Oracle® FMW

Deploying and Managing Oracle Unified Directory on Kubernetes

G22965-03
July 2025

Oracle FMW Deploying and Managing Oracle Unified Directory on Kubernetes,

G22965-03

Primary Author: Russell Hodgson

Contents

List of Figures

1

What's New in This Release?

This preface shows current and past versions of Oracle Unified Directory (OUD) 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 14c on Kubernetes

Date	Version	Change
July 2025	14.1.2.1.0 GitHub release version 25.3.1	Supports Oracle Unified Directory 14.1.2.1.0 domain deployment using the July 2025 container image which contains the July Patch Set Update (PSU) and other fixes released with the Critical Patch Update (CPU) program. The GitHub release version is the latest version of the deployment scripts used in Setting Up the Code Repository for OUD.
April 2025	14.1.2.1.0 GitHub release version 25.2.1	Supports Oracle Unified Directory 14.1.2.1.0 domain deployment using the April 2025 container image which contains the April Patch Set Update (PSU) and other fixes released with the Critical Patch Update (CPU) program. The GitHub release version is the latest version of the deployment scripts used in Setting Up the Code Repository for OUD.
March 2025	14.1.2.1.0 GitHub release version 25.1.3	Initial release of Oracle Unified Directory 14.1.2.1.0 on Kubernetes. Supports Oracle Unified Directory 14.1.2.1.0 deployment using the OUD container image. The GitHub release version is the latest version of the deployment scripts used in Setting Up the Code Repository for OUD.

Part I

Introduction to Oracle Unified Directory on Kubernetes

Oracle Unified Directory (OUD) can be deployed on Kubernetes.

This section includes the following chapters:

2

Introducing Oracle Unified Directory on Kubernetes

Oracle Unified Directory (OUD) is supported for deployment on Kubernetes.

This chapter includes the following topics:

2.1 Overview of Oracle Unified Directory on Kubernetes

Oracle Unified Directory provides a comprehensive Directory Solution for robust Identity Management. Oracle Unified Directory is an all-in-one directory solution with storage, proxy, synchronization and virtualization capabilities. While unifying the approach, it provides all the services required for high-performance Enterprise and carrier-grade environments. Oracle Unified Directory ensures scalability to billions of entries, ease of installation, elastic deployments, enterprise manageability and effective monitoring.

Oracle Unified Directory 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 on Kubernetes

The key features of using Oracle Unified Directory (OUD) 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 OUD can run seamlessly across different environments, including on-premises data centers, public clouds, and hybrid setups.
- **Scalability**: Containers allow organizations to scale their security components dynamically, ensuring that they can handle fluctuating workloads.
- **Improved Resource Efficiency**: Containers provide lightweight, efficient runtime environments that optimize resource utilization compared to traditional virtual machines.

3

About the Kubernetes Deployment

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

This chapter includes the following topics:

3.1 What is Kubernetes?

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

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

Kubernetes provides:

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

When deploying Kubernetes, Oracle highly recommends that you use the traditional recommendations of keeping different workloads in separate Kubernetes clusters. For

example, it is not a good practice to mix development and production workloads in the same Kubernetes cluster.

3.2 About the Kubernetes Architecture

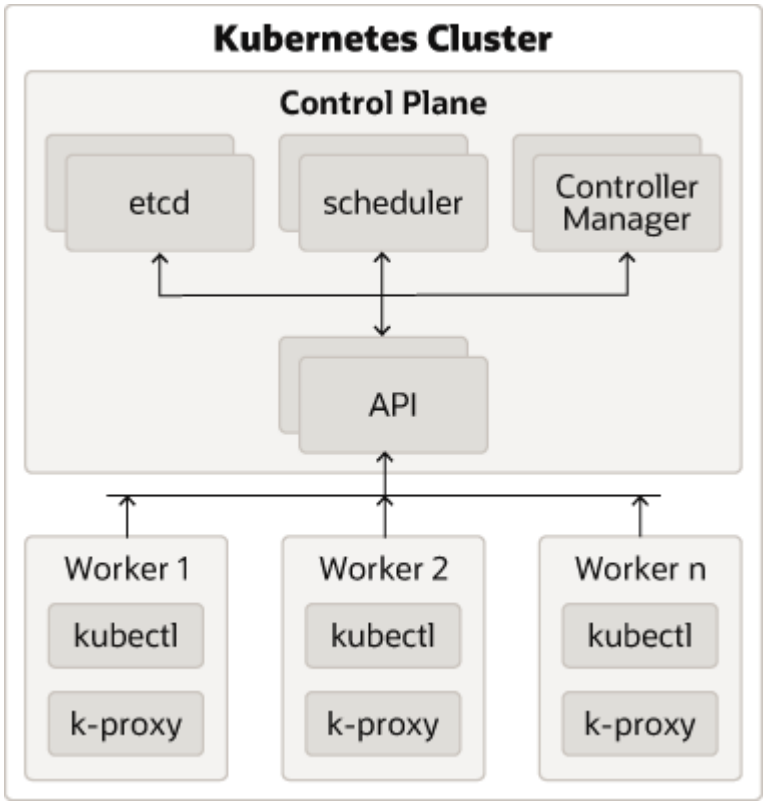
A Kubernetes host consists of a control plane and worker nodes.

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

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

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

Figure 3-1 An Illustration of the Kubernetes Cluster



Description of Components:

- **Control Plane**: The control plane comprises the following:
  - kube-api server: The API server is a component of the control plane that exposes the Kubernetes APIs.
  - etcd: It is used to store the Kubernetes backing store and all the cluster data.

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

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

3.3 Key Components Used By an OUD Deployment

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

Container Image

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

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

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

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

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

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

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

Pods

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

In an Oracle Unified Directory (OUD) deployment, each OUD 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 an Unknown state 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.

For pods in a Terminating state, the OUD deployment creates a cronjob that automatically deletes these pods for you.

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 OUD

domain configuration, is still available after you restart a pod, that is to say, that the data is persistent.

There are three 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.
3. Mount the PV to a block volume in OCI. The Oracle Cloud Infrastructure (OCI) Block Volume service enables you to dynamically provision and manage block storage volumes. You can create, attach, connect, move volumes, and adjust their performance as needed to meet your storage, performance, and application requirements. Once a volume is attached and connected to an instance, it behaves like a standard hard drive.
   In Kubernetes, the OUD StatefulSet controller manages a group of pods with stable network identities and persistent storage. This allows each pod to access dedicated OCI Block Volumes, ensuring data is preserved even during pod restarts, rescheduling, or scaling events.

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 OUD deployment uses the namespace oudns.

Part II

Installing Oracle Unified Directory on Kubernetes

Install Oracle Unified Directory (OUD) on Kubernetes.

This section contains the following chapters:

4

Before You Begin

This documentation explains how to configure Oracle Unified Directory (OUD) 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 Oracle Unified Directory 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 OUD 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 Oracle Unified Directory on Kubernetes

This section provides information about the system requirements and limitations for deploying and running Oracle Unified Directory (OUD) 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, a shared file system, or block storage. If you intend to use assured replication in OUD, you must have a persistent volume available that uses a Network File System (NFS) mount, or a shared file system for the config volume. See Enabling Assured Replication (Optional).
- 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 OUD 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 (OUD) deployment on Kubernetes, you must prepare your environment.

This chapter contains the following topics:

6.1 Confirming the Kubernetes Cluster is Ready

As per System Requirements for Oracle Unified Directory 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 OUD Container Image

The Oracle Unified Directory (OUD) Kubernetes deployment requires access to an OUD container image.

Prebuilt OUD Container Image

The latest prebuilt OUD 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.

> **Note**
>
> Administrators should be aware of the following:
>
> - The OUD container images available can be found on Oracle Container Registry, by navigating to **Middleware** > **oud** for the initial March 2025 release, and **Middleware** > **oud_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/oud_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/oud: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 OUD 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

> **Note**
>
> This section should not be followed if using block storage.

As referenced in System Requirements for Oracle Unified Directory 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/oudpv.

Perform the following steps:

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

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

For example:

```
cd /nfs_volumes/oudpv
mkdir oud_user_projects
sudo chown -R 1000:0 oud_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>/oud_user_projects
touch fileadmin.txt
ls fileadmin.txt
```

For example:

```
cd /nfs_volumes/oudpv/oud_user_projects
touch fileadmin.txt
ls fileadmin.txt
```

6.4 Setting Up the Code Repository for OUD

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

Oracle Unified Directory (OUD) deployment on Kubernetes leverages deployment scripts provided by Oracle for creating OUD containers, using the Helm charts provided.

Perform the following steps to set up the OUD 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 /oudscripts
```

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

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

For example:

```
cd /oudscripts
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/OracleUnifiedDirectory
```

For example:

```
export WORKDIR=/oudscripts/fmw-kubernetes/OracleUnifiedDirectory
```

7

Creating Oracle Unified Directory Instances

This chapter demonstrates how to deploy Oracle Unified Directory (OUD) 14c instance(s) and replicated instances using the Helm package manager for Kubernetes. The helm chart can be used to deploy an Oracle Unified Directory instance as a base, with configured sample entries, and multiple replicated Oracle Unified Directory instances, pods, and services, based on the specified replicaCount. Based on the configuration, this chart deploys the following objects in the specified namespace of a Kubernetes cluster:

- Service Account
- Secret
- Persistent Volume and Persistent Volume Claim
- Pod(s)/Container(s) for Oracle Unified Directory Instances
- Services for interfaces exposed through Oracle Unified Directory Instances
- Ingress configuration

This chapter contains the following topics:

7.1 Creating a Kubernetes Namespace

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

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

For example:

```
kubectl create namespace oudns
```

The output will look similar to the following:

```
namespace/oudns created
```

7.2 Creating a Kubernetes Secret for the Container Registry

Create a Kubernetes secret to store the credentials for the container registry where the Oracle Unified Directory (OUD) 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=oudns
```

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

- If using Oracle Container Registry to pull the OUD 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** > **oud** and accept the license agreement. For future releases (post March 25) that contain the latest Patch Set Update (PSU) and other fixes released with the Critical Patch Update (CPU) program, you should navigate to **Middleware** > **oud_cpu**.
- If using your own container registry to store the OUD 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 Create a Kubernetes Secret for Cronjob Images

Once Oracle Unified Directory (OUD) is deployed, if the Kubernetes node where the OUD pod(s) is/are running goes down after the pod eviction time-out, the pod(s) don't get evicted but move to a Terminating state. The pod(s) will then remain in that state forever. To avoid this problem, a cron-job is created during OUD deployment that checks for any pods in Terminating state. If there are any pods in Terminating state, the cron job will delete them. The pods will then start again automatically. This cron job requires access to images on hub.docker.com. A Kubernetes secret must therefore be created to enable access to these images.

1. 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=oudns
```

The output will look similar to the following:

```
secret/dockercred created
```

7.4 Creating OUD Instances

The oud-ds-rs Helm Chart

The oud-ds-rs Helm chart allows you to create or deploy a group of replicated Oracle Unified Directory (OUD) instances along with Kubernetes objects in a specified namespace.

The deployment can be initiated by running the following Helm command with reference to the oud-ds-rs 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 oud-ds-rs Helm Chart.

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

Deploying OUD Instances

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

- Deploying OUD Using a YAML File
- Deploying OUD Using --set Argument

> **Note**
>
> While it is possible to install sample data during the OUD deployment, it is not possible to load your own data via an ldif file. In order to load data in OUD, create the OUD deployment and then use ldapmodify post the ingress deployment. See Using LDAP Utilities.

7.4.1 Deploying OUD Using a YAML File

To deploy Oracle Unified Directory (OUD) using a YAML file:

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

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

2. Create an oud-ds-rs-values-override.yaml as follows:

   ```
   image:
     repository: <image_location>
     tag: <image_tag>
     pullPolicy: IfNotPresent
   imagePullSecrets:
     - name: orclcred
   oudConfig:
    # memory, cpu parameters for both requests and limits for oud instances
     resources:
       limits:
         cpu: "1"
         memory: "4Gi"
       requests:
         cpu: "500m"
         memory: "4Gi"
     rootUserPassword: <password>
     sampleData: "200"
   persistence:
     type: filesystem
     filesystem:
       hostPath:
         path: <persistent_volume>/oud_user_projects
   cronJob:
     kubectlImage:
       repository: bitnami/kubectl
       tag: <version>
       pullPolicy: IfNotPresent

     imagePullSecrets:
       - name: dockercred
   ```

   For example:

   ```
   image:
     repository: container-registry.oracle.com/middleware/oud_cpu
     tag: 14.1.2.1.0-jdk17-ol8-<YYMMDD>
   ```

```
  pullPolicy: IfNotPresent
imagePullSecrets:
  - name: orclcred
oudConfig:
 # memory, cpu parameters for both requests and limits for oud instances
  resources:
    limits:
      cpu: "1"
      memory: "8Gi"
    requests:
      cpu: "500m"
      memory: "4Gi"
  rootUserPassword: <password>
  sampleData: "200"
persistence:
  type: filesystem
  filesystem:
    hostPath:
      path: /nfs_volumes/oudpv/oud_user_projects
cronJob:
  kubectlImage:
    repository: bitnami/kubectl
    tag: 1.30.3
    pullPolicy: IfNotPresent

  imagePullSecrets:
    - name: dockercred
```

The following caveats exist:

- Replace `<password>` with the relevant password.
- `sampleData: "200"` will load 200 sample users into the default baseDN `dc=example,dc=com`. If you do not want sample data, remove this entry. If `sampleData` is set to `1,000,000` users or greater, then you must add the following entries to the yaml file to prevent inconsistencies in dsreplication:

  ```
  deploymentConfig:
    startupTime: 720
    period: 120
    timeout: 60
  ```

- The `<version>` in `kubectlImage: tag:` should be set to the same version as your Kubernetes version (`kubectl version`). For example if your Kubernetes version is 1.30.3 set to `1.30.3`.
- If you are not using Oracle Container Registry or your own container registry for your OUD container image, then you can remove the following:

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

- If your cluster does not have access to the internet to pull external images, such as bitnami/kubectl and busybox, you must load the images in a local container registry. You must then set the following:

```
cronJob:
  kubectlImage:
    repository: container-registry.example.com/bitnami/kubectl
    tag: 1.30.3
    pullPolicy: IfNotPresent

busybox:
  image: container-registry.example.com/busybox
```

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

> **Note**
>
> If you want to use NFS you should ensure that you have a default Kubernetes storage class defined for your environment that allows network storage. For more information on storage classes, see Storage Classes.

```
persistence:
  type: networkstorage
  networkstorage:
    nfs:
      path: <persistent_volume>/oud_user_projects
      server: <NFS IP address>
  # if true, it will create the storageclass. if value is false, please provide existing storage class
(storageClass) to be used.
  storageClassCreate: true
  storageClass: oud-sc
  # if storageClassCreate is true, please provide the custom provisioner if any to use. If you do
not have a custom provisioner, delete this line, and it will use the default class kubernetes.io/is-
default-class.
  provisioner:  kubernetes.io/is-default-class
```

The following caveats exist:

  - If you want to create your own storage class, set `storageClassCreate: true`. If `storageClassCreate: true` it is recommended to set `storageClass` to a value of your choice, and `provisioner` to the provisioner supported by your cloud vendor.
  - If you have an existing storageClass that supports network storage, set `storageClassCreate: false` and `storageClass` to the NAME value returned in "`kubectl get storageclass`". The `provisioner` can be ignored.

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

> **Note**
>
> If you want to use block devices you should ensure that you have a default Kubernetes storage class defined for your environment that allows dynamic storage. Each vendor has its own storage provider but it may not be configured to provide dynamic storage allocation. For more information on storage classes, see Storage Classes.

```
persistence:
  type: blockstorage
  # Specify Accessmode ReadWriteMany for NFS and for block ReadWriteOnce
  accessMode: ReadWriteOnce
  # if true, it will create the storageclass. if value is false, please provide existing storage class
(storageClass) to be used.
  storageClassCreate: true
  storageClass: oud-sc
  # if storageClassCreate is true, please provide the custom provisioner if any to use or else it will use
default.
  provisioner:  oracle.com/oci
```

The following caveats exist:

- If you want to create your own storage class, set `storageClassCreate: true`. If `storageClassCreate: true` it is recommended to set `storageClass` to a value of your choice, and `provisioner` to the provisioner supported by your cloud vendor.
- If you have an existing storageClass that supports dynamic storage, set `storageClassCreate: false` and `storageClass` to the NAME value returned in "`kubectl get storageclass`". The provisioner can be ignored.

- For `resources`, `limits`, and `requests`, the example CPU and memory values shown are for development environments only. For Enterprise Deployments, please review the performance recommendations and sizing requirements in Enterprise Deployment Guide for Oracle Identity and Access Management in a Kubernetes Cluster.

> **Note**
>
> Limits and requests for CPU resources are measured in CPU units. One CPU in Kubernetes is equivalent to 1 vCPU/Core for cloud providers, and 1 hyperthread on bare-metal Intel processors. An "`m`" suffix in a CPU attribute indicates 'milli-CPU', so 500m is 50% of a CPU. Memory can be expressed in various units, where one Mi is one IEC unit mega-byte (1024^2), and one Gi is one IEC unit giga-byte (1024^3). For more information, see Resource Management for Pods and Containers, Assign Memory Resources to Containers and Pods, and Assign CPU Resources to Containers and Pods

> **Note**
>
> The parameters above are also utilized by the Kubernetes Horizontal Pod Autoscaler (HPA). For more details on HPA, see Kubernetes Horizontal Pod Autoscaler.

- If you plan on integrating OUD with other Oracle components then you must specify the following under the oudConfig: section:

  ```
  integration: <Integration option>
  ```

  For example:

  ```
  oudConfig:
    etc...
    integration: <Integration option>
  ```

- If you want to enable Assured Replication, see Enabling Assured Replication (Optional).
- The examples given above are not an exhaustive list of all the parameters and environment variables that can be passed in the override yaml file. For more information, see Configuration Parameters for the oud-ds-rs Helm Chart and Environment Variables Used in the oud-ds-rs Helm Chart.

3. Run the following command to deploy OUD:

   ```
   helm install --namespace <namespace> \
   --values oud-ds-rs-values-override.yaml \
   <release_name> oud-ds-rs
   ```

   For example:

   ```
   helm install --namespace oudns \
   --values oud-ds-rs-values-override.yaml \
   oud-ds-rs oud-ds-rs
   ```

   The output will be similar to that shown in Helm Command Output.

4. Check the OUD deployment as per Verifying the OUD Deployment and Verifying OUD Assured Replication Status.

7.4.2 Deploying OUD Using --set Argument

To deploy Oracle Unified Directory (OUD) using the --set argument:

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

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

2. Run the following command to create OUD instances:

   ```
   helm install --namespace <namespace> \
   --set oudConfig.rootUserPassword=<password> \
   --set persistence.filesystem.hostPath.path=<persistent_volume>/oud_user_projects \
   --set image.repository=<image_location>,image.tag=<image_tag> \
   --set oudConfig.sampleData="200" \
   --set
   oudConfig.resources.limits.cpu="1",oudConfig.resources.limits.memory="8Gi",oudConfig.resources.requests.cp
   u="500m",oudConfig.resources.requests.memory="4Gi" \
   ```

```
--set cronJob.kubectlImage.repository=bitnami/kubectl,cronJob.kubectlImage.tag=<version> \
--set cronJob.imagePullSecrets[0].name="dockercred" \
--set imagePullSecrets[0].name="orclcred" \
<release_name> oud-ds-rs
```

For example:

```
helm install --namespace oudns \
--set oudConfig.rootUserPassword=<password> \
--set persistence.filesystem.hostPath.path=/nfs_volumes/oudpv/oud_user_projects \
--set image.repository=container-registry.oracle.com/middleware/oud_cpu,image.tag=14.1.2.1.0-jdk17-ol8-
<YYMMDD> \
--set oudConfig.sampleData="200" \
--set
oudConfig.resources.limits.cpu="1",oudConfig.resources.limits.memory="8Gi",oudConfig.resources.requests.cp
u="500m",oudConfig.resources.requests.memory="4Gi" \
--set cronJob.kubectlImage.repository=bitnami/kubectl,cronJob.kubectlImage.tag=1.30.3 \
--set cronJob.imagePullSecrets[0].name="dockercred" \
--set imagePullSecrets[0].name="orclcred" \
oud-ds-rs oud-ds-rs
```

The following caveats exist:

- Replace <password> with the relevant password.
- sampleData: "200" will load 200 sample users into the default baseDN dc=example,dc=com. If you do not want sample data, remove this entry. If sampleData is set to 1,000,000 users or greater, then you must set the following arguments to prevent inconsistencies in dsreplication:

  ```
  --set deploymentConfig.startupTime=720,deploymentConfig.period=120,deploymentConfig.timeout=60
  ```

- The <version> in kubectlImage: tag: should be set to the same version as your Kubernetes version (kubectl version). For example if your Kubernetes version is 1.30.3 set to 1.30.3.
- If you are not using Oracle Container Registry or your own container registry for your OUD container image, then you can remove the following:

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

- If using NFS for your persistent volume then use:

  ```
  --set persistence.networkstorage.nfs.path=<persistent_volume>/
  oud_user_projects,persistence.networkstorage.nfs.server=<NFS IP address>
  --set persistence.storageClassCreate="true",persistence.storageClass="oud-
  sc",persistence.provisioner="kubernetes.io/is-default-class"
  ```

- If using Block Device storage for your persistent volume then use:

  ```
  --set persistence.type="blockstorage",persistence.accessMode="ReadWriteOnce"
  --set persistence.storageClassCreate="true",persistence.storageClass="oud-
  sc",persistence.provisioner="oracle.com/oci"
  ```

- For resources, limits, and requests, the example CPU and memory values shown are for development environments only. For Enterprise Deployments, please review the

performance recommendations and sizing requirements in Enterprise Deployment Guide for Oracle Identity and Access Management in a Kubernetes Cluster.

> **Note**
>
> Limits and requests for CPU resources are measured in CPU units. One CPU in Kubernetes is equivalent to 1 vCPU/Core for cloud providers, and 1 hyperthread on bare-metal Intel processors. An "m" suffix in a CPU attribute indicates 'milli-CPU', so 500m is 50% of a CPU. Memory can be expressed in various units, where one Mi is one IEC unit mega-byte (1024^2), and one Gi is one IEC unit giga-byte (1024^3). For more information, see Resource Management for Pods and Containers, Assign Memory Resources to Containers and Pods, and Assign CPU Resources to Containers and Pods

> **Note**
>
> The parameters above are also utilized by the Kubernetes Horizontal Pod Autoscaler (HPA). For more details on HPA, see Kubernetes Horizontal Pod Autoscaler.

- If you plan on integrating OUD with other Oracle components then you must specify the following:

  ```
  --set oudConfig.integration=<Integration option>
  ```

  It is recommended to choose the option covering your minimal requirements. Allowed values include: `no-integration` (no integration), `basic` (Directory Integration Platform), `generic` (Directory Integration Platform, Database Net Services and E-Business Suite integration), `eus` (Directory Integration Platform, Database Net Services, E-Business Suite and Enterprise User Security integration). The default value is `no-integration`

  > **Note**
  >
  > This will enable the integration type only. To integrate OUD with the Oracle component referenced, refer to the relevant product component documentation.

- If you want to enable Assured Replication, see Enabling Assured Replication (Optional).

3. Check the OUD deployment as per Verifying the OUD Deployment and Verifying the OUD Replication.

7.4.3 Enabling Assured Replication (Optional)

If you want to enable assured replication, perform the following steps:

1. Create a directory on the persistent volume as follows:

   ```
   cd <persistent_volume>
   mkdir oud-repl-config
   sudo chown -R 1000:0 oud-repl-config
   ```

For example:

```
cd /nfs_volumes/oudpv/
mkdir oud-repl-config
sudo chown -R 1000:0 oud-repl-config
```

2. Add the following section in the oud-ds-rs-values-override.yaml:

```
replOUD:
  envVars:
    - name: post_dsreplication_dsconfig_3
      value: set-replication-domain-prop --domain-name ${baseDN} --advanced --set assured-type:safe-data --set assured-sd-level:2 --set assured-timeout:5s
    - name: execCmd_1
      value: /u01/oracle/user_projects/${OUD_INSTANCE_NAME}/OUD/bin/dsconfig --no-prompt --hostname ${sourceHost} --port ${adminConnectorPort} --bindDN "${rootUserDN}" --bindPasswordFile /u01/oracle/user_projects/${OUD_INSTANCE_NAME}/admin/rootPwdFile.txt  --trustAll set-replication-domain-prop --domain-name ${baseDN} --advanced --set assured-type:safe-data --set assured-sd-level:2 --set assured-timeout:5s --provider-name "Multimaster Synchronization"
configVolume:
  enabled: true
  type: networkstorage
  storageClassCreate: true
  storageClass: oud-config
  provisioner: kubernetes.io/is-default-class
  networkstorage:
    nfs:
      server: <IP_address>
      path: <persistent_volume>/oud-repl-config
  mountPath: /u01/oracle/config-input
```

The above will enable assured replication with assured type safe-data and assured-sd-level: 2.

> **Note**
>
> The above will enable assured replication with assured type safe-data and assured-sd-level: 2. For more information on OUD Assured Replication, and other options and levels, see, Understanding the Oracle Unified Directory Replication Model.

The following caveats exist:

- post_dsreplication_dsconfig_N and execCmd_N should be a unique key - change the suffix accordingly. For more information on the environment variable and respective keys, see Environment Variables Used in the oud-ds-rs Helm Chart.
- For configVolume the storage can be networkstorage(nfs) or filesystem(hostPath) as the config volume path has to be accessible from all the Kubernetes nodes. Please note that block storage is not supported for configVolume.
- If you want to create your own storage class, set storageClassCreate: true. If storageClassCreate: true it is recommended to set storageClass to a value of your choice, and provisioner to the provisioner supported by your cloud vendor.
- If you have an existing storageClass that supports network storage, set storageClassCreate: false and storageClass to the NAME value returned in "kubectl get storageclass".

Please note that the storage-class should not be the one you used for the persistent volume earlier. The provisioner can be ignored.

7.4.4 Helm Command Output

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

```
NAME: oud-ds-rs
LAST DEPLOYED:  <DATE>
NAMESPACE: oudns
STATUS: deployed
REVISION: 1
NOTES:
#
# Copyright (c) 2025, Oracle and/or its affiliates.
#
#  Licensed under the Universal Permissive License v 1.0 as shown at
# https://oss.oracle.com/licenses/upl
#
#
Since "nginx" has been chosen, follow the steps below to configure nginx ingress controller.
Add Repo reference to helm for retriving/installing Chart for nginx-ingress implementation.
command-# helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

Command helm install to install nginx-ingress related objects like pod, service, deployment, etc.
# helm install --namespace <namespace for ingress> --values nginx-ingress-values-override.yaml lbr-nginx ingress-nginx/ingress-nginx

For details of content of nginx-ingress-values-override.yaml refer README.md file of this chart.

Run these commands to check port mapping and services:
# kubectl --namespace <namespace for ingress> get services -o wide -w lbr-nginx-ingress-controller
# kubectl describe --namespace <namespace for oud-ds-rs chart> ingress.extensions/oud-ds-rs-http-ingress-nginx
# kubectl describe --namespace <namespace for oud-ds-rs chart> ingress.extensions/oud-ds-rs-admin-ingress-nginx

Accessible interfaces through ingress:
(External IP Address for LoadBalancer NGINX Controller can be determined through details associated with lbr-nginx-ingress-controller)

1. OUD Admin REST:
   Port: http/https

2. OUD Data REST:
   Port: http/https

3. OUD Data SCIM:
   Port: http/https

4. OUD LDAP/LDAPS:
   Port: ldap/ldaps

5. OUD Admin LDAPS:
   Port: ldaps
```

Please refer to README.md from Helm Chart to find more details about accessing interfaces and configuration parameters.

7.4.5 Verifying the OUD Deployment

Run the following command to verify the OUD deployment:

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

For example:

```
kubectl --namespace oudns 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/oud-ds-rs-0               1/1     Running     0          14m     10.244.1.180   <Worker Node>   <none>
<none>
pod/oud-ds-rs-1               1/1     Running     0          8m26s   10.244.1.181   <Worker Node>   <none>
<none>
pod/oud-ds-rs-2               0/1     Running     0          2m24s   10.244.1.182   <Worker Node>   <none>
<none>
pod/oud-pod-cron-job-27586680-p5d8q   0/1     Completed   0          50s     10.244.1.183   <Worker Node>
<none>         <none>

NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE   SELECTOR
service/oud-ds-rs           ClusterIP   None            <none>        1444/TCP,1888/TCP,1389/TCP,1636/TCP,1080/
TCP,1081/TCP,1898/TCP   14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/name=oud-ds-rs
service/oud-ds-rs-0         ClusterIP   None            <none>        1444/TCP,1888/
TCP,1898/TCP                    14m app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/name=oud-ds-
rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-0
service/oud-ds-rs-1         ClusterIP   None            <none>        1444/TCP,1888/
TCP,1898/TCP                    14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/name=oud-ds-
rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-1
service/oud-ds-rs-2         ClusterIP   None            <none>        1444/TCP,1888/
TCP,1898/TCP                    14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/name=oud-ds-
rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-2
service/oud-ds-rs-http-0    ClusterIP   10.104.112.93   <none>        1080/
TCP,1081/TCP                    14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-0
service/oud-ds-rs-http-1    ClusterIP   10.103.105.70   <none>        1080/
TCP,1081/TCP                    14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-1
service/oud-ds-rs-http-2    ClusterIP   10.110.160.107  <none>        1080/
TCP,1081/TCP                    14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-2
service/oud-ds-rs-lbr-admin   ClusterIP   10.99.238.222   <none>        1888/
TCP,1444/TCP                    14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs
service/oud-ds-rs-lbr-http    ClusterIP   10.101.250.196  <none>        1080/
TCP,1081/TCP                    14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
```

```
name=oud-ds-rs
service/oud-ds-rs-lbr-ldap    ClusterIP   10.104.149.90    <none>       1389/
TCP,1636/TCP                                14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs
service/oud-ds-rs-ldap-0      ClusterIP   10.109.255.221   <none>       1389/
TCP,1636/TCP                                14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-0
service/oud-ds-rs-ldap-1      ClusterIP   10.111.135.142   <none>       1389/
TCP,1636/TCP                                14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-1
service/oud-ds-rs-ldap-2      ClusterIP   10.100.8.145     <none>       1389/
TCP,1636/TCP                                14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-2

NAME                           TYPE                       DATA   AGE
secret/dockercred              kubernetes.io/dockerconfigjson   1      4h24m
secret/orclcred                kubernetes.io/dockerconfigjson   1      14m
secret/oud-ds-rs-creds         opaque                     8      14m
secret/oud-ds-rs-tls-cert      kubernetes.io/tls          2      14m
secret/sh.helm.release.v1.oud-ds-rs.v1   helm.sh/release.v1         1      14m


NAME                          CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM                 STORAGECLASS       REASON   AGE    VOLUMEMODE
persistentvolume/oud-ds-rs-pv       20Gi       RWX            Delete           Bound    oudns/oud-ds-rs-pvc
manual                14m    Filesystem

NAME                          STATUS   VOLUME       CAPACITY   ACCESS MODES   STORAGECLASS   AGE
VOLUMEMODE
persistentvolumeclaim/oud-ds-rs-pvc   Bound    oud-ds-rs-pv   20Gi       RWX            manual         14m    Filesystem

NAME                                        CLASS    HOSTS                                        ADDRESS
PORTS     AGE
ingress.networking.k8s.io/oud-ds-rs-admin-ingress-nginx   <none>   oud-ds-rs-admin-0,oud-ds-rs-admin-0,oud-ds-rs-
admin-1 + 3 more...           80, 443   14m
ingress.networking.k8s.io/oud-ds-rs-http-ingress-nginx    <none>   oud-ds-rs-http-0,oud-ds-rs-http-1,oud-ds-rs-http-2
+ 3 more...              80, 443   14m
```

If you are using block storage you will see slightly different entries for PV and PVC, for example:

```
NAME                                  CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM                    STORAGECLASS                REASON   AGE    VOLUMEMODE
persistentvolume/ocid1.volume.oc1.iad.<unique_ID>   50Gi       RWO            Delete           Bound       oudns/oud-
ds-rs-pv-oud-ds-rs-2   oud-sc                      60m    Filesystem
persistentvolume/ocid1.volume.oc1.iad.<unique_ID>   50Gi       RWO            Delete           Bound       oudns/oud-
ds-rs-pv-oud-ds-rs-1   oud-sc                      67m    Filesystem
persistentvolume/ocid1.volume.oc1.iad.<unique_ID>   50Gi       RWO            Delete           Bound       oudns/oud-
ds-rs-pv-oud-ds-rs-3   oud-sc                      45m    Filesystem

NAME                                        STATUS   VOLUME                                  CAPACITY   ACCESS MODES
STORAGECLASS   AGE    VOLUMEMODE
persistentvolumeclaim/oud-ds-rs-pv-oud-ds-rs-1   Bound    ocid1.volume.oc1.iad.<unique_ID>   50Gi
RWO            oud-sc         67m    Filesystem
persistentvolumeclaim/oud-ds-rs-pv-oud-ds-rs-2   Bound    ocid1.volume.oc1.iad.<unique_ID>   50Gi
```

```
RWO            oud-sc       60m   Filesystem
persistentvolumeclaim/oud-ds-rs-pv-oud-ds-rs-3   Bound    ocid1.volume.oc1.iad.<unique_ID>   50Gi
RWO            oud-sc       45m   Filesystem
```

> **Note**
>
> Initially pod/oud-ds-rs-0 will appear with a STATUS of 0/1 and it will take approximately 5 minutes before OUD is started (1/1). Once pod/oud-ds-rs-0 has a STATUS of 1/1, pod/oud-ds-rs-1 will appear with a STATUS of 0/1. Once pod/oud-ds-rs-1 is started (1/1), pod/oud-ds-rs-2 will appear. It will take around 15 minutes for all the pods to fully started.
> While the oud-ds-rs pods have a STATUS of 0/1 the pod is running but OUD 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 oudns
> ```
>
> For example:
>
> ```
> kubectl logs oud-ds-rs-0 -n oudns
> ```

If the OUD deployment fails, additionally refer to General Troubleshooting for instructions on how describe the failing pod(s). Once the problem is identified follow Deleting an OUD 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 'oud-ds-rs' as deployment/release name for the Helm chart installation.

Type	Name	Example Name	Purpose
Service Account	<deployment/release name>	oud-ds-rs	Kubernetes Service Account for the Helm Chart deployment.
Secret	<deployment/release name>-creds	oud-ds-rs-creds	Secret object for Oracle Unified Directory related critical values like passwords.
Persistent Volume	<deployment/release name>-pv	oud-ds-rs-pv	Persistent Volume for user_projects mount.
Persistent Volume Claim	<deployment/release name>-pvc	oud-ds-rs-pvc	Persistent Volume Claim for user_projects mount.

Type	Name	Example Name	Purpose
Persistent Volume	<deployment/release name>-pv-config	oud-ds-rs-pv-config	Persistent Volume for mounting volume in containers for configuration files like ldif, schema, jks, java.security, etc.
Persistent Volume Claim	<deployment/release name>-pvc-config	oud-ds-rs-pvc-config	Persistent Volume Claim for mounting volume in containers for configuration files like ldif, schema, jks, java.security, etc.
Pod	<deployment/release name>-0	oud-ds-rs-0	Pod/Container for base Oracle Unified Directory Instance which would be populated first with base configuration (like number of sample entries).
Pod	<deployment/release name>-N	oud-ds-rs-1, oud-ds-rs-2, …	Pod(s)/Container(s) for Oracle Unified Directory Instances - each would have replication enabled against base Oracle Unified Directory instance <deployment/release name>-0.
Service	<deployment/release name>-0	oud-ds-rs-0	Service for LDAPS Admin, REST Admin and Replication interfaces from base Oracle Unified Directory instance <deployment/release name>-0.
Service	<deployment/release name>-http-0	oud-ds-rs-http-0	Service for HTTP and HTTPS interfaces from base Oracle Unified Directory instance <deployment/release name>-0.
Service	<deployment/release name>-ldap-0	oud-ds-rs-ldap-0	Service for LDAP and LDAPS interfaces from base Oracle Unified Directory instance <deployment/release name>-0.
Service	<deployment/release name>-N	oud-ds-rs-1, oud-ds-rs-2, …	Service(s) for LDAPS Admin, REST Admin and Replication interfaces from base Oracle Unified Directory instance <deployment/release name>-N.

Type	Name	Example Name	Purpose
Service	<deployment/release name>-http-N	oud-ds-rs-http-1, oud-ds-rs-http-2, …	Service(s) for HTTP and HTTPS interfaces from base Oracle Unified Directory instance <deployment/release name>-N.
Service	<deployment/release name>-ldap-N	oud-ds-rs-ldap-1, oud-ds-rs-ldap-2, …	Service(s) for LDAP and LDAPS interfaces from base Oracle Unified Directory instance <deployment/release name>-N.
Service	<deployment/release name>-lbr-admin	oud-ds-rs-lbr-admin	Service for LDAPS Admin, REST Admin and Replication interfaces from all Oracle Unified Directory instances.
Service	<deployment/release name>-lbr-http	oud-ds-rs-lbr-http	Service for HTTP and HTTPS interfaces from all Oracle Unified Directory instances.
Service	<deployment/release name>-lbr-ldap	oud-ds-rs-lbr-ldap	Service for LDAP and LDAPS interfaces from all Oracle Unified Directory instances.
Ingress	<deployment/release name>-admin-ingress-nginx	oud-ds-rs-admin-ingress-nginx	Ingress Rules for HTTP Admin interfaces.
Ingress	<deployment/release name>-http-ingress-nginx	oud-ds-rs-http-ingress-nginx	Ingress Rules for HTTP (Data/REST) interfaces.

7.4.6 Verifying the OUD Replication

Once all the pods created are visible as READY (i.e. 1/1), you can verify your replication across multiple Oracle Unified Directory (OUD) instances.

To verify the replication group, connect to the container and issue an OUD administration command to show the details. The name of the container can be found by issuing the following:

```
kubectl get pods -n <namespace> -o jsonpath='{.items[*].spec.containers[*].name}'
```

For example:

```
kubectl get pods -n oudns -o jsonpath='{.items[*].spec.containers[*].name}'
```

The output will look similar to the following:

```
oud-ds-rs oud-ds-rs oud-ds-rs
```

Once you have the container name you can verify the replication status in the following ways:

- Run dresplication inside the pod
- Using kubectl commands

Run dresplication Inside the Pod

1. Run the following command to create a bash shell in the pod:

   ```
   kubectl --namespace <namespace> exec -it -c <containername> <podname> -- bash
   ```

   For example:

   ```
   kubectl --namespace oudns exec -it -c oud-ds-rs oud-ds-rs-0 -- bash
   ```

   This will take you into the pod:

   ```
   [oracle@oud-ds-rs-0 oracle]$
   ```

2. From the prompt, use the dsreplication command to check the status of your replication group:

   ```
   cd /u01/oracle/user_projects/oud-ds-rs-0/OUD/bin
   ./dsreplication status --trustAll \
   --hostname oud-ds-rs-0 --port 1444 --adminUID admin \
   --dataToDisplay compat-view --dataToDisplay rs-connections
   ```

   The output will look similar to the following. Enter credentials where prompted:

   ```
   >>>> Specify Oracle Unified Directory LDAP connection parameters

   Password for user 'admin':

   Establishing connections and reading configuration ..... Done.

   dc=example,dc=com - Replication Enabled
   ========================================

   Server              : Entries : M.C. [1] : A.O.M.C. [2] : Port [3] : Encryption [4] : Trust [5] : U.C. [6] : Status [7] :
   ChangeLog [8] : Group ID [9] : Connected To [10]
   --------------------:---------:----------:-------------:----------:----------------:-----------:----------:------------:---------------
   :-------------:----------------------------
   oud-ds-rs-0:1444    : 202     : 0       : 0          : 1898     : Disabled      : Trusted   : --      : Normal     :
   Enabled      : 1          : oud-ds-rs-0:1898
                  :       :         :           :        :            :         :         :          :            : (GID=1)
   oud-ds-rs-1:1444    : 202     : 0       : 0          : 1898     : Disabled      : Trusted   : --      : Normal     :
   Enabled      : 1          : oud-ds-rs-1:1898
                  :       :         :           :        :            :         :         :          :            : (GID=1)
   oud-ds-rs-2:1444    : 202     : 0       : 0          : 1898     : Disabled      : Trusted   : --      : Normal     :
   Enabled      : 1          : oud-ds-rs-2:1898
                  :       :         :           :        :            :         :         :          :            : (GID=1)

   Replication Server [11]       : RS #1 : RS #2 : RS #3
   ------------------------------:-------:-------:------
   oud-ds-rs-0:1898              : --    : Yes   : Yes
   (#1)                          :       :       :
   ```

```
oud-ds-rs-1:1898            : Yes  : --   : Yes
(#2)                  :     :     :
oud-ds-rs-2:1898            : Yes  : Yes  : --
(#3)                  :     :     :
```

[1] The number of changes that are still missing on this element (and that have been applied to at least one other server).
[2] Age of oldest missing change: the age (in seconds) of the oldest change that has not yet arrived on this element.
[3] The replication port used to communicate between the servers whose contents are being replicated.
[4] Whether the replication communication initiated by this element is encrypted or not.
[5] Whether the directory server is trusted or not. Updates coming from an untrusted server are discarded and not propagated.
[6] The number of untrusted changes. These are changes generated on this server while it is untrusted. Those changes are not propagated to the rest of the topology but are effective on the untrusted server.
[7] The status of the replication on this element.
[8] Whether the external change log is enabled for the base DN on this server or not.
[9] The ID of the replication group to which the server belongs.
[10] The replication server this server is connected to with its group ID between brackets.
[11] This table represents the connections between the replication servers. The headers of the columns use a number as identifier for each replication server. See the values of the first column to identify the corresponding replication server for each number.

3. Type `exit` to exit the pod.

Using kubectl commands

To verify the replication using kubectl commands:

1. The `dsreplication status` command can be invoked using the following kubectl command:

```
kubectl --namespace <namespace> exec -it -c <containername> <podname> -- \
/u01/oracle/user_projects/<OUD Instance/Pod Name>/OUD/bin/dsreplication status \
--trustAll --hostname <OUD Instance/Pod Name> --port 1444 --adminUID admin \
--dataToDisplay compat-view --dataToDisplay rs-connections
```

For example:

```
kubectl --namespace oudns exec -it -c oud-ds-rs oud-ds-rs-0 -- \
/u01/oracle/user_projects/oud-ds-rs-0/OUD/bin/dsreplication status \
--trustAll --hostname oud-ds-rs-0 --port 1444 --adminUID admin \
--dataToDisplay compat-view --dataToDisplay rs-connections
```

The output will be the same as in **Run dresplication Inside the Pod** above.

7.4.7 Verifying OUD Assured Replication Status

> **Note**
>
> This section only needs to be followed if you enabled assured replication as per Enabling Assured Replication (Optional).

To verify the OUD assured replication status:

1. Run the following command to create a bash shell in the pod:

   ```
   kubectl --namespace <namespace> exec -it -c <containername> <podname> -- bash
   ```

   For example:

   ```
   kubectl --namespace oudns exec -it -c oud-ds-rs oud-ds-rs-0 -- bash
   ```

   This will take you into the pod:

   ```
   [oracle@oud-ds-rs-0 oracle]$
   ```

2. At the prompt, enter the following commands:

   ```
   echo $bindPassword1 > /tmp/pwd.txt


   /u01/oracle/user_projects/${OUD_INSTANCE_NAME}/OUD/bin/dsconfig --no-prompt \
   --hostname ${OUD_INSTANCE_NAME} --port ${adminConnectorPort} --bindDN "${rootUserDN}" \
   --bindPasswordFile /tmp/pwd.txt  --trustAll get-replication-domain-prop --domain-name ${baseDN} \
   --advanced --property assured-type --property assured-sd-level --property assured-timeout \
   --provider-name "Multimaster Synchronization"
   ```

   The output will look similar to the following:

   ```
   Property        : Value(s)
   ----------------:----------
   assured-sd-level : 2
   assured-timeout  : 5 s
   assured-type     : safe-data
   ```

7.4.8 Verifying the Cronjob

1. Run the following command to make sure the cronjob is created:

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

   For example:

   ```
   kubectl get cronjob -n oudns
   ```

   The output will look similar to the following:

   ```
   NAME              SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
   oud-pod-cron-job   */30 * * * *   False     0       5m18s           19m
   ```

2. Run the following command to make sure the job(s) is created:

   ```
   kubectl get job -n <namespace> -o wide
   ```

For example:

```
kubectl get job -n oudns -o wide
```

The output will look similar to the following:

```
NAME                    COMPLETIONS   DURATION   AGE     CONTAINERS      IMAGES                          SELECTOR
oud-pod-cron-job-27586680   1/1           1s         5m36s   cron-kubectl    bitnami/kubectl:1.30.3   controller-uid=700ab9f7-6094-488a-854d-f1b914de5f61
```

Disabling the Cronjob

If you need to disable the job, for example if maintenance needs to be performed on the node, you can disable the job as follows:

1. Run the following command to edit the cronjob:

   ```
   kubectl edit cronjob pod-cron-job -n <namespace>
   ```

   For example:

   ```
   kubectl edit cronjob oud-pod-cron-job -n oudns
   ```

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

2. In the edit session search for suspend and change the value from `false` to `true`:

   ```
   ...
   schedule: '*/30 * * * *'
   successfulJobsHistoryLimit: 3
   suspend: true
   ...
   ```

3. Save the file and exit (`wq!`).
4. Run the following to make sure the cronjob is suspended:

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

   For example:

   ```
   kubectl get cronjob -n oudns
   ```

   The output will look similar to the following:

   ```
   NAME SCHEDULE SUSPEND ACTIVE LAST SCHEDULE AGE
   oud-pod-cron-job */30 * * * * True 0 7m47s 21m
   ```

5. To enable the cronjob again, repeat the above steps and set suspend to false.

8
Configuring Ingress

You must configure an ingress controller to allow access to Oracle Unified Directory (OUD).

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

By default the ingress configuration only supports HTTP and HTTPS ports. To allow LDAP and LDAPS communication over TCP, configuration is required at the ingress controller/ implementation level.

This chapter includes the following topics:

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 `oud-ds-rs` installed with value `oud-ds-rs` as a deployment/ release name in the namespace `oudns`. 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.

```
# Configuration for additional TCP ports to be exposed through Ingress
# Format for each port would be like:
# <PortNumber>: <Namespace>/<Service>
tcp:
  # Map 1389 TCP port to LBR LDAP service to get requests handled through any available POD/Endpoint
serving LDAP Port
  1389: oudns/oud-ds-rs-lbr-ldap:ldap
  # Map 1636 TCP port to LBR LDAP service to get requests handled through any available POD/Endpoint
serving LDAPS Port
  1636: oudns/oud-ds-rs-lbr-ldap:ldaps
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: oudns/oud-ds-rs-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 defied/configured, Node Port would be assigend 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
      tcp:
        # For LDAP Interface
        1389: 31389
        # For LDAPS Interface
        1636: 31636
```

> **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
```

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
```

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/v1beta1
 kind: Ingress
 metadata:
   annotations:
     kubernetes.io/ingress.class: nginx
   name: example
   namespace: foo
 spec:
   rules:
     - host: www.example.com
       http:
         paths:
           - backend:
               serviceName: exampleService
               servicePort: 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
```

Optional: Command Helm Upgrade to Update nginx-ingress

If required, an nginx-ingress deployment can be updated/upgraded with following command.

In this example, the ingress-nginx configuration is updated with an additional TCP port and Node Port for accessing the LDAP/LDAPS port of a specific pod:

1. Create a nginx-ingress-values-override.yaml that contains the following:

```
# Configuration for additional TCP ports to be exposed through Ingress
# Format for each port would be like:
# <PortNumber>: <Namespace>/<Service>
tcp:
  # Map 1389 TCP port to LBR LDAP service to get requests handled through any available POD/Endpoint serving LDAP Port
  1389: oudns/oud-ds-rs-lbr-ldap:ldap
  # Map 1636 TCP port to LBR LDAP service to get requests handled through any available POD/Endpoint serving LDAPS Port
  1636: oudns/oud-ds-rs-lbr-ldap:ldaps
  # Map specific ports for LDAP and LDAPS communication from individual Services/Pods
  # To redirect requests on 3890 port to oudns/oud-ds-rs-ldap-0:ldap
  3890: oudns/oud-ds-rs-ldap-0:ldap
  # To redirect requests on 6360 port to oudns/oud-ds-rs-ldaps-0:ldap
  6360: oudns/oud-ds-rs-ldap-0:ldaps
  # To redirect requests on 3891 port to oudns/oud-ds-rs-ldap-1:ldap
  3891: oudns/oud-ds-rs-ldap-1:ldap
  # To redirect requests on 6361 port to oudns/oud-ds-rs-ldaps-1:ldap
  6361: oudns/oud-ds-rs-ldap-1:ldaps
  # To redirect requests on 3892 port to oudns/oud-ds-rs-ldap-2:ldap
  3892: oudns/oud-ds-rs-ldap-2:ldap
  # To redirect requests on 6362 port to oudns/oud-ds-rs-ldaps-2:ldap
  6362: oudns/oud-ds-rs-ldap-2:ldaps
  # Map 1444 TCP port to LBR Admin service to get requests handled through any available POD/Endpoint serving Admin LDAPS Port
  1444: oudns/oud-ds-rs-lbr-admin:adminldaps
  # To redirect requests on 4440 port to oudns/oud-ds-rs-0:adminldaps
  4440: oudns/oud-ds-rs-0:adminldaps
  # To redirect requests on 4441 port to oudns/oud-ds-rs-1:adminldaps
  4441: oudns/oud-ds-rs-1:adminldaps
  # To redirect requests on 4442 port to oudns/oud-ds-rs-2:adminldaps
  4442: oudns/oud-ds-rs-2:adminldaps
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: oudns/oud-ds-rs-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 defied/configured, Node Port would be assigend 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
tcp:
 # For LDAP Interface referring to LBR LDAP services serving LDAP port
 1389: 31389
 # For LDAPS Interface referring to LBR LDAP services serving LDAPS port
 1636: 31636
 # For LDAP Interface from specific service oud-ds-rs-ldap-0
 3890: 30890
 # For LDAPS Interface from specific service oud-ds-rs-ldap-0
 6360: 30360
 # For LDAP Interface from specific service oud-ds-rs-ldap-1
 3891: 30891
 # For LDAPS Interface from specific service oud-ds-rs-ldap-1
 6361: 30361
 # For LDAP Interface from specific service oud-ds-rs-ldap-2
 3892: 30892
 # For LDAPS Interface from specific service oud-ds-rs-ldap-2
 6362: 30362
 # For LDAPS Interface referring to LBR Admin services serving adminldaps port
 1444: 31444
 # For Admin LDAPS Interface from specific service oud-ds-rs-0
 4440: 30440
 # For Admin LDAPS Interface from specific service oud-ds-rs-1
 4441: 30441
 # For Admin LDAPS Interface from specific service oud-ds-rs-2
 4442: 30442
```

2. Run the following command to upgrade the ingress:

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

Where:

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

For example:

```
helm upgrade --namespace mynginxns \
--values nginx-ingress-values-override.yaml \
lbr-nginx stable/ingress-nginx
```

8.4 Accessing OUD Interfaces 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 (OUD) HTTP/HTTPS interfaces:

Port	NodePort	Host	Example Hostname	Path	Backend Service:Port	Example Service Name:Port
http/https	30080/30443	<deployment/release name>-admin-0	oud-ds-rs-admin-0	*	<deployment/release name>-0:adminhttps	oud-ds-rs-0:adminhttps
http/https	30080/30443	<deployment/release name>-admin-N	oud-ds-rs-admin-N	*	<deployment/release name>-N:adminhttps	oud-ds-rs-1:adminhttps
http/https	30080/30443	<deployment/release name>-admin	oud-ds-rs-admin	*	<deployment/release name>-lbr-admin:adminhttps	oud-ds-rs-lbr-admin:adminhttps
http/https	30080/30443	*	*	/rest/v1/admin	<deployment/release name>-lbr-admin:adminhttps	oud-ds-rs-lbr-admin:adminhttps
http/https	30080/30443	<deployment/release name>-http-0	oud-ds-rs-http-0	*	<deployment/release name>-http-0:http	oud-ds-rs-http-0:http
http/https	30080/30443	<deployment/release name>-http-N	oud-ds-rs-http-N	*	<deployment/release name>-http-N:http	oud-ds-rs-http-N:http
http/https	30080/30443	<deployment/release name>-http	oud-ds-rs-http	*	<deployment/release name>-lbr-http:http	oud-ds-rs-lbr-http:http
http/https	*	*	*	/rest/v1/directory	<deployment/release name>-lbr-http:http	oud-ds-rs-lbr-http:http
http/https	*	*	*	/iam/directory	<deployment/release name>-lbr-http:http	oud-ds-rs-lbr-http:http

> **Note**
>
> In the table above, example values are based on the value 'oud-ds-rs' 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 External LoadBalancer is not available/configured, interfaces can be accessed through NodePort on a Kubernetes node.

The following table details the rules configured in ingress object(s) for access to Oracle Unified Directory (OUD) LDAP/LDAPS interfaces. This is based on using the updated/upgraded configuration referenced in Installing the NGINX Controller:

Port	NodePort	Backend Service:Port	Example Service Name:Port
1389	31389	<deployment/release name>-lbr-ldap:ldap	oud-ds-rs-lbr-ldap:ldap
1636	31636	<deployment/release name>-lbr-ldap:ldap	oud-ds-rs-lbr-ldap:ldaps
1444	31444	<deployment/release name>-lbr-admin:adminldaps	oud-ds-rs-lbr-admin:adminldaps
3890	30890	<deployment/release name>-ldap-0:ldap	oud-ds-rs-ldap-0:ldap
6360	30360	<deployment/release name>-ldap-0:ldaps	oud-ds-rs-ldap-0:ldaps
3891	30891	<deployment/release name>-ldap-1:ldap	oud-ds-rs-ldap-1:ldap
6361	30361	<deployment/release name>-ldap-1:ldaps	oud-ds-rs-ldap-1:ldaps
3892	30892	<deployment/release name>-ldap-2:ldap	oud-ds-rs-ldap-2:ldap
6362	30362	<deployment/release name>-ldap-2:ldaps	oud-ds-rs-ldap-2:ldaps
4440	30440	<deployment/release name>-0:adminldaps	oud-ds-rs-ldap-0:adminldaps
4441	30441	<deployment/release name>-1:adminldaps	oud-ds-rs-ldap-1:adminldaps
4442	30442	<deployment/release name>-2:adminldaps	oud-ds-rs-ldap-2:adminldaps

- In the table above, example values are based on value ‘oud-ds-rs’ as the deployment/ release name for helm chart installation.
- The NodePorts mentioned in the table are according to Ingress configuration described in Installing the NGINX Controller.
- When 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 OUD hostname interfaces, then the following entries can be added in /etc/hosts file on the host from where OUD interfaces will be accessed.

```
<IP Address of External LBR or Kubernetes Node>  oud-ds-rs-http oud-ds-rs-http-0 oud-ds-rs-http-1 oud-ds-rs-http-2 oud-ds-rs-http-N
<IP Address of External LBR or Kubernetes Node>  oud-ds-rs-admin oud-ds-rs-admin-0 oud-ds-rs-admin-1 oud-ds-rs-admin-2 oud-ds-rs-admin-N
```

- In the table above, hostnames are based on the value ‘oud-ds-rs’ 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 Using LDAP Utilities

To use Oracle LDAP utilities such as ldapbind, ldapsearch, ldapmodify, you can either:

- Run the LDAP commands from an Oracle Unified Directory (OUD) installation outside the Kubernetes cluster. This requires access to an On-Premises OUD installation outside the Kubernetes cluster.
- Run the LDAP commands from inside the OUD Kubernetes pod connecting to the internal port:

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

  For example:

  ```
  kubectl exec -ti oud-ds-rs-0 -n oudns -- bash
  ```

  This will take you into a bash session in the pod:

  ```
  [oracle@oud-ds-rs-0 oracle]$
  ```

  Inside the container navigate to /u01/oracle/oud/bin to view the LDAP utilities:

  ```
  cd /u01/oracle/oud/bin
  ```

  ```
  ls ldap* ldapcompare ldapdelete ldapmodify ldappasswordmodify ldapsearch
  ```

> **Note**
>
> For commands that require an ldif file, copy the file into the <persistent_volume>/oud_user_projects directory:
>
> ```
> cp file.ldif <peristent_volume>/oud_user_projects
> ```
>
> For example:
>
> ```
> cp file.ldif /nfs_volumes/oudpv/oud_user_projects
> ```
>
> The file can then be viewed inside the pod:
>
> ```
> [oracle@oud-ds-rs-0 bin]$ cd /u01/oracle/user_projects
> [oracle@oud-ds-rs-0 user_projects]$ ls *.ldif
> file.ldif
> ```

8.6 Validating Access Using LDAP

Use the following steps to validate you can access Oracle Unified Directory (OUD) via LDAP:

> **Note**
>
> The examples assume sample data was installed when creating the OUD instance.

LDAP Against an External Load Balancer

If your ingress is configured with type: LoadBalancer then you cannot connect to the external LoadBalancer hostname and ports from inside the pod and must connect from an OUD installation outside the cluster.

- Example 1: Performing ldapsearch against external load balancer and LDAP port:

```
$OUD_HOME/bin/ldapsearch --hostname <External LBR> --port 1389 \
-D "<Root User DN>" -w <Password for Root User DN> \
-b "" -s base "(objectClass=*)" "*"
```

The output will look similar to the following:

```
dn:
objectClass: top
objectClass: ds-root-dse
lastChangeNumber: 0
firstChangeNumber: 0
changelog: cn=changelog
entryDN:
pwdPolicySubentry: cn=Default Password Policy,cn=Password Policies,cn=config
subschemaSubentry: cn=schema
supportedAuthPasswordSchemes: SHA256
supportedAuthPasswordSchemes: SHA1
supportedAuthPasswordSchemes: SHA384
supportedAuthPasswordSchemes: SHA512
supportedAuthPasswordSchemes: MD5
numSubordinates: 1
supportedFeatures: 1.3.6.1.1.14
supportedFeatures: 1.3.6.1.4.1.4203.1.5.1
supportedFeatures: 1.3.6.1.4.1.4203.1.5.2
supportedFeatures: 1.3.6.1.4.1.4203.1.5.3
lastExternalChangelogCookie:
vendorName: Oracle Corporation
vendorVersion: Oracle Unified Directory 12.2.1.4.0
componentVersion: 4
releaseVersion: 1
platformVersion: 0
supportedLDAPVersion: 2
supportedLDAPVersion: 3
supportedControl: 1.2.826.0.1.3344810.2.3
supportedControl: 1.2.840.113556.1.4.1413
supportedControl: 1.2.840.113556.1.4.319
```

```
supportedControl: 1.2.840.113556.1.4.473
supportedControl: 1.2.840.113556.1.4.805
supportedControl: 1.3.6.1.1.12
supportedControl: 1.3.6.1.1.13.1
supportedControl: 1.3.6.1.1.13.2
supportedControl: 1.3.6.1.4.1.26027.1.5.2
supportedControl: 1.3.6.1.4.1.26027.1.5.4
supportedControl: 1.3.6.1.4.1.26027.1.5.5
supportedControl: 1.3.6.1.4.1.26027.1.5.6
supportedControl: 1.3.6.1.4.1.26027.2.3.1
supportedControl: 1.3.6.1.4.1.26027.2.3.2
supportedControl: 1.3.6.1.4.1.26027.2.3.4
supportedControl: 1.3.6.1.4.1.42.2.27.8.5.1
supportedControl: 1.3.6.1.4.1.42.2.27.9.5.2
supportedControl: 1.3.6.1.4.1.42.2.27.9.5.8
supportedControl: 1.3.6.1.4.1.4203.1.10.1
supportedControl: 1.3.6.1.4.1.4203.1.10.2
supportedControl: 2.16.840.1.113730.3.4.12
supportedControl: 2.16.840.1.113730.3.4.16
supportedControl: 2.16.840.1.113730.3.4.17
supportedControl: 2.16.840.1.113730.3.4.18
supportedControl: 2.16.840.1.113730.3.4.19
supportedControl: 2.16.840.1.113730.3.4.2
supportedControl: 2.16.840.1.113730.3.4.3
supportedControl: 2.16.840.1.113730.3.4.4
supportedControl: 2.16.840.1.113730.3.4.5
supportedControl: 2.16.840.1.113730.3.4.9
supportedControl: 2.16.840.1.113894.1.8.21
supportedControl: 2.16.840.1.113894.1.8.31
supportedControl: 2.16.840.1.113894.1.8.36
maintenanceVersion: 2
supportedSASLMechanisms: PLAIN
supportedSASLMechanisms: EXTERNAL
supportedSASLMechanisms: CRAM-MD5
supportedSASLMechanisms: DIGEST-MD5
majorVersion: 12
orclGUID: D41D8CD98F003204A9800998ECF8427E
entryUUID: d41d8cd9-8f00-3204-a980-0998ecf8427e
ds-private-naming-contexts: cn=schema
hasSubordinates: true
nsUniqueId: d41d8cd9-8f003204-a9800998-ecf8427e
structuralObjectClass: ds-root-dse
supportedExtension: 1.3.6.1.4.1.4203.1.11.1
supportedExtension: 1.3.6.1.4.1.4203.1.11.3
supportedExtension: 1.3.6.1.1.8
supportedExtension: 1.3.6.1.4.1.26027.1.6.3
supportedExtension: 1.3.6.1.4.1.26027.1.6.2
supportedExtension: 1.3.6.1.4.1.26027.1.6.1
supportedExtension: 1.3.6.1.4.1.1466.20037
namingContexts: cn=changelog
namingContexts: dc=example,dc=com
```

- Example 2 - Performing ldapsearch against external load balancer and LDAP port for specific Oracle Unified Directory Interface:

```
$OUD_HOME/bin/ldapsearch --hostname <External LBR> --port 3890 \
-D "<Root User DN>" -w <Password for Root User DN> \
-b "" -s base "(objectClass=*)" "*"
```

LDAPS Against Kubernetes NodePort for Ingress Controller Service

In the example below LDAP utilities are executed from inside the oud-ds-rs-0 pod. If your ingress is configured with type: LoadBalancer you can connect to the Kubernetes hostname where the ingress is deployed using the NodePorts.

The following command performs an ldapsearch against the Kubernetes NodePort and LDAP port:

```
[oracle@oud-ds-rs-0 bin]$ ./ldapsearch --hostname <Kubernetes Node> --port 31636 \
--useSSL --trustAll \
-D "<Root User DN>" -w <Password for Root User DN> \
-b "" -s base "(objectClass=*)" "*"
```

8.7 Validating Access Using HTTPS

Use the following steps to validate you can access Oracle Unified Directory (OUD) via HTTPS:

> **Note**
>
> The examples assume sample data was installed when creating the OUD instance.

HTTPS/REST API against External LBR Host:Port

> **Note**
>
> In all the examples below:
>
> - You need to have an external IP assigned at ingress level.
> - "| json_pp" is used to format output in readable json format on the client side. It can be ignored if you do not have the json_pp library.
> - BASE64 of userDN:userPassword can be generated using:
>
>   ```
>   echo -n "userDN:userPassword" | base64
>   ```

- Example 1: Invoking the Data REST API:

```
curl --noproxy "*" -k  --location \
--request GET 'https://<External LBR Host>/rest/v1/directory/uid=user.1,ou=People,dc=example,dc=com?
scope=sub&attributes=*' \
--header 'Authorization: Basic <Base64 of userDN:userPassword>' | json_pp
```

The output will look similar to the following:


```
{
  "msgType" : "urn:ietf:params:rest:schemas:oracle:oud:1.0:SearchResponse",
  "totalResults" : 1,
  "searchResultEntries" : [
    {
      "dn" : "uid=user.1,ou=People,dc=example,dc=com",
      "attributes" : {
        "st" : "OH",
        "employeeNumber" : "1",
        "postalCode" : "93694",
        "description" : "This is the description for Aaren Atp.",
        "telephoneNumber" : "+1 390 103 6917",
        "homePhone" : "+1 280 375 4325",
        "initials" : "ALA",
        "objectClass" : [
          "top",
          "inetorgperson",
          "organizationalperson",
          "person"
        ],
        "uid" : "user.1",
        "sn" : "Atp",
        "street" : "70110 Fourth Street",
        "mobile" : "+1 680 734 6300",
        "givenName" : "Aaren",
        "mail" : "user.1@maildomain.net",
        "l" : "New Haven",
        "postalAddress" : "Aaren Atp$70110 Fourth Street$New Haven, OH  93694",
        "pager" : "+1 850 883 8888",
        "cn" : "Aaren Atp"
      }
    }
  ]
}
```


- Example 2 - Invoking the Data REST API against a specific Oracle Unified Directory interface:

```
curl --noproxy "*" -k  --location \
--request GET 'https://oud-ds-rs-http-0/rest/v1/directory/uid=user.1,ou=People,dc=example,dc=com?
scope=sub&attributes=*' \
--header 'Authorization: Basic <Base64 of userDN:userPassword>' | json_pp
```

  - In the above example it is assumed that the value ‘oud-ds-rs’ is used as the deployment/ release name for helm chart installation. It is assumed that ‘oud-ds-rs-http-0’ points to an External LoadBalancer

HTTPS/REST API Against Kubernetes NodePort for Ingress Controller Service

> **Note**
>
> In all the examples below:
>
> - "| json_pp" is used to format output in readable json format on the client side. It can be ignored if you do not have the json_pp library.
> - BASE64 of userDN:userPassword can be generated using:
>
>   echo -n "userDN:userPassword" | base64
>
> - It is assumed that the value 'oud-ds-rs' is used as the deployment/release name for helm chart installation

- Example 1: Invoking Data SCIM API against a specific Kubernetes node:

```
curl --noproxy "*" -k --location \
--request GET 'https://<Kubernetes Node>:30443/iam/directory/oud/scim/v1/Users' \
--header 'Authorization: Basic <Base64 of userDN:userPassword>' | json_pp
```

The output will look similar to the following:

```
{
  "Resources" : [
    {
      "id" : "ad55a34a-763f-358f-93f9-da86f9ecd9e4",
      "userName" : [
        {
          "value" : "user.0"
        }
      ],
      "schemas" : [
        "urn:ietf:params:scim:schemas:core:2.0:User",
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:OUD:User",
        "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User"
      ],
      "meta" : {
        "location" : "http://<Kubernetes Node>:30443/iam/directory/oud/scim/v1/Users/
ad55a34a-763f-358f-93f9-da86f9ecd9e4",
        "resourceType" : "User"
      },
      "addresses" : [
        {
          "postalCode" : "50369",
          "formatted" : "Aaccf Amar$01251 Chestnut Street$Panama City, DE  50369",
          "streetAddress" : "01251 Chestnut Street",
          "locality" : "Panama City",
          "region" : "DE"
        }
      ],
      "urn:ietf:params:scim:schemas:extension:oracle:2.0:OUD:User" : {
```

```
  "description" : [
    {
      "value" : "This is the description for Aaccf Amar."
    }
  ],
  "mobile" : [
    {
      "value" : "+1 010 154 3228"
    }
  ],
  "pager" : [
    {
      "value" : "+1 779 041 6341"
    }
  ],
  "objectClass" : [
    {
      "value" : "top"
    },
    {
      "value" : "organizationalperson"
    },
    {
      "value" : "person"
    },
    {
      "value" : "inetorgperson"
    }
  ],
  "initials" : [
    {
      "value" : "ASA"
    }
  ],
  "homePhone" : [
    {
      "value" : "+1 225 216 5900"
    }
  ]
},
"name" : [
  {
    "givenName" : "Aaccf",
    "familyName" : "Amar",
    "formatted" : "Aaccf Amar"
  }
],
"emails" : [
  {
    "value" : "user.0@maildomain.net"
  }
],
"phoneNumbers" : [
  {
    "value" : "+1 685 622 6202"
  }
```

```
    ],
    "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User" : {
      "employeeNumber" : [
        {
          "value" : "0"
        }
      ]
    }
  }
  ,
.
.
.
}
```

- Example 2 - Invoking the Data SCIM API against a specific Oracle Unified Directory Interface:

```
curl --noproxy "*" -k --location \
--request GET 'https://oud-ds-rs-http-0:30443/iam/directory/oud/scim/v1/Users' \
--header 'Authorization: Basic <Base64 of userDN:userPassword>' | json_pp
```

HTTPS/REST Admin API

> **Note**
>
> In all the examples below:
>
> - "| json_pp" is used to format output in readable json format on the client side. It can be ignored if you do not have the json_pp library.
> - BASE64 of userDN:userPassword can be generated using:
>
>   ```
>   echo -n "userDN:userPassword" | base64
>   ```

- Example 1: Invoking the Admin REST API against External LBR:

```
curl --noproxy "*" -k --insecure --location \
--request GET 'https://<External LBR Host>/rest/v1/admin/?
scope=base&attributes=vendorName&attributes=vendorVersion&attributes=ds-private-naming-
contexts&attributes=subschemaSubentry' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic <Base64 of userDN:userPassword>' | json_pp
```

The output will look similar to the following:

```
{
  "totalResults" : 1,
  "searchResultEntries" : [
    {
      "dn" : "",
      "attributes" : {
        "vendorVersion" : "Oracle Unified Directory 12.2.1.4.0",
```

```
        "ds-private-naming-contexts" : [
          "cn=admin data",
          "cn=ads-truststore",
          "cn=backups",
          "cn=config",
          "cn=monitor",
          "cn=schema",
          "cn=tasks",
          "cn=virtual acis",
          "dc=replicationchanges"
        ],
        "subschemaSubentry" : "cn=schema",
        "vendorName" : "Oracle Corporation"
      }
    }
  ],
  "msgType" : "urn:ietf:params:rest:schemas:oracle:oud:1.0:SearchResponse"
}
```


- Example 2 - Invoking the Admin REST API against specific Oracle Unified Directory Admin Interface:

```
 curl --noproxy "*" -k --insecure --location \
--request GET 'https://oud-ds-rs-admin-0/rest/v1/admin/?
scope=base&attributes=vendorName&attributes=vendorVersion&attributes=ds-private-naming-
contexts&attributes=subschemaSubentry' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic <Base64 of userDN:userPassword>' | json_pp
```

- Example 3 - Invoking the Admin REST API against Kubernetes NodePort for Ingress Controller Service:

```
curl --noproxy "*" -k --insecure --location \
--request GET 'https://oud-ds-rs-admin-0:30443/rest/v1/admin/?
scope=base&attributes=vendorName&attributes=vendorVersion&attributes=ds-private-naming-
contexts&attributes=subschemaSubentry' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic <Base64 of userDN:userPassword>' | json_pp
```

Part III

Administering Oracle Unified Directory on Kubernetes

Administer Oracle Unified Directory (OUD) on Kubernetes.

This section contains the following chapters:

9
Scaling OUD Instances

Learn the basic operations to scale Oracle Unified Directory (OUD) instances in Kubernetes.

> **Note**
>
> The instructions below are for scaling servers up or down manually. If you wish to use autoscaling, see Kubernetes Horizontal Pod Autoscaler. Please note, if you have enabled autoscaling, and then decide to run the commands manually, it is recommended to delete the autoscaler before running the commands in the topics below.

This chapter includes the following topics:

9.1 Viewing Existing OUD Instances

By default the `oud-ds-rs` helm chart deployment starts three pods: `oud-ds-rs-0` and two replica pods, `oud-ds-rs-1` and `oud-ds-rs-2`.

The number of pods started is determined by the `replicaCount`, which is set to `3` by default. A value of `3` starts the three pods above.

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

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

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

For example:

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

The output will look similar to the following:

```
NAME          READY   STATUS    RESTARTS   AGE    IP          NODE          NOMINATED NODE
READINESS GATES
pod/oud-ds-rs-0   1/1     Running   0          22h    10.244.0.195   <Worker Node>   <none>           <none>
pod/oud-ds-rs-1   1/1     Running   0          22h    10.244.0.194   <Worker Node>   <none>           <none>
pod/oud-ds-rs-2   1/1     Running   0          22h    10.244.0.193   <Worker Node>   <none>           <none>
```

9.2 Scaling Up OUD Instances

In the examples below, replicaCount is increased to 4 which creates a new OUD pod oud-ds-rs-3, with associated services created.

You can scale up the number of OUD 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/helm directory:

   ```
   cd $WORKDIR/kubernetes/helm
   ```

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

   ```
   replicaCount: 4
   ```

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

   ```
   helm upgrade --namespace <namespace> \
   --values oud-scaleup-override.yaml \
   <release_name> oud-ds-rs --reuse-values
   ```

   For example:

   ```
   helm upgrade --namespace oudns \
   --values oud-scaleup-override.yaml \
   oud-ds-rs oud-ds-rs --reuse-values
   ```

9.2.2 Scaling Up Using --set Argument

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

   ```
   helm upgrade --namespace <namespace> \
   --set replicaCount=4 \
   <release_name> oud-ds-rs --reuse-values
   ```

   For example:

   ```
   helm upgrade --namespace oudns \
   --set replicaCount=4 \
   oud-ds-rs oud-ds-rs --reuse-values
   ```

9.2.3 Verifying Scaling Up

1. Verify the new OUD pod oud-ds-rs-3 and has started:

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

   For example:

   kubectl get pods,service -n oudns

   The output will look similar to the following:

```
NAME           READY   STATUS    RESTARTS   AGE   IP          NODE          NOMINATED NODE
READINESS GATES
pod/oud-ds-rs-0   1/1     Running   0          22h   10.244.0.195   <Worker Node>   <none>         <none>
pod/oud-ds-rs-1   1/1     Running   0          22h   10.244.0.194   <Worker Node>   <none>         <none>
pod/oud-ds-rs-2   1/1     Running   0          22h   10.244.0.193   <Worker Node>   <none>         <none>
pod/oud-ds-rs-3   1/1     Running   0          17m   10.244.0.193   <Worker Node>   <none>         <none>

NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                  AGE
SELECTOR
service/oud-ds-rs           ClusterIP   None            <none>        1444/TCP,1888/TCP,1389/TCP,1636/TCP,1080/
TCP,1081/TCP,1898/TCP   22h    app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/name=oud-ds-rs
service/oud-ds-rs-0         ClusterIP   None            <none>        1444/TCP,1888/
TCP,1898/TCP                    22h    app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-0
service/oud-ds-rs-1         ClusterIP   None            <none>        1444/TCP,1888/
TCP,1898/TCP                    22h    app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-1
service/oud-ds-rs-2         ClusterIP   None            <none>        1444/TCP,1888/
TCP,1898/TCP                    22h    app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-2
service/oud-ds-rs-3         ClusterIP   None            <none>        1444/TCP,1888/
TCP,1898/TCP                    9m9s   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-3
service/oud-ds-rs-http-0    ClusterIP   10.104.112.93   <none>        1080/
TCP,1081/TCP                    22h    app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-0
service/oud-ds-rs-http-1    ClusterIP   10.103.105.70   <none>        1080/
TCP,1081/TCP                    22h    app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-1
service/oud-ds-rs-http-2    ClusterIP   10.110.160.107  <none>        1080/
TCP,1081/TCP                    22h    app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-2
service/oud-ds-rs-http-3    ClusterIP   10.102.93.179   <none>        1080/
TCP,1081/TCP                    9m9s   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-3
service/oud-ds-rs-lbr-admin ClusterIP   10.99.238.222   <none>        1888/
TCP,1444/TCP                    22h    app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs
service/oud-ds-rs-lbr-http  ClusterIP   10.101.250.196  <none>        1080/
TCP,1081/TCP                    22h    app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs
```

```
service/oud-ds-rs-lbr-ldap    ClusterIP   10.104.149.90   <none>       1389/
TCP,1636/TCP                              22h    app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs
service/oud-ds-rs-ldap-0      ClusterIP   10.109.255.221  <none>       1389/
TCP,1636/TCP                              22h    app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-0
service/oud-ds-rs-ldap-1      ClusterIP   10.111.135.142  <none>       1389/
TCP,1636/TCP                              22h    app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-1
service/oud-ds-rs-ldap-2      ClusterIP   10.100.8.145    <none>       1389/
TCP,1636/TCP                              22h    app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-2
service/oud-ds-rs-ldap-3      ClusterIP   10.111.177.46   <none>       1389/
TCP,1636/TCP                              9m9s   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-3
```

Note

It will take several minutes before all the services listed above show. While the oud-ds-rs-3 pod has a STATUS of 0/1 the pod is started but the OUD 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 oud-ds-rs-3 -n oudns
```

9.3 Scaling Down OUD Instances

Scaling down OUD pods is performed in exactly the same way as in Scaling Up OUD Instances except the replicaCount is reduced to the required number of pods.

In the examples below, replicaCount is decreased to 3 from 4which terminates the oud-ds-rs-3 pod and associated services.

You can scale down the number of OUD 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/helm directory:

   ```
   cd $WORKDIR/kubernetes/helm
   ```

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

   ```
   replicaCount: 3
   ```

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

```
helm upgrade --namespace <namespace> \
--values oud-scaledown-override.yaml \
<release_name> oud-ds-rs --reuse-values
```

For example:

```
helm upgrade --namespace oudns \
--values oud-scaledown-override.yaml \
oud-ds-rs oud-ds-rs --reuse-values
```

9.3.2 Scaling Down Using --set Argument

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

```
helm upgrade --namespace <namespace> \
--set replicaCount=3 \
<release_name> oud-ds-rs --reuse-values
```

For example:

```
helm upgrade --namespace oudns \
--set replicaCount=3 \
oud-ds-rs oud-ds-rs --reuse-values
```

9.3.3 Verifying Scaling Down

1. Verify the new OUD pod oud-ds-rs-3 and has terminated:

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

For example:

```
kubectl get pods -n oudns
```

The output will look similar to the following:

```
NAME           READY   STATUS      RESTARTS   AGE
pod/oud-ds-rs-0   1/1     Running     0          22h
pod/oud-ds-rs-1   1/1     Running     0          22h
pod/oud-ds-rs-2   1/1     Running     0          22h
pod/oud-ds-rs-3   1/1     Terminating 0          21m
```

The oud-ds-rs-3 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/oud-ds-rs-0   1/1     Running     0          22h
```

```
pod/oud-ds-rs-1   1/1   Running   0   22h
pod/oud-ds-rs-2   1/1   Running   0   22h
```

10

Logging and Visualization

This chapter describes how to install and configure logging and visualization for the oud-ds-rs 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:

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:

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 oudns --from-literal password=*<ELK_APIKEY>*

      The output will look similar to the following:

      secret/elasticsearch-pw-elastic created

   b. If ELK uses a password for authentication:

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

      For example:

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

The output will look similar to the following:

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

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

2. Check that the dockercred secret that was created previously in Create a Kubernetes Secret for Cronjob Images exists:

```
kubectl get secret -n <domain_namespace> | grep dockercred
```

For example:

```
kubectl get secret -n oudns | grep dockercred
```

The output will look similar to the following:

```
dockercred                kubernetes.io/dockerconfigjson      1      149m
```

If the secret does not exist, create it as per Create a Kubernetes Secret for Cronjob Images.

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: oudlogs-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: oudlogs-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 OUD Deployment for ELK

1. Run the following command to upgrade the OUD deployment with the ELK configuration:

   ```
   helm upgrade --namespace <namespace> --values <valuesfile.yaml> <releasename> oud-ds-rs --reuse-values
   ```

   For example:

   ```
   helm upgrade --namespace oudns --values logging-override-values.yaml oud-ds-rs oud-ds-rs --reuse-values
   ```

   The output should look similar to the following:

   ```
   Release "oud-ds-rs" has been upgraded. Happy Helming!
   NAME: oud-ds-rs
   LAST DEPLOYED: <DATE>
   NAMESPACE: oudns
   STATUS: deployed
   REVISION: 2
   NOTES:
   #
   # Copyright (c) 2020, 2022, Oracle and/or its affiliates.
   #
   # Licensed under the Universal Permissive License v 1.0 as shown at
   # https://oss.oracle.com/licenses/upl
   #
   #
   Since "nginx" has been chosen, follow the steps below to configure nginx ingress controller.
   Add Repo reference to helm for retriving/installing Chart for nginx-ingress implementation.
   command-# helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

   Command helm install to install nginx-ingress related objects like pod, service, deployment, etc.
   # helm install --namespace <namespace for ingress> --values nginx-ingress-values-override.yaml lbr-nginx
   ```

ingress-nginx/ingress-nginx

For details of content of nginx-ingress-values-override.yaml refer README.md file of this chart.

Run these commands to check port mapping and services:
kubectl --namespace <namespace for ingress> get services -o wide -w lbr-nginx-ingress-controller
kubectl describe --namespace <namespace for oud-ds-rs chart> ingress.extensions/oud-ds-rs-http-ingress-nginx
kubectl describe --namespace <namespace for oud-ds-rs chart> ingress.extensions/oud-ds-rs-admin-ingress-nginx

Accessible interfaces through ingress:
(External IP Address for LoadBalancer NGINX Controller can be determined through details associated with lbr-nginx-ingress-controller)

1. OUD Admin REST:
   Port: http/https

2. OUD Data REST:
   Port: http/https

3. OUD Data SCIM:
   Port: http/https

4. OUD LDAP/LDAPS:
   Port: ldap/ldaps

5. OUD Admin LDAPS:
   Port: ldaps

Please refer to README.md from Helm Chart to find more details about accessing interfaces and configuration parameters.

Accessible interfaces through ingress:

1. OUD Admin REST:
   Port: http/https

2. OUD Data REST:
   Port: http/https

3. OUD Data SCIM:
   Port: http/https

Please refer to README.md from Helm Chart to find more details about accessing interfaces and configuration parameters.

10.3 Verifying the Pods

Verifying the pods for ELK depends on whether you are using NFS or file system storage, or block storage for your persistent volume.

NFS or File System Storage

If you are using Oracle Unified Directory (OUD) with NFS or file system storage, a new logstash pod will be created.

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

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

   For example:

   ```
   kubectl get pods -n oudns
   ```

   The output should look similar to the following:

   ```
   NAME                                 READY   STATUS      RESTARTS   AGE
   oud-ds-rs-0                          1/1     Running     0          150m
   oud-ds-rs-1                          1/1     Running     0          143m
   oud-ds-rs-2                          1/1     Running     0          137m
   oud-ds-rs-logstash-5dc8d94597-knk8g  1/1     Running     0          2m12s
   oud-pod-cron-job-27758370-wpfq7      0/1     Completed   0          66m
   oud-pod-cron-job-27758400-kd6pn      0/1     Completed   0          36m
   oud-pod-cron-job-27758430-ndmgj      0/1     Completed   0          6m33s
   ```

   > **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 oud-ds-rs-logstash-<pod> -n oudns
   > ```

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

Block Storage

If you are using OUD with block devices, the logstash pod will run as a separate sidecar container in the OUD pods.

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

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

   For example:

   ```
   kubectl get pods -n oudns
   ```

   The output should look similar to the following:

   ```
   NAME                                 READY   STATUS      RESTARTS   AGE
   oud-ds-rs-0                          2/2     Running     0          24m
   ```

```
oud-ds-rs-1                 2/2   Running   0     17m
oud-ds-rs-2                 2/2   Running   0     10m
etc..
```

Notice the pods oud-ds-rs-0 - oud-ds-rs-2 have a READY status of 2/2.

> **Note**
>
> The pods will terminate one at a time and restart to add the sidecar logstash container. For example oud-ds-rs-2 will terminate and restart. oud-ds-rs1 will not terminate until oud-ds-rs-2 is at READY 2/2. If any of the pods fail, you can view the pod logs using:
>
> ```
> kubectl logs -f oud-ds-rs-<pod> -c oud-ds-rs-logstash -n oudns
> ```

If the pods have problems starting, 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 oud-ds-rs -n oudns
```

The output will look similar to the following:

```
REVISION    UPDATED     STATUS      CHART         APP VERSION   DESCRIPTION
1           <DATE>      superseded  oud-ds-rs-0.2 14.1.2.1.0    Install complete
2           <DATE>      deployed    oud-ds-rs-0.2 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 oud-ds-rs 1 -n oudns
```

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 oudlogs* 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 OUD 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: oudlogs*
   - 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 oudlogs* to view the log file entries.

11

Monitoring an Oracle Unified Directory Instance

You can monitor and Oracle Unified Directory (OUD) instance using Prometheus and Grafana.

This chapter includes the following topics:

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

Kubernetes Horizontal Pod Autoscaler

Kubernetes Horizontal Pod Autoscaler (HPA) allows automatic scaling (up and down) of the OUD servers. If load increases then extra OUD servers will be started as required. Similarly, if load decreases, OUD servers will be automatically shutdown.

For more information on HPA, see Horizontal Pod Autoscaling.

The instructions below show you how to configure and run an HPA to scale OUD servers, based on CPU utilization or memory resource metrics.

> **Note**
>
> If you enable HPA and then decide you want to start,stop, or scale OUD servers manually as per Scaling OUD Instances, it is recommended to delete HPA beforehand as per Deleting the Horizontal Pod Autoscaler.

This chapter includes the following topics:

12.1 Prerequisite Configurations

In order to use HPA, Oracle Unified Directory (OUD) must have been created with the required `resources` parameter as per Creating Oracle Unified Directory Instances.

For example:

```
oudConfig:
 # memory, cpu parameters for both requests and limits for oud instances
  resources:
    limits:
      cpu: "1"
      memory: "8Gi"
    requests:
      cpu: "500m"
      memory: "4Gi"
```

You can check the values by running the following command:

```
helm get values oud-ds-rs -n oudns
```

The output will look similar to the following:

```
...oudConfig:
 resources:
   limits:
     cpu: "1"
     memory: 4Gi
   requests:
     cpu: 500m
     memory: 4Gi
 ...
```

12.2 Deploying the Kubernetes Metrics Server

Before deploying Horizontal Pod Autoscaler (HPA) you must deploy the Kubernetes Metrics Server.

1. Check to see if the Kubernetes Metrics Server is already deployed:

   ```
   kubectl get pods -n kube-system | grep metric
   ```

   If a row is returned as follows, then Kubernetes Metric Server is deployed and you can move to Deploying the Horizontal Pod Autoscaler:

   ```
   metrics-server-d9694457-mf69d        1/1     Running   0          5m13s
   ```

2. If no rows are returned by the previous command, then the Kubernetes Metric Server needs to be deployed. Run the following commands to get the components.yaml:

   ```
   mkdir $WORKDIR/kubernetes/hpa
   ```

   ```
   cd $WORKDIR/kubernetes/hpa
   ```

   ```
   wget https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
   ```

3. Deploy the Kubernetes Metrics Server by running the following command:

   ```
   kubectl apply -f components.yaml
   ```

   The output will look similar to the following:

   ```
   serviceaccount/metrics-server created
   clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created
   clusterrole.rbac.authorization.k8s.io/system:metrics-server created
   rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created
   clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator created
   clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created
   service/metrics-server created
   ```

```
deployment.apps/metrics-server created
apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io created
```

4. Run the following command to check Kubernetes Metric Server is running:

   ```
   kubectl get pods -n kube-system | grep metric
   ```

   Make sure the pod has a READY status of 1/1:

   ```
   metrics-server-d9694457-mf69d        1/1     Running   0          39s
   ```

12.3 Troubleshooting the Metrics Server

If the Kubernetes Metric Server does not reach the READY 1/1 state, run the following commands:

```
kubectl describe pod <metrics-server-pod> -n kube-system
```

```
kubectl logs <metrics-server-pod> -n kube-system
```

If you see errors such as:

```
Readiness probe failed: HTTP probe failed with statuscode: 500
```

and:

```
E0907 13:07:50.937308       1 scraper.go:140] "Failed to scrape node" err="Get \"https://X.X.X.X:10250/metrics/resource\": x509: cannot validate certificate for 100.105.18.113 because it doesn't contain any IP SANs" node="worker-node1"
```

then you may need to install a valid cluster certificate for your Kubernetes cluster.

For testing purposes, you can resolve this issue by:

1. Delete the Kubernetes Metrics Server by running the following command:

   ```
   kubectl delete -f $WORKDIR/kubernetes/hpa/components.yaml
   ```

2. Edit the $WORKDIR/hpa/components.yaml and locate the args: section. Add kubelet-insecure-tls to the arguments. For example:

   ```
   spec:
     containers:
     - args:
       - --cert-dir=/tmp
       - --secure-port=4443
       - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
       - --kubelet-use-node-status-port
       - --kubelet-insecure-tls
       - --metric-resolution=15s
   ```

```
    image: registry.k8s.io/metrics-server/metrics-server:v0.6.4
...
```

3. Deploy the Kubernetes Metrics Server using the command:

   ```
   kubectl apply -f components.yaml
   ```

4. Run the following and make sure the READY status shows 1/1:

   ```
   kubectl get pods -n kube-system | grep metric
   ```

   The output should look similar to the following:

   ```
   metrics-server-d9694457-mf69d          1/1     Running   0          40s
   ```

12.4 Deploying the Horizontal Pod Autoscaler

The steps below show how to configure and run the Horizontal Pod Autoscaler (HPA) to scale Oracle Unified Directory (OUD), based on the CPU or memory utilization resource metrics.

Assuming the example OUD configuration in Creating Oracle Unified Directory Instances, three OUD servers are started by default (oud-ds-rs-0, oud-ds-rs-1, oud-ds-rs-2).

In the following example an HPA resource is created, targeted at the statefulset oud-ds-rs. This resource will autoscale OUD servers from a minimum of 3 OUD servers up to 5 OUD servers. Scaling up will occur when the average CPU is consistently over 70%. Scaling down will occur when the average CPU is consistently below 70%.

1. Find the statefulset used by OUD:

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

   For example:

   ```
   kubectl get statefulset -n oudns
   ```

   The output will look similar to the following:

   ```
   NAME       READY   AGE
   oud-ds-rs   3/3     23h
   ```

2. Navigate to the $WORKDIR/kubernetes/hpa and create an autoscalehpa.yaml file that contains the following:

   ```
   #
   apiVersion: autoscaling/v2
   kind: HorizontalPodAutoscaler
   metadata:
     name: oud-sts-hpa
     namespace: oudns
   spec:
     scaleTargetRef:
       apiVersion: apps/v1
   ```

```
  kind: StatefulSet
  name: oud-ds-rs #statefulset name of oud
 behavior:
  scaleDown:
    stabilizationWindowSeconds: 60
  scaleUp:
    stabilizationWindowSeconds: 60
 minReplicas: 3
 maxReplicas: 5
 metrics:
 - type: Resource
  resource:
    name: cpu
    target:
      type: Utilization
      averageUtilization: 70
```

where:

- oud-ds-rs is the stateful set returned earlier
- minReplicas should match the number of OUD servers started by default.
- maxReplicas should be set to the maximum amount of OUD servers that can be started by HPA.

> **Note**
>
> For setting HPA based on Memory Metrics, update the metrics block with the following content. Please note, Oracle recommends using only CPU or Memory, not both:
>
> ```
> metrics:
> - type: Resource
>  resource:
>    name: memory
>    target:
>      type: Utilization
>      averageUtilization: 70
> ```

3. Run the following command to create the autoscaler:

   ```
   kubectl apply -f autoscalehpa.yaml
   ```

   The output will look similar to the following:

   ```
   horizontalpodautoscaler.autoscaling/oud-sts-hpa created
   ```

4. Verify the status of the autoscaler by running the following:

   ```
   kubectl get hpa -n oudns
   ```

The output will look similar to the following:

```
NAME        REFERENCE             TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
oud-sts-hpa   StatefulSet/oud-ds-rs   5%/70%   3         5         3          33s
```

12.5 Verifying the Horizontal Pod Autoscaler

To verify the Horizontal Pod Autoscaler (HPA) works, perform the following steps:

1. Check the current status of the Oracle Unified Directory (OUD) servers:

   ```
   kubectl get pods -n oudns
   ```

   The output will look similar to the following:

   ```
   NAME                                READY   STATUS      RESTARTS   AGE
   oud-ds-rs-0                         1/1     Running     0          5h15m
   oud-ds-rs-1                         1/1     Running     0          5h9m
   oud-ds-rs-2                         1/1     Running     0          5h2m
   oud-pod-cron-job-28242120-bwtcz     0/1     Completed   0          61m
   oud-pod-cron-job-28242150-qf8fg     0/1     Completed   0          31m
   oud-pod-cron-job-28242180-q69lm     0/1     Completed   0          92s
   ```

   In the above example. `oud-ds-rs-0`, `oud-ds-rs-1`, and `oud-ds-rs-2` are running.

2. To test HPA can scale up the OUD servers, run the following commands:

   ```
   kubectl exec --stdin --tty oud-ds-rs-0 -n oudns -- /bin/bash
   ```

   This will take you inside a bash shell inside the `oud-ds-rs-0` pod:

   ```
   [oracle@oud-ds-rs-0 oracle]$
   ```

3. Inside the bash shell, run the following command to increase the load on the CPU:

   ```
   [oracle@oud-ds-rs-0 oracle]$ dd if=/dev/zero of=/dev/null
   ```

   This command will continue to run in the foreground.

4. Repeat the step above for the `oud-ds-rs-1` pod:

   ```
   kubectl exec --stdin --tty oud-ds-rs-1 -n oudns -- /bin/bash
   [oracle@oud-ds-rs-1 oracle]$
   [oracle@oud-ds-rs-1 oracle]$ dd if=/dev/zero of=/dev/null
   ```

5. In a command window outside the bash shells, run the following command to view the current CPU usage:

   ```
   kubectl get hpa -n oudns
   ```

The output will look similar to the following:

```
NAME        REFERENCE             TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
oud-sts-hpa   StatefulSet/oud-ds-rs   125%/70%   3         5         3          5m15s
```

In the above example the CPU has increased to 125%. As this is above the 70% limit, the autoscaler increases the replicas by starting additional OUD servers.

6. Run the following to see if any more OUD servers are started:

```
kubectl get pods -n oudns
```

The output will look similar to the following:

```
NAME                                READY   STATUS      RESTARTS   AGE
oud-ds-rs-0                         1/1     Running     0          5h50m
oud-ds-rs-1                         1/1     Running     0          5h44m
oud-ds-rs-2                         1/1     Running     0          5h37m
oud-ds-rs-3                         1/1     Running     0          9m29s
oud-ds-rs-4                         1/1     Running     0          5m17s
oud-pod-cron-job-28242150-qf8fg     0/1     Completed   0          66m
oud-pod-cron-job-28242180-q69lm     0/1     Completed   0          36m
oud-pod-cron-job-28242210-kn7sv     0/1     Completed   0          6m28s
```

In the example above one more OUD server has started (oud-ds-rs-4).

> **Note**
>
> It may take some time for the server to appear and start. Once the server is at READY status of 1/1, the server is started.

7. To stop the load on the CPU, in both bash shells, issue a Control C, and then exit the bash shell:

```
[oracle@oud-ds-rs-0 oracle]$ dd if=/dev/zero of=/dev/null
^C
[oracle@oud-ds-rs-0 oracle]$ exit
```

8. Run the following command to view the current CPU usage:

```
kubectl get hpa -n oudns
```

The output will look similar to the following:

```
NAME        REFERENCE             TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
oud-sts-hpa   StatefulSet/oud-ds-rs   4%/70%    3         5         5          40m
```

In the above example CPU has dropped to 4%. As this is below the 70% threshold, you should see the autoscaler scale down the servers:

```
kubectl get pods -n oudns
```

The output will look similar to the following:

```
NAME                                    READY   STATUS      RESTARTS   AGE
oud-ds-rs-0                             1/1     Running     0          5h54m
oud-ds-rs-1                             1/1     Running     0          5h48m
oud-ds-rs-2                             1/1     Running     0          5h41m
oud-ds-rs-3                             1/1     Running     0          13m
oud-ds-rs-4                             1/1     Terminating 0          8m27s
oud-pod-cron-job-28242150-qf8fg         0/1     Completed   0          70m
oud-pod-cron-job-28242180-q69lm         0/1     Completed   0          40m
oud-pod-cron-job-28242210-kn7sv         0/1     Completed   0          10m
```

Eventually, the extra OUD pod will disappear:

```
NAME                                    READY   STATUS      RESTARTS   AGE
oud-ds-rs-0                             1/1     Running     0          5h57m
oud-ds-rs-1                             1/1     Running     0          5h51m
oud-ds-rs-2                             1/1     Running     0          5h44m
oud-ds-rs-3                             1/1     Running     0          16m
oud-pod-cron-job-28242150-qf8fg         0/1     Completed   0          73m
oud-pod-cron-job-28242180-q69lm         0/1     Completed   0          43m
oud-pod-cron-job-28242210-kn7sv         0/1     Completed   0          13m
```

12.6 Deleting the Horizontal Pod Autoscaler

If you need to delete the Horizontal Pod Autoscaler (HPA), you can do so by running the following commands:

```
cd $WORKDIR/kubernetes/hpa
```

```
kubectl delete -f autoscalehpa.yaml
```

12.7 Other Considerations for Horizontal Pod Autoscaler

Administrators should be aware of the following considerations after deploying the Horizontal Pod Autoscaler (HPA):

- If HPA is deployed and you need to upgrade the Oracle Unified Directory (OUD) container image, then you must delete the HPA before upgrading. To delete the HPA, see Deleting the Horizontal Pod Autoscaler. Once the upgrade is successful you can deploy HPA again.
- If you choose to scale up or scale down an OUD server manually as per Scaling Up/Down OUD Pods, then it is recommended to delete the HPA before doing so.

13
Patching and Upgrading

This chapter includes the following topics:

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

13.1 Patching and Upgrading Within Oracle Unified Directory 14.1.2

The instructions in this section relate to patching or upgrading an existing 14.1.2.1.0 OUD Kubernetes deployment with a new OUD 14c container image.

This section contains the following topics:

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

13.1.1 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 (OUD) Kubernetes deployment with a new OUD 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.
> - If you have Kubernetes Horizontal Pod Autoscaler (HPA) enabled, you must disable HPA before performing the steps below. See, Deleting the Horizontal Pod Autoscaler.

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

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

2. Create an `oud-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/oud_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 OUD container image, then you can remove the following:
>
> ```
> imagePullSecrets:
>   - name: orclcred
> ```

If you have also upgraded your version of Kubernetes since the last container image update, you also need to add the following to the file:

```
cronJob:
  kubectlImage:
    repository: bitnami/kubectl
    tag: <version>
    pullPolicy: IfNotPresent
```

Where the <version> in kubectlImage: tag: should be set to the same version as your Kubernetes version (kubectl version). For example if your Kubernetes version is 1.30.3 set to 1.30.3.
If your cluster does not have access to the internet to pull the bitnami/kubectl image, you must load the images in a local container registry and set the repository tag appropriately.

3. Take a backup of the persistent volume directory:

```
sudo cp -rp <persistent_volume>/oud_user_projects <persistent_volume>/oud_user_projects_bkp14c_<tag>
```

For example:

```
sudo cp -rp /nfs_volumes/oudpv/oud_user_projects /nfs_volumes/oudpv/oud_user_projects_bkp14c_old
```

4. Run the following command to upgrade the deployment:

```
helm upgrade --namespace <namespace> \
--values oud-patch-override.yaml \
<release_name> oud-ds-rs --reuse-values
```

For example:

```
helm upgrade --namespace oudns \
--values oud-patch-override.yaml \
oud-ds-rs oud-ds-rs --reuse-values
```

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

5. Run the following command and make sure all the OUD 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 oudns -w
```

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

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

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

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

For example:

```
kubectl describe pod oud-ds-rs-0 -n oudns | grep image
```

The output will look similar to the following:

```
...
Containers:
  oud-ds-rs:
    Container ID:   cri-o://6a35ef3a0721015aa99b2aaeebdc96528c8166db7bf36176f0b9665e43c10ded
    Image:          container-registry.oracle.com/middleware/oud_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD>
    Image ID:       container-registry.oracle.com/middleware/
oud_cpu@sha256:2ae38d6bdca4c411d6b62289cf80563f611a1fdcbaf01632be7b4fa6a4169000
```

7. Verify the OUD deployment where applicable:
   - Verifying the OUD Deployment
   - Verifying OUD Assured Replication Status
   - Verifying the Cronjob
   - Accessing OUD Interfaces Through Ingress

13.1.2 Rolling Back the Upgrade Within 14.1.2

If the Oracle Unified Directory (OUD) upgrade fails, you can rollback to the previous OUD 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 (OUD) installation perform the following steps:

1. Rollback the OUD deployment using the following command:

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

   For example:

   ```
   helm rollback oud-ds-rs -n oudns
   ```

   The output will look similar to the following:

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

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

2. Run the following command and make sure all the OUD 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 oudns -w
   ```

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

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

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

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

   For example:

   ```
   kubectl describe pod oud-ds-rs-0 -n oudns | grep image
   ```

   The output will look similar to the following:

   ```
   ...
   Containers:
     oud-ds-rs:
       Container ID:   cri-o://6a35ef3a0721015aa99b2aaeebdc96528c8166db7bf36176f0b9665e43c10ded
       Image:          container-registry.oracle.com/middleware/oud_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD>
   ```

Image ID: container-registry.oracle.com/middleware/oud_cpu@sha256:2ae38d6bdca4c411d6b62289cf80563f611a1fdcbaf01632be7b4fa6a4169000

4. Verify the OUD deployment where applicable:
   - Verifying the OUD Deployment
   - Verifying OUD Assured Replication Status
   - Verifying the Cronjob
   - Accessing OUD Interfaces Through Ingress

13.2 Upgrading from Oracle Unified Directory 12.2.1.4 to 14.1.2

The instructions in this section are for upgrading an existing Oracle Unified Directory (OUD) 12.2.1.4 deployment on Kubernetes to OUD 14.1.2.1.0.

This section contains the following topics:

- Performing the Upgrade from 12c to 14c
- Rolling Back the 14c Upgrade to 12c

13.2.1 Performing the Upgrade from 12c to 14c

Run the following steps to upgrade the Oracle Unified Directory (OUD) 12.2.1.4 deployment to 14.1.2.1.0:

> **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 OUD 14c container image on all nodes in your Kubernetes cluster.
> - If you have Kubernetes Horizontal Pod Autoscaler (HPA) enabled, you must disable HPA before performing the steps below. See, Deleting the Horizontal Pod Autoscaler.
> - Your <persistent_volume> must have enough free storage to back up the contents of the persistent volume and to allow for the upgrade. It's advisable to have at least 1GB spare before performing the upgrade.

1. Download the OUD 14c deployment scripts as per Setting Up the Code Repository for OUD.
2. Take a backup of the persistent volume directory:

   ```
   sudo cp -rp <persistent_volume>/oud_user_projects <persistent_volume>/oud_user_projects_bkp12c
   ```

   For example:

   ```
   sudo cp -rp /nfs_volumes/oudpv/oud_user_projects /nfs_volumes/oudpv/oud_user_projects_bkp12c
   ```

3. Navigate to the $WORKDIR/kubernetes/helm14c directory:

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

4. Create an oud-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/oud_cpu
  tag: 14.1.2.1.0-jdk17-ol8-<YYMMDD>
imagePullSecrets:
  - name: orclcred
```

Note

Administrators should be aware of the following:

- If you are not using Oracle Container Registry, or your own password protected container registry for your OUD container image, then you can remove the following:

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

- If you have also upgraded your version of Kubernetes since the last container image update, you also need to add the following to the file:

```
cronJob:
  kubectlImage:
    repository: bitnami/kubectl
    tag: <version>
    pullPolicy: IfNotPresent
```

  Where the <version> in kubectlImage: tag: should be set to the same version as your Kubernetes version (kubectl version). For example if your Kubernetes version is 1.30.3 set to 1.30.3.
  If your cluster does not have access to the internet to pull the bitnami/kubectl image, you must load the images in a local container registry and set the repository tag appropriately.

- If you are upgrading an Enterprise Deployment then:
  - Add the following to the oud-patch-override.yaml:

```
baseOUD:
 envVars:
   - name: serverTuning
     value: -Xms1024m -Xmx2048m -server -Xmn1g -XX:MaxTenuringThreshold=1 -
XX:+UseG1GC
replOUD:
 envVars:
   - name: serverTuning
     value: -Xms1024m -Xmx2048m -server -Xmn1g -XX:MaxTenuringThreshold=1 -
XX:+UseG1GC
```

  - Edit the <persistent_volume>/edg-oud-ds-rs-0/OUD/config/java.properties file and remove all lines that start with start-ds.java-args from the file. Repeat for all edg-oud-ds-rs-N pods.

5. Run the following command to upgrade the deployment:

```
helm upgrade --namespace <namespace> \
--values oud-patch-override.yaml \
<release_name> oud-ds-rs --reuse-values
```

For example:

```
helm upgrade --namespace oudns \
--values oud-patch-override.yaml \
oud-ds-rs oud-ds-rs --reuse-values
```

The output will look similar to the following:

```
Release "oud-ds-rs" has been upgraded. Happy Helming!
NAME: oud-ds-rs
LAST DEPLOYED: <DATE>
NAMESPACE: oudns
STATUS: deployed
REVISION: 2
NOTES:
#
# Copyright (c) 2025, Oracle and/or its affiliates.
#
# Licensed under the Universal Permissive License v 1.0 as shown at
# https://oss.oracle.com/licenses/upl
#
#
Since "nginx" has been chosen, follow the steps below to configure nginx ingress controller.
Add Repo reference to helm for retriving/installing Chart for nginx-ingress implementation.
command-# helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

Command helm install to install nginx-ingress related objects like pod, service, deployment, etc.
# helm install --namespace <namespace for ingress> --values nginx-ingress-values-override.yaml lbr-nginx
ingress-nginx/ingress-nginx

For details of content of nginx-ingress-values-override.yaml refer README.md file of this chart.

Run these commands to check port mapping and services:
# kubectl --namespace <namespace for ingress> get services -o wide -w lbr-nginx-ingress-controller
# kubectl describe --namespace <namespace for oud-ds-rs chart> ingress.extensions/oud-ds-rs-http-ingress-nginx
# kubectl describe --namespace <namespace for oud-ds-rs chart> ingress.extensions/oud-ds-rs-admin-ingress-
nginx

Accessible interfaces through ingress:
(External IP Address for LoadBalancer NGINX Controller can be determined through details associated with
lbr-nginx-ingress-controller)

1. OUD Admin REST:
   Port: http/https

2. OUD Data REST:
   Port: http/https

3. OUD Data SCIM:
   Port: http/https

4. OUD LDAP/LDAPS:
   Port: ldap/ldaps

5. OUD Admin LDAPS:
```

```
  Port: ldaps

Please refer to README.md from Helm Chart to find more details about accessing interfaces and configuration parameters.


Accessible interfaces through ingress:

1. OUD Admin REST:
  Port: http/https

2. OUD Data REST:
  Port: http/https

3. OUD Data SCIM:
  Port: http/https

Please refer to README.md from Helm Chart to find more details about accessing interfaces and configuration parameters.
```

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

6. Run the following command and make sure all the OUD pods are started:

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

For example:

```
kubectl get pods -n oudns -w
```

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

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

For example:

```
kubectl describe pod oud-ds-rs-0 -n oudns
```

The output will look similar to the following:

```
...
Containers:
 oud-ds-rs:
   Container ID:   cri-o://6a35ef3a0721015aa99b2aaeebdc96528c8166db7bf36176f0b9665e43c10ded
   Image:          container-registry.oracle.com/middleware/oud_cpu:14.1.2.1.0-jdk17-ol8-<DDYYMM>
   Image ID:       container-registry.oracle.com/middleware/
oud_cpu@sha256:2ae38d6bdca4c411d6b62289cf80563f611a1fdcbaf01632be7b4fa6a4169000
```

8. Verify the OUD deployment where applicable:
   - Verifying the OUD Deployment
   - Verifying OUD Assured Replication Status

- Verifying the Cronjob
- Accessing OUD Interfaces Through Ingress

13.2.2 Rolling Back the 14c Upgrade to 12c

If the Oracle Unified Directory (OUD) 14c upgrade fails, you can rollback to OUD 12c, 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 (OUD) installation perform the following steps:

1. Run the following command to find the name of the StatefulSet:

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

   For example:

   ```
   kubectl get statefulsets -n oudns
   ```

   The output will look similar to the following:

   ```
   NAME       READY   AGE
   oud-ds-rs   3/3     54m
   ```

2. Run the following commands to scale the replicas of the StatefulSet to 0. This ensures all pods are terminated gracefully:

   ```
   kubectl scale statefulset <name> --replicas=0 -n <namespace>
   ```

   For example:

   ```
   kubectl scale statefulset oud-ds-rs --replicas=0 -n oudns
   ```

   The output will look similar to the following:

   ```
   statefulset.apps/oud-ds-rs scaled
   ```

3. Run the following command and make sure all the OUD pods are terminated before proceeding:

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

   For example:

   ```
   kubectl get pods -n oudns
   ```

4. Restore the backup of the OUD 12c persistent volume:

```
sudo cp -rp <persistent_volume>/oud_user_projects <persistent_volume>/oud_user_projects_bkp14c

sudo rm -rf <persistent_volume>/oud_user_projects

sudo cp -rp <persistent_volume>/oud_user_projects_bkp12c <persistent_volume>/oud_user_projects
```

For example:

```
sudo cp -rp /nfs_volumes/oudpv/oud_user_projects /nfs_volumes/oudpv/oud_user_projects_bkp14c

sudo rm -rf /nfs_volumes/oudpv/oud_user_projects

sudo cp -rp /nfs_volumes/oudpv/oud_user_projects_bkp12c /nfs_volumes/oudpv/oud_user_projects/
oud_user_projects
```

5. Rollback the OUD deployment using the following command:

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

For example:

```
helm rollback oud-ds-rs -n oudns
```

The output will look similar to the following:

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

6. Run the following command and make sure all the OUD pods are started:

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

For example:

```
kubectl get pods -n oudns -w
```

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

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

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

7. Run the following command to check the pods are now using the previous image:

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

   For example:

   ```
   kubectl describe pod oud-ds-rs-0 -n oudns | grep image
   ```

   The output will look similar to the following:

   ```
   ...
   Containers:
     oud-ds-rs:
       Container ID:   cri-o://6a35ef3a0721015aa99b2aaeebdc96528c8166db7bf36176f0b9665e43c10ded
       Image:          container-registry.oracle.com/middleware/oud_cpu:12.2.1.2.4-jdk8-ol8-<YYMMDD>
       Image ID:       container-registry.oracle.com/middleware/
   oud_cpu@sha256:2ae38d6bdca4c411d6b62289cf80563f611a1fdcbaf01632be7b4fa6a4169000
   ```

8. Verify the OUD deployment where applicable:
   - Verifying the OUD Deployment
   - Verifying OUD Assured Replication Status
   - Verifying the Cronjob
   - Accessing OUD Interfaces Through Ingress

14

General Troubleshooting

This chapter includes the following topics:

14.1 Checking the Status of an Oracle Unified Directory 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 oudns 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/oud-ds-rs-0               1/1     Running     0          14m     10.244.1.180   <Worker Node>   <none>
<none>
pod/oud-ds-rs-1               1/1     Running     0          8m26s   10.244.1.181   <Worker Node>   <none>
<none>
pod/oud-ds-rs-2               0/1     Running     0          2m24s   10.244.1.182   <Worker Node>   <none>
<none>
pod/oud-pod-cron-job-27586680-p5d8q   0/1     Completed   0          50s     10.244.1.183   <Worker Node>
<none>          <none>

NAME                  TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE   SELECTOR
service/oud-ds-rs           ClusterIP   None          <none>       1444/TCP,1888/TCP,1389/TCP,1636/TCP,1080/
TCP,1081/TCP,1898/TCP   14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/name=oud-ds-rs
service/oud-ds-rs-0         ClusterIP   None          <none>       1444/TCP,1888/
TCP,1898/TCP                14m app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/name=oud-ds-
rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-0
service/oud-ds-rs-1         ClusterIP   None          <none>       1444/TCP,1888/
TCP,1898/TCP                14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/name=oud-ds-
rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-1
service/oud-ds-rs-2         ClusterIP   None          <none>       1444/TCP,1888/
TCP,1898/TCP                14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/name=oud-ds-
```

```
rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-2
service/oud-ds-rs-http-0     ClusterIP   10.104.112.93   <none>       1080/
TCP,1081/TCP                              14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-0
service/oud-ds-rs-http-1     ClusterIP   10.103.105.70   <none>       1080/
TCP,1081/TCP                              14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-1
service/oud-ds-rs-http-2     ClusterIP   10.110.160.107  <none>       1080/
TCP,1081/TCP                              14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-2
service/oud-ds-rs-lbr-admin  ClusterIP   10.99.238.222   <none>       1888/
TCP,1444/TCP                              14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs
service/oud-ds-rs-lbr-http   ClusterIP   10.101.250.196  <none>       1080/
TCP,1081/TCP                              14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs
service/oud-ds-rs-lbr-ldap   ClusterIP   10.104.149.90   <none>       1389/
TCP,1636/TCP                              14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs
service/oud-ds-rs-ldap-0     ClusterIP   10.109.255.221  <none>       1389/
TCP,1636/TCP                              14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-0
service/oud-ds-rs-ldap-1     ClusterIP   10.111.135.142  <none>       1389/
TCP,1636/TCP                              14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-1
service/oud-ds-rs-ldap-2     ClusterIP   10.100.8.145    <none>       1389/
TCP,1636/TCP                              14m   app.kubernetes.io/instance=oud-ds-rs,app.kubernetes.io/
name=oud-ds-rs,statefulset.kubernetes.io/pod-name=oud-ds-rs-2

NAME                                TYPE                  DATA   AGE
secret/dockercred                   kubernetes.io/dockerconfigjson   1      4h24m
secret/orclcred                     kubernetes.io/dockerconfigjson   1      14m
secret/oud-ds-rs-creds              opaque                8      14m
secret/oud-ds-rs-tls-cert           kubernetes.io/tls     2      14m
secret/sh.helm.release.v1.oud-ds-rs.v1   helm.sh/release.v1    1      14m


NAME                          CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM                 STORAGECLASS       REASON   AGE    VOLUMEMODE
persistentvolume/oud-ds-rs-pv       20Gi       RWX            Delete           Bound    oudns/oud-ds-rs-pvc
manual                14m    Filesystem

NAME                          STATUS   VOLUME       CAPACITY   ACCESS MODES   STORAGECLASS   AGE
VOLUMEMODE
persistentvolumeclaim/oud-ds-rs-pvc   Bound    oud-ds-rs-pv   20Gi       RWX            manual         14m   Filesystem

NAME                                        CLASS    HOSTS                                              ADDRESS
PORTS     AGE
ingress.networking.k8s.io/oud-ds-rs-admin-ingress-nginx   <none>   oud-ds-rs-admin-0,oud-ds-rs-admin-0,oud-ds-rs-
admin-1 + 3 more...   80, 443   14m
ingress.networking.k8s.io/oud-ds-rs-http-ingress-nginx    <none>   oud-ds-rs-http-0,oud-ds-rs-http-1,oud-ds-rs-http-2
+ 3 more...   80, 443   14m
```

14.2 Viewing Pod Logs

To view logs for a pod use the following command:

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

For example:

```
kubectl logs oud-ds-rs-0 -n oudns
```

> **Note**
>
> If you add -f to the command, then the log will be streamed.

14.3 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 oud-ds-rs-0 -n oudns
```

The output will look similar to the following:

```
Name:         oud-ds-rs-0
Namespace:    oudns
Priority:     0
Node:         <Worker Node>/100.105.18.114
Start Time:   <DATE>
Labels:       app.kubernetes.io/instance=oud-ds-rs
              app.kubernetes.io/name=oud-ds-rs
              controller-revision-hash=oud-ds-rs-5c8b8f67c9
              statefulset.kubernetes.io/pod-name=oud-ds-rs-0
Annotations:  <none>
Status:       Running
IP:           10.244.2.48
IPs:
  IP:         10.244.2.48
Controlled By:  StatefulSet/oud-ds-rs
Init Containers:
  mount-pv:
    Container ID:  cri-o://905af11c6f032f2dfa18b1e3956d7936cb7dd04d9d0df0cfcf8ed061e6930b52
    Image:         <location>/busybox
    Image ID:      <location>@sha256:2c8ed5408179ff4f53242a4bdd2706110ce000be239fe37a61be9c52f704c437
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sh
      -c
    Args:
```

```
    ordinal=${OUD_INSTANCE_NAME##*-}; if [[ ${CLEANUP_BEFORE_START} == "true" ]]; then if
[[ "$ordinal" != "0" ]]; then cd /u01/oracle; rm -fr /u01/oracle/user_projects/$(OUD_INSTANCE_NAME)/OUD; fi; fi
    if [[ ${CONFIGVOLUME_ENABLED} == "true" ]]; then if [[ "$ordinal" == "0" ]]; then cp "/mnt/
baseOUD.props" "${CONFIGVOLUME_MOUNTPATH}/config-baseOUD.props"; else cp "/mnt/replOUD.props" "$
{CONFIGVOLUME_MOUNTPATH}/config-replOUD.props"; fi; fi;
  State:          Terminated
    Reason:       Completed
    Exit Code:    0
    Started:      <DATE>
    Finished:     <DATE>
  Ready:          True
  Restart Count:  0
  Environment:
    OUD_INSTANCE_NAME:       oud-ds-rs-0 (v1:metadata.name)
    CONFIGVOLUME_ENABLED:    false
    CONFIGVOLUME_MOUNTPATH:  /u01/oracle/config-input
    CLEANUP_BEFORE_START:    false
  Mounts:
    /u01/oracle/user_projects from oud-ds-rs-pv (rw)
    /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-65skp (ro)
Containers:
 oud-ds-rs:
  Container ID:   cri-o://d691b090dfbb1ee1b8606952497d80642424a82a2290071b325ea720098817c3
  Image:          container-registry.oracle.com/middleware/oud_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD>
  Image ID:       container-registry.oracle.com/middleware/
oud_cpu@sha256:faca16dbbcda1985ff567eefe3f2ca7bae6cbbb7ebcd296fffb040ce61e9396a
  Ports:          1444/TCP, 1888/TCP, 1389/TCP, 1636/TCP, 1080/TCP, 1081/TCP, 1898/TCP
  Host Ports:     0/TCP, 0/TCP, 0/TCP, 0/TCP, 0/TCP, 0/TCP, 0/TCP
  State:          Running
    Started:      <DATE>
  Ready:          True
  Restart Count:  0
  Limits:
    cpu:     1
    memory:  4Gi
  Requests:
    cpu:      500m
    memory:   4Gi
  Liveness:   tcp-socket :ldap delay=300s timeout=30s period=60s #success=1 #failure=5
  Readiness:  exec [/u01/oracle/container-scripts/checkOUDInstance.sh] delay=300s timeout=30s period=60s
#success=1 #failure=10
  Environment:
    instanceType:               DS2RS_STS
    OUD_INSTANCE_NAME:          oud-ds-rs-0 (v1:metadata.name)
    MY_NODE_NAME:               (v1:spec.nodeName)
    MY_POD_NAME:                oud-ds-rs-0 (v1:metadata.name)
    sleepBeforeConfig:          3
    sourceHost:                 oud-ds-rs-0
    baseDN:                     dc=example,dc=com
    rootUserDN:                 <set to the key 'rootUserDN' in secret 'oud-ds-rs-creds'>       Optional: false
    rootUserPassword:           <set to the key 'rootUserPassword' in secret 'oud-ds-rs-creds'>  Optional: false
    adminUID:                   <set to the key 'adminUID' in secret 'oud-ds-rs-creds'>         Optional: false
    adminPassword:              <set to the key 'adminPassword' in secret 'oud-ds-rs-creds'>    Optional: false
    bindDN1:                    <set to the key 'bindDN1' in secret 'oud-ds-rs-creds'>          Optional: false
    bindPassword1:              <set to the key 'bindPassword1' in secret 'oud-ds-rs-creds'>    Optional: false
    bindDN2:                    <set to the key 'bindDN2' in secret 'oud-ds-rs-creds'>          Optional: false
```

```
    bindPassword2:              <set to the key 'bindPassword2' in secret 'oud-ds-rs-creds'>  Optional: false
    sourceServerPorts:          oud-ds-rs-0:1444
    sourceAdminConnectorPort:   1444
    sourceReplicationPort:      1898
    sampleData:                 200
    adminConnectorPort:         1444
    httpAdminConnectorPort:     1888
    ldapPort:                   1389
    ldapsPort:                  1636
    httpPort:                   1080
    httpsPort:                  1081
    replicationPort:            1898
    dsreplication_1:            verify --hostname ${sourceHost} --port ${sourceAdminConnectorPort} --baseDN ${baseDN} --serverToRemove $(OUD_INSTANCE_NAME):${adminConnectorPort} --connectTimeout 600000 --readTimeout 600000
    dsreplication_2:            enable --host1 ${sourceHost} --port1 ${sourceAdminConnectorPort} --replicationPort1 ${sourceReplicationPort} --host2 $(OUD_INSTANCE_NAME) --port2 ${adminConnectorPort} --replicationPort2 ${replicationPort} --baseDN ${baseDN} --connectTimeout 600000 --readTimeout 600000
    dsreplication_3:            initialize --hostSource ${initializeFromHost} --portSource ${sourceAdminConnectorPort} --hostDestination $(OUD_INSTANCE_NAME) --portDestination ${adminConnectorPort} --baseDN ${baseDN} --connectTimeout 600000 --readTimeout 600000
    dsreplication_4:            verify --hostname $(OUD_INSTANCE_NAME) --port ${adminConnectorPort} --baseDN ${baseDN} --connectTimeout 600000 --readTimeout 600000
    post_dsreplication_dsconfig_1:  set-replication-domain-prop --domain-name ${baseDN} --set group-id:1
    post_dsreplication_dsconfig_2:  set-replication-server-prop --set group-id:1
  Mounts:
    /u01/oracle/user_projects from oud-ds-rs-pv (rw)
    /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-65skp (ro)
Conditions:
 Type              Status
 Initialized       True
 Ready             True
 ContainersReady   True
 PodScheduled      True
Volumes:
 oud-ds-rs-pv:
   Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
   ClaimName:  oud-ds-rs-pvc
   ReadOnly:   false
 kube-api-access-65skp:
   Type:                    Projected (a volume that contains injected data from multiple sources)
   TokenExpirationSeconds:  3607
   ConfigMapName:           kube-root-ca.crt
   ConfigMapOptional:       <nil>
   DownwardAPI:             true
QoS Class:                   Burstable
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:                      <none>
```

14.4 Known Issues

This section contains information about known issues.

dsreplication Output After Scale Up/Down Shows Pod in Unknown State

Sometimes when scaling up or down, it is possible to get incorrect data in the dsreplication output. In the example below the replicaCount was changed from 4 to 3. The oud-ds-rs-3 server appears as <Unknown> when it should have disappeared:

```
dc=example,dc=com - Replication Enabled
========================================

Server                 : Entries : M.C. [1] : A.O.M.C. [2] : Port [3] : Encryption [4] : Trust [5] : U.C. [6] : Status [7] :
ChangeLog [8] : Group ID [9] : Connected To [10]
-----------------------------:---------:----------:--------------:----------:----------------:-----------:----------:------------:-----------
----:--------------:-------------------------------
oud-ds-rs-3:<Unknown>        : --      : N/A      : --          : 1898     : Disabled      : --       : --       : Unknown    :
--           : N/A         : --
[11]                         :         :          :             :          :               :          :          :            :            :
oud-ds-rs-0:1444             : 39135   : 0        : 0           : 1898     : Disabled      : Trusted  : --       : Normal     :
Enabled      : 1             : oud-ds-rs-2:1898
                             :         :          :             :          :               :          :          :            :            : (GID=1)
oud-ds-rs-1:1444             : 39135   : 0        : 0           : 1898     : Disabled      : Trusted  : --       : Normal     :
Enabled      : 1             : oud-ds-rs-1:1898
                             :         :          :             :          :               :          :          :            :            : (GID=1)
oud-ds-rs-2:1444             : 39135   : 0        : 0           : 1898     : Disabled      : Trusted  : --       : Normal     :
Enabled      : 1             : oud-ds-rs-2:1898
                             :         :          :             :          :               :          :          :            :            : (GID=1)

Replication Server [12]      : RS #1 : RS #2 : RS #3 : RS #4
-----------------------------:-------:-------:-------:------
oud-ds-rs-0:1898 (#1)  : --    : Yes   : Yes   : N/A
oud-ds-rs-1:1898 (#2)  : Yes   : --    : Yes   : N/A
oud-ds-rs-2:1898 (#3)  : Yes   : Yes   : --    : N/A
oud-ds-rs-3:1898 (#4)  : No    : No    : No    : --
```

In this situation, perform the following steps to remove the server:

1. Run the following command to enter the OUD Kubernetes pod:

   ```
   kubectl --namespace <namespace> exec -it -c <containername> <podname> -- bash
   ```

   For example:

   ```
   kubectl --namespace oudns exec -it -c oud-ds-rs oud-ds-rs-0 -- bash
   ```

   This will take you into the pod:

   ```
   [oracle@oud-ds-rs-0 oracle]$
   ```

2. Once inside the pod run the following command to create a password file:

   ```
   echo <ADMIN_PASSWORD> > /tmp/adminpassword.txt
   ```

3. Run the following command to remove the replicationPort:

   ```
   /u01/oracle/oud/bin/dsreplication disable --hostname localhost --port $adminConnectorPort --adminUID admin
   --trustAll --adminPasswordFile /tmp/adminpassword.txt --no-prompt --unreachableServer oud-ds-
   rs-3:$replicationPort
   ```

   The output will look similar to the following:

   ```
   Establishing connections and reading configuration ..... Done.

   The following errors were encountered reading the configuration of the
   existing servers:
   Could not connect to the server oud-ds-rs-3:1444.  Check that the
   server is running and that is accessible from the local machine.  Details:
   oud-ds-rs-3:1444
   The tool will try to update the configuration in a best effort mode.

   Removing references to replication server oud-ds-rs-3:1898 ..... Done.
   ```

4. Run the following command to remove the adminConnectorPort:

   ```
   /u01/oracle/oud/bin/dsreplication disable --hostname localhost --port $adminConnectorPort --adminUID admin
   --trustAll --adminPasswordFile /tmp/adminpassword.txt --no-prompt --unreachableServer oud-ds-
   rs-3:$adminConnectorPort
   ```

   The output will look similar to the following:

   ```
   Establishing connections and reading configuration ..... Done.

   Removing server oud-ds-rs-3:1444 from the registration information ..... Done.
   ```

5. Delete the password file:

   ```
   rm /tmp/adminpassword.txt
   ```

15

Deleting an OUD Deployment

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

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

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

   For example:

   ```
   helm --namespace oudns list
   ```

   The output will look similar to the following:

   ```
   NAME        NAMESPACE    REVISION    UPDATED   STATUS      CHART         APP VERSION
   oud-ds-rs   oudns        1           <DATE>    deployed    oud-ds-rs-0.2 12.2.1.4.0
   ```

2. Delete the deployment using the following command:

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

   For example:

   ```
   helm uninstall --namespace oudns oud-ds-rs
   ```

   The output will look similar to the following:

   ```
   release "oud-ds-rs" uninstalled
   ```

3. Run the following command to view the status:

   ```
   kubectl --namespace oudns 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/oud-ds-rs-0   1/1     Terminating   0          24m   10.244.1.180   <Worker Node>   <none>         <none>
   pod/oud-ds-rs-1   1/1     Terminating   0          18m   10.244.1.181   <Worker Node>   <none>         <none>
   pod/oud-ds-rs-2   1/1     Terminating   0          12m   10.244.1.182   <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/oud-ds-rs-pv      20Gi       RWX            Delete           Terminating   oudns/oud-ds-rs-pvc
manual                24m   Filesystem

NAME                          STATUS       VOLUME       CAPACITY   ACCESS MODES
STORAGECLASS   AGE   VOLUMEMODE
persistentvolumeclaim/oud-ds-rs-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 oud-ds-rs-pvc -n oudns
kubectl delete pv oud-ds-rs-pv -n oudns
```

> **Note**
>
> If using block storage, you will see a PV and PVC for each pod. Delete all of the PVC's and PV's using the above commands.

6. Delete the persistent volume contents:

> **Note**
>
> The steps below are not relevant for block storage.

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

For example:

```
cd /nfs_volumes/oudpv/oud_user_projects
rm -rf *
```

Part IV
Appendices

This section includes the following topics:

A

Configuration Parameters for the oud-ds-rs Helm Chart

The following table lists the configurable parameters of the oud-ds-rs chart and their default values.

Parameter	Description	Default Value
replicaCount	Number of DS+RS instances/ pods/services to be created with replication enabled against a base Oracle Unified Directory instance/pod.	3
restartPolicyName	restartPolicy to be configured for each POD containing Oracle Unified Directory instance	OnFailure
image.repository	Oracle Unified Directory Image Registry/Repository and name. Based on this, image parameter would be configured for Oracle Unified Directory pods/containers.	oracle/oud
image.tag	Oracle Unified Directory Image Tag. Based on this, image parameter would be configured for Oracle Unified Directory 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.	oud-ds-rs-< 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.	
tolerations	Node taints to tolerate.	
affinity	Node/pod affinities.	
ingress.enabled		true

Parameter	Description	Default Value
ingress.type	Supported value: nginx.	nginx
ingress.nginx.http.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.nginx.http.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.nginx.http.backendPort		http
ingress.nginx.http.nginxAnnotation		{ ingressClassName: “nginx" }
ingress.nginx.admin.host	Hostname to be used with Ingress Rules. If not set, hostname would be configured according to fullname. Hosts would be configured as < fullname >-admin.< domain >, < fullname >-admin-0.< domain >, < fullname >-admin-1.< domain >, etc.	
ingress.nginx.admin.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.nginx.admin.nginxAnnotations		{ ingressClassName: “nginx” nginx.ingress.kubernetes.io/ backend-protocol: “https"}
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 >
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.	oud-ds-rs-< fullname >-creds
secret.type	Specifies the type of the secret	Opaque

Parameter	Description	Default Value
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.	oud-ds-rs-< fullname >-pv
persistence.pvcname	pvcname to use an already created Persistent Volume Claim , If blank will use default name.	oud-ds-rs-< 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/oud_user_projects
persistence.networkstorage.nfs.path	Path of NFS Share location.	/scratch/shared/oud_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.	{ }
configVolume.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
configVolume.mountPath	If enabled, it will use the persistent volume. If value is false, PV and PVC would not be used and there would not be any mount point available for config.	false

Parameter	Description	Default Value
configVolume.pvname	pvname to use an already created Persistent Volume , If blank will use the default name.	oud-ds-rs-< fullname >-pv-config
configVolume.pvcname	pvcname to use an already created Persistent Volume Claim , If blank will use default name	oud-ds-rs-< fullname >-pvc-config
configVolume.type	supported values: either filesystem or networkstorage or custom.	filesystem
configVolume.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/ oud_user_projects
configVolume.networkstorage.nfs. path	Path of NFS Share location.	/scratch/shared/oud_config
configVolume.networkstorage.nfs. server	IP or hostname of NFS Server.	0.0.0.0
configVolume.custom.*	Based on values/data, YAML content would be included in PersistenceVolume Object.	
configVolume.accessMode	Specifies the access mode of the location provided.	ReadWriteMany
configVolume.size	Specifies the size of the storage.	10Gi
configVolume.storageClass	Specifies the storageclass of the persistence volume.	empty
configVolume.annotations	Specifies any annotations that will be used.	{ }
configVolume.storageClassCreat e	If true, it will create the storageclass. if value is false, provide existing storage class (storageClass) to be used.	true
configVolume.provisioner	If configVolume.storageClassCreat e is true, please provide the custom provisioner if any.	kubernetes.io/is-default-class
oudPorts.adminldaps	Port on which Oracle Unified Directory Instance in the container should listen for Administration Communication over LDAPS Protocol.	1444
oudPorts.adminhttps	Port on which Oracle Unified Directory Instance in the container should listen for Administration Communication over HTTPS Protocol.	1888
oudPorts.ldap	Port on which Oracle Unified Directory Instance in the container should listen for LDAP Communication.	1389
oudPorts.ldaps	Port on which Oracle Unified Directory Instance in the container should listen for LDAPS Communication.	1636

Parameter	Description	Default Value
oudPorts.http	Port on which Oracle Unified Directory Instance in the container should listen for HTTP Communication.	1080
oudPorts.https	Port on which Oracle Unified Directory Instance in the container should listen for HTTPS Communication.	1081
oudPorts.replication	Port value to be used while setting up replication server.	1898
oudConfig.baseDN	BaseDN for Oracle Unified Directory Instances.	dc=example,dc=com
oudConfig.rootUserDN	Root User DN for Oracle Unified Directory Instances.	cn=Directory Manager
oudConfig.rootUserPassword	Password for Root User DN.	RandomAlphanum
oudConfig.sampleData	To specify that the database should be populated with the specified number of sample entries.	0
oudConfig.sleepBeforeConfig	Based on the value for this parameter, initialization/ configuration of each Oracle Unified Directory replica would be delayed.	120
oudConfig.adminUID	AdminUID to be configured with each replicated Oracle Unified Directory instance.	admin
oudConfig.adminPassword	Password for AdminUID. If the value is not passed, value of rootUserPassword would be used as password for AdminUID.	rootUserPassword
baseOUD.envVarsConfigMap	Reference to ConfigMap which can contain additional environment variables to be passed on to POD for Base Oracle Unified Directory Instance. Following are the environment variables which would not be honored from the ConfigMap. instanceType, sleepBeforeConfig, OUD_INSTANCE_NAME, hostname, baseDN, rootUserDN, rootUserPassword, adminConnectorPort, httpAdminConnectorPort, ldapPort, ldapsPort, httpPort, httpsPort, replicationPort, sampleData.	rootUserPassword

Parameter	Description	Default Value
baseOUD.envVarsConfigMap	Reference to ConfigMap which can contain additional environment variables to be passed on to POD for Base Oracle Unified Directory Instance. Following are the environment variables which would not be honored from the ConfigMap. instanceType, sleepBeforeConfig, OUD_INSTANCE_NAME, hostname, baseDN, rootUserDN, rootUserPassword, adminConnectorPort, httpAdminConnectorPort, ldapPort, ldapsPort, httpPort, httpsPort, replicationPort, sampleData.	
baseOUD.envVars	Environment variables in Yaml Map format. This is helpful when its requried to pass environment variables through –values file. List of env variables which would not be honored from envVars map is same as list of env var names mentioned for envVarsConfigMap. For a full list of environment variables, see **< Environment Variables>**.	

Parameter	Description	Default Value
replOUD.envVarsConfigMap	Reference to ConfigMap which can contain additional environment variables to be passed on to PODs for Replicated Oracle Unified Directory Instances. Following are the environment variables which would not be honored from the ConfigMap. instanceType, sleepBeforeConfig, OUD_INSTANCE_NAME, hostname, baseDN, rootUserDN, rootUserPassword, adminConnectorPort, httpAdminConnectorPort, ldapPort, ldapsPort, httpPort, httpsPort, replicationPort, sampleData, sourceHost, sourceServerPorts, sourceAdminConnectorPort, sourceReplicationPort, dsreplication_1, dsreplication_2, dsreplication_3, dsreplication_4, post_dsreplication_dsconfig_1, post_dsreplication_dsconfig_2 - replOUD.envVars Environment variables in Yaml Map format. This is helpful when its required to pass environment variables through –values file. List of env variables which would not be honored from envVars map is same as list of env var names mentioned for envVarsConfigMap. For a full list of environment variables, see **<Environment Variables>**.	
podManagementPolicy	Defines the policy for pod management within the statefulset. Typical values are OrderedReady/Parallel.	OrderedReady
updateStrategy	Allows you to configure and disable automated rolling updates for containers, labels, resource request/limits, and annotations for the Pods in a StatefulSet. Typical values are OnDelete/ RollingUpdate.	RollingUpdate
podManagementPolicy	Defines the policy for pod management within the statefulset. Typical values are OrderedReady/Parallel.	OrderedReady

Parameter	Description	Default Value
updateStrategy	Allows you to configure and disable automated rolling updates for containers, labels, resource request/limits, and annotations for the Pods in a StatefulSet. Typical values are OnDelete/ RollingUpdate	RollingUpdate
busybox.image	busy box image name. Used for initcontainers.	busybox
oudConfig.cleanupbeforeStart	Used to remove the individual pod directories during restart. Recommended value is false. Note: Do not change the default value (false) as it will delete the existing data and clone it from base pod again.	false
oudConfig.disablereplicationbeforeStop	This parameter is used to disable replication when a pod is restarted. Recommended value is false. Note Do not change the default value (false), as changing the value will result in an issue where the pod won't join the replication topology after a restart.	false
oudConfig.resources.requests.memory	This parameter is used to set the memory request for the OUD pod.	4Gi
oudConfig.resources.requests.cpu	This parameter is used to set the cpu request for the OUD pod.	0.5
oudConfig.resources.limits.memory	This parameter is used to set the memory limit for the OUD pod.	4Gi
oudConfig.resources.limits.cpu	This parameter is used to set the cpu limit for the OUD pod.	1
replOUD.groupId	Group ID to be used/configured with each Oracle Unified Directory instance in replicated topology.	1
service.lbrtype	Type of load balancer Service to be created for admin, http,ldap services. Values allowed: ClusterIP/NodePort.	ClusterIP
oudPorts.nodePorts.adminldaps	Public port on which the OUD instance in the container should listen for administration communication over LDAPS Protocol. The port number should be between 30000-32767. No duplicate values are allowed. Note: Set only if service.lbrtype is set as NodePort. If left blank then k8s will assign random ports in between 30000 and 32767.	

Parameter	Description	Default Value
oudPorts.nodePorts.adminhttps	Public port on which the OUD instance in the container should listen for administration communication over HTTPS Protocol. The port number should be between 30000-32767. No duplicate values are allowed. Note: Set only if service.lbrtype is set as NodePort. If left blank then k8s will assign random ports in between 30000 and 32767.	
oudPorts.nodePorts.ldap	Public port on which the OUD instance in the container should listen for LDAP communication. The port number should be between 30000-32767. No duplicate values are allowed. Note: Set only if service.lbrtype is set as NodePort. If left blank then k8s will assign random ports in between 30000 and 32767.	
oudPorts.nodePorts.ldaps	Public port on which the OUD instance in the container should listen for LDAPS communication. The port number should be between 30000-32767. No duplicate values are allowed. Note: Set only if service.lbrtype is set as NodePort. If left blank then k8s will assign random ports in between 30000 and 32767.	
oudPorts.nodePorts.http	Public port on which the OUD instance in the container should listen for HTTP communication. The port number should be between 30000-32767. No duplicate values are allowed. Note: Set only if service.lbrtype is set as NodePort. If left blank then k8s will assign random ports in between 30000 and 32767.	
oudPorts.nodePorts.https	Public port on which the OUD instance in the container should listen for HTTPS communication. The port number should be between 30000-32767. No duplicate values are allowed. Note: Set only if service.lbrtype is set as NodePort. If left blank then k8s will assign random ports in between 30000 and 32767.	

Parameter	Description	Default Value
oudConfig.integration	Specifies which Oracle components the server can be integrated with. It is recommended to choose the option covering your minimal requirements. Allowed values: no-integration (no integration), basic (Directory Integration Platform), generic (Directory Integration Platform, Database Net Services and E-Business Suite integration), eus (Directory Integration Platform, Database Net Services, E-Business Suite and Enterprise User Security integration).	no-integration
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

B

Environment Variables Used in the oud-ds-rs Helm Chart

The following table lists the environment variables of the oud-ds-rs chart and their default values.

Environment Variable	Description	Default Value
ldapPort	Port on which the Oracle Unified Directory instance in the container should listen for LDAP communication. Use ‘disabled’ if you do not want to enable it.	1389
ldapsPort	Port on which the Oracle Unified Directory instance in the container should listen for LDAPS communication. Use ‘disabled’ if you do not want to enable it.	1636
rootUserDN	DN for the Oracle Unified Directory instance root user.	——
rootUserPassword	Password for the Oracle Unified Directory instance root user.	——
adminConnectorPort	Port on which the Oracle Unified Directory instance in the container should listen for administration communication over LDAPS. Use ‘disabled’ if you do not want to enable it. Note that at least one of the LDAP or the HTTP administration ports must be enabled.	1444
httpAdminConnectorPort	Port on which the Oracle Unified Directory Instance in the container should listen for Administration Communication over HTTPS Protocol. Use ‘disabled’ if you do not want to enable it. Note that at least one of the LDAP or the HTTP administration ports must be enabled.	1888
httpPort	Port on which the Oracle Unified Directory Instance in the container should listen for HTTP Communication. Use ‘disabled’ if you do not want to enable it.	1080
httpsPort	Port on which the Oracle Unified Directory Instance in the container should listen for HTTPS Communication. Use ‘disabled’ if you do not want to enable it.	1081

Environment Variable	Description	Default Value
sampleData	Specifies the number of sample entries to populate the Oracle Unified Directory instance with on creation. If this parameter has a non-numeric value, the parameter addBaseEntry is added to the command instead of sampleData. Similarly, when the ldifFile_n parameter is specified sampleData will not be considered and ldifFile entries will be populated.	0
adminUID	User ID of the Global Administrator to use to bind to the server. This parameter is primarily used with the dsreplication command.	——
adminPassword	Password for adminUID.	——
bindDN1	BindDN to be used while setting up replication using dsreplication to connect to First Directory/ Replication Instance.	——
bindPassword1	Password for bindDN1.	——
bindDN2	BindDN to be used while setting up replication using dsreplication to connect to Second Directory/ Replication Instance.	——
bindPassword2	Password for bindDN2.	——
replicationPort	Port value to be used while setting up a replication server. This variable is used to substitute values in dsreplication parameters.	1898
sourceHost	Value for the hostname to be used while setting up a replication server. This variable is used to substitute values in dsreplication parameters.	——
initializeFromHost	Value for the hostname to be used while initializing data on a new Oracle Unified Directory instance replicated from an existing instance. This variable is used to substitute values in dsreplication parameters. It is possible to have a different value for sourceHost and initializeFromHost while setting up replication with Replication Server, sourceHost can be used for the Replication Server and initializeFromHost can be used for an existing Directory instance from which data will be initialized.	$sourceHost

Environment Variable	Description	Default Value
serverTuning	Values to be used to tune JVM settings. The default value is jvm-default. If specific tuning parameters are required, they can be added using this variable.	jvm-default
offlineToolsTuning	Values to be used to specify the tuning for offline tools. This variable if not specified will consider jvm-default as the default or specify the complete set of values with options if wanted to set to specific tuning.	jvm-default
generateSelfSignedCertificate	Set to “true” if the requirement is to generate a self signed certificate when creating an Oracle Unified Directory instance. If no value is provided this value takes the default, “true”. If using a certificate generated separately this value should be set to “false”.	true
usePkcs11Keystore	Use a certificate in a PKCS#11 token that the replication gateway will use as servercertificate when accepting encrypted connections from the Oracle Directory Server Enterprise Edition server. Set to “true” if the requirement is to use the usePkcs11Keystore parameter when creating an Oracle Unified Directory instance. By default this parameter is not set. To use this option generateSelfSignedCertificate should be set to “false”.	——
enableStartTLS	Enable StartTLS to allow secure communication with the directory server by using the LDAP port. By default this parameter is not set. To use this option generateSelfSignedCertificate should be set to “false”.	——
useJCEKS	Specifies the path of a JCEKS that contains a certificate that the replication gateway will use as server certificate when accepting encrypted connections from the Oracle Directory Server Enterprise Edition server. If required this should specify the keyStorePath, for example, /u01/oracle/config/keystore.	——

Environment Variable	Description	Default Value
useJavaKeystore	Specify the path to the Java Keystore (JKS) that contains the server certificate. If required this should specify the path to the JKS, for example, /u01/oracle/config/keystore. By default this parameter is not set. To use this option generateSelfSignedCertificate should be set to “false”.	——
usePkcs12keyStore	Specify the path to the PKCS#12 keystore that contains the server certificate. If required this should specify the path, for example, /u01/oracle/config/keystore.p12. By default this parameter is not set.	——
keyStorePasswordFile	Set password storage scheme, if configuring Oracle Unified Directory for Enterprise User Security. Set this to a value of either “sha1” or “sha2”. By default this parameter is not set.	——
eusPasswordScheme	Specifies the type of the secret.	——
jmxPort	Port on which the Directory Server should listen for JMX communication. Use ‘disabled’ if you do not want to enable it.	disabled
javaSecurityFile	Specify the path to the Java security file. If required this should specify the path, for example, /u01/oracle/config/new_security_file. By default this parameter is not set.	——
schemaConfigFile_n	‘n’ in the variable name represents a numeric value between 1 and 50. This variable is used to set the full path of LDIF files that need to be passed to the Oracle Unified Directory instance for schema configuration/extension. If required this should specify the path, for example, schemaConfigFile_1=/u01/oracle/config/00_test.ldif.	——
ldifFile_n	‘n’ in the variable name represents a numeric value between 1 and 50. This variable is used to set the full path of LDIF files that need to be passed to the Oracle Unified Directory instance for initial data population. If required this should specify the path, for example, ldifFile_1=/u01/oracle/config/test1.ldif.	——

Environment Variable	Description	Default Value
dsconfigBatchFile_n	'n' in the variable name represents a numeric value between 1 and 50. This variable is used to set the full path of LDIF files that need to be passed to the Oracle Unified Directory instance for batch processing by the dsconfig command. If required this should specify the path, for example, dsconfigBatchFile_1=/u01/oracle/config/dsconfig_1.txt. When executing the dsconfig command the following values are added implicitly to the arguments contained in the batch file : ${hostname}, ${adminConnectorPort}, ${bindDN} and ${bindPasswordFile}.	——
dstune_n	'n' in the variable name represents a numeric value between 1 and 50. Allows commands and options to be passed to the dstune utility as a full command.	——
dsconfig_n	'n' in the variable name represents a numeric value between 1 and 300. Each file represents a set of execution parameters for the dsconfig command. For each dsconfig execution, the following variables are added implicitly : ${hostname}, ${adminConnectorPort}, ${bindDN}, ${bindPasswordFile}.	——
dsreplication_n	'n' in the variable name represents a numeric value between 1 and 50. Each file represents a set of execution parameters for the dsreplication command. For each dsreplication execution, the following variables are added implicitly : ${hostname}, ${ldapPort}, ${ldapsPort}, ${adminConnectorPort}, ${replicationPort}, ${sourceHost}, ${initializeFromHost}, and ${baseDN}. Depending on the dsreplication sub-command, the following variables are added implicitly : ${bindDN1}, ${bindPasswordFile1}, ${bindDN2}, ${bindPasswordFile2}, ${adminUID}, and ${adminPasswordFile}.	——

Environment Variable	Description	Default Value
post_dsreplication_dsconfig_n	‘n’ in the variable name represents a numeric value between 1 and 300. Each file represents a set of execution parameters for the dsconfig command to be run following execution of the dsreplication command. For each dsconfig execution, the following variables/ values are added implicitly : – provider-name “Multimaster Synchronization”, ${hostname}, $ {adminConnectorPort}, $ {bindDN}, ${bindPasswordFile}.	——
rebuildIndex_n	‘n’ in the variable name represents a numeric value between 1 and 50. Each file represents a set of execution parameters for the rebuild-index command. For each rebuild-index execution, the following variables are added implicitly : $ {hostname}, $ {adminConnectorPort}, $ {bindDN}, ${bindPasswordFile}, and ${baseDN}.	——
manageSuffix_n	‘n’ in the variable name represents a numeric value between 1 and 50. Each file represents a set of execution parameters for the manage-suffix command. For each manage-suffix execution, the following variables are added implicitly : $ {hostname}, $ {adminConnectorPort}, $ {bindDN}, ${bindPasswordFile}.	——
importLdif_n	‘n’ in the variable name represents a numeric value between 1 and 50. Each file represents a set of execution parameters for the import-ldif command. For each import-ldif execution, the following variables are added implicitly : $ {hostname}, $ {adminConnectorPort}, $ {bindDN}, ${bindPasswordFile}.	——

Environment Variable	Description	Default Value
execCmd_n	'n' in the variable name represents a numeric value between 1 and 300. Each file represents a command to be executed in the container. For each command execution, the following variables are replaced, if present in the command : ${hostname}, ${ldapPort}, ${ldapsPort}, ${adminConnectorPort}.	——
persistence.annotations	specifies any annotations that will be used.	——
restartAfterRebuildIndex	Specifies whether to restart the server after building the index.	false
restartAfterSchemaConfig	Specifies whether to restart the server after configuring the schema.	false

Note

For the following parameters above, the following statement applies:

- dsconfig_n
- dsreplication_n
- post_dsreplication_dsconfig_n
- rebuildIndex_n
- manageSuffix_n
- importLdif_n
- execCmd_n

If values are provided, the following variables will be substituted with their values:

${hostname},${ldapPort},${ldapsPort},${adminConnectorPort},${replicationPort},${sourceHost},${initializeFromHost},${sourceAdminConnectorPort},${sourceReplicationPort},${baseDN},${rootUserDN},${adminUID},${rootPwdFile},${bindPasswordFile},${adminPwdFile},${bindPwdFile1},${bindPwdFile2}