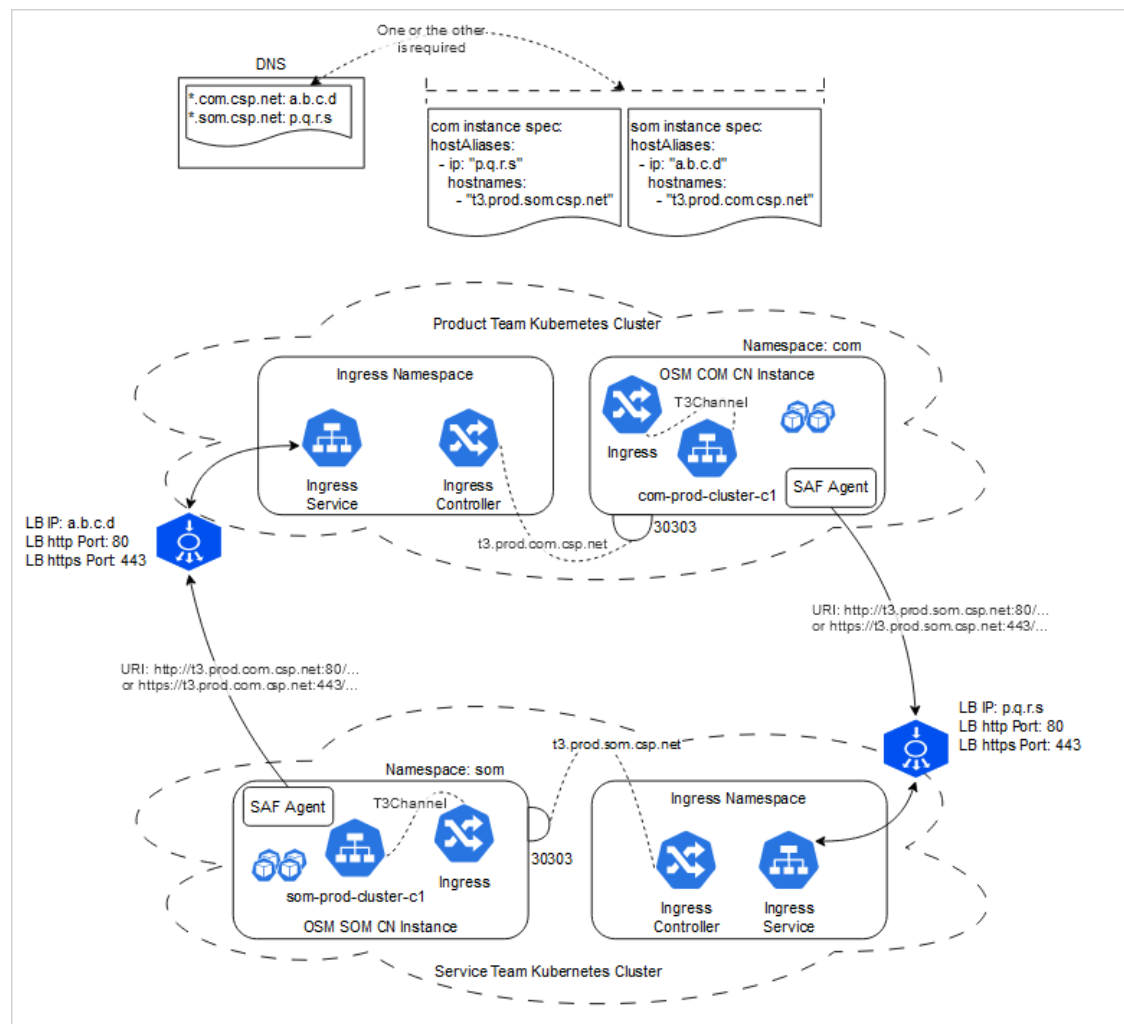
The external system must ensure its SAF traffic hits the Load Balancer using http or https. However, for the Ingress Controller to apply the instance's routing rule internally, the SAF traffic has to be addressed to `t3.com.osm.csp.net`. To satisfy both these requirements, two options exist:

1. The external WebLogic system has its `/etc/hosts` file update on each machine to include a line that matches the Load Balancer IP with the `t3.prod.com.csp.net` hostname.
2. An external DNS used by the external WebLogic system (for example, a corporate DNS) is updated to return the Load Balancer IP for all hostnames with the pattern `"*.com.csp.net"` or `"*.csp.net"`.

Regardless of the DNS approach chosen, the external WebLogic system must ensure it has WebLogic patch 30656708 or its equivalent for the WebLogic version or patch-level in use.

Cloud Native to Cloud Native Remote SAF

A special case of the above is when an OSM cloud native instance running in one Kubernetes cluster needs to have SAF communication with another OSM cloud native instance running in a different Kubernetes cluster. An example might be an OSM COM instance running in one organization's cluster while needing to send service orders to an OSM SOM instance running in another organization's cluster. This would include the need for OSM SOM to send events back to OSM COM as well.



The above image, CN to CN Remote SAF, can be accessed as a draw.io file from the OSM SDK. See the [Raw Sources](#) section for more details.

This is an extrapolation or doubling of the Remote SAF scenario. The main difference is that in the absence of a ubiquitous DNS mechanism, the integrator must leverage OSM cloud native capability to inject Linux /etc/hosts based name resolution via the instance specification.