Oracle® FMW

Deploying and Managing Oracle Unified Directory Services Manager on Kubernetes

G22971-01
March 2025

Oracle FMW Deploying and Managing Oracle Unified Directory Services Manager on Kubernetes,

G22971-01

Primary Author: Russell Hodgson

Contents

1 What's New in This Release?

Part I Introduction to Oracle Unified Directory Services Manager on Kubernetes

2 Introducing Oracle Unified Directory Services Manager on Kubernetes

2.1 Overview of Oracle Unified Directory Services Manager on Kubernetes 2-1
2.2 Key Features of Oracle Unified Directory Services Manager on Kubernetes 2-1

3 About the Kubernetes Deployment

3.1 What is Kubernetes? 3-1
3.2 About the Kubernetes Architecture 3-2
3.3 Key Components Used By an OUDSM Deployment 3-3

Part II Installing Oracle Unified Directory Services Manager on Kubernetes

4 Before You Begin

5 System Requirements for OUDSM on Kubernetes

6 Preparing Your Environment

6.1 Confirming the Kubernetes Cluster is Ready 6-1
6.2 Obtaining the OUDSM Container image 6-1
6.3 Creating a Persistent Volume Directory 6-2
6.4 Setting Up the Code Repository for OUDSM 6-3

7 Creating Oracle Unified Directory Services Manager Instances

7.1 Creating a Kubernetes Namespace 7-1
7.2 Creating a Kubernetes Secret for the Container Registry 7-1
7.3 Creating OUDSM Instances 7-2
7.3.1 Deploying OUDSM Using a YAML File 7-3
7.3.2 Deploying OUDSM Using Set Argument 7-4
7.3.3 Helm Command Output 7-5
7.3.4 Verifying the OUDSM Deployment 7-5

8 Configuring Ingress

8.1 Installing the NGINX Repository 8-1
8.2 Creating a Kubernetes Namespace for NGINX 8-1
8.3 Installing the NGINX Controller 8-2
8.4 Accessing OUDSM Through Ingress 8-4
8.5 Validating OUDSM URLs 8-5

Part III Administering Oracle Unified Directory Services Manager on Kubernetes

9 Scaling OUDSM Pods

9.1 Viewing Existing OUDSM Instances 9-1
9.2 Scaling Up OUDSM Instances 9-1
9.2.1 Scaling Up Using a YAML File 9-2
9.2.2 Scaling Up Using set Argument 9-2
9.2.3 Verifying the Scaling Up 9-2
9.3 Scaling Down OUDSM Instances 9-3
9.3.1 Scaling Down Using a YAML File 9-3
9.3.2 Scaling Down Using --set Argument 9-4
9.3.3 Verifying Scaling Down 9-4

10 Logging and Visualization

10.1 Installing Elasticsearch and Kibana 10-1
10.2 Creating the Logstash Pod 10-1
10.2.1 Variables Used in This Section 10-1
10.2.2 Creating a Kubernetes Secret for ELK 10-2
10.2.3 Enabling Logstash 10-3
10.2.4 Upgrading the OUDSM Deployment for ELK 10-4
10.3 Verifying the Pods 10-5

10.4 Troubleshooting Pod and Logstash Errors 10-5
10.5 Verifying and Accessing the Kibana Console 10-6

11 Monitoring an Oracle Unified Directory Services Manager Instance

11.1 Creating a Kubernetes Namespace for Monitoring 11-1
11.2 Adding Prometheus and Grafana Helm Repositories 11-1
11.3 Installing the Prometheus Operator 11-2
11.4 Viewing Prometheus and Grafana Objects 11-3
11.5 Adding the NodePort for Grafana 11-5
11.6 Verifying Monitoring Using the Grafana GUI 11-6

12 Patching and Upgrading

12.1 Patching and Upgrading Within 14.1.2 12-1
12.1.1 Performing the Upgrade Within 14.1.2 12-1
12.1.2 Rolling Back the Upgrade Within 14.1.2 12-3
12.2 Upgrading from Oracle Unified Directory Services Manager 12.2.1.4 to 14.1.2 12-4

13 General Troubleshooting

13.1 Checking the Status of an OUDSM Namespace 13-1
13.2 Viewing Pod Logs 13-2
13.3 Viewing Pod Descriptions 13-2

14 Deleting an OUDSM Deployment

Part IV Appendices

A Configuration Parameters for the oudsm Helm Chart

List of Figures

3-1 An Illustration of the Kubernetes Cluster 3-2

1

What's New in This Release?

This preface shows current and past versions of Oracle Unified Directory Services Manager (OUDSM) 14c container images and deployment scripts on Kubernetes. If any new functionality is added, details are outlined.

Table 1-1   Release Notes for Oracle Unified Directory Service Manager 14c on Kubernetes

Date	Version	Change
March 2025	14.1.2.1.0 GitHub release version 25.1.3	Initial release of Oracle Unified Directory Services Manager 14.1.2.1.0 on Kubernetes. Supports Oracle Unified Directory Services Manager 14.1.2.1.0 deployment using the OUDSM container image. The GitHub release version is the latest version of the deployment scripts used in Setting Up the Code Repository for OUDSM.

Part I

Introduction to Oracle Unified Directory Services Manager on Kubernetes

Oracle Unified Directory Services Manager (OUDSM) can be deployed on Kubernetes.

This section includes the following chapters:

- Introducing Oracle Unified Directory Services Manager on Kubernetes
- About the Kubernetes Deployment

2

Introducing Oracle Unified Directory Services Manager on Kubernetes

Oracle Unified Directory Services Manager (OUDSM) is supported for deployment on Kubernetes.

This chapter includes the following topics:

- Overview of Oracle Unified Directory Services Manager on Kubernetes
- Key Features of Oracle Unified Directory Services Manager on Kubernetes

2.1 Overview of Oracle Unified Directory Services Manager on Kubernetes

Oracle Unified Directory Services Manager (OUDSM) is an interface for managing instances of Oracle Unified Directory. OUDSM enables you to configure the structure of the directory, define objects in the directory, add and configure users, groups, and other entries. OUDSM is also the interface you use to manage entries, schema, security, and other directory features.

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

2.2 Key Features of Oracle Unified Directory Services Manager on Kubernetes

The key features of using Oracle Unified Directory Services Manager (OUDSM) on Kubernetes are:

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

3

About the Kubernetes Deployment

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

This chapter includes the following topics:

- What is Kubernetes?
- About the Kubernetes Architecture
- Key Components Used By an OUDSM Deployment

3.1 What is Kubernetes?

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

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

Kubernetes provides:

- **Service Discovery and Load Balancing**: Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes balances the load and distributes the network traffic so that the deployment remains stable.
- **Storage Orchestration**: Kubernetes enables you to automatically mount a storage system of your choice, such as local storages, NAS storages, public cloud providers, and more.
- **Automated Rollouts and Rollbacks**: You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers, and adopt all their resources to the new container.
- **Automatic Bin Packing**: If you provide Kubernetes with a cluster of nodes that it can use to run containerized tasks, and indicate the CPU and memory (RAM) each container needs, Kubernetes can fit containers onto the nodes to make the best use of the available resource.
- **Self-healing**: Kubernetes restarts containers that fail, replaces containers, kills containers that do not respond to your user-defined health check, and does not advertise them to clients until they are ready to serve.
- **Secret and Configuration Management**: Kubernetes lets you store and manage sensitive information such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.

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

3.2 About the Kubernetes Architecture

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

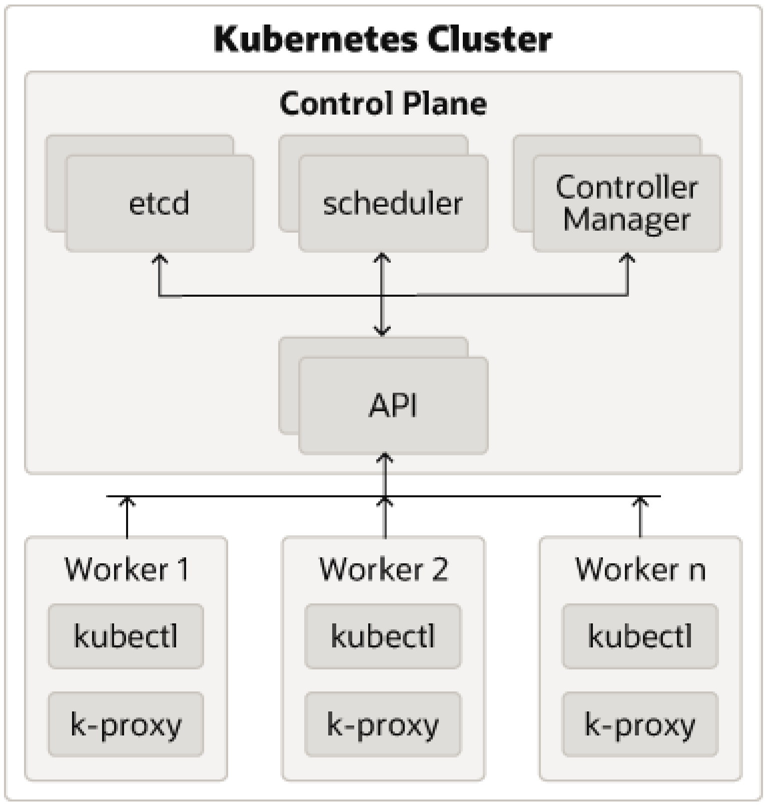
Control Plane: A control plane is responsible for managing the Kubernetes components and deploying applications. In an enterprise deployment, you need to ensure that the Kubernetes control plane is highly available so that the failure of a control plane host does not fail the Kubernetes cluster.

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

Note:

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

Figure 3-1 An Illustration of the Kubernetes Cluster



Description of Components:

- **Control Plane**: The control plane comprises the following:
  - kube-api server: The API server is a component of the control plane that exposes the Kubernetes APIs.

  - etcd: It is used to store the Kubernetes backing store and all the cluster data.
  - Scheduler: The scheduler is responsible for the placement of containers on the worker nodes. It takes into account resource requirements, hardware and software policy constraints, affinity specifications, and data affinity.
  - Control Manager: It is responsible for running the controller processes. Controller processes consist of:
    * Node Controller
    * Route Controller
    * Service Controller

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

3.3 Key Components Used By an OUDSM Deployment

An Oracle Unified Directory Services Manager (OUDSM) deployment uses the Kubernetes components such as pods and Kubernetes services.

Container Image

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

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

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

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

When you upgrade a pod, you are actually instructing the pod to use a different container image. For example, if the container image for OUDSM is based on the July Critical Patch Update (CPU), then to upgrade the pod to use the October CPU image, you have to tell the pod to use the October CPU image and restart the pod. Further information on upgrading can be found in Patching and Upgrading.

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

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

Pods

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

In an OUDSM deployment, each OUDSM instance runs in a different pod.

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

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

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

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

Pod Scheduling

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

Persistent Volumes

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

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

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

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

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

Kubernetes Services

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

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

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

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

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

The Kubernetes services use a small port range. Therefore, when a Kubernetes service is created, there will be a port mapping. For instance, if a pod is using port 1389, then a Kubernetes/Ingress service may use 31389 as its port, mapping port 31389 to 1389 internally. It is worth noting that if you are using individual NodePort Services, then the corresponding Kubernetes service port will be reserved on every worker node in the cluster.

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

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

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

- Load balancer
- Direct proxy calls
- DNS CNames

Ingress Controller

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

Ingress is a proxy server which sits inside the Kubernetes cluster, unlike the NodePort Services which reserve a port per service on every worker node in the cluster. With an ingress service, you can reserve single ports for all HTTP / HTTPS traffic. An Ingress service has the concept of virtual hosts and can terminate SSL, if required. There are various implementations of Ingress. However, this guide describes the installation and configuration of NGNIX. The installation will be similar for other Ingress services but the command syntax may be different. Therefore, when you use a different Ingress, see the appropriate vendor documentation for the equivalent commands. Ingress can proxy HTTP, HTTPS, LDAP, and LDAPS protocols. Ingress is not mandatory.

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

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

Domain Name System

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

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

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

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

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

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

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

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

- **Option 2 -** Add individual host entries to CoreDNS for the external hosts.

Namespaces

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

In this guide the OUDSM deployment uses the namespace `oudsmns`.

Part II

Installing Oracle Unified Directory Services Manager on Kubernetes

Install Oracle Unified Directory Services Manager (OUDSM) on Kubernetes.

This section contains the following chapters:

- Before You Begin
- System Requirements for OUDSM on Kubernetes
- Preparing Your Environment
- Creating Oracle Unified Directory Services Manager Instances
- Configuring Ingress

4
Before You Begin

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

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

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

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

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

5

System Requirements for OUDSM on Kubernetes

This section provides information about the system requirements and limitations for deploying and running Oracle Unified Directory Services Manager (OUDSM) on Kubernetes.

Kubernetes Requirements

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

- The Kubernetes cluster and container engine must meet the minimum version requirements outlined in document ID 2723908.1 on My Oracle Support.
- An administrative host from which to deploy the products: This host could be a Kubernetes Control host, a Kubernetes Worker host, or an independent host. This host must have `kubectl` deployed using the same version as your cluster.
- The Kubernetes cluster must have sufficient nodes and resources.
- An installation of Helm is required on the Kubernetes cluster. Helm is used to create and deploy the necessary resources on the Kubernetes cluster.
- A supported container engine such as CRI-O or Docker must be installed and running on the Kubernetes cluster.
- The nodes in the Kubernetes cluster must have access to a persistent volume such as a Network File System (NFS) mount, or a shared file system.
- The system clocks on node of the Kubernetes cluster must be synchronized. Run the date command simultaneously on all the nodes in each cluster and then synchronize accordingly.

Note:

This documentation does not tell you how to install a Kubernetes cluster, Helm, or the container engine. Please refer to your vendor specific documentation for this information. Also see Before You Begin.

Container Registry Requirements

If your Kubernetes cluster does not have network access to Oracle Container Registry, then you must have your own container registry to store the OUDSM container images.

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

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

Note:

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

6 Preparing Your Environment

Before embarking on Oracle Unified Directory Services Manager (OUDSM) deployment on Kubernetes, you must prepare your environment.

This chapter contains the following topics:

- Confirming the Kubernetes Cluster is Ready
- Obtaining the OUDSM Container image
- Creating a Persistent Volume Directory
- Setting Up the Code Repository for OUDSM

6.1 Confirming the Kubernetes Cluster is Ready

As per System Requirements for OUDSM on Kubernetes, a Kubernetes cluster should have already been configured.

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

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

   The output will look similar to the following:

   ```
   NAME                      STATUS   ROLES                  AGE   VERSION
    node/worker-node1         Ready    <none>                 17h   1.30.3+1.el8
    node/worker-node2         Ready    <none>                 17h   1.30.3+1.el8
    node/master-node          Ready    control-plane,master   23h   1.30.3+1.el8

    NAME                                      READY   STATUS    RESTARTS   AGE
    pod/coredns-66bff467f8-fnhbq              1/1     Running   0          23h
    pod/coredns-66bff467f8-xtc8k              1/1     Running   0          23h
    pod/etcd-master                           1/1     Running   0          21h
    pod/kube-apiserver-master-node            1/1     Running   0          21h
    pod/kube-controller-manager-master-node   1/1     Running   0          21h
    pod/kube-flannel-ds-amd64-lxsfw           1/1     Running   0          17h
    pod/kube-flannel-ds-amd64-pqrqr           1/1     Running   0          17h
    pod/kube-flannel-ds-amd64-wj5nh           1/1     Running   0          17h
    pod/kube-proxy-2kxv2                      1/1     Running   0          17h
    pod/kube-proxy-82vvj                      1/1     Running   0          17h
    pod/kube-proxy-nrgw9                      1/1     Running   0          23h
    pod/kube-scheduler-master                 1/1     Running   0          21h
   ```

6.2 Obtaining the OUDSM Container image

The Oracle Unified Directory Services Manager (OUDSM) Kubernetes deployment requires access to an OUDSM container image.

Prebuilt OUDSM Container Image

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

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

You can use this image in the following ways:

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

6.3 Creating a Persistent Volume Directory

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

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

Perform the following steps:

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

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

   For example:

   ```
   cd /nfs_volumes/oudsmpv
   mkdir oudsm_user_projects
   sudo chown -R 1000:0 oudsm_user_projects
   ```

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

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

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

For example:

```
cd /nfs_volumes/oudsmpv/oudsm_user_projects
touch fileadmin.txt
ls fileadmin.txt
```

6.4 Setting Up the Code Repository for OUDSM

To deploy Oracle Unified Directory Services Manager (OUDSM) you need to set up the code repository which provides sample deployment yaml files.

The OUDSM deployment on Kubernetes leverages deployment scripts provided by Oracle for creating OUDSM containers, using the Helm charts provided.

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

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

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

   ```
   mkdir <workdir>
   ```

   For example:

   ```
   mkdir /oudsmscripts
   ```

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

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

   For example:

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

The output will look similar to the following:

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

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

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

   For example:

   ```
   export WORKDIR=/oudsmscripts/fmw-kubernetes/OracleUnifiedDirectorySM
   ```

7

Creating Oracle Unified Directory Services Manager Instances

This chapter demonstrates how to deploy Oracle Unified Directory Services Manager (OUDSM) 14c instance(s) using the Helm package manager for Kubernetes.

This chapter contains the following topics:

- Creating a Kubernetes Namespace
- Creating a Kubernetes Secret for the Container Registry
- Creating OUDSM Instances

7.1 Creating a Kubernetes Namespace

Create a Kubernetes namespace for the Oracle Unified Directory Services Manager (OUDSM) deployment by running the following command:

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

For example:

```
kubectl create namespace oudsmns
```

The output will look similar to the following:

```
namespace/oudsmns created
```

7.2 Creating a Kubernetes Secret for the Container Registry

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

1. Run the following command to create the secret:

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

For example, if using Oracle Container Registry:

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

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

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

The output will look similar to the following:

```
secret/orclcred created
```

7.3 Creating OUDSM Instances

The oudsm Helm Chart

The `oudsm` Helm chart allows you to create or deploy Oracle Unified Directory Services Manager (OUDSM) instances along with Kubernetes objects in a specified namespace.

The deployment can be initiated by running the following Helm command with reference to the `oudsm` Helm chart, along with configuration parameters according to your environment:

```
cd $WORKDIR/kubernetes/helm14c
helm install --namespace <namespace> \
<Configuration Parameters> \
<deployment/release name> \
<Helm Chart Path/Name>
```

Configuration Parameters (override values in chart) can be passed on with `--set` arguments on the command line and/or with `-f / --values` arguments when referring to files.

Note:

The examples in the following sections provide values which allow the user to override the default values provided by the Helm chart. A full list of configuration parameters and their default values is shown in Configuration Parameters for the oudsm Helm Chart.

For more details about the helm command and parameters, execute `helm --help` and `helm install --help`.

Deploying OUDSM Instances

OUDSM instances can be deployed using one of the following methods:

- Deploying OUDSM Using a YAML File
- Deploying OUDSM Using Set Argument

7.3.1 Deploying OUDSM Using a YAML File

To deploy Oracle Unified Directory Services Manager (OUDSM) using a YAML file:

1. Navigate to the `$WORKDIR/kubernetes/helm14c` directory:

   ```
   cd $WORKDIR/kubernetes/helm14c
   ```

2. Create an `oudsm-values-override.yaml` as follows:

   ```
   image:
     repository: <image_location>
     tag: <image_tag>
     pullPolicy: IfNotPresent
   imagePullSecrets:
     - name: orclcred
   oudsm:
     adminUser: weblogic
     adminPass: <password>
   ingress:
     tlsEnabled: false
   persistence:
     type: filesystem
     filesystem:
       hostPath:
         path: <persistent_volume>/oudsm_user_projects
   ```

   For example:

   ```
   image:
     repository: container-registry.oracle.com/middleware/oudsm_cpu
     tag: 14.1.2.1.0-jdk17-ol8-<YYMMDD>
     pullPolicy: IfNotPresent
   imagePullSecrets:
     - name: orclcred
   oudsm:
     adminUser: weblogic
     adminPass: <password>
   ingress:
     tlsEnabled: false
   persistence:
     type: filesystem
     filesystem:
       hostPath:
         path: /nfs_volumes/oudsmpv/oudsm_user_projects
   ```

The following caveats exist:

- Replace `<password>` with the relevant password.
- If you are not using Oracle Container Registry or your own container registry for your OUDSM container image, then you can remove the following:

  ```
  imagePullSecrets:
    - name: orclcred
  ```

- If using NFS for your persistent volume then change the `persistence` section as follows:

  ```
  persistence:
    type: networkstorage
    networkstorage:
      nfs:
        path: <persistent_volume>/oudsm_user_projects
        server: <NFS IP address>
  ```

3. Run the following command to deploy OUDSM:

   ```
   helm install --namespace <namespace> \
   --values oudsm-values-override.yaml \
   <release_name> oudsm
   ```

   For example:

   ```
   helm install --namespace oudsmns \
   --values oudsm-values-override.yaml \
   oudsm oudsm
   ```

   The output will be similar to that shown in Helm Command Output.
4. Check the OUDSM deployment as per Verifying the OUDSM Deployment.

7.3.2 Deploying OUDSM Using Set Argument

To deploy Oracle Unified Directory Services Manager (OUDSM) using the `--set` argument:

1. Navigate to the `$WORKDIR/kubernetes/helm14c` directory:

   ```
   cd $WORKDIR/kubernetes/helm14c
   ```

2. Run the following command to create OUDSM instances:

   ```
   helm install --namespace oudsmns \
   --set
   oudsm.adminUser=weblogic,oudsm.adminPass=<password>,ingress.tlsEnabled=fals
   e,persistence.filesystem.hostPath.path=<persistent_volume>/
   oudsm_user_projects,image.repository=<image_location>,image.tag=<image_tag>
    \
   --set imagePullSecrets[0].name="orclcred" \
   <release_name> oudsm
   ```

For example:

```
helm install --namespace oudsmns \
--set
oudsm.adminUser=weblogic,oudsm.adminPass=<password>,ingress.tlsEnabled=fals
e,persistence.filesystem.hostPath.path=/nfs_volumes/oudsmpv/
oudsm_user_projects,image.repository=container-registry.oracle.com/
middleware/oudsm_cpu,image.tag=14.1.2.1.0-jdk17-ol8-<YYMMDD> \
--set imagePullSecrets[0].name="orclcred" \
oudsm oudsm
```

The following caveats exist:

- Replace `<password>` with the relevant password.
- If you are not using Oracle Container Registry or your own container registry for your OUDSM container image, then you can remove the following:

  ```
  --set imagePullSecrets[0].name="orclcred"
  ```

- If using NFS for your persistent volume then use:

  ```
  persistence.networkstorage.nfs.path=<persistent_volume>/
  oudsm_user_projects,persistence.networkstorage.nfs.server:<NFS IP
  address>
  ```

3. Check the OUDSM deployment as per Verifying the OUDSM Deployment.

7.3.3 Helm Command Output

In all the examples above, the following output is shown following a successful execution of the helm install command:

```
NAME: oudsm
LAST DEPLOYED: <DATE>
NAMESPACE: oudsmns
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

7.3.4 Verifying the OUDSM Deployment

Run the following command to verify the OUDSM deployment:

```
kubectl --namespace <namespace> get pod,service,secret,pv,pvc,ingress -o wide
```

For example:

```
kubectl --namespace oudsmns get pod,service,secret,pv,pvc,ingress -o wide
```

The output will look similar to the following:

```
NAME           READY   STATUS    RESTARTS   AGE   IP
NODE              NOMINATED NODE   READINESS GATES
pod/oudsm-1   1/1     Running   0          73m   10.244.0.19   <worker-
node>   <none>           <none>

NAME                  TYPE        CLUSTER-IP      EXTERNAL-IP
PORT(S)               AGE   SELECTOR
service/oudsm-1      ClusterIP   10.96.108.200   <none>        7001/
TCP,7002/TCP   73m   app.kubernetes.io/instance=oudsm,app.kubernetes.io/
name=oudsm,oudsm/instance=oudsm-1
service/oudsm-lbr   ClusterIP   10.96.41.201    <none>        7001/
TCP,7002/TCP   73m   app.kubernetes.io/instance=oudsm,app.kubernetes.io/
name=oudsm

NAME                                 TYPE
DATA   AGE
secret/orclcred                      kubernetes.io/dockerconfigjson
1      3h13m
secret/oudsm-creds                   opaque
2      73m
secret/oudsm-token-ksr4g             kubernetes.io/service-account-token
3      73m
secret/sh.helm.release.v1.oudsm.v1   helm.sh/release.v1
1      73m

NAME                            CAPACITY   ACCESS MODES   RECLAIM POLICY
STATUS   CLAIM                   STORAGECLASS   REASON   AGE   VOLUMEMODE
persistentvolume/oudsm-pv       30Gi       RWX            Retain
Bound    myoudsmns/oudsm-pvc     manual                  73m   Filesystem

NAME                                  STATUS   VOLUME     CAPACITY   ACCESS
MODES   STORAGECLASS   AGE   VOLUMEMODE
persistentvolumeclaim/oudsm-pvc   Bound    oudsm-pv   30Gi
RWX            manual         73m   Filesystem

NAME                                    HOSTS
ADDRESS          PORTS   AGE
ingress.extensions/oudsm-ingress-nginx   oudsm-1,oudsm-2,oudsm + 1 more...
100.102.51.230   80      73m
```

Note:

It will take several minutes before all the services listed above show. While the oudsm pods have a `STATUS` of `0/1` the pod is started but the OUDSM server associated with it is currently starting. While the pod is starting you can check the startup status in the pod logs, by running the following command:

```
kubectl logs <pod> -n oudsmns
```

For example:

```
kubectl logs oudsm-1 -n oudsmns
```

If the OUDSM deployment fails, additionally refer to General Troubleshooting for instructions on how describe the failing pod(s). Once the problem is identified follow Deleting an OUDSM Deployment to clean down the deployment before deploying again.

Kubernetes Objects

Kubernetes objects created by the Helm chart are detailed in the table below:

Note:

The '**Example Name**' for each Object below is based on the value '`oudsm`' as deployment/release name for the Helm chart installation.

Type	Name	Example Name	Purpose
Service Account	<deployment/release name>	oudsm	Kubernetes Service Account for the Helm Chart deployment.
Secret	<deployment/release name>-creds	oudsm-creds	Secret object for Oracle Unified Directory Services Manager related critical values like passwords.
Persistent Volume	<deployment/release name>-pv	oudsm-pv	Persistent Volume for user_projects mount.
Persistent Volume Claim	<deployment/release name>-pvc	oudsm-pvc	Persistent Volume Claim for user_projects mount.
Pod	<deployment/release name>-N	oudsm-1, oudsm-2, …	Pod(s)/Container(s) for Oracle Unified Directory Services Manager Instances.
Service	<deployment/release name>-N	oudsm-1, oudsm-2, …	Service(s) for HTTP and HTTPS interfaces from Oracle Unified Directory Services Manager instance <deployment/release name>-N.

Type	Name	Example Name	Purpose
Ingress	<deployment/release name>-ingress-nginx	oudsm-ingress-nginx	Ingress Rules for HTTP and HTTPS interfaces.

8

Configuring Ingress

You must configure an ingress controller to allow access to Oracle Unified Directory Services Manager (OUDSM).

The instructions below explain how to set up NGINX as the ingress controller for OUDSM.

This chapter includes the following topics:

- Installing the NGINX Repository
- Creating a Kubernetes Namespace for NGINX
- Installing the NGINX Controller
- Accessing OUDSM Through Ingress
- Validating OUDSM URLs

8.1 Installing the NGINX Repository

To install the NGINX ingress controller:

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

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

   The output will look similar to the following:

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

2. Update the repository using the following command:

   ```
   helm repo update
   ```

   The output will look similar to the following:

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

8.2 Creating a Kubernetes Namespace for NGINX

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

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

For example:

```
kubectl create namespace mynginxns
```

The output will look similar to the following:

```
namespace/mynginxns created
```

8.3 Installing the NGINX Controller

To install the NGINX controller:

1. Navigate to the `$WORKDIR/kubernetes/helm14c/` and create a `nginx-ingress-values-override.yaml` that contains the following:

   Note:

   The configuration below:

   - Assumes that you have `oudsm` installed with value `oudsm` as a deployment/release name in the namespace `oudsmns`. If using a different deployment name and/or namespace change appropriately.
   - Deploys an ingress using LoadBalancer. If you prefer to use NodePort, change the configuration accordingly. For more details about NGINX configuration see: NGINX Ingress Controller.

```
controller:
  admissionWebhooks:
    enabled: false
  extraArgs:
    # The secret referred to by this flag contains the default certificate
to be used when accessing the catch-all server.
    # If this flag is not provided NGINX will use a self-signed
certificate.
    # If the TLS Secret is in different namespace, name can be mentioned
as <namespace>/<tlsSecretName>
    default-ssl-certificate: oudsmns/oudsm-tls-cert
  service:
    # controller service external IP addresses
    # externalIPs:
    #   - < External IP Address >
    # To configure Ingress Controller Service as LoadBalancer type of
Service
    # Based on the Kubernetes configuration, External LoadBalancer would
be linked to the Ingress Controller Service
    type: LoadBalancer
    # Configuration for NodePort to be used for Ports exposed through
Ingress
    # If NodePorts are not defined/configured, Node Port would be assigned
automatically by Kubernetes
    # These NodePorts are helpful while accessing services directly
```

```
through Ingress and without having External Load Balancer.
    nodePorts:
      # For HTTP Interface exposed through LoadBalancer/Ingress
      http: 30080
      # For HTTPS Interface exposed through LoadBalancer/Ingress
      https: 30443
```

> **Note:**
>
> If you do not have an external load balancer configured for your Kubernetes configuration, change `type: LoadBalancer` to `type: NodePort`.

2. To install and configure NGINX Ingress issue the following commands:

```
cd $WORKDIR/kubernetes/helm14c/


helm install --namespace <namespace> \
--values nginx-ingress-values-override.yaml \
lbr-nginx stable/ingress-nginx \
--version 4.7.2
```

Where:

- `lbr-nginx` is your deployment name
- `stable/ingress-nginx` is the chart reference

For example:

```
cd $WORKDIR/kubernetes/helm14c/


helm install --namespace mynginxns \
--values nginx-ingress-values-override.yaml \
lbr-nginx stable/ingress-nginx \
--version 4.7.2
```

The output will look similar to the following:

```
NAME: lbr-nginx
LAST DEPLOYED: <DATE>
NAMESPACE: mynginxns
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The ingress-nginx controller has been installed.
It may take a few minutes for the LoadBalancer IP to be available.
You can watch the status by running 'kubectl --namespace mynginxns get
services -o wide -w lbr-nginx-ingress-nginx-controller'

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

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

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

8.4 Accessing OUDSM Through Ingress

Using the Helm chart, ingress objects are created according to configuration.

The following table details the rules configured in ingress object(s) for access to Oracle Unified Directory Services Manager (OUDSM) interfaces through ingress.

Port	NodePort	Host	Example Hostname	Path	Backend Service:Port	Example Service Name:Port
http/https	30080/30443	<deployment/release name>-N	oudsm-N	N	<deployment/release name>-N:http	oudsm-1:http
http/https	30080/30443	*	*	/oudsm /console	<deployment/release name>-lbr:http	oudsm-lbr:http

Note:

In the table above, example values are based on the value ‘`oudsm`’ as the deployment/release name for Helm chart installation. The NodePorts mentioned in the table are according to ingress configuration described in the previous section. When an External LoadBalancer is not available/configured, interfaces can be accessed through NodePort on a Kubernetes node.

Changes in /etc/hosts to Validate Hostname Based Ingress Rules

If it is not possible to update the DNS with the OUDSM hostname interfaces, then the following entries can be added in `/etc/hosts` file on the host from where OUDSM interfaces will be accessed.

```
<IP Address of External LBR or Kubernetes Node>    oudsm oudsm-1 oudsm-2
oudsm-N
```

- In the table above, hostnames are based on the value ‘`oudsm`’ as the deployment/release name for Helm chart installation.
- When External LoadBalancer is not available/configured, interfaces can be accessed through NodePort on the Kubernetes Node.

8.5 Validating OUDSM URLs

Launch a browser and access the Oracle Unified Directory Services Manager (OUDSM) console:

- If using an External LoadBalancer: `https://<External LBR Host>/oudsm`
- If not using an External LoadBalancer: `https://<Kubernetes Node>:30443/oudsm`

Note:

Administrators should be aware of the following:

- To monitor the OUDSM WebLogic Server domain in 14.1.2.1.0 you must use the Oracle WebLogic Remote Console. When connecting in WebLogic Remote Console to the OUDSM domain, login with `weblogic/<password>` where `weblogic/<password>` is the `adminUser` and `adminPass` set when creating the OUDSM instance. For more information about installing and configuring the console, see Getting Started Using Administration Console.
- The Oracle WebLogic Remote Console and Oracle Enterprise Manager Console should only be used to monitor the servers in the OUDSM domain. To control the Administration Server and OUDSM managed servers (start/stop) you must use Kubernetes. See Scaling OUDSM Pods for more information.

Part III

Administering Oracle Unified Directory Services Manager on Kubernetes

Administer Oracle Unified Directory Services Manager (OUDSM) on Kubernetes.

This section contains the following chapters:

- Scaling OUDSM Pods
- Logging and Visualization
- Monitoring an Oracle Unified Directory Services Manager Instance
- Patching and Upgrading
- General Troubleshooting
- Deleting an OUDSM Deployment

9

Scaling OUDSM Pods

Learn the basic operations to scale Oracle Unified Directory Services Manager (OUDSM) instances in Kubernetes.

This chapter includes the following topics:

- Viewing Existing OUDSM Instances
- Scaling Up OUDSM Instances
- Scaling Down OUDSM Instances

9.1 Viewing Existing OUDSM Instances

By default the `oudsm` helm chart deployment starts one pod: `oudsm-1`.

The number of pods started is determined by the `replicaCount`, which is set to `1` by default. A value of `1` starts the pod above.

To scale up or down the number of OUDSM pods, set `replicaCount` accordingly.

Run the following command to view the number of pods in the OUDSM deployment:

```
kubectl --namespace <namespace> get pods -o wide
```

For example:

```
$ kubectl --namespace oudsmns get pods -o wide
```

The output will look similar to the following:

```
NAME           READY   STATUS    RESTARTS   AGE   IP
NODE              NOMINATED NODE   READINESS GATES
pod/oudsm-1    1/1     Running   0          73m   10.244.0.19   <worker-
node>    <none>           <none>
```

9.2 Scaling Up OUDSM Instances

In the examples below, `replicaCount` is increased to `2` which creates a new Oracle Unified Directory Services Manager (OUDSM) pod `oudsm-2`, with associated services created.

You can scale up the number of OUDSM pods using one of the following methods:

- Scaling Up Using a YAML File
- Scaling Up Using set Argument

9.2.1 Scaling Up Using a YAML File

1. Navigate to the `$WORKDIR/kubernetes/helm14c` directory:

   ```
   cd $WORKDIR/kubernetes/helm14c
   ```

2. Create an `oudsm-scaleup-override.yaml` file that contains:

   ```
   replicaCount: 2
   ```

3. Run the following command to scale up the OUDSM pods:

   ```
   helm upgrade --namespace <namespace> \
   --values oudsm-scaleup-override.yaml \
   <release_name> oudsm --reuse-values
   ```

   For example:

   ```
   helm upgrade --namespace oudsmns \
   --values oudsm-scaleup-override.yaml \
   oudsm oudsm --reuse-values
   ```

9.2.2 Scaling Up Using set Argument

1. Run the following command to scale up the OUDSM pods:

   ```
   helm upgrade --namespace <namespace> \
   --set replicaCount=2 \
   <release_name> oudsm --reuse-values
   ```

   For example:

   ```
   helm upgrade --namespace oudsmns \
   --set replicaCount=2 \
   oudsm oudsm --reuse-values
   ```

9.2.3 Verifying the Scaling Up

1. Verify the new OUDSM pod `oudsm-2` has started:

   ```
   kubectl get pod,service -o wide -n <namespace>
   ```

   For example:

   ```
   kubectl get pods,service -n oudsmns
   ```

The output will look similar to the following:

```
NAME          READY   STATUS    RESTARTS   AGE   IP
NODE            NOMINATED NODE   READINESS GATES
pod/oudsm-1   1/1     Running   0          88m   10.244.0.19   <worker-
node>   <none>           <none>
pod/oudsm-2   1/1     Running   0          15m   10.245.3.45   <worker-
node>   <none>           <none>

NAME                TYPE        CLUSTER-IP      EXTERNAL-IP
PORT(S)             AGE   SELECTOR
service/oudsm-1     ClusterIP   10.96.108.200   <none>        7001/
TCP,7002/TCP   88m   app.kubernetes.io/instance=oudsm,app.kubernetes.io/
name=oudsm,oudsm/instance=oudsm-1
service/oudsm-2     ClusterIP   10.96.31.201    <none>        7001/
TCP,7002/TCP   15m   app.kubernetes.io/instance=oudsm,app.kubernetes.io/
name=oudsm,oudsm/instance=oudsm-2
service/oudsm-lbr   ClusterIP   10.96.41.201    <none>        7001/
TCP,7002/TCP   73m   app.kubernetes.io/instance=oudsm,app.kubernetes.io/
name=oudsm
```

Note:

It will take several minutes before all the services listed above show. While the `oudsm-2` pod has a `STATUS` of `0/1` the pod is started but the OUDSM server associated with it is currently starting. While the pod is starting, you can check the startup status in the pod log, by running the following command:

```
kubectl logs oudsm-2 -n oudsmns
```

9.3 Scaling Down OUDSM Instances

Scaling down Oracle Unified Directory Services Manager (OUDSM) pods is performed in exactly the same way as in Scaling Up OUDSM Instances except the `replicaCount` is reduced to the required number of pods.

In the examples below, `replicaCount` is decreased to `1` from `2` which terminates the `oudsm-2` pod and associated services.

You can scale down the number of OUDSM pods using one of the following methods:

- Scaling Down Using a YAML File
- Scaling Down Using --set Argument

9.3.1 Scaling Down Using a YAML File

1. Navigate to the `$WORKDIR/kubernetes/helm14c` directory:

   ```
   cd $WORKDIR/kubernetes14c/helm
   ```

2. Create an `oudsm-scaledown-override.yaml` file and set the `replicaCount`:

   ```
   replicaCount: 1
   ```

3. Run the following command to scale down the OUD pods:

   ```
   helm upgrade --namespace <namespace> \
   --values oudsm-scaledown-override.yaml \
   <release_name> oudsm --reuse-values
   ```

   For example:

   ```
   helm upgrade --namespace oudsmns \
   --values oudsm-scaledown-override.yaml \
   oudsm oudsm --reuse-values
   ```

9.3.2 Scaling Down Using --set Argument

1. Run the following command to scale down the OUDSM pods:

   ```
   helm upgrade --namespace <namespace> \
   --set replicaCount=1 \
   <release_name> oudsm --reuse-values
   ```

   For example:

   ```
   helm upgrade --namespace oudsmns \
   --set replicaCount=1 \
   oudsm oudsm --reuse-values
   ```

9.3.3 Verifying Scaling Down

1. Verify the OUDSM pod `oudsm-2` is terminated:

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

   For example:

   ```
   kubectl get pods -n oudsmns
   ```

   The output will look similar to the following:

   ```
   NAME           READY   STATUS        RESTARTS   AGE
   pod/oudsm-1    1/1     Running       0          92m
   pod/oudsm-2    1/1     Terminating   0          19m
   ```

   The `oudsm-2` pod has moved to a `STATUS` of `Terminating`.

The pod will take a minute or two to stop and then will disappear:

```
NAME           READY    STATUS         RESTARTS    AGE
pod/oudsm-1    1/1      Running        0           94m
```

10 Logging and Visualization

This chapter describes how to install and configure logging and visualization for the `oudsm` Helm chart deployment.

The ELK stack consists of Elasticsearch, Logstash, and Kibana. Using ELK you can gain insights in real-time from the log data from your applications.

Elasticsearch is a distributed, RESTful search and analytics engine capable of solving a growing number of use cases. As the heart of the Elastic Stack, it centrally stores your data so you can discover the expected and uncover the unexpected.

Logstash is an open source, server-side data processing pipeline that ingests data from a multitude of sources simultaneously, transforms it, and then sends it to your favorite “stash.”

Kibana lets you visualize your Elasticsearch data and navigate the Elastic Stack. It gives you the freedom to select the way you give shape to your data, and you don’t always have to know what you’re looking for.

This chapter includes the following topics:

- Installing Elasticsearch and Kibana
- Creating the Logstash Pod
- Verifying the Pods
- Troubleshooting Pod and Logstash Errors
- Verifying and Accessing the Kibana Console

10.1 Installing Elasticsearch and Kibana

If you do not already have a centralized Elasticsearch (ELK) stack then you must configure this first.

For details on how to configure the ELK stack, see Installing the Monitoring and Visualization Software.

10.2 Creating the Logstash Pod

Topics in the section include:

- Variables Used in This Section
- Creating a Kubernetes Secret for ELK
- Enabling Logstash
- Upgrading the OUDSM Deployment for ELK

10.2.1 Variables Used in This Section

In order to create the logstash pod, you must create a yaml file. This file contains variables which you must substitute with variables applicable to your ELK environment.

Most of the values for the variables will be based on your ELK deployment as per Installing the Monitoring and Visualization Software.

The table below outlines the variables and values you must set:

Variable	Sample Value	Description
<ELK_VER>	`8.3.1`	The version of logstash you want to install.
<ELK_SSL>	true	If SSL is enabled for ELK set the value to `true`, or if `NON-SSL` set to `false`. This value must be lowercase.
<ELK_HOSTS>	`https://elasticsearch.example.com:9200`	The URL for sending logs to Elasticsearch. HTTP if `NON-SSL` is used.
<ELK_USER>	`logstash_internal`	The name of the user for logstash to access Elasticsearch.
<ELK_PASSWORD>	`password`	The password for *<ELK_USER>*.
<ELK_APIKEY>	`apikey`	The API key details.

You will also need the BASE64 version of the Certificate Authority (CA) certificate(s) that signed the certificate of the Elasticsearch server. If using a self-signed certificate, this is the self signed certificate of the Elasticsearch server. See Copying the Elasticsearch Certificate, for details on how to get the correct certificate. In the example below the certificate is called `elk.crt`.

10.2.2 Creating a Kubernetes Secret for ELK

1. Create a Kubernetes secret for Elasticsearch using the API Key or Password:
   a. If ELK uses an API Key for authentication:

      ```
      kubectl create secret generic elasticsearch-pw-elastic -n
      <domain_namespace> --from-literal password=<ELK_APIKEY>
      ```

      For example:

      ```
      kubectl create secret generic elasticsearch-pw-elastic -n oudsmns --
      from-literal password=<ELK_APIKEY>
      ```

      The output will look similar to the following:

      ```
      secret/elasticsearch-pw-elastic created
      ```

   b. If ELK uses a password for authentication:

      ```
      kubectl create secret generic elasticsearch-pw-elastic -n
      <domain_namespace> --from-literal password=<ELK_PASSWORD>
      ```

For example:

```
kubectl create secret generic elasticsearch-pw-elastic -n oudsmns --
from-literal password=<ELK_PASSWORD>
```

The output will look similar to the following:

```
secret/elasticsearch-pw-elastic created
```

Note:

It is recommended that the ELK Stack is created with authentication enabled. If no authentication is enabled you may create a secret using the values above.

2. Create a Kubernetes secret to access the required images on hub.docker.com:

   Note:

   You must first have a user account on hub.docker.com

```
kubectl create secret docker-registry "dockercred" \
--docker-server="https://index.docker.io/v1/" \
--docker-username="<docker_username>" \
--docker-password=<password> \
--docker-email=<docker_email_credentials> \
--namespace=<domain_namespace>
```

   For example:

```
kubectl create secret docker-registry "dockercred" \
--docker-server="https://index.docker.io/v1/" \
--docker-username="username" \
--docker-password=<password> \
--docker-email=user@example.com \
--namespace=oudsmns
```

   The output will look similar to the following:

```
secret/dockercred created
```

10.2.3 Enabling Logstash

Navigate to the `$WORKDIR/kubernetes/helm` directory and create a `logging-override-values.yaml` file as follows:

```
elk:
  imagePullSecrets:
    - name: dockercred
  IntegrationEnabled: true
  logStashImage: logstash:<ELK_VER>
  logstashConfigMap: false
  esindex: oudsmlogs-00001
  sslenabled: <ELK_SSL>
  eshosts: <ELK_HOSTS>
  # Note: We need to provide either esuser,espassword or esapikey
  esuser: <ELK_USER>
  espassword: elasticsearch-pw-elastic
  esapikey: elasticsearch-pw-elastic
```

- Change the <*ELK_VER*>, <*ELK_SSL*>, <*ELK_HOSTS*>, and <*ELK_USER*> to match the values for your environment.
- If using SSL, replace the `elk.crt` in `$WORKDIR/kubernetes/helm/oud-ds-rs/certs/` with the `elk.crt` for your ElasticSearch server.
- If using API KEY for your ELK authentication, leave both `esuser:` and `espassword:` with no value.
- If using a password for ELK authentication, leave `esapi_key:` but delete `elasticsearch-pw-elastic`.
- If no authentication is used for ELK, leave `esuser`, `espassword`, and `esapi_key` with no value assigned.
- The rest of the lines in the yaml file should not be changed.

For example:

```
elk:
  imagePullSecrets:
    - name: dockercred
  IntegrationEnabled: true
  logStashImage: logstash:8.3.1
  logstashConfigMap: false
  esindex: oudsmlogs-00001
  sslenabled: true
  eshosts: https://elasticsearch.example.com:9200
  # Note: We need to provide either esuser,espassword or esapikey
  esuser: logstash_internal
  espassword: elasticsearch-pw-elastic
  esapikey:
```

10.2.4 Upgrading the OUDSM Deployment for ELK

1. Run the following command to upgrade the Oracle Unified Directory Services Manager (OUDSM) deployment with the ELK configuration:

   ```
   helm upgrade --namespace <namespace> --values <valuesfile.yaml>
   <releasename> oudsm --reuse-values
   ```

For example:

```
helm upgrade --namespace oudsmns --values logging-override-values.yaml
oudsm oudsm --reuse-values
```

The output should look similar to the following:

```
Release "oudsm" has been upgraded. Happy Helming!
NAME: oudsm
LAST DEPLOYED: <DATE>
NAMESPACE: oudsmns
STATUS: deployed
REVISION: 2
TEST SUITE: None
```

10.3 Verifying the Pods

1. Run the following command to check the logstash pod is created correctly:

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

   For example:

   ```
   kubectl get pods -n oudsmns
   ```

   The output should look similar to the following:

   ```
   NAME                                 READY   STATUS    RESTARTS   AGE
   oudsm-1                              1/1     Running   0          51m
   oudsm-logstash-56dbcc6d9f-mxsgj      1/1     Running   0          2m7s
   ```

   Note:

   Wait a couple of minutes to make sure the pod has not had any failures or restarts. If the pod fails you can view the pod log using:

   ```
   kubectl logs -f oudsm-logstash-<pod> -n oudsmns
   ```

   If the logstash pod has problems, see Troubleshooting Pod and Logstash Errors.

10.4 Troubleshooting Pod and Logstash Errors

Most errors occur due to misconfiguration of the `logging-override-values.yaml`. This is usually because of an incorrect value set, or the certificate was not pasted with the correct indentation.

If the pod has errors, view the helm history to find the last working revision, for example:

```
helm history oudsm -n oudsmns
```

The output will look similar to the following:

```
REVISION        UPDATED       STATUS          CHART           APP VERSION
DESCRIPTION
1               <DATE>        superseded      oudsm-0.1       14.1.2.1.0
Install complete
2               <DATE>        deployed        oudsm-0.1       14.1.2.1.0
Upgrade complete
```

Rollback to the previous working revision by running:

```
helm rollback <release> <revision> -n <domain_namespace>
```

For example:

```
helm rollback oudsm 1 -n oudsmns
```

Once you have resolved the issue in the yaml files, run the helm upgrade command outlined earlier to recreate the logstash pod.

10.5 Verifying and Accessing the Kibana Console

To access the Kibana console you will need the Kibana URL as per Installing the Monitoring and Visualization Software.

Kibana Version 7.8.X or Higher

1. Access the Kibana console with `http://<hostname>:<port>/app/kibana` and login with your username and password.
2. From the Navigation menu, navigate to **Management** > **Kibana** > **Index Patterns**.
3. In the **Create Index Pattern** page enter `oudsmlogs*` for the **Index pattern** and click **Next Step**.
4. In the **Configure settings** page, from the **Time Filter field name** drop down menu select `@timestamp` and click **Create index pattern**.
5. Once the index pattern is created click on **Discover** in the navigation menu to view the OUDSM logs.

Kibana 7.7.x or Lower

1. Access the Kibana console with `http://<hostname>:<port>/app/kibana` and login with your username and password.
2. From the Navigation menu, navigate to **Management** > **Stack Management**.
3. Click **Data Views** in the **Kibana** section.
4. Click **Create Data View** and enter the following information:
   - Name: `oudsmlogs*`

- Timestamp: `@timestamp`

5. Click **Create Data View**.
6. From the Navigation menu, click **Discover** to view the log file entries.
7. From the drop down menu, select `oudsmlogs*` to view the log file entries.

11

Monitoring an Oracle Unified Directory Services Manager Instance

You can monitor and Oracle Unified Directory Services Manager (OUDSM) instance using Prometheus and Grafana.

This chapter includes the following topics:

- Creating a Kubernetes Namespace for Monitoring
- Adding Prometheus and Grafana Helm Repositories
- Installing the Prometheus Operator
- Viewing Prometheus and Grafana Objects
- Adding the NodePort for Grafana
- Verifying Monitoring Using the Grafana GUI

11.1 Creating a Kubernetes Namespace for Monitoring

Create a Kubernetes namespace to provide a scope for Prometheus and Grafana objects, such as pods and services, that you create in the environment.

To create your namespace issue the following command:

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

For example:

```
kubectl create namespace monitoring
```

The output will look similar to the following:

```
namespace/monitoring created
```

11.2 Adding Prometheus and Grafana Helm Repositories

1. Add the Prometheus and Grafana Helm repositories by issuing the following command:

   ```
   helm repo add prometheus https://prometheus-community.github.io/helm-charts
   ```

   The output will look similar to the following:

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

2. Run the following command to update the repositories:

```
helm repo update
```

   The output will look similar to the following:

```
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "stable" chart repository
...Successfully got an update from the "prometheus" chart repository
...Successfully got an update from the "prometheus-community" chart
repository

Update Complete.  Happy Helming!
```

11.3 Installing the Prometheus Operator

Install the Prometheus operator using the helm command:

```
helm install <release_name> prometheus/kube-prometheus-stack -n <namespace>
```

For example:

```
helm install monitoring prometheus/kube-prometheus-stack -n monitoring
```

The output should look similar to the following:

```
NAME: monitoring
LAST DEPLOYED: <DATE>
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace monitoring get pods -l "release=monitoring"

Visit https://github.com/prometheus-operator/kube-prometheus for instructions
on how to create & configure Alertmanager and Prometheus instances using the
Operator.
```

Note:

If your cluster does not have access to the internet to pull external images, such as prometheus or grafana, you must load the images in a local container registry. You must then install as follows:

```
helm install \
--set grafana.image.registry="container-registry.example.com" \
--set grafana.image.repository="grafana/grafana" \
--set grafana.image.tag=8.4.2 \
monitoring prometheus/kube-prometheus-stack \
-n monitoring
```

11.4 Viewing Prometheus and Grafana Objects

View the objects created for Prometheus and Grafana by issuing the following command

```
kubectl get all,service,pod -o wide -n <namespace>
```

For example:

```
kubectl get all,service,pod -o wide -n monitoring
```

The output will look similar to the following:

```
NAME                                                          READY
STATUS     RESTARTS   AGE   IP               NODE               NOMINATED
NODE   READINESS GATES
pod/alertmanager-monitoring-kube-prometheus-alertmanager-0    2/2
Running    0          36s   10.244.1.78      <worker-node>    <none>
<none>
pod/monitoring-grafana-578f79599c-qc9gd                       3/3
Running    0          47s   10.244.2.200     <worker-node>    <none>
<none>
pod/monitoring-kube-prometheus-operator-65cdf7995-kndgg       1/1
Running    0          47s   10.244.2.199     <worker-node>    <none>
<none>
pod/monitoring-kube-state-metrics-56bfd4f44f-85l4p            1/1
Running    0          47s   10.244.1.76      <worker-node>    <none>
<none>
pod/monitoring-prometheus-node-exporter-g2x9g                 1/1
Running    0          47s   100.102.48.121   <master-node>    <none>
<none>
pod/monitoring-prometheus-node-exporter-p9kkq                 1/1
Running    0          47s   100.102.48.84    <worker-node>    <none>
<none>
pod/monitoring-prometheus-node-exporter-rzhrd                 1/1
Running    0          47s   100.102.48.28    <worker-node>    <none>
<none>
pod/prometheus-monitoring-kube-prometheus-prometheus-0        2/2
```

```
Running   0          35s   10.244.1.79      <worker-node>   <none>
<none>

NAME                                              TYPE        CLUSTER-
IP        EXTERNAL-IP   PORT(S)                     AGE   SELECTOR
service/alertmanager-operated                     ClusterIP
None             <none>        9093/TCP,9094/TCP,9094/UDP   36s
app.kubernetes.io/name=alertmanager
service/monitoring-grafana                        ClusterIP
10.110.193.30    <none>        80/TCP                       47s
app.kubernetes.io/instance=monitoring,app.kubernetes.io/name=grafana
service/monitoring-kube-prometheus-alertmanager   ClusterIP
10.104.2.37      <none>        9093/TCP                     47s
alertmanager=monitoring-kube-prometheus-alertmanager,app.kubernetes.io/
name=alertmanager
service/monitoring-kube-prometheus-operator       ClusterIP
10.99.162.229    <none>        443/TCP                      47s   app=kube-
prometheus-stack-operator,release=monitoring
service/monitoring-kube-prometheus-prometheus     ClusterIP
10.108.161.46    <none>        9090/TCP                     47s
app.kubernetes.io/name=prometheus,prometheus=monitoring-kube-prometheus-
prometheus
service/monitoring-kube-state-metrics             ClusterIP
10.111.162.185   <none>        8080/TCP                     47s
app.kubernetes.io/instance=monitoring,app.kubernetes.io/name=kube-state-
metrics
service/monitoring-prometheus-node-exporter       ClusterIP
10.109.21.136    <none>        9100/TCP                     47s
app=prometheus-node-exporter,release=monitoring
service/prometheus-operated                       ClusterIP
None             <none>        9090/TCP                     35s
app.kubernetes.io/name=prometheus

NAME                                                 DESIRED   CURRENT
READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE   CONTAINERS
IMAGES                                     SELECTOR
daemonset.apps/monitoring-prometheus-node-exporter   3         3
3       3            3           <none>          47s   node-exporter
quay.io/prometheus/node-exporter:v1.3.1   app=prometheus-node-
exporter,release=monitoring

NAME                                                  READY   UP-TO-DATE
AVAILABLE   AGE   CONTAINERS
IMAGES
                  SELECTOR
deployment.apps/monitoring-grafana                    1/1     1
1           47s   grafana-sc-dashboard,grafana-sc-datasources,grafana
quay.io/kiwigrid/k8s-sidecar:1.15.6,quay.io/kiwigrid/k8s-
sidecar:1.15.6,grafana/grafana:8.4.2   app.kubernetes.io/
instance=monitoring,app.kubernetes.io/name=grafana
deployment.apps/monitoring-kube-prometheus-operator   1/1     1
1           47s   kube-prometheus-stack
quay.io/prometheus-operator/prometheus-
operator:v0.55.0                                          app=kube-prometheus-
stack-operator,release=monitoring
deployment.apps/monitoring-kube-state-metrics         1/1     1
```

```
1           47s   kube-state-metrics
k8s.gcr.io/kube-state-metrics/kube-state-
metrics:v2.4.1                                         app.kubernetes.io/
instance=monitoring,app.kubernetes.io/name=kube-state-metrics

NAME                                                            DESIRED
CURRENT   READY   AGE   CONTAINERS
IMAGES
                  SELECTOR
replicaset.apps/monitoring-grafana-578f79599c                   1
1         1       47s   grafana-sc-dashboard,grafana-sc-datasources,grafana
quay.io/kiwigrid/k8s-sidecar:1.15.6,quay.io/kiwigrid/k8s-
sidecar:1.15.6,grafana/grafana:8.4.2   app.kubernetes.io/
instance=monitoring,app.kubernetes.io/name=grafana,pod-template-
hash=578f79599c
replicaset.apps/monitoring-kube-prometheus-operator-65cdf7995   1
1         1       47s   kube-prometheus-stack
quay.io/prometheus-operator/prometheus-
operator:v0.55.0                                         app=kube-prometheus-
stack-operator,pod-template-hash=65cdf7995,release=monitoring
replicaset.apps/monitoring-kube-state-metrics-56bfd4f44f        1
1         1       47s   kube-state-metrics
k8s.gcr.io/kube-state-metrics/kube-state-
metrics:v2.4.1                                         app.kubernetes.io/
instance=monitoring,app.kubernetes.io/name=kube-state-metrics,pod-template-
hash=56bfd4f44f

NAME
READY   AGE   CONTAINERS                     IMAGES
statefulset.apps/alertmanager-monitoring-kube-prometheus-alertmanager
1/1     36s   alertmanager,config-reloader   quay.io/prometheus/
alertmanager:v0.23.0,quay.io/prometheus-operator/prometheus-config-
reloader:v0.55.0
statefulset.apps/prometheus-monitoring-kube-prometheus-prometheus
1/1     35s   prometheus,config-reloader     quay.io/prometheus/
prometheus:v2.33.5,quay.io/prometheus-operator/prometheus-config-
reloader:v0.55.0
```

11.5 Adding the NodePort for Grafana

1. Edit the grafana service to add the NodePort:

   ```
   kubectl edit service/<deployment_name>-grafana -n <namespace>
   ```

   For example:

   ```
   kubectl edit service/monitoring-grafana -n monitoring
   ```

   Note:

   This opens an edit session for the domain where parameters can be changed using standard vi commands.

2. Change the ports entry and add `nodePort: 30091` and `type: NodePort`:

```
ports:
- name: http-web
  nodePort: 30091
  port: 80
  protocol: TCP
  targetPort: 3000
selector:
  app.kubernetes.io/instance: monitoring
  app.kubernetes.io/name: grafana
sessionAffinity: None
type: NodePort
```

3. Save the file and exit (`:wq`).

11.6 Verifying Monitoring Using the Grafana GUI

1. Access the Grafana GUI using `http://<HostIP>:<nodeport>` and login with `admin/prom-operator`. Change the password when prompted.
2. Download the K8 Cluster Detail Dashboard json file from: K8 Cluster Detail Dashboard.
3. Import the Grafana dashboard by navigating on the left hand menu to **Dashboards** > **Import**.
4. Click **Upload JSON file** and select the json downloaded file.
5. In the **Prometheus** drop down box select `Prometheus`. Click **Import**. The dashboard should be displayed.
6. Verify your installation by viewing some of the customized dashboard views.

12

Patching and Upgrading

This chapter includes the following topics:

- Patching and Upgrading Within 14.1.2
- Upgrading from Oracle Unified Directory Services Manager 12.2.1.4 to 14.1.2

12.1 Patching and Upgrading Within 14.1.2

The instructions in this section relate to patching or upgrading an existing 14.1.2.1.0 Oracle Unified Directory Services Manager (OUDSM) Kubernetes deployment with a new OUDSM 14c container image.

This section contains the following topics:

- Performing the Upgrade Within 14.1.2
- Rolling Back the Upgrade Within 14.1.2

12.1.1 Performing the Upgrade Within 14.1.2

Run the following steps to patch or upgrade an existing 14.1.2.1.0 Oracle Unified Directory Services Manager (OUDSM) Kubernetes deployment with a new OUDSM 14c container image:

Note:

Administrators should be aware of the following:

- If you are not using Oracle Container Registry or your own container registry, then you must first load the new container image on all nodes in your Kubernetes cluster.

1. Navigate to the `$WORKDIR/kubernetes/helm14c` directory:

   ```
   cd $WORKDIR/kubernetes/helm14c
   ```

2. Create an `oudsm-patch-override.yaml` file that contains:

   ```
   image:
     repository: <image_location>
     tag: <image_tag>
    imagePullSecrets:
      - name: orclcred
   ```

For example:

```
image:
  repository: container-registry.oracle.com/middleware/oudsm_cpu
  tag: 14.1.2.1.0-jdk17-ol8-<YYMMDD>
imagePullSecrets:
  - name: orclcred
```

> **Note:**
>
> If you are not using Oracle Container Registry or your own container registry for your OUDSM container image, then you can remove the following:
>
> ```
> imagePullSecrets:
>   - name: orclcred
> ```

3. Take a backup of the persistent volume directory:

   ```
   sudo cp -rp <persistent_volume>/oudsm_user_projects <persistent_volume>/
   oudsm_user_projects_bkp14c_<tag>
   ```

   For example:

   ```
   sudo cp -rp /nfs_volumes/oudsmpv/oudsm_user_projects /nfs_volumes/oudsmpv/
   oudsm_user_projects_bkp14c_old
   ```

4. Run the following command to upgrade the deployment:

   ```
   helm upgrade --namespace <namespace> \
   --values oudsm-patch-override.yaml \
   <release_name> oudsm --reuse-values
   ```

   For example:

   ```
   helm upgrade --namespace oudsmns \
   --values oudsm-patch-override.yaml \
   oudsm oudsm --reuse-values
   ```

   The `helm upgrade` will perform a rolling restart of the OUDSM pods.

5. Run the following command and make sure all the OUDSM pods are started:

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

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

For example:

```
kubectl get pods -n oudsmns -w
```

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

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

6. Once the pods are up and running, you can run the following command to show the new OUDSM 14c container image is used by the pods:

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

   For example:

   ```
   kubectl describe pod oudsm-1 -n oudsmns | grep image
   ```

   The output will look similar to the following:

   ```
   ...
   Containers:
     oudsm-1:
       Container ID:   cri-o://
   6a35ef3a0721015aa99b2aaeebdc96528c8166db7bf36176f0b9665e43c10ded
       Image:          container-registry.oracle.com/middleware/
   oudsm_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD>
       Image ID:       container-registry.oracle.com/middleware/
   oudsm_cpu@sha256:2ae38d6bdca4c411d6b62289cf80563f611a1fdcbaf01632be7b4fa6a4
   169000
   ```

7. Verify the OUDSM deployment. See, Validating OUDSM URLs.

12.1.2 Rolling Back the Upgrade Within 14.1.2

If the Oracle Unified Directory Services Manager (OUDSM) upgrade fails, you can rollback to the previous OUDSM 14c container image, fix the issue, and then retry the upgrade.

You can also rollback if the upgrade was successful but you subsequently have functional issues.

To rollback the Oracle Unified Directory (OUDSM) installation perform the following steps:

1. Rollback the OUDSM deployment using the following command:

   ```
   helm rollback <release_name> -n <namespace>
   ```

   For example:

   ```
    helm rollback oudsm -n oudsmns
   ```

The output will look similar to the following:

```
Rollback was a success! Happy Helming!
```

The `helm rollback` will perform a rolling restart of the OUDSM pods.

2. Run the following command and make sure all the OUDSM pods are started:

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

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

   For example:

   ```
   kubectl get pods -n oudsmns -w
   ```

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

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

3. Once the pods are up and running, you can run the following command to show the previous OUDSM 14c container image is used by the pods:

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

   For example:

   ```
   kubectl describe pod oudsm-1 -n oudsmns | grep image
   ```

   The output will look similar to the following:

   ```
   ...
   Containers:
     oudsm-1:
       Container ID:   cri-o://
   6a35ef3a0721015aa99b2aaeebdc96528c8166db7bf36176f0b9665e43c10ded
       Image:          container-registry.oracle.com/middleware/
   oudsm_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD>
       Image ID:       container-registry.oracle.com/middleware/
   oudsm_cpu@sha256:2ae38d6bdca4c411d6b62289cf80563f611a1fdcbaf01632be7b4fa6a4
   169000
   ```

4. Verify the OUDSM deployment. See, Validating OUDSM URLs.

12.2 Upgrading from Oracle Unified Directory Services Manager 12.2.1.4 to 14.1.2

Upgrading an existing Oracle Unified Directory Serivces Manager (OUDSM) 12.2.1.4 deployment on Kubernetes to OUDSM 14.1.2.1.0, is not supported.

You must install a new 14.1.2.1.0 OUDSM deployment as outlined in Installing Oracle Unified Directory Services Manager on Kubernetes.

13
General Troubleshooting

This chapter includes the following topics:

- Checking the Status of an OUDSM Namespace
- Viewing Pod Logs
- Viewing Pod Descriptions

13.1 Checking the Status of an OUDSM Namespace

To check the status of objects in a namespace use the following command:

```
kubectl --namespace <namespace> get nodes,pod,service,secret,pv,pvc,ingress -
o wide
```

For example:

```
kubectl --namespace oudsmns get pod,service,secret,pv,pvc,ingress -o wide
```

The output will look similar to the following:

```
NAME          READY   STATUS    RESTARTS   AGE      IP
NODE             NOMINATED NODE   READINESS GATES
pod/oudsm-1   1/1     Running   0          3d21h    10.244.0.173 <Worker
Node>    <none>           <none>

NAME                 TYPE          CLUSTER-IP        EXTERNAL-IP
PORT(S)            AGE     SELECTOR
service/oudsm-1      ClusterIP     10.96.102.147     <none>          7001/
TCP,7002/TCP 3d21h   app.kubernetes.io/instance=oudsm,app.kubernetes.io/
name=oudsm,oudsm/instance=oudsm-1
service/oudsm-lbr    ClusterIP     10.96.148.14      <none>          7001/
TCP,7002/TCP 3d21h   app.kubernetes.io/instance=oudsm,app.kubernetes.io/
name=oudsm

NAME                                TYPE                              DATA AGE
secret/orclcred                     kubernetes.io/dockerconfigjson    1
4d17h
secret/oudsm-creds                  opaque                            2
3d21h
secret/oudsm-tls-cert               kubernetes.io/tls                 2
3d21h
secret/sh.helm.release.v1.oudsm.v1  helm.sh/release.v1                1
3d21h
```

13.2 Viewing Pod Logs

To view logs for a pod use the following command:

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

For example:

```
kubectl logs oudsm-1 -n oudsmns
```

Note:

If you add `-f` to the command, then the log will be streamed.

13.3 Viewing Pod Descriptions

Details about a pod can be viewed using the `kubectl describe` command:

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

For example:

```
kubectl describe pod oudsm-1 -n oudsmns
```

14 Deleting an OUDSM Deployment

The following steps can be followed to delete an Oracle Unified Directory Services Manager (OUDSM) deployment:

1. Run the following command to find the deployment release name:

   ```
   helm --namespace <namespace> list
   ```

   For example:

   ```
   helm --namespace oudsmns list
   ```

   The output will look similar to the following:

   ```
   NAME    NAMESPACE       REVISION        UPDATED    STATUS
   CHART           APP VERSION
   oudsm   oudsmns         2               <DATE>     deployed
   oudsm-0.1       14.1.2.1.0
   ```

2. Delete the deployment using the following command:

   ```
   helm uninstall --namespace <namespace> <release>
   ```

   For example:

   ```
   helm uninstall --namespace oudsmns oudsm
   ```

   The output will look similar to the following:

   ```
   release "oudsm" uninstalled
   ```

3. Run the following command to view the status:

   ```
   kubectl --namespace oudsmns get pod,service,secret,pv,pvc,ingress -o wide
   ```

   Initially the pods and persistent volume (PV) and persistent volume claim (PVC) will move to a `Terminating` status:

   ```
   NAME           READY   STATUS        RESTARTS   AGE   IP
   NODE             NOMINATED NODE   READINESS GATES

   pod/oudsm-1    1/1     Terminating   0          24m   10.244.1.180
   <Worker Node>    <none>           <none>

   NAME                        TYPE                                   DATA
   ```

```
AGE
secret/default-token-msmmd   kubernetes.io/service-account-token   3
3d20h
secret/dockercred            kubernetes.io/dockerconfigjson        1
3d20h
secret/orclcred              kubernetes.io/dockerconfigjson        1
3d20h

NAME                             CAPACITY   ACCESS MODES   RECLAIM
POLICY   STATUS        CLAIM                         STORAGECLASS
REASON   AGE    VOLUMEMODE
persistentvolume/oudsm-pv        20Gi       RWX
Delete           Terminating   oudsmns/oud-ds-rs-pvc
manual                       24m    Filesystem

NAME                              STATUS        VOLUME         CAPACITY
ACCESS MODES   STORAGECLASS   AGE   VOLUMEMODE
persistentvolumeclaim/oudsm-pvc   Terminating   oud-ds-rs-pv   20Gi
RWX            manual         24m   Filesystem
```

4. Run the command again until the pods, PV and PVC disappear.
5. If the PV or PVC's don't delete, remove them manually:

   ```
   kubectl delete pvc oudsm-pvc -n oudsmns
   kubectl delete pv oudsm-pv -n oudsmns
   ```

6. Delete the persistent volume contents:

   ```
   cd <persistent_volume>/oudsm_user_projects
   rm -rf *
   ```

   For example:

   ```
   cd /nfs_volumes/oudsmpv/oudsm_user_projects
   rm -rf *
   ```

7. Delete the ingress controller:

   ```
   helm delete lbr-nginx -n <namespace>
   ```

   For example:

   ```
   helm delete lbr-nginx -n mynginxns
   ```

Part IV
Appendices

This section includes the following topics:

- Configuration Parameters for the oudsm Helm Chart

A

Configuration Parameters for the oudsm Helm Chart

The following table lists the configurable parameters of the `oudsm` chart and their default values.

Parameter	Description	Default Value
replicaCount	Number of Oracle Unified Directory Services Manager instances/pods/services to be created	1
restartPolicyName	restartPolicy to be configured for each POD containing Oracle Unified Directory Services Manager instance	OnFailure
image.repository	Oracle Unified Directory Services Manager Image Registry/ Repository and name. Based on this, image parameter would be configured for Oracle Unified Directory Services Manager pods/containers.	oracle/oudsm
image.tag	Oracle Unified Directory Services Manager Image Tag. Based on this, image parameter would be configured for Oracle Unified Directory Services Manager pods/containers	14.1.2.1.0
image.pullPolicy	Policy to pull the image.	IfnotPresent
imagePullSecrets.name	Name of Secret resource containing private registry credentials.	regcred
nameOverride	override the fullname with this name.	
fullnameOverride	Overrides the fullname with the provided string.	
serviceAccount.create	Specifies whether a service account should be created.	true
serviceAccount.name	If not set and create is true, a name is generated using the fullname template.	oudsm-< fullname >-token-< randomalphanum >
podSecurityContext	Security context policies to add to the controller pod.	
securityContext	Security context policies to add by default.	
service.type	Type of controller service to create.	ClusterIP
nodeSelector	Node labels for pod assignment.	

Parameter	Description	Default Value
tolerations	Node taints to tolerate.	
affinity	Node/pod affinities.	
ingress.enabled		true
ingress.type	Supported value: nginx.	nginx
ingress.host	Hostname to be used with Ingress Rules. If not set, hostname would be configured according to fullname. Hosts would be configured as < fullname >-http.< domain >, < fullname >-http-0.< domain >, < fullname >-http-1.< domain >, etc.	
ingress.domain	Domain name to be used with Ingress Rules. In ingress rules, hosts would be configured as < host >.< domain >, < host >-0.< domain >, < host >-1.< domain >, etc.	
ingress.backendPort		http
ingress.nginxAnnotations		{ kubernetes.io/ingress.class: “nginx" nginx.ingress.kubernetes.io/ affinity-mode: “persistent” nginx.ingress.kubernetes.io/ affinity: “cookie” }
ingress.ingress.tlsSecret	Secret name to use an already created TLS Secret. If such secret is not provided, one would be created with name < fullname >-tls-cert. If the TLS Secret is in different namespace, name can be mentioned as < namespace >/< tlsSecretName >	
ingress.certCN	Subject’s common name (cn) for SelfSigned Cert.	< fullname >
ingress.certValidityDays	Validity of Self-Signed Cert in days	365
ingress.ingress.tlsSecret	Secret name to use an already created TLS Secret. If such secret is not provided, one would be created with name < fullname >-tls-cert. If the TLS Secret is in different namespace, name can be mentioned as < namespace >/< tlsSecretName >	
ingress.certCN	Subject’s common name (cn) for SelfSigned Cert.	< fullname >
ingress.certValidityDays	Validity of Self-Signed Cert in days	365

Parameter	Description	Default Value
secret.enabled	If enabled it will use the secret created with base64 encoding. if value is false, secret would not be used and input values (through –set, –values, etc.) would be used while creation of pods.	true
secret.name	Secret name to use an already created xecret.	oudsm-< fullname >-creds
secret.type	Specifies the type of the secret	Opaque
persistence.enabled	If enabled, it will use the persistent volume. if value is false, PV and PVC would not be used and pods would be using the default emptyDir mount volume.	true
persistence.pvname	pvname to use an already created Persistent Volume , If blank will use the default name.	oudsm-< fullname >-pv
persistence.pvcname	pvcname to use an already created Persistent Volume Claim , If blank will use default name.	oudsm-< fullname >-pvc
persistence.type	supported values: either filesystem or networkstorage or blockstorage or custom.	filesystem
persistence.filesystem.hostPath.path	The path location mentioned should be created and accessible from the local host provided with necessary privileges for the user.	/scratch/shared/oudsm_user_projects
persistence.networkstorage.nfs.path	Path of NFS Share location.	/scratch/shared/oudsm_user_projects
persistence.networkstorage.nfs.server	IP or hostname of NFS Server.	0.0.0.0
persistence.custom.*	Based on values/data, YAML content would be included in PersistenceVolume Object.	
persistence.accessMode	Specifies the access mode of the location provided. ReadWriteMany for Filesystem/NFS, ReadWriteOnce for block storage.	ReadWriteMany
persistence.size	Specifies the size of the storage.	10Gi
persistence.storageClassCreate	If true, it will create the storageclass. if value is false, please provide existing storage class (storageClass) to be used.	empty
persistence.storageClass	Specifies the storageclass of the persistence volume.	empty
persistence.provisioner	If storageClassCreate is true, provide the custom provisioner if any.	kubernetes.io/is-default-class
persistence.annotations	specifies any annotations that will be used.	{ }
oudsm.adminUser	oudsm.adminUser	weblogic

Parameter	Description	Default Value
oudsm.adminPass	Password for Weblogic Administration User	
oudsm.startupTime	Expected startup time. After specified seconds readinessProbe would start	900
oudsm.livenessProbeInitialDelay	Parameter to decide livenessProbe initialDelaySeconds	1200
elk.logStashImage	The version of logstash you want to install.	logstash:8.3.1
elk.sslenabled	If SSL is enabled for ELK set the value to true, or if NON-SSL set to false. This value must be lowercase.	TRUE
elk.eshosts	The URL for sending logs to Elasticsearch. HTTP if NON-SSL is used.	https://elasticsearch.example.com:9200
elk.esuser	The name of the user for logstash to access Elasticsearch.	logstash_internal
elk.espassword	The password for ELK_USER.	password
elk.esapikey	The API key details.	apikey
elk.esindex	The log name.	oudlogs-00001
elk.imagePullSecrets	Secret to be used for pulling logstash image.	dockercred