Oracle® FMW

Deploying and Managing Oracle Identity Governance on Kubernetes

G22497-03
July 2025

Oracle FMW Deploying and Managing Oracle Identity Governance on Kubernetes,

G22497-03

Primary Author: Russell Hodgson

Contents

15 Kubernetes Horizontal Pod Autoscaler

16 Patching and Upgrading

17 General Troubleshooting

18 Deleting an OIG Deployment

List of Figures

1

What's New in This Release?

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

Table 1-1 Release Notes for Oracle Identity Governance 14c on Kubernetes

Date	Version	Change
July 2025	14.1.2.1.0 GitHub release version 25.3.1	Supports Oracle Identity Governance 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 OIG.
April 2025	14.1.2.1.0 GitHub release version 25.2.1	Supports Oracle Identity Governance 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 OIG.
March 2025	14.1.2.1.0 GitHub release version 25.1.3	Initial release of Oracle Identity Governance 14.1.2.1.0 on Kubernetes. Supports Oracle Identity Governance 14.1.2.1.0 deployment using the OIG container image and WebLogic Operator 4.2.10. The GitHub release version is the latest version of the deployment scripts used in Setting Up the Code Repository for OIG.

Part I

Introduction to Oracle Identity Governance on Kubernetes

Oracle Identity Governance (OIG) can be deployed on Kubernetes.

This section includes the following chapters:

2

Introducing Oracle Identity Governance on Kubernetes

Oracle Identity Governance (OIG) is supported for deployment on Kubernetes.

This chapter includes the following topics:

2.1 Overview of Oracle Identity Governance on Kubernetes

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

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

2.2 Key Features of Oracle Identity Governance on Kubernetes

The key features of using Oracle Identity Governance (OIG) 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 OIG 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

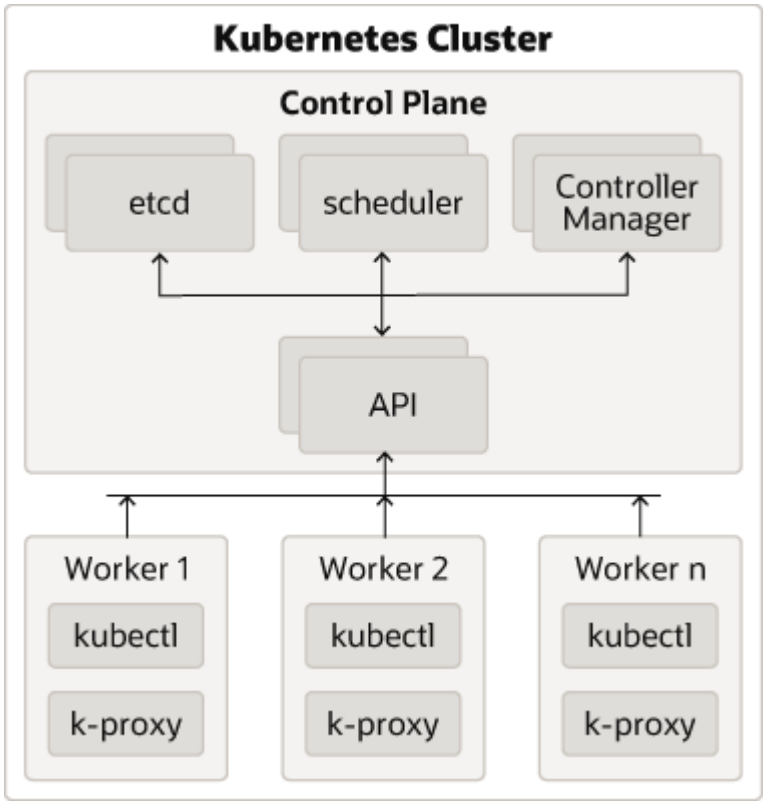
A Kubernetes host consists of a control plane and worker nodes.

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 OIG Deployment

An Oracle Identity Governance (OIG) deployment uses the Kubernetes components such as pods and Kubernetes services.

Container Image

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

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

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

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

When you upgrade a pod, you are actually instructing the pod to use a different container image. For example, if the container image for Oracle Identity Governance 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 Within 14.1.2.

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 Identity Governance (OIG) deployment, each OIG server runs in a different pod.

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

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

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

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

Pod Scheduling

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

Persistent Volumes

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

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

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

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

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

Kubernetes Services

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

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

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

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

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

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

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

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

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

- Load balancer
- Direct proxy calls
- DNS CNames

Ingress Controller

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

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

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

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

Domain Name System

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

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

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

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

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

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

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

- **Option 1** - Add a secondary DNS server to CoreDNS for the company domain.
- **Option 2** - Add individual host entries to CoreDNS for the external hosts.

Namespaces

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

In this guide the OIG deployment uses the namespace oigns.

3.4 Overview of WebLogic Operator

The WebLogic Kubernetes Operator (the “operator”) supports running Oracle Identity Governance (OIG) domains on Kubernetes.

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

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

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

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

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

- Create OIG instances in a Kubernetes persistent volume. This persistent volume can reside in an NFS file system or other Kubernetes volume types.
- Start servers based on declarative startup parameters and desired states.
- Expose the OIG Services through external access.

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

WebLogic Kubernetes Operator Limitations with OIG

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

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

3.5 OIG Deployment Methods

Oracle Identity Governance (OIG) can be deployed using one of the following methods:

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

WebLogic Scripting Tool Configuration Scripts

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

WebLogic Deploy Tooling Models

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

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

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

The main benefits of WDT models are:

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

Part II

Installing Oracle Identity Governance on Kubernetes

Install Oracle Identity Governance (OIG) on Kubernetes.

This section contains the following chapters:

4

Before You Begin

This documentation explains how to configure Oracle Identity Governance (OIG) 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 OIG on Kubernetes and follow the documentation sequentially.

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

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

The Enterprise Deployment Automation section in that guide also contains details on automation scripts that can:

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

5

System Requirements for OIG on Kubernetes

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

Kubernetes Requirements

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

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

> **Note**
>
> This documentation does not tell you how to install a Kubernetes cluster, Helm, or the container engine. Please refer to your vendor specific documentation for this information. Also see System Requirements for OIG on Kubernetes.

Database Requirements

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

- Oracle Database 19.23 or later. The database must be a supported version for OIG as outlined in Oracle Fusion Middleware 14c Certifications.
- The database must meet the requirements as outlined in About Database Requirements for an Oracle Fusion Middleware Installation and in RCU Requirements for Oracle Databases.
- It is recommended that the database initialization parameters are set as per Minimum Initialization Parameters.

Container Registry Requirements

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

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

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

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

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

6
Preparing Your Environment

Before embarking on Oracle Identity Management (OIG) 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 OIG 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 OIG Container Image

The Oracle Identity Governance (OIG) Kubernetes deployment requires access to an OIG container image.

Prebuilt OIG Container Image

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

- The OIG container images available can be found on Oracle Container Registry, by navigating to **Middleware** > **oig** for the initial March 2025 release, and **Middleware** > **oig_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/oig_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/oig: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 OIG Kubernetes deployment.
- Manually pull the container image from the Oracle Container Registry and then upload it to your own container registry.
- Manually pull the container image from the Oracle Container Registry and manually stage it on each worker node.

6.3 Creating a Persistent Volume Directory

As referenced in System Requirements for OIG 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/oig/governancedomainpv.

Perform the following steps:

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

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

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

For example:

```
cd /nfs_volumes/oig
mkdir governancedomainpv
sudo chown -R 1000:0 governancedomainpv
```

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

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

For example:

```
cd /nfs_volumes/oig/governancedomainpv
touch fileadmin.txt
ls fileadmin.txt
```

6.4 Setting Up the Code Repository for OIG

To deploy Oracle Identity Governance (OIG) you need to set up the code repository which provides sample deployment yaml files.

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

Perform the following steps to set up the OIG 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 /OIGK8S
```

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

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

For example:

```
cd /OIGK8S
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/OracleIdentityGovernance
```

For example:

```
export WORKDIR=/OIGK8S/fmw-kubernetes/OracleIdentityGovernance
```

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

```
kubectl get crd
```

In the output you should see:

```
No resources found
```

If you see any of the following:

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

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

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

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

6.5 Installing the WebLogic Kubernetes Operator

Oracle Identity Governance (OIG) on Kubernetes leverages the WebLogic Kubernetes Operator.

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

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

   For example:

   ```
   kubectl create namespace opns
   ```

   The output will look similar to the following:

   ```
   namespace/opns created
   ```

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

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

   For example:

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

   The output will look similar to the following:

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

3. Navigate to the $WORKDIR:

   ```
   cd $WORKDIR
   ```

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

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

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

For example:

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

The output will look similar to the following:

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

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

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

For example:

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

The output will look similar to the following:

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

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

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

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

6. Verify the operator pod's log:

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

For example:

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

The output will look similar to the following:

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

6.6 Creating a Kubernetes Namespace

You must create a namespace to store the Kubernetes objects for Oracle Identity Governance (OIG).

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

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

   For example:

   ```
   kubectl create namespace oigns
   ```

   The output will look similar to the following:

   ```
   namespace/oigns created
   ```

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

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

   For example:

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

   The output will look similar to the following:

   ```
   namespace/oigns labeled
   ```

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

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

For example:

```
kubectl describe namespace oigns
```

The output will look similar to the following:

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

No resource quota.

No LimitRange resource.
```

6.7 Creating a Kubernetes Secret for the Container Registry

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

1. Run the following command to create the secret:

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

For example, if using Oracle Container Registry:

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

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

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

The output will look similar to the following:

```
secret/orclcred created
```

7

Creating Oracle Identity Governance Domains

Choose one of the following supported methods to create an Oracle Identity Governance (OIG) domain:

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

7.1 Creating OIG Domains Using WLST Offline Scripts

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

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

This section includes the following topics:

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

7.1.1 Creating the RCU Schemas

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

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

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

- If using Oracle Container Registry or your own container registry for the Oracle Identity Governance (OIG) container image:

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

  For example:

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

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

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

  For example:

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

The output will look similar to the following:

```
pod/helper created
```

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

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

   For example:

   ```
   kubectl get pods -n oigns
   ```

   The output will look similar to the following:

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

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

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

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

   For example:

   kubectl exec -it helper -n oigns -- /bin/bash

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

   [oracle@helper ~]$

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

   export DB_HOST=<db_host.domain>

   export DB_PORT=<db_port>

   export DB_SERVICE=<service_name>

   export RCUPREFIX=<rcu_schema_prefix>

   export RCU_SCHEMA_PWD=<rcu_schema_pwd>

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

   cat /tmp/pwd.txt

   Where:

   - <db_host.domain> is the database server hostname.
   - <db_port> is the database listener port.
   - <service_name> is the database service name.
   - <rcu_schema_prefix> is the RCU schema prefix you want to set.
   - <db_pwd> is the SYS password for the database.

- <rcu_schema_pwd> is the password you want to set for the <rcu_schema_prefix>.

For example:

```
export DB_HOST=mydatabasehost.example.com

export DB_PORT=1521

export DB_SERVICE=orcl.example.com

export RCUPREFIX=OIGK8S

export RCU_SCHEMA_PWD=<password>

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

cat /tmp/pwd.txt
```

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

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

```
/u01/oracle/oracle_common/bin/rcu -silent -createRepository -databaseType ORACLE \
-connectString $DB_HOST:$DB_PORT/$DB_SERVICE \
-dbUser sys -dbRole sysdba -useSamePasswordForAllSchemaUsers true \
-selectDependentsForComponents true -schemaPrefix $RCUPREFIX -component OIM -component MDS -
component SOAINFRA -component OPSS \
-f < /tmp/pwd.txt
```

The output will look similar to the following:

```
RCU Logfile: /tmp/RCU<DATE>/logs/rcu.log

Processing command line ....
Repository Creation Utility - Checking Prerequisites
Checking Global Prerequisites


Repository Creation Utility - Checking Prerequisites
Checking Component Prerequisites
Repository Creation Utility - Creating Tablespaces
Validating and Creating Tablespaces
Create tablespaces in the repository database
Repository Creation Utility - Create
Repository Create in progress.
      Percent Complete: 10
Executing pre create operations
      Percent Complete: 25
```

```
        Percent Complete: 25
        Percent Complete: 26
        Percent Complete: 27
        Percent Complete: 28
        Percent Complete: 28
        Percent Complete: 29
        Percent Complete: 29
    Creating Common Infrastructure Services(STB)
        Percent Complete: 36
        Percent Complete: 36
        Percent Complete: 44
        Percent Complete: 44
        Percent Complete: 44
    Creating Audit Services Append(IAU_APPEND)
        Percent Complete: 51
        Percent Complete: 51
        Percent Complete: 59
        Percent Complete: 59
        Percent Complete: 59
    Creating Audit Services Viewer(IAU_VIEWER)
        Percent Complete: 66
        Percent Complete: 66
        Percent Complete: 67
        Percent Complete: 67
        Percent Complete: 68
        Percent Complete: 68
    Creating Metadata Services(MDS)
        Percent Complete: 76
        Percent Complete: 76
        Percent Complete: 76
        Percent Complete: 77
        Percent Complete: 77
        Percent Complete: 78
        Percent Complete: 78
        Percent Complete: 78
    Creating Weblogic Services(WLS)
        Percent Complete: 82
        Percent Complete: 82
        Percent Complete: 83
        Percent Complete: 84
        Percent Complete: 86
        Percent Complete: 88
        Percent Complete: 88
        Percent Complete: 88
    Creating User Messaging Service(UCSUMS)
        Percent Complete: 92
        Percent Complete: 92
        Percent Complete: 95
        Percent Complete: 95
        Percent Complete: 100
    Creating Audit Services(IAU)
    Creating Oracle Platform Security Services(OPSS)
    Creating SOA Infrastructure(SOAINFRA)
    Creating Oracle Identity Manager(OIM)
    Executing post create operations
```

```
Repository Creation Utility: Create - Completion Summary

Database details:
-----------------------------
Host Name                      : mydatabasehost.example.com
Port                           : 1521
Service Name                   : ORCL.EXAMPLE.COM
Connected As                   : sys
Prefix for (prefixable) Schema Owners       : OIGK8S
RCU Logfile                    : /tmp/RCU<DATE>/logs/rcu.log

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

Common Infrastructure Services              Success       /tmp/RCU<DATE>/logs/stb.log
Oracle Platform Security Services           Success       /tmp/RCU<DATE>/logs/opss.log
SOA Infrastructure                  Success       /tmp/RCU<DATE>/logs/soainfra.log
Oracle Identity Manager                 Success       /tmp/RCU<DATE>/logs/oim.log
User Messaging Service                  Success       /tmp/RCU<DATE>/logs/ucsums.log
Audit Services                  Success       /tmp/RCU<DATE>/logs/iau.log
Audit Services Append                   Success       /tmp/RCU<DATE>/logs/iau_append.log
Audit Services Viewer                   Success       /tmp/RCU<DATE>/logs/iau_viewer.log
Metadata Services                   Success       /tmp/RCU<DATE>/logs/mds.log
WebLogic Services                   Success       /tmp/RCU<DATE>/logs/wls.log

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

6. Run the following command inside the helper pod to patch schemas in the database:

> **Note**
>
> This command should only be run when using July OIG Patch Set Update (PSU) and Critical Patch Update (CPU) or later.

```
/u01/oracle/oracle_common/modules/thirdparty/org.apache.ant/apache-ant/bin/ant \
-f /u01/oracle/idm/server/setup/deploy-files/automation.xml \
run-patched-sql-files \
-logger org.apache.tools.ant.NoBannerLogger \
-logfile /u01/oracle/idm/server/bin/patch_oim_wls.log \
-DoperationsDB.host=$DB_HOST \
-DoperationsDB.port=$DB_PORT \
-DoperationsDB.serviceName=$DB_SERVICE \
-DoperationsDB.user=$RCUPREFIX_OIM \
-DOIM.DBPassword=$RCU_SCHEMA_PWD \
-Dojdbc=/u01/oracle/oracle_common/modules/oracle.jdbc/ojdbc11.jar
```

The output will look similar to the following:

```
Buildfile: /u01/oracle/idm/server/setup/deploy-files/automation.xml
```

7. Verify the database was patched successfully by viewing the patch_oim_wls.log:

   ```
   cat /u01/oracle/idm/server/bin/patch_oim_wls.log
   ```

   The output should look similar to below:

   ```
   ...
   run-patched-sql-files:
       [sql] Executing resource: /u01/oracle/idm/server/db/oim/oracle/StoredProcedures/Recon/
   OIM_SP_ReconBlkAccountChglog.sql
       [sql] Executing resource: /u01/oracle/idm/server/db/oim/oracle/Upgrade/oim14cBP/list/
   oim14c_dml_pty_insert_self_assignment_allowed.sql
       [sql] Executing resource: /u01/oracle/idm/server/db/oim/oracle/StoredProcedures/API/
   oim_role_mgmt_pkg_body.sql
       [sql] Executing resource: /u01/oracle/idm/server/db/oim/oracle/StoredProcedures/API/
   oim_usr_mgmt_pkg_body.sql
       [sql] Executing resource: /u01/oracle/idm/server/db/oim/oracle/StoredProcedures/DBDiagnostics/
   oim_db_diagnostics_pkg_body.sql
       [sql] 5 of 5 SQL statements executed successfully
   BUILD SUCCESSFUL
   Total time: 1 second
   ```

8. Exit the helper bash shell by issuing the command exit.

7.1.2 Creating a Kubernetes Secret for the WLST Domain

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

1. Run the following command to create the secret:The output will look similar to the following:

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

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

   Where:

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

For example:

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

```
./create-weblogic-credentials.sh -u weblogic -p <password> -n oigns -d governancedomain -s oig-domain-credentials
```

The output will look similar to the following:

```
secret/oig-domain-credentials created
secret/oig-domain-credentials labeled
The secret oig-domain-credentials has been successfully created in the oigns namespace.
```

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

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

For example:

```
kubectl get secret oig-domain-credentials -o yaml -n oigns
```

The output will look similar to the following:

```
apiVersion: v1
data:
  password: V2VsY29tZTE=
  username: d2VibG9naWM=
kind: Secret
metadata:
  creationTimestamp: "<DATE>"
  labels:
    weblogic.domainName: governancedomain
    weblogic.domainUID: governancedomain
  name: oig-domain-credentials
  namespace: oigns
  resourceVersion: "3216738"
  uid: c2ec07e0-0135-458d-bceb-c648d2a9ac54
type: Opaque
```

7.1.3 Creating a Kubernetes Secret for RCU in WLST

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

1. Run the following command to create the secret:

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

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

Where:

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

For example:

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

```
./create-rcu-credentials.sh -u OIGK8S -p <password> -a sys -q <password> -d governancedomain -n oigns -s oig-rcu-credentials
```

The output will look similar to the following:

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

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

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

For example:

```
kubectl get secret oig-rcu-credentials -o yaml -n oigns
```

The output will look similar to the following:

```
apiVersion: v1
data:
  password: V2VsY29tZTE=
  sys_password: V2VsY29tZTE=
  sys_username: c3lz
  username: T0lHSzhT
kind: Secret
metadata:
  creationTimestamp: "<DATE>"
  labels:
    weblogic.domainName: governancedomain
    weblogic.domainUID: governancedomain
  name: oig-rcu-credentials
  namespace: oigns
  resourceVersion: "3217023"
```

```
  uid: ce70b91a-fbbc-4839-9616-4cc2c1adeb4f
type: Opaque
```

7.1.4 Creating a Kubernetes Persistent Volume and Persistent Volume Claim

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

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

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

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

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

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

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

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

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

   ```
   mkdir output
   ```

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

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

   For example:

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

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

```
Kubernetes cluster.
domainUID: governancedomain

# Name of the namespace for the persistent volume claim
namespace: oigns

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

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

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

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

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

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

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

The output will be similar to the following:

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

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

```
Completed
```

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

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

The output will look similar to the following:

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

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

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

For example:

```
kubectl create -f output/pv-pvcs/governancedomain-domain-pv.yaml -n oigns
```

The output will look similar to the following:

```
persistentvolume/governancedomain-domain-pv created
```

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

```
kubectl describe pv governancedomain-domain-pv
```

The output will look similar to the following:

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

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

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

For example:

```
kubectl create -f output/pv-pvcs/governancedomain-domain-pvc.yaml -n oigns
```

The output will look similar to the following:

```
persistentvolumeclaim/governancedomain-domain-pvc created
```

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

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

For example:

```
kubectl describe pvc governancedomain-domain-pvc -n oigns
```

The output will look similar to the following:

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

7.1.5 Preparing the Create Domain Script

The sample scripts for Oracle Identity Governance (OIG) domain deployment are available in the $WORKDIR/kubernetes/create-oim-domain directory. You must prepare the scripts before deploying OIG.

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

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

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

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

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

```
domainUID: <domain_uid>
domainHome: /u01/oracle/user_projects/domains/<domain_uid>
```

```
image: <image_name>
imagePullSecretName: <container_registry_secret>
weblogicCredentialsSecretName: <kubernetes_domain_secret>
logHome: /u01/oracle/user_projects/domains/logs/<domain_id>
namespace: <domain_namespace>
persistentVolumeClaimName: <pvc_name>
rcuSchemaPrefix: <rcu_prefix>
rcuDatabaseURL: <rcu_db_host>:<rcu_db_port>/<rcu_db_service_name>
rcuCredentialsSecret: <kubernetes_rcu_secret>
frontEndHost: <front_end_hostname>
frontEndPort: <front_end_port>
```

For example:

```
domainUID: governancedomain
domainHome: /u01/oracle/user_projects/domains/governancedomain
image: container-registry.oracle.com/middleware/oig_cpu:14.1.2.1.0-jdk8-ol8-<YYMMDD>
imagePullSecretName: orclcred
weblogicCredentialsSecretName: oig-domain-credentials
logHome: /u01/oracle/user_projects/domains/logs/governancedomain
namespace: oigns
persistentVolumeClaimName: governancedomain-domain-pvc
rcuSchemaPrefix: OIGK8S
rcuDatabaseURL: mydatabasehost.example.com:1521/orcl.example.com
rcuCredentialsSecret: oig-rcu-credentials
frontEndHost: example.com
frontEndPort: 14100
```

> **Note**
>
> For now frontEndHost and frontEndPort should be set to example.com and 14100 respectively. These values will be changed to the correct values in post installation tasks in Setting OIMFrontendURL Using MBeans.

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

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

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

Parameter	Definition	Default
domainHome	Home directory of the OIG domain. If not specified, the value is derived from the domainUID as /shared/domains/<domainUID>.	/u01/oracle/user_projects/domains/governancedomain
domainPVMountPath	Mount path of the domain persistent volume.	/u01/oracle/user_projects/domains
domainUID	Unique ID that will be used to identify this particular domain. Used as the name of the generated WebLogic domain as well as the name of the Kubernetes domain resource. This ID must be unique across all domains in a Kubernetes cluster. This ID cannot contain any character that is not valid in a Kubernetes service name.	governancedomain
exposeAdminNodePort	Boolean indicating if the Administration Server is exposed outside of the Kubernetes cluster.	false
exposeAdminT3Channel	Boolean indicating if the T3 administrative channel is exposed outside the Kubernetes cluster.	true
image	OIG container image. The operator requires OIG 14.1.2. Refer to Obtaining the OIG Container Image for details on how to obtain or create the image.	oracle/oig:14.1.2.1.0
imagePullPolicy	WebLogic container image pull policy. Legal values are IfNotPresent, Always, or Never	IfNotPresent
imagePullSecretName	Name of the Kubernetes secret to access the container registry to pull the OIG container image. The presence of the secret will be validated when this parameter is specified.	orclcred
includeServerOutInPodLog	Boolean indicating whether to include the server .out to the pod's stdout.	true
initialManagedServerReplicas	Number of Managed Servers to initially start for the domain.	2
javaOptions	Java options for starting the Administration Server and Managed Servers. A Java option can have references to one or more of the following pre-defined variables to obtain WebLogic domain information: $(DOMAIN_NAME), $(DOMAIN_HOME), $(ADMIN_NAME), $(ADMIN_PORT), and $(SERVER_NAME).	-Dweblogic.StdoutDebugEnabled=false

Parameter	Definition	Default
logHome	The in-pod location for the domain log, server logs, server out, and Node Manager log files. If not specified, the value is derived from the domainUID as /shared/logs/<domainUID>.	/u01/oracle/user_projects/domains/logs/governancedomain
managedServerNameBase	Base string used to generate Managed Server names.	oim_server
managedServerPort	Port number for each Managed Server.	14000
namespace	Kubernetes namespace in which to create the domain.	oigns
persistentVolumeClaimName	Name of the persistent volume claim created to host the domain home. If not specified, the value is derived from the domainUID as <domainUID>-weblogic-sample-pvc.	governancedomain-domain-pvc
productionModeEnabled	Boolean indicating if production mode is enabled for the domain.	true
serverStartPolicy	Determines which WebLogic Server instances will be started. Legal values are Never, IfNeeded, AdminOnly.	IfNeeded
t3ChannelPort	Port for the T3 channel of the NetworkAccessPoint.	30012
t3PublicAddress	Public address for the T3 channel. This should be set to the public address of the Kubernetes cluster. This would typically be a load balancer address. For development environments only: In a single server (all-in-one) Kubernetes deployment, this may be set to the address of the master, or at the very least, it must be set to the address of one of the worker nodes.	If not provided, the script will attempt to set it to the IP address of the Kubernetes cluster
weblogicCredentialsSecretName	Name of the Kubernetes secret for the Administration Server's user name and password. If not specified, then the value is derived from the domainUID as <domainUID>-weblogic-credentials.	oig-domain-credentials
weblogicImagePullSecretName	Name of the Kubernetes secret for the container registry, used to pull the WebLogic Server image.	
serverPodCpuRequest, serverPodMemoryRequest, serverPodCpuCLimit, serverPodMemoryLimit	The maximum amount of compute resources allowed, and minimum amount of compute resources required, for each server pod. Please refer to the Kubernetes documentation on Managing Compute Resources for Containers for details.	Resource requests and resource limits are not specified.

Parameter	Definition	Default
rcuSchemaPrefix	The schema prefix to use in the database, for example OIGK8S. You may wish to make this the same as the domainUID in order to simplify matching domains to their RCU schemas. **Note** The RCU schema prefix can only contain alpha-numeric characters, and contain no spaces, or other special characters. It must begin with a letter and be no longer than 8 characters.	OIGK8S
rcuDatabaseURL	The database URL.	oracle-db.default.svc.cluster.local:1521/devpdb.k8s
rcuCredentialsSecret	The Kubernetes secret containing the database credentials.	oig-rcu-credentials

Parameter	Definition	Default
frontEndHost	The entry point URL for the OIG.	Not set
frontEndPort	The entry point port for the OIF.	Not set
datasourceType	Type of JDBC datasource applicable for the OIG domain. Legal values are agl and generic. Choose agl for Active GridLink datasource and generic for Generic datasource. For enterprise deployments, Oracle recommends that you use GridLink data sources to connect to Oracle RAC databases. See the Enterprise Deployment Guide for Oracle Identity and Access Management in a Kubernetes Cluster for further details.	generic

Note

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

7.1.6 Creating the domain.yaml

To create the domain.yaml file used in the Oracle Identity Governance (OIG) deployment:

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

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

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

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

   The output will look similar to the following:

   ```
   Input parameters being used
   export version="create-weblogic-sample-domain-inputs-v1"
   ```

```
export adminPort="7001"
export adminServerName="AdminServer"
export domainUID="governancedomain"
export domainHome="/u01/oracle/user_projects/domains/governancedomain"
export serverStartPolicy="IfNeeded"
export clusterName="oim_cluster"
export configuredManagedServerCount="5"
export initialManagedServerReplicas="1"
export managedServerNameBase="oim_server"
export managedServerPort="14000"
export image="container-registry.oracle.com/middleware/oig_cpu:14.1.2.1.0-jdk8-ol8-<YYMMDD>"
export imagePullPolicy="IfNotPresent"
export imagePullSecretName="orclcred"
export productionModeEnabled="true"
export weblogicCredentialsSecretName="oig-domain-credentials"
export includeServerOutInPodLog="true"
export logHome="/u01/oracle/user_projects/domains/logs/governancedomain"
export t3ChannelPort="30012"
export exposeAdminT3Channel="false"
export adminNodePort="30701"
export exposeAdminNodePort="false"
export namespace="oigns"
javaOptions=-Dweblogic.StdoutDebugEnabled=false
export persistentVolumeClaimName="governancedomain-domain-pvc"
export domainPVMountPath="/u01/oracle/user_projects/domains"
export createDomainScriptsMountPath="/u01/weblogic"
export createDomainScriptName="create-domain-job.sh"
export createDomainFilesDir="wlst"
export rcuSchemaPrefix="OIGK8S"
export rcuDatabaseURL="mydatabasehost.example.com:1521/orcl.example.com"
export rcuCredentialsSecret="oig-rcu-credentials"
export frontEndHost="example.com"
export frontEndPort="14100"
export datasourceType="generic"

validateWlsDomainName called with governancedomain
createFiles - valuesInputFile is create-domain-inputs.yaml
createDomainScriptName is create-domain-job.sh
Generating output/weblogic-domains/governancedomain/create-domain-job.yaml
Generating output/weblogic-domains/governancedomain/delete-domain-job.yaml
Generating output/weblogic-domains/governancedomain/domain.yaml
Checking to see if the secret governancedomain-domain-credentials exists in namespace oigns
configmap/governancedomain-create-fmw-infra-sample-domain-job-cm created
Checking the configmap governancedomain-create-fmw-infra-sample-domain-job-cm was created
configmap/governancedomain-create-fmw-infra-sample-domain-job-cm labeled
Checking if object type job with name governancedomain-create-fmw-infra-sample-domain-job exists
No resources found in oigns namespace.
Creating the domain by creating the job output/weblogic-domains/governancedomain/create-domain-job.yaml
job.batch/governancedomain-create-fmw-infra-sample-domain-job created
Waiting for the job to complete...
status on iteration 1 of 40
pod governancedomain-create-fmw-infra-sample-domain-job-8cww8 status is Running
status on iteration 2 of 40
pod governancedomain-create-fmw-infra-sample-domain-job-8cww8 status is Running
status on iteration 3 of 40
pod governancedomain-create-fmw-infra-sample-domain-job-8cww8 status is Running
```

```
status on iteration 4 of 40
pod governancedomain-create-fmw-infra-sample-domain-job-8cww8 status is Running
status on iteration 5 of 40
pod governancedomain-create-fmw-infra-sample-domain-job-8cww8 status is Running
status on iteration 6 of 40
pod governancedomain-create-fmw-infra-sample-domain-job-8cww8 status is Running
status on iteration 7 of 40
pod governancedomain-create-fmw-infra-sample-domain-job-8cww8 status is Running
status on iteration 8 of 40
pod governancedomain-create-fmw-infra-sample-domain-job-8cww8 status is Running
status on iteration 9 of 40
pod governancedomain-create-fmw-infra-sample-domain-job-8cww8 status is Running
status on iteration 10 of 40
pod governancedomain-create-fmw-infra-sample-domain-job-8cww8 status is Running
status on iteration 11 of 40
pod governancedomain-create-fmw-infra-sample-domain-job-8cww8 status is Completed

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

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

Completed
```

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

7.1.7 Setting the OIG Server Memory Parameters

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

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

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

   For example:

   ```
   cd $WORKDIR/kubernetes/create-oim-domain/domain-home-on-pv/output/weblogic-domains/
   governancedomain
   ```

2. Edit the domain.yaml file. Locate the section of the file starting with: clusterName: oim_cluster under governancedomain-oim-cluster. Add the memory settings as below:

   ```
   serverPod:
     env:
     - name: USER_MEM_ARGS
   ```

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

For example:

```
apiVersion: weblogic.oracle/v1
kind: Cluster
metadata:
  name: governancedomain-oim-cluster
  namespace: oigns
spec:
  clusterName: oim_cluster
  serverService:
    precreateService: true
  serverPod:
    env:
    - name: USER_MEM_ARGS
      value: "-XX:+UseContainerSupport -Djava.security.egd=file:/dev/./urandom -Xms8192m -Xmx8192m"
    resources:
      limits:
        cpu: "2"
        memory: "8Gi"
      requests:
        cpu: "1000m"
        memory: "4Gi"
  replicas: 0
```

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

3. Save the changes to domain.yaml.

7.1.8 Deploying the WLST OIG Domain

Deploy the Oracle Identity Governance (OIG) domain using the domain.yaml.

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

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

   For example:

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

   The output will look similar to the following:

   ```
   domain.weblogic.oracle/governancedomain unchanged
   cluster.weblogic.oracle/governancedomain-oim-cluster created
   cluster.weblogic.oracle/governancedomain-soa-cluster created
   ```

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

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

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

For example:

```
kubectl get pods -n oigns -w
```

The output will initially look similar to the following:

```
NAME                                                READY   STATUS      RESTARTS   AGE
governancedomain-create-fmw-infra-sample-domain-job-8cww8   0/1     Completed   0          27m
governancedomain-introspector-rctsv                 1/1     Running     0          6s
helper                                              1/1     Running     0          3h30m
```

The introspector pod will be displayed first.
After several minutes the Administration Server and SOA Server are both started.

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

When started, the pods should have aSTATUS of Running and READY = 1/1:

```
NAME                                                READY   STATUS      RESTARTS   AGE/
governancedomain-adminserver                        1/1     Running     0          7m30s
governancedomain-create-fmw-infra-sample-domain-job-8cww8   0/1     Completed   0          35m
governancedomain-soa-server1                        1/1     Running     0          4m
helper                                              1/1     Running     0          3h38m
```

> **Note**
>
> If there any failures, follow **Domain Creation Failure with WLST** in Known Issues.

3. Check the clusters using the following command:

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

For example:

```
kubectl get cluster -n oigns
```

The output will look similar to the following:

```
NAME                        AGE
governancedomain-oim-cluster   9m
governancedomain-soa-cluster   9m
```

4. Start the OIM server using the following command:

```
kubectl patch cluster -n <domain_namespace> <OIMClusterName> --type=merge -p '{"spec":
{"replicas":<initialManagedServerReplicas>}}'
```

For example:

```
kubectl patch cluster -n oigns governancedomain-oim-cluster --type=merge -p '{"spec":{"replicas":1}}'
```

The output will look similar to the following:

```
cluster.weblogic.oracle/governancedomain-oim-cluster patched
```

Run the following command to view the status of the OIG pods:

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

For example:

```
kubectl get pods -n oigns -w
```

The output will initially look similar to the following:

```
NAME                                                READY   STATUS      RESTARTS   AGE
governancedomain-adminserver                        1/1     Running     0          7m30s
governancedomain-create-fmw-infra-sample-domain-job-8cww8   0/1     Completed   0          35m
governancedomain-oim-server1                        1/1     Running     0          4m25s
governancedomain-soa-server1                        1/1     Running     0          4m
helper                                              1/1     Running     0          3h38m
```

> **Note**
>
> It will take several minutes before the governancedomain-oim-server1 pod has a STATUS of 1/1. While the pod is starting you can check the startup status in the pod log, by running the following command:
>
> kubectl logs governancedomain-oim-server1 -n oigns

> **Note**
>
> If there any failures, follow **Domain Creation Failure with WLST** in Known Issues.

7.1.9 Verifying the OIG WLST Deployment

Verifying the Domain, Pods and Services

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

kubectl get all,domains -n <domain_namespace>

For example:

kubectl get all,domains -n oigns

The output will look similar to the following:

```
NAME                                                        READY   STATUS      RESTARTS   AGE
pod/governancedomain-adminserver                            1/1     Running     0          19m30s
pod/governancedomain-create-fmw-infra-sample-domain-job-8cww8   0/1     Completed   0          47m
pod/governancedomain-oim-server1                            1/1     Running     0          16m25s
pod/governancedomain-soa-server1                            1/1     Running     0          16m
pod/helper                                                  1/1     Running     0          3h50m

NAME                                                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)              AGE
service/governancedomain-adminserver                ClusterIP   None            <none>        7001/TCP             28m
service/governancedomain-cluster-oim-cluster        ClusterIP   10.106.198.40   <none>        14002/TCP,14000/TCP
25m
service/governancedomain-cluster-soa-cluster        ClusterIP   10.102.218.11   <none>        7003/TCP             25m
service/governancedomain-oim-server1                ClusterIP   None            <none>        14002/TCP,14000/TCP  16m24s
service/governancedomain-oim-server2                ClusterIP   10.97.32.112    <none>        14002/TCP,14000/TCP  25m
service/governancedomain-oim-server3                ClusterIP   10.100.233.109  <none>        14002/TCP,14000/TCP  25m
service/governancedomain-oim-server4                ClusterIP   10.96.154.17    <none>        14002/TCP,14000/TCP  25m
service/governancedomain-oim-server5                ClusterIP   10.103.222.213  <none>        14002/TCP,14000/TCP  25m
service/governancedomain-soa-server1                ClusterIP   None            <none>        7003/TCP             25m
service/governancedomain-soa-server2                ClusterIP   10.104.43.118   <none>        7003/TCP             25m
service/governancedomain-soa-server3                ClusterIP   10.110.180.120  <none>        7003/TCP             25m
service/governancedomain-soa-server4                ClusterIP   10.99.161.73    <none>        7003/TCP             25m
```

```
service/governancedomain-soa-server5          ClusterIP   10.97.67.196    <none>        7003/TCP            25m

NAME                                                       COMPLETIONS   DURATION   AGE
job.batch/governancedomain-create-fmw-infra-sample-domain-job   1/1           3m6s       125m

NAME                                    AGE
domain.weblogic.oracle/governancedomain   24m

NAME                                                 AGE
cluster.weblogic.oracle/governancedomain-oim-cluster   23m
cluster.weblogic.oracle/governancedomain-soa-cluster   23m
```

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

- An Administration Server named AdminServer listening on port 7001.
- A configured OIG cluster named oim_cluster of size 5.
- A configured SOA cluster named soa_cluster of size 5.
- One started OIG managed server, named oim_server1, listening on port 14000.
- One started SOA managed server named soa_server1, listening on port 7003.
- Log files that are located in <persistent_volume>/logs/<domainUID>.

Verifying the Domain

Run the following command to describe the domain:

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

For example:

```
kubectl describe domain governancedomain -n oigns
```

The output will look similar to the following:

```
Name:         governancedomain
Namespace:    oigns
Labels:       weblogic.domainUID=governancedomain
Annotations:  <none>
API Version:  weblogic.oracle/v9
Kind:         Domain
Metadata:
  Creation Timestamp:  <DATE>
  Generation:          1
  Managed Fields:
    API Version:  weblogic.oracle/v9
    Fields Type:  FieldsV1
    fieldsV1:
      f:metadata:
        f:annotations:
          .:
          f:kubectl.kubernetes.io/last-applied-configuration:
        f:labels:
          .:
```

```
      f:weblogic.domainUID:
    f:spec:
      .:
      f:adminServer:
        .:
        f:adminChannelPortForwardingEnabled:
        f:serverPod:
          .:
          f:env:
        f:serverStartPolicy:
      f:clusters:
      f:dataHome:
      f:domainHome:
      f:domainHomeSourceType:
      f:failureRetryIntervalSeconds:
      f:failureRetryLimitMinutes:
      f:httpAccessLogInLogHome:
      f:image:
      f:imagePullPolicy:
      f:imagePullSecrets:
      f:includeServerOutInPodLog:
      f:logHome:
      f:logHomeEnabled:
      f:logHomeLayout:
      f:maxClusterConcurrentShutdown:
      f:maxClusterConcurrentStartup:
      f:maxClusterUnavailable:
      f:replicas:
      f:serverPod:
        .:
        f:env:
        f:volumeMounts:
        f:volumes:
      f:serverStartPolicy:
      f:webLogicCredentialsSecret:
        .:
        f:name:
  Manager:      kubectl-client-side-apply
  Operation:    Update
  Time:         <DATE>
  API Version:  weblogic.oracle/v9
  Fields Type:  FieldsV1
  fieldsV1:
    f:status:
      .:
      f:clusters:
      f:conditions:
      f:observedGeneration:
      f:servers:
      f:startTime:
  Manager:         Kubernetes Java Client
  Operation:       Update
  Subresource:     status
  Time:            <DATE>
Resource Version:  1247307
UID:               4933be73-df97-493f-a20c-bf1e24f6b3f2
```

```
Spec:
  Admin Server:
    Admin Channel Port Forwarding Enabled:  true
    Server Pod:
      Env:
        Name:           USER_MEM_ARGS
        Value:          -Djava.security.egd=file:/dev/./urandom -Xms512m -Xmx1024m
   Server Start Policy:  IfNeeded
  Clusters:
    Name:                    governancedomain-oim-cluster
    Name:                    governancedomain-soa-cluster
  Data Home:
  Domain Home:                  /u01/oracle/user_projects/domains/governancedomain
  Domain Home Source Type:      PersistentVolume
  Failure Retry Interval Seconds:  120
  Failure Retry Limit Minutes:     1440
  Http Access Log In Log Home:     true
  Image:                        container-registry.oracle.com/middleware/oig_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD>
  Image Pull Policy:            IfNotPresent
  Image Pull Secrets:
    Name:                       orclcred
  Include Server Out In Pod Log:    true
  Log Home:                     /u01/oracle/user_projects/domains/logs/governancedomain
  Log Home Enabled:             true
  Log Home Layout:              ByServers
  Max Cluster Concurrent Shutdown:  1
  Max Cluster Concurrent Startup:   0
  Max Cluster Unavailable:      1
  Replicas:                     1
  Server Pod:
    Env:
      Name:   JAVA_OPTIONS
      Value:  -Dweblogic.StdoutDebugEnabled=false
      Name:   USER_MEM_ARGS
      Value:  -Djava.security.egd=file:/dev/./urandom -Xms256m -Xmx1024m
    Volume Mounts:
      Mount Path:  /u01/oracle/user_projects/domains
      Name:        weblogic-domain-storage-volume
    Volumes:
      Name:  weblogic-domain-storage-volume
      Persistent Volume Claim:
        Claim Name:     governancedomain-domain-pvc
  Server Start Policy:  IfNeeded
  Web Logic Credentials Secret:
    Name:  oig-domain-credentials
Status:
  Clusters:
    Cluster Name:  oim_cluster
    Conditions:
      Last Transition Time:  <DATE>
      Status:                True
      Type:                  Available
      Last Transition Time:  <DATE>
      Status:                True
      Type:                  Completed
    Label Selector:          weblogic.domainUID=governancedomain,weblogic.clusterName=oim_cluster
```

```
  Maximum Replicas:  5
  Minimum Replicas:  0
  Observed Generation:  2
  Ready Replicas:  1
  Replicas:  1
  Replicas Goal:  1
  Cluster Name:  soa_cluster
  Conditions:
    Last Transition Time:  <DATE>
    Status:  True
    Type:  Available
    Last Transition Time:  <DATE>
    Status:  True
    Type:  Completed
  Label Selector:  weblogic.domainUID=governancedomain,weblogic.clusterName=soa_cluster
  Maximum Replicas:  5
  Minimum Replicas:  0
  Observed Generation:  1
  Ready Replicas:  1
  Replicas:  1
  Replicas Goal:  1
Conditions:
  Last Transition Time:  <DATE>
  Status:  True
  Type:  Available
  Last Transition Time:  <DATE>
  Status:  True
  Type:  Completed
Observed Generation:  1
Servers:
 Health:
    Activation Time:  <DATE>
    Overall Health:  ok
    Subsystems:
      Subsystem Name:  ServerRuntime
      Symptoms:
  Node Name:  worker-node2
  Pod Phase:  Running
  Pod Ready:  True
  Server Name:  AdminServer
  State:  RUNNING
  State Goal:  RUNNING
  Cluster Name:  oim_cluster
  Health:
    Activation Time:  <DATE>
    Overall Health:  ok
    Subsystems:
      Subsystem Name:  ServerRuntime
      Symptoms:
  Node Name:  worker-node1
  Pod Phase:  Running
  Pod Ready:  True
  Server Name:  oim_server1
  State:  RUNNING
  State Goal:  RUNNING
  Cluster Name:  oim_cluster
```

```
    Server Name:   oim_server2
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
    Cluster Name:  oim_cluster
    Server Name:   oim_server3
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
    Cluster Name:  oim_cluster
    Server Name:   oim_server4
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
    Cluster Name:  oim_cluster
    Server Name:   oim_server5
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
    Cluster Name:  soa_cluster
    Health:
      Activation Time:  <DATE>
      Overall Health:   ok
      Subsystems:
        Subsystem Name:  ServerRuntime
        Symptoms:
    Node Name:     worker-node1
    Pod Phase:     Running
    Pod Ready:     True
    Server Name:   soa_server1
    State:         RUNNING
    State Goal:    RUNNING
    Cluster Name:  soa_cluster
    Server Name:   soa_server2
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
    Cluster Name:  soa_cluster
    Server Name:   soa_server3
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
    Cluster Name:  soa_cluster
    Server Name:   soa_server4
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
    Cluster Name:  soa_cluster
    Server Name:   soa_server5
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
  Start Time:      <DATE>
Events:
  Type     Reason   Age                From              Message
  ----     ------   ----               ----              -------
  Normal   Created  35m                weblogic.operator  Domain governancedomain was created.
  Normal   Changed  34m (x1127 over 35m)  weblogic.operator  Domain governancedomain was changed.
  Warning  Failed   34m (x227 over 35m)   weblogic.operator  Domain governancedomain failed due to 'Domain
validation error': Cluster resource 'governancedomain-oim-cluster' not found in namespace 'oigns'
    Cluster resource 'governancedomain-soa-cluster' not found in namespace 'oigns'. Update the domain resource to
correct the validation error.
  Warning  Unavailable  17m                weblogic.operator  Domain governancedomain is unavailable: an insufficient
number of its servers that are expected to be running are ready.";
```

Warning Incomplete 17m weblogic.operator Domain governancedomain is incomplete for one or more of the following reasons: there are failures detected, there are pending server shutdowns, or not all servers expected to be running are ready and at their target image, auxiliary images, restart version, and introspect version.
Normal Completed 13m (x2 over 26m) weblogic.operator Domain governancedomain is complete because all of the following are true: there is no failure detected, there are no pending server shutdowns, and all servers expected to be running are ready and at their target image, auxiliary images, restart version, and introspect version.
Normal Available 13m (x2 over 26m) weblogic.operator Domain governancedomain is available: a sufficient number of its servers have reached the ready state.

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

Verifying the Pods

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

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

For example:

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

The output will look similar to the following:

```
NAME                                                READY   STATUS      RESTARTS   AGE     IP            NODE
NOMINATED NODE   READINESS GATES
governancedomain-adminserver                        1/1     Running     0          24m     10.244.1.42   worker-node2
<none>           <none>
governancedomain-create-fmw-infra-sample-domain-job-8cww8   0/1     Completed   0          52m     10.244.1.40
worker-node2   <none>           <none>
governancedomain-oim-server1                        1/1     Running     0          52m     10.244.1.44   worker-node2
<none>           <none>
governancedomain-soa-server1                        1/1     Running     0          21m     10.244.1.43   worker-node2
<none>           <none>
helper                                              1/1     Running     0          3h55m   10.244.1.39   worker-node2
<none>           <none>
```

Configuring the Ingress

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

7.2 Creating OIG Domains Using WDT Models

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

In this section a domain creation image is built using the supplied model files and that image is used for domain creation. You will need your own container registry to upload the domain image to. Having your own container repository is a prerequisite before creating an Oracle Identity Governance (OIG) domain with WDT models. If you don't have your own container registry, you can load the image on each node in the cluster instead. This documentation does

not explain how to create your own container registry, or how to load the image onto each node. Consult your vendor specific documentation for more information.

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

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

This section includes the following topics:

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

7.2.1 Creating a Kubernetes Secret for the WDT Domain

Create a Kubernetes secret for the Oracle Identity Governance (OIG) domain using the create-secret.sh script.

1. Navigate to the wdt-utils directory:

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

2. Run the following command to create the secret:

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

   Where:

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

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

For example:

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

The output will look similar to the following:

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

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

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

For example:

```
kubectl get secret governancedomain-weblogic-credentials -o yaml -n oigns
```

The output will look similar to the following:

```
apiVersion: v1
data:
  password: <password>
  username: d2VibG9naWM=
kind: Secret
metadata:
  creationTimestamp: "<DATE>"
  labels:
    weblogic.domainName: governancedomain
    weblogic.domainUID: governancedomain
  name: governancedomain-weblogic-credentials
  namespace: oigns
  resourceVersion: "3216738"
  uid: c2ec07e0-0135-458d-bceb-c648d2a9ac54
type: Opaque
```

7.2.2 Creating a Kubernetes Secret for RCU in WDT

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

1. Navigate to the wdt-utils directory:

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

2. Run the following command to create the secret:

```
./create-secret.sh -l "rcu_prefix=<rcu_prefix>" \
-l "rcu_schema_password=<rcu_schema_pwd>" \
```

```
-l "db_host=<db_host.domain>" \
-l "db_port=<db_port>" \
-l "db_service=<service_name>" \
-l "dba_user=<sys_db_user>" \
-l "dba_password=<sys_db_pwd>" \
-n <domain_namespace> \
-d <domain_uid> \
-s <domain_uid>-rcu-credentials
```

Where:

- <rcu_prefix> is the name of the RCU schema prefix to be created.
- <rcu_schema_pwd> is the password you want to create for the RCU schema prefix.
- <db_host.domain> is the hostname.domain of the database.
- <db_port> is the database listener port.
- <service_name> is the service name of the database.
- <sys_db_user> is the database user with SYSDBA privilege.
- <sys_db_pwd> is the SYS database password.
- <domain_uid> is the domain_uid that you want to create. This must be the same domain_uid used in Creating a Kubernetes Secret for the WDT Domain.
- <domain_namespace> is the OIG domain namespace.
- <domain_uid>-rcu-credentials is the name you want to create for the RCU secret for this namespace.

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

For example:

```
./create-secret.sh -l "rcu_prefix=OIGK8S" \
-l "rcu_schema_password=<password>" \
-l "db_host=mydatabasehost.example.com" \
-l "db_port=1521" \
-l "db_service=orcl.example.com" \
-l "dba_user=sys" \
-l "dba_password=<password>" \
-n oigns \
-d governancedomain \
-s governancedomain-rcu-credentials
```

The output will look similar to the following:

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

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

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

For example:

```
kubectl get secret governancedomain-rcu-credentials -o yaml -n oigns
```

The output will look similar to the following:

```
apiVersion: v1
data:
  db_host: <DB_HOST>
  db_port: MTUyMQ==
  db_service: <SERVICE_NAME>
  dba_password: <PASSWORD>
  dba_user: c3lz
  rcu_prefix: <RCU_PREFIX>
  rcu_schema_password: <RCU_PWD>
kind: Secret
metadata:
  creationTimestamp: "<DATE>"
  labels:
    weblogic.domainUID: governancedomain
  name: governancedomain-rcu-credentials
  namespace: oigns
  resourceVersion: "31695660"
  uid: 71cfcc73-4c96-42bd-b9a5-988ea9ed27ff
type: Opaque
```

7.2.3 Preparing the WDT Create Domain YAML Files

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

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

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

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

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

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

```
appVersion: 14c
domainUID: <domain_uid>
domainHome: /u01/oracle/user_projects/domains/<domain_uid>
image: <image_name>:<tag>
imagePullSecretName: <container_registry_secret>
logHome: /u01/oracle/user_projects/domains/logs/<domain_uid>
namespace: <domain_namespace>
```

```
weblogicDomainStorageType: NFS
weblogicDomainStorageNFSServer: <nfs_server>
weblogicDomainStoragePath: <physical_path_of_persistent_storage>
weblogicDomainStorageSize: 10Gi
```

For example:

```
appVersion: 14c
domainUID: governancedomain
domainHome: /u01/oracle/user_projects/domains/governancedomain
image: container-registry.oracle.com/middleware/oig_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD>
imagePullSecretName: orclcred
logHome: /u01/oracle/user_projects/domains/logs/governancedomain
namespace: oigns
weblogicDomainStorageType: NFS
weblogicDomainStorageNFSServer: mynfsserver
weblogicDomainStoragePath: /nfs_volumes/oig/governancedomainpv
weblogicDomainStorageSize: 10Gi
```

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

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

Parameter	Definition	Default
adminPort	Port number for the Administration Server inside the Kubernetes cluster.	7001
adminNodePort	Port number of the Administration Server outside the Kubernetes cluster.	30701
configuredManagedServerCount	Number of Managed Server instances to generate for the domain.	5
datasourceType	Type of JDBC datasource applicable for the OIG domain. Legal values are `agl` and `generic`. Choose `agl` for Active GridLink datasource and `generic` for Generic datasource. For enterprise deployments, Oracle recommends that you use GridLink data sources to connect to Oracle RAC databases. See the Preparing an Existing Database for an Enterprise Deploymentfor further details.	generic
domainHome	Home directory of the OIG domain. If not specified, the value is derived from the `domainUID` as `/shared/domains/<domainUID>`.	/u01/oracle/user_projects/domains/governancedomain

Parameter	Definition	Default
domainPVMountPath	Mount path of the domain persistent volume.	/u01/oracle/user_projects/domains
domainUID	Unique ID that will be used to identify this particular domain. Used as the name of the generated WebLogic domain as well as the name of the Kubernetes domain resource. This ID must be unique across all domains in a Kubernetes cluster. This ID cannot contain any character that is not valid in a Kubernetes service name.	governancedomain
edgInstall	Used only if performing an install using the Enterprise Deployment Guide. See, Enterprise Deployment Guide for Oracle Identity and Access Management in a Kubernetes Cluster	false
exposeAdminNodePort	Boolean indicating if the Administration Server is exposed outside of the Kubernetes cluster.	false
exposeAdminT3Channel	Boolean indicating if the T3 administrative channel is exposed outside the Kubernetes cluster.	true
frontEndHost	The entry point URL for the OIM.	example.com
frontEndPort	The entry point port for the OIM.	14000
image	OIG container image. The operator requires OIG 14.1.2. Refer to Obtaining the OIG Container Image for details on how to obtain or create the image.	oracle/oig:14.1.2.1.0
imagePullSecretName	Name of the Kubernetes secret to access the container registry to pull the OIG container image. The presence of the secret will be validated when this parameter is specified.	orclcred
initialManagedServerReplicas	Number of Managed Servers to initially start for the domain.	2
javaOptions	Java options for starting the Administration Server and Managed Servers. A Java option can have references to one or more of the following pre-defined variables to obtain WebLogic domain information: $(DOMAIN_NAME), $(DOMAIN_HOME), $(ADMIN_NAME), $(ADMIN_PORT), and $(SERVER_NAME).	-Dweblogic.StdoutDebugEnabled=false

Parameter	Definition	Default
logHome	The in-pod location for the domain log, server logs, server out, and Node Manager log files. If not specified, the value is derived from the domainUID as /shared/logs/<domainUID>.	/u01/oracle/user_projects/domains/logs/governancedomain
namespace	Kubernetes namespace in which to create the domain.	oigns
oimCPU	Initial CPU Units, 1000m = 1 CPU core.	1000m
oimMaxCPU	Max CPU a pod is allowed to consume.	2
oimMemory	Initial memory allocated to a pod.	4Gi
oimMaxMemory	Max memory a pod is allowed to consume.	8Gi
oimServerJavaParams	The memory parameters to use for the OIG managed servers.	"-Xms8192m -Xmx8192m"
productionModeEnabled	Boolean indicating if production mode is enabled for the domain.	true
soaCPU	Initial CPU Units, 1000m = 1 CPU core.	1000m
soaMaxCPU	Max CPU Cores pod is allowed to consume.	1
soaMemory	Initial Memory pod allocated to a pod.	4Gi
soaMaxMemory	Max Memory pod is allowed to consume.	10Gi
soaServerJavaParams	The memory parameters to use for the SOA managed servers	"-Xms8192m -Xmx8192m"
t3PublicAddress	Public address for the T3 channel. This should be set to the public address of the Kubernetes cluster. This would typically be a load balancer address. For development environments only: In a single server (all-in-one) Kubernetes deployment, this may be set to the address of the master, or at the very least, it must be set to the address of one of the worker nodes.	If not provided, the script will attempt to set it to the IP address of the Kubernetes cluster
weblogicDomainStorageType	Persistent volume storage type. Options are NFS for NFS volumes or HOST_PATH for shared file system.	NFS
weblogicDomainStorageNFSServer	Hostname or IP address of the NFS Server.	nfsServer
weblogicDomainStoragePath	Physical path to the persistent volume.	/scratch/governancedomainpv
weblogicDomainStorageSize	Total storage allocated to the persistent storage.	10Gi

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

7.2.4 Creating the WDT YAML files

Generate the required WDT models for the Oracle Identity Governance (OIG) domain, along with the domain resource yaml files.

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

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

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

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

   For example:

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

   The output will look similar to the following:

   ```
   input parameters being used
   export version="create-weblogic-sample-domain-inputs-v1"
   export appVersion="14c"
   export adminPort="7001"
   export domainUID="governancedomain"
   export configuredManagedServerCount="5"
   export initialManagedServerReplicas="1"
   export productionModeEnabled="true"
   export t3ChannelPort="30012"
   export datasourceType="generic"
   export edgInstall="false"
   export domainHome="/u01/oracle/user_projects/domains/governancedomain"
   export image="container-registry.oracle.com/middleware/oig_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD>"
   export imagePullSecretName="orclcred"
   export logHome="/u01/oracle/user_projects/domains/logs/governancedomain"
   export exposeAdminT3Channel="false"
   export adminNodePort="30701"
   export exposeAdminNodePort="false"
   export namespace="oigns"
   javaOptions=-Dweblogic.StdoutDebugEnabled=false
   export domainPVMountPath="/u01/oracle/user_projects"
   export weblogicDomainStorageType="NFS"
   export weblogicDomainStorageNFSServer="mynfsServer"
   export weblogicDomainStoragePath="/scratch/shared/governancedomainpv"
   export weblogicDomainStorageReclaimPolicy="Retain"
   ```

```
export weblogicDomainStorageSize="10Gi"
export frontEndHost="example.com"
export frontEndPort="14000"
export oimServerJavaParams="-Xms8192m -Xmx8192m "
export soaServerJavaParams="-Xms8192m -Xmx8192m "
export oimMaxCPU="2"
export oimCPU="1000m"
export oimMaxMemory="8Gi"
export oimMemory="4Gi"
export soaMaxCPU="1"
export soaCPU="1000m"
export soaMaxMemory="10Gi"
export soaMemory="4Gi"

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

> **Note**
>
> This will generate the domain.yaml, oig.yaml and oig.properties in output/weblogic-domains/governancedomain.

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

```
mkdir $WORKDIR/yaml


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

7.2.5 Building the Domain Creation Image

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

Domain creation images are used for supplying WDT model files, WDT variables files, WDT application archive files (collectively known as WDT model files), and the directory where the WebLogic Deploy Tooling software is installed (known as the WDT Home), when deploying a domain using a Domain on PV model. You distribute WDT model files and the WDT executable using these images, and the WebLogic Kubernetes Operator uses them to manage the domain.

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

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

Prerequisites

Verify that your environment meets the following prerequisites:

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

Preparing the Build Domain Image Script

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

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

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

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

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

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

   For example:

   ```
   JAVA_HOME=/scratch/jdk
   IMAGE_TAG=oig-aux-generic-v1
   BASE_IMAGE=ghcr.io/oracle/oraclelinux:8-slim
   REPOSITORY=container-registry.example.com/mytenancy/idm
   REG_USER=mytenancy/myemail@example.com
   IMAGE_PUSH_REQUIRES_AUTH=true
   WDT_MODEL_FILE="/OIGK8S/fmw-kubernetes/OracleIdentityGovernance/yaml/oig.yaml"
   WDT_VARIABLE_FILE="/OIGK8S/fmw-kubernetes/OracleIdentityGovernance/yaml/oig.properties"
   WDT_ARCHIVE_FILE=""
   WDT_VERSION="4.2.0"
   WIT_VERSION="1.14.3"
   ```

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

Parameter	Definition	Default
JAVA_HOME	Path to the JAVA_HOME for the JDK8+.	/scratch/jdk/jdk1.8.0_351
IMAGE_TAG	Image tag for the final domain creation image.	oig-aux-generic-v1
BASE_IMAGE	The Oracle Linux product container image to use as a base image.	ghcr.io/oracle/oraclelinux:8-slim
REPOSITORY	Container image repository that will host the domain creation image.	iad.ocir.io/mytenancy/idm
REG_USER	Username to authenticate to the <REGISTRY> and push the domain creation image.	mytenancy/ oracleidentitycloudservice/ myemail@example.com
IMAGE_PUSH_REQUIRES_AUTH	If authentication to <REGISTRY> is required then set to true, else set to false. If set to false, <REG_USER> is not required.	true
WDT_MODEL_FILE	Absolute path to WDT model file oig.yaml. For example $WORKDIR/yaml/ oig.yaml.	/scratch/model/oig.yaml
WDT_MODEL_FILE	Absolute path to WDT variable file oig.properties. For example $WORKDIR/yaml/ oig.properties.	/scratch/model/oig.properties
WDT_ARCHIVE_FILE	Absolute path to WDT archive file.	
WDT_VERSION	WebLogic Deploy Tool version. If not specified the latest available version will be downloaded. It is recommended to use the default value.	4.2.0
WIT_VERSION	WebLogic Image Tool Version. If not specified the latest available version will be downloaded. It is recommended to use the default value.	1.14.3
TARGET	Select the target environment in which the created image will be used. Supported values: Default or OpenShift. See Additional Information.	Default
CHOWN	userid:groupid to be used for creating files within the image, such as the WDT installer, WDT model, and WDT archive. If the user or group does not exist in the image, they will be added with useradd/groupadd.	oracle:oracle

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

Running the build-domain-creation-image Script

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

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

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

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

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

   The output will look similar to the following:

   ```
   using WDT_DIR: /OIGK8S/fmw-kubernetes/OracleIdentityGovernance/kubernetes/create-oim-domain/domain-home-on-pv/wdt-utils/build-domain-creation-image/workdir
   Using WDT_VERSION 4.2.0
   Using WIT_DIR /OIGK8S/fmw-kubernetes/OracleIdentityGovernance/kubernetes/create-oim-domain/domain-home-on-pv/wdt-utils/build-domain-creation-image/workdir
   Using WIT_VERSION 1.14.3
   Using Image tag: oig-aux-generic-v1
   using Base Image: ghcr.io/oracle/oraclelinux:8-slim
   using IMAGE_BUILDER_EXE /usr/bin/podman
   JAVA_HOME is set to /scratch/jdk
   @@  Info: WIT_INSTALL_ZIP_URL is ''
   @@ WIT_INSTALL_ZIP_URL is not set
   @@ imagetool.sh not found in /OIGK8S/fmw-kubernetes/OracleIdentityGovernance/kubernetes/create-oim-domain/domain-home-on-pv/wdt-utils/build-domain-creation-image/workdir/imagetool/bin. Installing imagetool...
   ```

```
@@ Info:  Downloading https://github.com/oracle/weblogic-image-tool/releases/download/release-1.14.3/imagetool.zip
@@ Info:  Downloading https://github.com/oracle/weblogic-image-tool/releases/download/release-1.14.3/imagetool.zip with https_proxy="http://proxy.example.com:80"
@@ Info: Archive downloaded to /OIGK8S/fmw-kubernetes/OracleIdentityGovernance/kubernetes/create-oim-domain/domain-home-on-pv/wdt-utils/build-domain-creation-image/workdir/imagetool.zip, about to unzip via '/home/opc/jdk/bin/jar xf'.
@@ Info: imageTool cache does not contain a valid entry for wdt_4.2.0. Installing WDT
@@  Info: WDT_INSTALL_ZIP_URL is ''
@@ WDT_INSTALL_ZIP_URL is not set
@@ Info:  Downloading https://github.com/oracle/weblogic-deploy-tooling/releases/download/release-4.2.0/weblogic-deploy.zip
@@ Info:  Downloading https://github.com/oracle/weblogic-deploy-tooling/releases/download/release-4.2.0/weblogic-deploy.zip with https_proxy="http://proxy.example.com:80"
@@ Info: Archive downloaded to /OIGK8S/fmw-kubernetes/OracleIdentityGovernance/kubernetes/create-oim-domain/domain-home-on-pv/wdt-utils/build-domain-creation-image/workdir/weblogic-deploy.zip
[INFO   ] Successfully added to cache. wdt_4.2.0=/OIGK8S/fmw-kubernetes/OracleIdentityGovernance/kubernetes/create-oim-domain/domain-home-on-pv/wdt-utils/build-domain-creation-image/workdir/weblogic-deploy.zip
@@ Info: Install succeeded, imagetool install is in the /OIGK8S/fmw-kubernetes/OracleIdentityGovernance/kubernetes/create-oim-domain/domain-home-on-pv/wdt-utils/build-domain-creation-image/workdir/imagetool directory.
Starting Building Image registry.example.com/mytenancy/idm:oig-aux-generic-v1
 Login Succeeded!
WDT_MODEL_FILE is set to /OIGK8S/fmw-kubernetes/OracleIdentityGovernance/yaml/oig.yaml
WDT_VARIABLE_FILE is set to /OIGK8S/fmw-kubernetes/OracleIdentityGovernance/yaml/oig.properties
Additional Build Commands file is set to /OIGK8S/fmw-kubernetes/OracleIdentityGovernance/kubernetes/create-oim-domain/domain-home-on-pv/wdt-utils/build-domain-creation-image/additonal-build-files/build-files.txt
Additonal Build file is set to /OIGK8S/fmw-kubernetes/OracleIdentityGovernance/kubernetes/create-oim-domain/domain-home-on-pv/wdt-utils/build-domain-creation-image/additonal-build-files/OIG.json
[INFO   ] WebLogic Image Tool version 1.14.3
[INFO   ] Image Tool build ID: 0c9aa58f-808b-4707-a11a-7766fb301cbb
[INFO   ] Temporary directory used for image build context: /home/oracle/wlsimgbuilder_temp1198331326550546381
[INFO   ] Copying /OIGK8S/fmw-kubernetes/OracleIdentityGovernance/kubernetes/create-oim-domain/domain-home-on-pv/wdt-utils/build-domain-creation-image/additonal-build-files/OIG.json to build context folder.
[INFO   ] User specified fromImage ghcr.io/oracle/oraclelinux:8-slim
[INFO   ] Inspecting ghcr.io/oracle/oraclelinux:8-slim, this may take a few minutes if the image is not available locally.
[INFO   ] Copying /OIGK8S/fmw-kubernetes/OracleIdentityGovernance/yaml/oig.yaml to build context folder.
[INFO   ] Copying /OIGK8S/fmw-kubernetes/OracleIdentityGovernance/yaml/oig.properties to build context folder.
[INFO   ] Copying /OIGK8S/fmw-kubernetes/OracleIdentityGovernance/kubernetes/create-oim-domain/domain-home-on-pv/wdt-utils/build-domain-creation-image/workdir/weblogic-deploy.zip to build context folder.
[INFO   ] Starting build: /usr/bin/podman build --no-cache --force-rm --tag registry.example.com/mytenancy/idm:oig-aux-generic-v1 --pull --build-arg http_proxy=http://proxy.example.com:80 --build-arg https_proxy=http://proxy.example.com:80 --build-arg no_proxy=localhost,127.0.0.0/8,.example.com,,/var/run/crio/crio.sock,X.X.X.X /home/oracle/wlsimgbuilder_temp1198331326550546381
[1/3] STEP 1/5: FROM ghcr.io/oracle/oraclelinux:8-slim AS os_update
[1/3] STEP 2/5: LABEL com.oracle.weblogic.imagetool.buildid="0c9aa58f-808b-4707-a11a-7766fb301cbb"
--> ba91c351bf94
[1/3] STEP 3/5: USER root
--> d8f89c65892a
[1/3] STEP 4/5: RUN microdnf update     && microdnf install gzip tar unzip libaio libnsl jq findutils diffutils shadow-utils     && microdnf clean all
```

```
Downloading metadata...
Downloading metadata...
Package                                 Repository          Size
Upgrading:
 libgcc-8.5.0-20.0.3.el8.x86_64               ol8_baseos_latest  93.4 kB
  replacing libgcc-8.5.0-20.0.2.el8.x86_64
 libstdc++-8.5.0-20.0.3.el8.x86_64            ol8_baseos_latest 474.6 kB
  replacing libstdc++-8.5.0-20.0.2.el8.x86_64
 systemd-libs-239-78.0.4.el8.x86_64           ol8_baseos_latest   1.2 MB
   replacing systemd-libs-239-78.0.3.el8.x86_64
Transaction Summary:
 Installing:       0 packages
 Reinstalling:     0 packages
 Upgrading:        3 packages
 Obsoleting:       0 packages
 Removing:         0 packages
 Downgrading:      0 packages
Downloading packages...
Running transaction test...
Updating: libgcc;8.5.0-20.0.3.el8;x86_64;ol8_baseos_latest
Updating: libstdc++;8.5.0-20.0.3.el8;x86_64;ol8_baseos_latest
Updating: systemd-libs;239-78.0.4.el8;x86_64;ol8_baseos_latest
Cleanup: libstdc++;8.5.0-20.0.2.el8;x86_64;installed
Cleanup: systemd-libs;239-78.0.3.el8;x86_64;installed
Cleanup: libgcc;8.5.0-20.0.2.el8;x86_64;installed
Complete.
Package                       Repository          Size
Installing:
 diffutils-3.6-6.el8.x86_64       ol8_baseos_latest 369.3 kB
 findutils-1:4.6.0-21.el8.x86_64  ol8_baseos_latest 539.8 kB
 gzip-1.9-13.el8_5.x86_64         ol8_baseos_latest 170.7 kB
 jq-1.6-7.0.3.el8.x86_64          ol8_appstream     206.5 kB
 libaio-0.3.112-1.el8.x86_64      ol8_baseos_latest  33.4 kB
 libnsl-2.28-236.0.1.el8.7.x86_64 ol8_baseos_latest 111.4 kB
 oniguruma-6.8.2-2.1.el8_9.x86_64 ol8_appstream     191.5 kB
 unzip-6.0-46.0.1.el8.x86_64      ol8_baseos_latest 201.0 kB
Transaction Summary:
Installing:       8 packages
 Reinstalling:     0 packages
 Upgrading:        0 packages
 Obsoleting:       0 packages
 Removing:         0 packages
 Downgrading:      0 packages
Downloading packages...
Running transaction test...
Installing: oniguruma;6.8.2-2.1.el8_9;x86_64;ol8_appstream
Installing: jq;1.6-7.0.3.el8;x86_64;ol8_appstream
Installing: unzip;6.0-46.0.1.el8;x86_64;ol8_baseos_latest
Installing: libnsl;2.28-236.0.1.el8.7;x86_64;ol8_baseos_latest
Installing: libaio;0.3.112-1.el8;x86_64;ol8_baseos_latest
Installing: gzip;1.9-13.el8_5;x86_64;ol8_baseos_latest
Installing: findutils;1:4.6.0-21.el8;x86_64;ol8_baseos_latest
Installing: diffutils;3.6-6.el8;x86_64;ol8_baseos_latest
Complete.
Complete.
--> 73fb79fa40b2
```

```
[1/3] STEP 5/5: RUN if [ -z "$(getent group oracle)" ]; then groupadd oracle || exit 1 ; fi  && if [ -z "$(getent
group oracle)" ]; then groupadd oracle || exit 1 ; fi  && if [ -z "$(getent passwd oracle)" ]; then useradd -g oracle
oracle || exit 1; fi  && mkdir -p /u01  && chown oracle:oracle /u01  && chmod 775 /u01
--> ff6cf74351d1
[2/3] STEP 1/4: FROM ff6cf74351d1e0124121321174eaa64ebefa0bc3eef80ec88caec12feb9e8fb3 AS wdt_build
[2/3] STEP 2/4: RUN mkdir -p /auxiliary  && mkdir -p /auxiliary/models  && chown oracle:oracle /auxiliary
--> a061b678fa0a
[2/3] STEP 3/4: COPY --chown=oracle:oracle ["weblogic-deploy.zip", "/tmp/imagetool/"]
--> 3daccfef2f06
[2/3] STEP 4/4: RUN test -d /auxiliary/weblogic-deploy && rm -rf /auxiliary/weblogic-deploy || echo Initial
WDT install         && unzip -q "/tmp/imagetool/weblogic-deploy.zip" -d /auxiliary
Initial WDT install
--> b77b02f66a83
[3/3] STEP 1/12: FROM ff6cf74351d1e0124121321174eaa64ebefa0bc3eef80ec88caec12feb9e8fb3 AS final
[3/3] STEP 2/12: ENV AUXILIARY_IMAGE_PATH=/auxiliary     WDT_HOME=/auxiliary
WDT_MODEL_HOME=/auxiliary/models
--> 10dc1832266f
[3/3] STEP 3/12: RUN mkdir -p /auxiliary && chown oracle:oracle /auxiliary
--> 0b85f8e7399a
[3/3] STEP 4/12: COPY --from=wdt_build --chown=oracle:oracle /auxiliary /auxiliary/
--> c64bf2bef430
[3/3] STEP 5/12: RUN mkdir -p /auxiliary/models && chown oracle:oracle /auxiliary/models
--> d8817f84ab58
[3/3] STEP 6/12: COPY --chown=oracle:oracle ["oig.yaml", "/auxiliary/models/"]
--> 45b1d25264b9
[3/3] STEP 7/12: COPY --chown=oracle:oracle ["oig.properties", "/auxiliary/models/"]
--> 2ceba77ee226
[3/3] STEP 8/12: RUN chmod -R 640 /auxiliary/models/*
--> 34385bac7974
[3/3] STEP 9/12: USER oracle
--> 409f6e3ccce4
[3/3] STEP 10/12: WORKDIR /auxiliary
--> aaa2f154f512
[3/3] STEP 11/12: COPY --chown=oracle:oracle files/OIG.json /auxiliary/weblogic-deploy/lib/typedefs
--> c8a9d29106d3
[3/3] STEP 12/12: RUN chmod -R 755 /auxiliary
[3/3] COMMIT registry.example.com/mytenancy/idm:oig-aux-generic-v1
--> 0797418499a1
Successfully tagged registry.example.com/mytenancy/idm:oig-aux-generic-v1
0797418499a1dfd6d2a28672948c17ed747291ad069cebca5fac1b0410978d75
[INFO   ] Build successful. Build time=72s. Image tag=registry.example.com/mytenancy/idm:oig-aux-generic-v1
Getting image source signatures
 Copying blob 462ffb36555c done
Copying blob 3db4d3748983 done
Copying blob 7e9f3f6c7a0a done
Copying blob 32aa5f13e19b done
Copying blob d979da323f64 done
Copying blob f18b9e5f415f done
Copying blob aaaea7c1392f done
Copying blob 5504fa641a87 done
Copying blob 5aa81493c602 done
Copying blob f56f992ba90d done
Copying blob 2b1e0644fbd3 done
Copying config a39dc6ae7f done
Writing manifest to image destination
Pushed image registry.example.com/mytenancy/idm/oig-aux-generic-v1 to image registry Docker Hub
```

7.2.6 Deploying the WDT OIG Domain

You must modify the Oracle Identity Governance (OIG) domain.yaml and deploy the OIG domain using the build image created.

Modify the OIG domain.yaml

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

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

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

For example:

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

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

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

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

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

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

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

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

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

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

A sample domain.yaml is shown below:

```
# Copyright (c) 2024, Oracle and/or its affiliates.
# Licensed under the Universal Permissive License v 1.0 as shown at https://oss.oracle.com/licenses/upl.
#
# This is an example of how to define an OIG Domain. For details about the fields in domain specification, refer
https://oracle.github.io/weblogic-kubernetes-operator/managing-domains/domain-resource/
#
apiVersion: "weblogic.oracle/v9"
kind: Domain
metadata:
  name: governancedomain
  namespace: oigns
  labels:
    weblogic.domainUID: governancedomain
spec:
  # The WebLogic Domain Home
  domainHome: /u01/oracle/user_projects/domains/governancedomain

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

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

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

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

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

```
  # Whether to enable log home
  logHomeEnabled: true

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

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

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

  serverPod:
    initContainers:
      #DO NOT CHANGE THE NAME OF THIS INIT CONTAINER
      - name: compat-connector-init
        # OIG Product image, same as spec.image mentioned above
        image: "container-registry.oracle.com/middleware/oig_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD>"
        imagePullPolicy: IfNotPresent
        command: [ "/bin/bash", "-c", "mkdir -p  /u01/oracle/user_projects/domains/ConnectorDefaultDirectory",
"mkdir -p  /u01/oracle/user_projects/domains/wdt-logs"]
        volumeMounts:
          - mountPath: /u01/oracle/user_projects
            name: weblogic-domain-storage-volume
    # a mandatory list of environment variable to be set on the servers
    env:
    - name: JAVA_OPTIONS
      value: -Dweblogic.StdoutDebugEnabled=false
    - name: USER_MEM_ARGS
      value: "-Djava.security.egd=file:/dev/./urandom -Xms256m -Xmx1024m "
    - name: WLSDEPLOY_LOG_DIRECTORY
      value: "/u01/oracle/user_projects/domains/wdt-logs"
    - name: FRONTENDHOST
      value: example.com
    - name: FRONTENDPORT
      value: "14000"
    - name: WLSDEPLOY_PROPERTIES
      value: "-Dwdt.config.disable.rcu.drop.schema=true"
    envFrom:
    - secretRef:
        name: governancedomain-rcu-credentials
    volumes:
    - name: weblogic-domain-storage-volume
      persistentVolumeClaim:
        claimName: governancedomain-domain-pvc
    volumeMounts:
```

```
  - mountPath: /u01/oracle/user_projects
    name: weblogic-domain-storage-volume

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

configuration:
    secrets: [ governancedomain-rcu-credentials ]
    initializeDomainOnPV:
      persistentVolume:
        metadata:
            name: governancedomain-domain-pv
        spec:
          storageClassName: governancedomain-domain-storage-class
          capacity:
        # Total storage allocated to the persistent storage.
              storage: 10Gi
        # Reclaim policy of the persistent storage
        # # The valid values are: 'Retain', 'Delete', and 'Recycle'
          persistentVolumeReclaimPolicy: Retain
        # Persistent volume type for the persistent storage.
        # # The value must be 'hostPath' or 'nfs'.
        # # If using 'nfs', server must be specified.
          nfs:
            server: mynfsserver
          # hostPath:
            path: "/scratch/shared/governancedomain"
      persistentVolumeClaim:
        metadata:
            name: governancedomain-domain-pvc
        spec:
          storageClassName: governancedomain-domain-storage-class
          resources:
              requests:
                  storage: 10Gi
          volumeName: governancedomain-domain-pv
      domain:
          # Domain | DomainAndRCU
          createIfNotExists: DomainAndRCU
          # Image containing WDT installer and Model files.
          domainCreationImages:
              - image: 'container-registry.example.com/mytenancy/idm:oig-aux-generic-v1'
          domainType: OIG
# References to Cluster resources that describe the lifecycle options for all
```

```
  # the Managed Server members of a WebLogic cluster, including Java
  # options, environment variables, additional Pod content, and the ability to
  # explicitly start, stop, or restart cluster members. The Cluster resource
  # must describe a cluster that already exists in the WebLogic domain
  # configuration.
  clusters:
  - name: governancedomain-oim-cluster
  - name: governancedomain-soa-cluster

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

---
# This is an example of how to define a Cluster resource.
apiVersion: weblogic.oracle/v1
kind: Cluster
metadata:
  name: governancedomain-oim-cluster
  namespace: oigns
spec:
  clusterName: oim_cluster
  serverService:
    precreateService: true
  replicas: 0
  serverPod:
    env:
    - name: USER_MEM_ARGS
      value: "-Djava.security.egd=file:/dev/./urandom -Xms8192m -Xmx8192m  "
    resources:
      limits:
        cpu: "2"
        memory: "8Gi"
      requests:
        cpu: "1000m"
        memory: "4Gi"


---
# This is an example of how to define a Cluster resource.
apiVersion: weblogic.oracle/v1
kind: Cluster
metadata:
  name: governancedomain-soa-cluster
  namespace: oigns
spec:
  clusterName: soa_cluster
  serverService:
    precreateService: true
  replicas: 1
  serverPod:
    env:
    - name: USER_MEM_ARGS
      value: "-Xms8192m -Xmx8192m "
    resources:
      limits:
        cpu: "1"
```

```
    memory: "10Gi"
  requests:
    cpu: "1000m"
    memory:  "4Gi"
```

Optional WDT Models ConfigMap

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

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

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

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

To create a configmap run the following commands:

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

```
./create-configmap.sh -n oigns -d governancedomain -c mymodel-domain-configmap -f wdt_models/mymodel.yaml
```

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

Deploying the OIG Domain

Deploy the OIG domain using the domain.yaml:

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

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

   The following steps will be performed by WebLogic Kubernetes Operator:

   - Run the introspector job.
   - The introspection job will create the RCU Schemas.
   - The introspector job pod will create the domain on PV using the model provided in the domain creation image.
   - The introspector job pod will execute OIG offline configuration actions post successful creation of domain via WDT.
   - Brings up the Administration Server, and the SOA Managed Server (soa_server1).

   The output will look similar to the following:

   ```
   domain.weblogic.oracle/governancedomain created
   cluster.weblogic.oracle/governancedomain-oim-cluster created
   cluster.weblogic.oracle/governancedomain-soa-cluster created
   ```

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

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

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

For example:

```
kubectl get pods -n oigns -w
```

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

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

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

2. Once everything is started you should see the Administration Server and SOA server are running:

```
NAME                          READY   STATUS    RESTARTS   AGE
governancedomain-adminserver   1/1     Running   0          13m
governancedomain-soa-server1   1/1     Running   0          10m
```

> **Note**
>
> Depending on the speed of your cluster, it can take around 25 minutes for all the pods to be in READY 1/1 state.

If there are any failures, follow **Domain creation failure with WDT models** in Known Issues.

3. Start the OIM server by running the following command:

```
kubectl patch cluster -n <domain_namespace> <domainUID>-oim-cluster --type=merge -p '{"spec":
{"replicas":1}}'
```

For example:

```
kubectl patch cluster -n oigns governancedomain-oim-cluster --type=merge -p '{"spec":{"replicas":1}}'
```

The output will look similar to the following:

```
cluster.weblogic.oracle/governancedomain-oim-cluster patched
```

4. Run the following command to view the status of the OIM server:

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

For example:

```
kubectl get pods -n oigns -w
```

Once the OIM server is running, the output will look similar to the following:

```
NAME                          READY   STATUS    RESTARTS   AGE
governancedomain-adminserver   1/1     Running   0          16m
governancedomain-soa-server1   1/1     Running   0          13m
governancedomain-oim-server1   1/1     Running   0          5m22s
```

If there are any failures, follow **Domain creation failure with WDT models** in Known Issues.

7.2.7 Verifying the WDT OIG Deployment

Verifying the Domain, Pods and Services

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

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

For example:

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

The output will look similar to the following:

```
NAME                              READY   STATUS    RESTARTS   AGE
pod/governancedomain-adminserver   1/1     Running   0          25m
pod/governancedomain-oim-server1   1/1     Running   0          7m18s
pod/governancedomain-soa-server1   1/1     Running   0          20m

NAME                                      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)              AGE
service/governancedomain-adminserver          ClusterIP   None            <none>        7001/TCP             25m
service/governancedomain-cluster-oim-cluster   ClusterIP   10.102.36.107   <none>        14002/TCP,14000/TCP
20m
service/governancedomain-cluster-soa-cluster   ClusterIP   10.102.230.187   <none>        7003/TCP             20m
service/governancedomain-oim-server1          ClusterIP   None            <none>        14002/TCP,14000/TCP   7m18s
service/governancedomain-oim-server2          ClusterIP   10.111.183.16   <none>        14002/TCP,14000/TCP   20m
service/governancedomain-oim-server3          ClusterIP   10.107.144.169   <none>        14002/TCP,14000/TCP   20m
service/governancedomain-oim-server4          ClusterIP   10.110.18.114   <none>        14002/TCP,14000/TCP   20m
service/governancedomain-oim-server5          ClusterIP   10.106.220.13   <none>        14002/TCP,14000/TCP   20m
service/governancedomain-soa-server1          ClusterIP   None            <none>        7003/TCP             20m
```

```
service/governancedomain-soa-server2      ClusterIP   10.104.204.68    <none>        7003/TCP          20m
service/governancedomain-soa-server3      ClusterIP   10.110.104.108   <none>        7003/TCP          20m
service/governancedomain-soa-server4      ClusterIP   10.103.117.118   <none>        7003/TCP          20m
service/governancedomain-soa-server5      ClusterIP   10.101.65.38     <none>        7003/TCP          20m

NAME                                  AGE
domain.weblogic.oracle/governancedomain   32m

NAME                                             AGE
cluster.weblogic.oracle/governancedomain-oim-cluster   32m
cluster.weblogic.oracle/governancedomain-soa-cluster   32m
```

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

- An Administration Server named AdminServer listening on port 7001.
- A configured OIG cluster named oig_cluster of size 5.
- A configured SOA cluster named soa_cluster of size 5.
- One started OIG Managed Server, named oim_server1, listening on port 14100.
- One started SOA Managed Server named soa_server1, listening on port 7003.
- Log files that are located in <persistent_volume>/logs/<domainUID>.

If the OIG deployment fails refer to General Troubleshooting.

Verifying the Domain

Run the following command to describe the domain:

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

For example:

```
kubectl describe domain governancedomain -n oigns
```

The output will look similar to the following:

```
Name:         governancedomain
Namespace:    oigns
Labels:       weblogic.domainUID=governancedomain
Annotations:  <none>
API Version:  weblogic.oracle/v9
Kind:         Domain
Metadata:
  Creation Timestamp:  <DATE>
  Generation:          1
  Resource Version:    1013312
  UID:                 b5b4446b-b056-431f-8ae4-db470ac7731e
Spec:
  Admin Server:
    Admin Channel Port Forwarding Enabled:  true
    Server Pod:
      Env:
        Name:            USER_MEM_ARGS
```

```
      Value:          -Djava.security.egd=file:/dev/./urandom -Xms512m -Xmx1024m
    Server Start Policy:  IfNeeded
  Clusters:
    Name:  governancedomain-oim-cluster
    Name:  governancedomain-soa-cluster
  Configuration:
    Initialize Domain On PV:
      Domain:
        Create If Not Exists:  DomainAndRCU
        Domain Creation Images:
          Image:     container-registry.example.com/mytenancy/idm:oig-aux-generic-v1
        Domain Type:  OIG
      Persistent Volume:
        Metadata:
          Name:  governancedomain-domain-pv
        Spec:
          Capacity:
            Storage:  10Gi
          Nfs:
            Path:                       /nfs_volumes/oig/governancedomainpv
            Server:                     mynfsserver
          Persistent Volume Reclaim Policy:  Retain
          Storage Class Name:           governancedomain-domain-storage-class
      Persistent Volume Claim:
        Metadata:
          Name:  governancedomain-domain-pvc
        Spec:
          Resources:
            Requests:
              Storage:          10Gi
          Storage Class Name:       governancedomain-domain-storage-class
          Volume Name:          governancedomain-domain-pv
    Override Distribution Strategy:  Dynamic
    Secrets:
      governancedomain-rcu-credentials
  Data Home:
  Domain Home:                  /u01/oracle/user_projects/domains/governancedomain
  Domain Home Source Type:      PersistentVolume
  Failure Retry Interval Seconds:  120
  Failure Retry Limit Minutes:    1440
  Http Access Log In Log Home:    true
  Image:                        container-registry.oracle.com/middleware/oig_cpu:14.1.2.1.0-jdk8-ol8-<YYMMDD>
  Image Pull Policy:            IfNotPresent
  Image Pull Secrets:
    Name:                       orclcred
    Name:                       privatecred
  Include Server Out In Pod Log:    true
  Log Home:                     /u01/oracle/user_projects/domains/logs/governancedomain
  Log Home Enabled:             true
  Max Cluster Concurrent Shutdown:  1
  Max Cluster Concurrent Startup:   0
  Max Cluster Unavailable:          1
  Replace Variables In Java Options:  false
  Replicas:                     1
  Server Pod:
    Env:
```

```
      Name:   JAVA_OPTIONS
      Value:  -Dweblogic.StdoutDebugEnabled=false
      Name:   USER_MEM_ARGS
      Value:  -Djava.security.egd=file:/dev/./urandom -Xms256m -Xmx1024m
      Name:   WLSDEPLOY_LOG_DIRECTORY
      Value:  /u01/oracle/user_projects/domains/wdt-logs
      Name:   FRONTENDHOST
      Value:  example.com
      Name:   FRONTENDPORT
      Value:  14000
      Name:   WLSDEPLOY_PROPERTIES
      Value:  -Dwdt.config.disable.rcu.drop.schema=true
    Env From:
      Secret Ref:
        Name:  governancedomain-rcu-credentials
    Init Containers:
      Command:
        /bin/bash
        -c
        mkdir -p  /u01/oracle/user_projects/domains/ConnectorDefaultDirectory
        mkdir -p  /u01/oracle/user_projects/domains/wdt-logs
      Image:             container-registry.oracle.com/middleware/oig_cpu:14.1.2.1.0-jdk8-ol8-<YYMMDD>
      Image Pull Policy:  IfNotPresent
      Name:              compat-connector-init
      Volume Mounts:
        Mount Path:  /u01/oracle/user_projects
        Name:        weblogic-domain-storage-volume
    Volume Mounts:
      Mount Path:  /u01/oracle/user_projects
      Name:        weblogic-domain-storage-volume
    Volumes:
      Name:  weblogic-domain-storage-volume
      Persistent Volume Claim:
        Claim Name:     governancedomain-domain-pvc
  Server Start Policy:  IfNeeded
  Web Logic Credentials Secret:
    Name:  governancedomain-weblogic-credentials
Status:
  Clusters:
    Cluster Name:  oim_cluster
    Conditions:
      Last Transition Time:  <DATE>
      Status:                True
      Type:                  Available
      Last Transition Time:  <DATE>
      Status:                True
      Type:                  Completed
    Label Selector:         weblogic.domainUID=governancedomain,weblogic.clusterName=oim_cluster
    Maximum Replicas:       5
    Minimum Replicas:       0
    Observed Generation:    2
    Ready Replicas:         1
    Replicas:               1
    Replicas Goal:          1
    Cluster Name:           soa_cluster
    Conditions:
```

```
    Last Transition Time:  <DATE>
    Status:                True
    Type:                  Available
    Last Transition Time:  <DATE>
    Status:                True
    Type:                  Completed
  Label Selector:          weblogic.domainUID=governancedomain,weblogic.clusterName=soa_cluster
  Maximum Replicas:        5
  Minimum Replicas:        0
  Observed Generation:     1
  Ready Replicas:          1
  Replicas:                1
  Replicas Goal:           1
Conditions:
  Last Transition Time:  <DATE>
  Status:                True
  Type:                  Available
  Last Transition Time:  <DATE>
  Status:                True
  Type:                  Completed
Observed Generation:     1
Servers:
  Health:
    Activation Time:  <DATE>
    Overall Health:   ok
    Subsystems:
      Subsystem Name:  ServerRuntime
      Symptoms:
  Node Name:     doc-worker2
  Pod Phase:     Running
  Pod Ready:     True
  Server Name:   AdminServer
  State:         RUNNING
  State Goal:    RUNNING
  Cluster Name:  oim_cluster
  Health:
    Activation Time:  <DATE>
    Overall Health:   ok
    Subsystems:
      Subsystem Name:  ServerRuntime
      Symptoms:
  Node Name:     doc-worker1
  Pod Phase:     Running
  Pod Ready:     True
  Server Name:   oim_server1
  State:         RUNNING
  State Goal:    RUNNING
  Cluster Name:  oim_cluster
  Server Name:   oim_server2
  State:         SHUTDOWN
  State Goal:    SHUTDOWN
  Cluster Name:  oim_cluster
  Server Name:   oim_server3
  State:         SHUTDOWN
  State Goal:    SHUTDOWN
  Cluster Name:  oim_cluster
```

```
    Server Name:   oim_server4
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
    Cluster Name:  oim_cluster
    Server Name:   oim_server5
    State:         SHUTDOWN
    State Goal:    SHUTDOWN
    Cluster Name:  soa_cluster
    Health:
      Activation Time:  <DATE>
      Overall Health:   ok
      Subsystems:
        Subsystem Name:  ServerRuntime
        Symptoms:
  Node Name:     doc-worker1
  Pod Phase:     Running
  Pod Ready:     True
  Server Name:   soa_server1
  State:         RUNNING
  State Goal:    RUNNING
  Cluster Name:  soa_cluster
  Server Name:   soa_server2
  State:         SHUTDOWN
  State Goal:    SHUTDOWN
  Cluster Name:  soa_cluster
  Server Name:   soa_server3
  State:         SHUTDOWN
  State Goal:    SHUTDOWN
  Cluster Name:  soa_cluster
  Server Name:   soa_server4
  State:         SHUTDOWN
  State Goal:    SHUTDOWN
  Cluster Name:  soa_cluster
  Server Name:   soa_server5
  State:         SHUTDOWN
  State Goal:    SHUTDOWN
 Start Time:     <DATE>
Events:          <none>
```

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

Verifying the Pods

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

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

For example:

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

The output will look similar to the following:

```
NAME                                READY   STATUS      RESTARTS   AGE     IP           NODE
NOMINATED NODE   READINESS GATES
governancedomain-adminserver                1/1     Running     0          26m     10.244.1.42   worker-node2
<none>           <none>
governancedomain-oim-server1                1/1     Running     0          7m56s   10.244.1.44   worker-node2
<none>           <none>
governancedomain-soa-server1                1/1     Running     0          21m     10.244.1.43   worker-node2
<none>           <none>
```

Configuring the Ingress

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

8
Configuring Ingress

You must configure an ingress controller to allow access to Oracle Identity Governance (OIG).

The ingress can be configured in the following ways:

- Without SSL
- With SSL

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 Generating SSL Certificates

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

For production environments it is recommended to use a commercially available certificate, traceable to a trusted Certificate Authority. For sandbox environments, you can generate your own self-signed certificates.

Using a Third Party CA for Generating Certificates

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

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

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

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

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

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

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

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

2. Copy the files listed above to the $WORKDIR>/ssl directory.

3. If your CA has multiple certificates in a chain, create a bundle.pem that contains all the CA certificates:

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

Using Self-Signed Certificates

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

   ```
   mkdir $WORKDIR/ssl
   ```

   ```
   cd $WORKDIR/ssl
   ```

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

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

   For example:

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

   The output will look similar to the following:

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

Creating a Kubernetes Secret for SSL

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

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

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

For example:

```
kubectl -n mynginxns create secret tls governancedomain-tls-cert --key /OIGK8S/ssl/oig.key --cert /OIGK8S/ssl/oig.crt
```

The output will look similar to the following:

```
secret/governancedomain-tls-cert created
```

8.4 Installing the NGINX Controller

In this section you install the NGINX controller.

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

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

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

Configuring an Ingress Controller with SSL

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

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

Where:

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

For example:

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

The output will look similar to the following:

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

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

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

An example Ingress that makes use of the controller:

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

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

```
  apiVersion: v1
  kind: Secret
  metadata:
    name: example-tls
    namespace: foo
  data:
```

```
  tls.crt: <base64 encoded cert>
  tls.key: <base64 encoded key>
type: kubernetes.io/tls
```

Configure an Ingress Controller Without SSL

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

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

Where:

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

For example:

```
helm install nginx-ingress \
-n mynginxns \
--set controller.service.nodePorts.http=30777 \
--set controller.service.type=NodePort \
--set controller.config.use-forwarded-headers=true \
--set controller.config.enable-underscores-in-headers=true \
--set controller.admissionWebhooks.enabled=false \
stable/ingress-nginx \
--version 4.7.2
```

The output will look similar to the following:

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

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

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

An example Ingress that makes use of the controller:

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

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

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

8.5 Preparing the Ingress values.yaml

To prepare the values.yaml for the ingress:

1. Navigate to the following directory and make a copy of the values.yaml

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

2. Make a copy of the values.yaml:

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

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

   - domainUID: - If you created your OIG domain with anything other than the default governancedomain, change accordingly.
   - sslType: - Values supported are SSL and NONSSL. If you created your ingress controller to use SSL then set to SSL, otherwise set to NONSSL.

The following show example files based on different configuration types:

SSL values.yaml

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

# SSL configuration Type. Supported Values are : NONSSL,SSL
sslType: SSL

# domainType. Supported values are: oim
domainType: oim

#WLS domain as backend to the load balancer
wlsDomain:
  domainUID: governancedomain
  adminServerName: AdminServer
  adminServerPort: 7001
  adminServerSSLPort:
  soaClusterName: soa_cluster
  soaManagedServerPort: 7003
  soaManagedServerSSLPort:
  oimClusterName: oim_cluster
  oimManagedServerPort: 14000
  oimManagedServerSSLPort:

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

# Ngnix specific values
nginx:
  nginxTimeOut: 180
```

NONSSL values.yaml

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

# SSL configuration Type. Supported Values are : NONSSL,SSL
```

```
sslType: NONSSL

# domainType. Supported values are: oim
domainType: oim

#WLS domain as backend to the load balancer
wlsDomain:
  domainUID: governancedomain
  adminServerName: AdminServer
  adminServerPort: 7001
  adminServerSSLPort:
  soaClusterName: soa_cluster
  soaManagedServerPort: 7003
  soaManagedServerSSLPort:
  oimClusterName: oim_cluster
  oimManagedServerPort: 14000
  oimManagedServerSSLPort:

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

# Ngnix specific values
nginx:
  nginxTimeOut: 180
```

8.6 Creating the Ingress

Run the following commands to create the ingress:

1. Navigate to the $WORKDIR:

   ```
   cd $WORKDIR
   ```

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

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

> **Note**
>
> The $WORKDIR/kubernetes/charts/ingress-per-domain/templates/nginx-ingress-ssl.yaml has nginx.ingress.kubernetes.io/enable-access-log set to false. If you want to enable access logs then set this value to true before executing the command. Enabling access-logs can cause issues with disk space if not regularly maintained.

For example:

```
helm install governancedomain-nginx kubernetes/charts/ingress-per-domain --namespace oigns --values
kubernetes/charts/ingress-per-domain/values.yaml
```

The output will look similar to the following:

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

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

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

For example

```
kubectl get ing -n oigns
```

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

```
NAME                  CLASS    HOSTS   ADDRESS   PORTS   AGE
governancedomain-nginx   <none>   *                 80     49s
```

4. Run the following command to check the ingress:

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

For example:

```
kubectl describe ing governancedomain-nginx -n oigns
```

The output will look similar to the following:

```
Name:             governancedomain-nginx
Labels:           app.kubernetes.io/managed-by=Helm
Namespace:        oigns
Address:          10.109.22.22
Ingress Class:    nginx
Default backend:  <default>
Rules:
  Host        Path  Backends
  ----        ----  --------
  *
              /console                governancedomain-adminserver:7001 (10.244.1.43:7001)
              /consolehelp            governancedomain-adminserver:7001 (10.244.1.43:7001)
              /em                     governancedomain-adminserver:7001 (10.244.1.43:7001)
              /management             governancedomain-adminserver:7001 (10.244.1.43:7001)
```

```
              /ws_utc                governancedomain-cluster-soa-cluster:7003 (10.244.2.247:7003)
              /soa                   governancedomain-cluster-soa-cluster:7003 (10.244.2.247:7003)
              /integration           governancedomain-cluster-soa-cluster:7003 (10.244.2.247:7003)
              /soa-infra             governancedomain-cluster-soa-cluster:7003 (10.244.2.247:7003)
              /identity              governancedomain-cluster-oim-cluster:14000 (10.244.2.248:14000)
              /admin                 governancedomain-cluster-oim-cluster:14000 (10.244.2.248:14000)
              /dms                   governancedomain-adminserver:7001 (10.244.1.43:7001)
              /oim                   governancedomain-cluster-oim-cluster:14000 (10.244.2.248:14000)
              /sysadmin              governancedomain-cluster-oim-cluster:14000 (10.244.2.248:14000)
              /workflowservice       governancedomain-cluster-oim-cluster:14000 (10.244.2.248:14000)
              /callbackResponseService   governancedomain-cluster-oim-cluster:14000 (10.244.2.248:14000)
              /spml-xsd              governancedomain-cluster-oim-cluster:14000 (10.244.2.248:14000)
              /HTTPClnt              governancedomain-cluster-oim-cluster:14000 (10.244.2.248:14000)
              /reqsvc                governancedomain-cluster-oim-cluster:14000 (10.244.2.248:14000)
              /iam                   governancedomain-cluster-oim-cluster:14000 (10.244.2.248:14000)
              /provisioning-callback     governancedomain-cluster-oim-cluster:14000 (10.244.2.248:14000)
              /CertificationCallbackService   governancedomain-cluster-oim-cluster:14000 (10.244.2.248:14000)
              /ucs                   governancedomain-cluster-oim-cluster:14000 (10.244.2.248:14000)
              /FacadeWebApp          governancedomain-cluster-oim-cluster:14000 (10.244.2.248:14000)
              /OIGUI                 governancedomain-cluster-oim-cluster:14000 (10.244.2.248:14000)
              /weblogic              governancedomain-cluster-oim-cluster:14000 (10.244.2.248:14000)
Annotations:  meta.helm.sh/release-name: governancedomain-nginx
              meta.helm.sh/release-namespace: oigns
              nginx.ingress.kubernetes.io/affinity: cookie
              nginx.ingress.kubernetes.io/affinity-mode: persistent
              nginx.ingress.kubernetes.io/enable-access-log: false
              nginx.ingress.kubernetes.io/proxy-read-timeout: 180
              nginx.ingress.kubernetes.io/proxy-send-timeout: 180
              nginx.ingress.kubernetes.io/session-cookie-name: sticky
Events:
  Type    Reason  Age                From                      Message
  ----    ------  ----               ----                      -------
  Normal  Sync    72s (x2 over 83s)  nginx-ingress-controller  Scheduled for sync
```

5. To confirm that the new ingress is successfully routing to the domain's server pods, run the following command to send a request to the OIG Enterprise Manager Fusion Middleware Control:

   - For SSL:

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

   - For NONSSL:

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

   Where ${HOSTNAME} is the host.domain of any of the nodes in the Kubernetes cluster, and ${PORT} is the ingress controller port. For example http://oig.example.com:30777/em. For example:

   ```
   curl -v http://oig.example.com:30777/em
   ```

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

```
> GET /em HTTP/1.1
> Host: oig.example.com:30777
> User-Agent: curl/7.61.1
> Accept: */*
>
< HTTP/1.1 302 Moved Temporarily
< Date: <DATE>
< Content-Type: text/html
< Content-Length: 353
< Connection: keep-alive
< Set-Cookie: sticky=fda412d59b87742b3b045e51cea11ade|59a94680c4648be4c706b7db494ae03c; Path=/em;
HttpOnly
< Location: http://oig.example.com:30777/em/console/home
< X-ORACLE-DMS-ECID: 9f09f338-557f-494c-b93c-a73817b15ef0-00000838
< X-ORACLE-DMS-RID: 0
< X-Content-Type-Options: nosniff
< Set-Cookie:
ADMINCONSOLESESSION=yCmKRBfyGrPN5ZUFb1Ys4CRyKisRHhcnuJfhUaSgswd5aPfTZGhO!
1679590153; path=/; HttpOnly
< X-Frame-Options: DENY
<
<html><head><title>302 Moved Temporarily</title></head>
<body bgcolor="#FFFFFF">
<p>This document you requested has moved
temporarily.</p>

<p>It's now at <a href="http://oig.example.com:30777/em/">http://oig.example.com:30777/em/</a>.</p>
</body></html>
* Connection #0 to host oig.example.com left intact
```

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

9

Validating the Domain URLs

Launch a browser and access the following URL's.

Login to Oracle Enterprise Manager Console with the weblogic username and password (weblogic/<password>).

Login to Oracle Identity Governance with the xelsysadm username and password (xelsysadm/<password>).

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

Console or Page	URL
Oracle Enterprise Manager Console	http(s)://${HOSTNAME}:${PORT}/em
Oracle Identity System Administration	http(s)://${HOSTNAME}:${PORT}/sysadmin
Oracle Identity Self Service	http(s)://${HOSTNAME}:${PORT}/identity

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

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

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

10

Post Installation Configuration

After the Oracle Identity Governance (OIG) domain is successfully deployed, you must perform some post configuration steps.

This chapter includes the following topics:

10.1 Creating a Server Overrides File

Perform the following steps to create a server overrides file for Oracle Identity Governance (OIG):

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

     ```
     cd $WORKDIR/kubernetes/create-oim-domain/domain-home-on-pv/output/weblogic-domains/governancedomain
     ```

   - For OIG domains created with WDT:

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

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

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

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

   ```
   chmod 755 setUserOverrides.sh
   ```

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

   For example:

   ```
   kubectl cp setUserOverrides.sh oigns/governancedomain-adminserver:/u01/oracle/user_projects/domains/governancedomain/bin/setUserOverrides.sh
   ```

4. Stop the OIG domain using the following command:

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

   For example:

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

   The output will look similar to the following:

   ```
   domain.weblogic.oracle/governancedomain patched
   ```

5. Check that all the pods are stopped:

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

   For example:

   ```
   kubectl get pods -n oigns
   ```

   The output will look similar to the following:

   ```
   NAME                                READY   STATUS        RESTARTS   AGE
   governancedomain-adminserver        1/1     Terminating   0          18h
   governancedomain-oim-server1        1/1     Terminating   0          18h
   governancedomain-soa-server1        1/1     Terminating   0          18h
   ```

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

6. Start the domain using the following command:

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

   For example:

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

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

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

   For example:

   ```
   kubectl get pods -n oigns
   ```

The output will look similar to the following:

```
NAME                                READY   STATUS     RESTARTS   AGE
governancedomain-introspector-jhjtj   1/1     Running    0          8s
```

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

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

```
NAME                                READY   STATUS     RESTARTS   AGE
governancedomain-adminserver          1/1     Running    0          6m4s
governancedomain-oim-server1          1/1     Running    0          3m5s
governancedomain-soa-server1          1/1     Running    0          3m5s
```

10.2 Setting OIMFrontendURL Using MBeans

Perform the following steps to set the OIMFrontendURL:

1. Login to Oracle Enterprise Manager Console, for example https://${HOSTNAME}:${PORT}/em.
2. Click the Target Navigation icon in the top left of the screen and navigate to the following:
   - Expand **Identity and Access** > **Access** > **OIM** > **oim**.
   - Right click the instance **oim** and select **System MBean Browser**
   - Under **Application Defined MBeans**, navigate to **oracle.iam**, **Server:oim_server1** > **Application:oim** > **XMLConfig** > **Config** > **XMLConfig.DiscoveryConfig** > **Discovery**.
3. In **OimFrontEndURL** enter the value as the URL entry point for OIG, for example http(s)://${HOSTNAME}:${PORT}.

   > **Note**
   >
   > The http(s)://${HOSTNAME}:${PORT} depends on the architecture configured, and your ingress setup as per Configuring Ingress.

4. Click **Apply**.

10.3 Updating the OIM Integration URLs

Perform the following steps to set the OIM Integration URL's:

1. Login to Oracle Enterprise Manager Console, for example https://${HOSTNAME}:${PORT}/em.

2. Click **Weblogic Domain**, and then click **System Mbean Browser**.
3. In the search box, enter OIMSOAIntegrationMBean, and click **Search**. The mbean is displayed.
4. In the Operations tab of the mbean, select **integrateWithSOAServer**.
5. Enter the following information:
   - **WebLogic Administrator User Name**: Enter the name of the WebLogic administrator. For example: weblogic.
   - **WebLogic Administrator Password**: Enter the password for the above account.
   - **OIM Front end URL**: Enter the value as the URL entry point for OIG, for example http(s)://${HOSTNAME}:${PORT}.

     > **Note**
     >
     > The http(s)://${HOSTNAME}:${PORT} depends on the architecture configured, and your ingress setup as per Configuring Ingress.

   - **OIM External Front End URL**: Enter the value as the URL entry point for OIG, for example http(s)://${HOSTNAME}:${PORT}.
   - **SOA SOAP URL**: Set this URL to the SOA Kubernetes Service used for internal call backs: http://<domain-UID>-cluster-soa-cluster.<domain_namespace>.svc.cluster.local:7003, for example: http://governancedomain-cluster-soa-cluster.oigns.svc.cluster.local:7003.
   - **SOA RMI URL**: Set this URL to the SOA Kubernetes Service used for internal call backs: t3://<domain-UID>-cluster-soa-cluster.<domain-namespace>.svc.cluster.local:7003, for example: http://governancedomain-cluster-soa-cluster.oigns.svc.cluster.local:7003.
   - **UMS Webservice URL**: Set this to the following URL: http://<domain-UID>-cluster-soa-cluster.<domain_namespace>.svc.cluster.local:7003/ucs/messaging/webservice, for example http://governancedomain-cluster-soa-cluster.oigns.svc.cluster.local:7003/ucs/messaging/webservice.
6. Click **Invoke**.

10.4 Installing and Configuring Connectors

If you need to use Oracle Identity Governance (OIG) connectors in Kubernetes you must download the connector, copy it so it can be used by the OIG deployment, and install it.

This section contains the following topics:

- Downloading OIG Connectors
- Copying the OIG Connector
- Installing the OIG Connector

10.4.1 Downloading OIG Connectors

Perform the following steps to download any Oracle Identity Govenance (OIG) connectors you require:

1. Download the connector you require. See, Oracle Identity Manager Connector Downloads.

2. Copy the connector zip file to a staging directory on the Kubernetes administrative host and unzip it:

```
mkdir <workdir>/connectors
```

```
cp <download_location><connector>.zip <workdir>/connectors
```

```
unzip <connector>.zip
```

For example:

```
cp $HOME/Downloads/Exchange-12.2.1.3.0.zip /OIGK8S/connectors
```

```
cd /OIGK8S/connectors
```

```
unzip exchange-12.2.1.3.0.zip
```

3. Change the permissions on the extracted files and directories:

```
chmod -R 755 *
```

10.4.2 Copying the OIG Connector

There are two options to copy Oracle Identity Governance (OIG) connectors to your Kubernetes cluster:

- Copy the connector directly to the persistent volume.
- Use the `kubectl cp` command to copy the connector to the persistent volume.

> **Note**
>
> It is recommended to copy the connector directly to the persistent volume, however there may be cases, for example when using a Managed Service such as Oracle Kubernetes Engine on Oracle Cloud Infrastructure, where it may not be feasible to directly mount the domain directory. In such cases the `kubectl cp` command should be used.

Copying the Connector Directly to the Persistent Volume

Run the following command to copy the connector zip file to the persistent volume:

```
cp -R <workdir>/connectors/<connector> <persistent_volume>/governancedomainpv/ConnectorDefaultDirectory
```

For example:

```
cp -R /OIGK8S/connectors/Exchange-12.2.1.3.0 /nfs_volumes/oig/governancedomainpv/ConnectorDefaultDirectory/
```

Using the kubectl cp Command

Run the following command to copy the connector zip file:

```
kubectl -n <domain_namespace> cp <workdir>/connectors/<connector> <cluster_name>:/u01/oracle/idm/server/ConnectorDefaultDirectory/
```

For example:

```
kubectl -n oigns cp /OIGK8S/connectors/Exchange-12.2.1.3.0 governancedomain-oim-server1:/u01/oracle/idm/server/ConnectorDefaultDirectory/
```

10.4.3 Installing the OIG Connector

The connectors are installed as they are on a standard on-premises setup, via Application On Boarding or via the Connector Installer.

Refer to your connector specific documentation for instructions.

10.5 Configuring Design Console

If you need to use the Oracle Identity Governance (OIG) Design Console you must configure an ingress to allow Design Console to connect to OIG in your Kubernetes cluster.

Before following this section, make sure you have installed the ingress controller as per Configuring Ingress, and that you know the entry point to OIG, for example http(s)://${HOSTNAME}:${PORT}.

If your ingress controller is configured for SSL, you will need the names of the secrets created for the certificates in Generating SSL Certificates.

This section contains the following topics:

- Configuring the Design Console Ingress
- Updating the T3 Channel
- Using the Design Console Client
- Logging in to the Design Console

10.5.1 Configuring the Design Console Ingress

To prepare the values.yaml for the Design Console ingress:

1. Navigate to the following directory and make a copy of the values.yaml:

   ```
   cd $WORKDIR/kubernetes/design-console-ingress
   ```

2. Make a copy of the values.yaml:

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

3. Edit the $WORKDIR/kubernetes/design-console-ingress/values.yaml and modify the following parameters if required:

- domainUID: - If you created your OIG domain with anything other than the default governancedomain, change accordingly.
- sslType: - Values supported are SSL and NONSSL. If you created your ingress controller to use SSL then set to SSL, otherwise set to NONSSL.
- secretName: If using SSL, change the name to the secret for your ingress controller, or if using NONSSL leave the default value.

For example:

```
# Load balancer type.  Supported values are: NGINX
type: NGINX
# Type of Configuration Supported Values are : NONSSL,SSL
# tls: NONSSL
tls: SSL
# TLS secret name if the mode is SSL
secretName: governancedomain-tls-cert


# WLS domain as backend to the load balancer
wlsDomain:
  domainUID: governancedomain
  oimClusterName: oim_cluster
  oimServerT3Port: 14002
```

4. Navigate to the $WORKDIR directory:

```
cd $WORKDIR
```

5. Run the following commands to create the ingress:

```
helm install governancedomain-nginx-designconsole kubernetes/design-console-ingress  --namespace
<domain_namespace>  --values kubernetes/design-console-ingress/values.yaml
```

For example:

```
helm install governancedomain-nginx-designconsole kubernetes/design-console-ingress  --namespace oigns  --
values kubernetes/design-console-ingress/values.yaml
```

The output will look similar to the following:

```
NAME: governancedomain-nginx-designconsole
<DATE>
NAMESPACE: oigns
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

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

```
kubectl describe ing governancedomain-nginx-designconsole -n <domain_namespace>
```

For example:

```
kubectl describe ing governancedomain-nginx-designconsole -n oigns
```

7. The output will look similar to the following:

   - For Non-SSL:

```
Name:             governancedomain-nginx-designconsole
Labels:           app.kubernetes.io/managed-by=Helm
                  weblogic.resourceVersion=domain-v2
Namespace:        oigns
Address:
Ingress Class:    nginx
Default backend:  <default>
Rules:
  Host        Path  Backends
  ----        ----  --------
  *
              governancedomain-cluster-oim-cluster:14002 (10.244.2.250:14002)
Annotations:  meta.helm.sh/release-name: governancedomain-nginx-designconsole
              meta.helm.sh/release-namespace: oigns
              nginx.ingress.kubernetes.io/affinity: cookie
              nginx.ingress.kubernetes.io/enable-access-log: false
Events:
  Type    Reason  Age   From                      Message
  ----    ------  ----  ----                      -------
  Normal  Sync    5s    nginx-ingress-controller  Scheduled for sync
```

   - For SSL:

```
Name:             governancedomain-nginx-designconsole
Labels:           app.kubernetes.io/managed-by=Helm
                  weblogic.resourceVersion=domain-v2
Namespace:        oigns
Address:
Ingress Class:    nginx
Default backend:  <default>
Rules:
  Host        Path  Backends
  ----        ----  --------
  *
              governancedomain-cluster-oim-cluster:14002 (10.244.2.250:14002)
Annotations:  meta.helm.sh/release-name: governancedomain-nginx-designconsole
              meta.helm.sh/release-namespace: oigns
              nginx.ingress.kubernetes.io/affinity: cookie
              nginx.ingress.kubernetes.io/enable-access-log: false
              nginx.ingress.kubernetes.io/configuration-snippet:
                more_set_input_headers "X-Forwarded-Proto: https";
                more_set_input_headers "WL-Proxy-SSL: true";
              nginx.ingress.kubernetes.io/enable-access-log: false
              nginx.ingress.kubernetes.io/ingress.allow-http: false
              nginx.ingress.kubernetes.io/proxy-buffer-size: 2000k
Events:
  Type    Reason  Age   From                      Message
```

```
----   ------  ----  ----                 -------
Normal  Sync    5s    nginx-ingress-controller  Scheduled for sync
```

10.5.2 Updating the T3 Channel

To update the T3 channel, perform the following steps:

1. Connect to the OIG Administration Server in the WebLogic Remote Console.
2. Select **Edit Tree** and In the left pane of the console, expand **Environment** > **Servers** > **oim_server1** > **Channels**. Click on **T3Channel**.
3. In the **Channel General** tab, set the **External Listen Address** to the hostname of the OIG URL entry point, for example ${HOSTNAME} and the **External Listen Port** to the port of the OIG URL entry point, for example ${PORT}.
4. Click **Save**.
5. Click the **Shopping Cart** in the top right of the console, and click **Commit Changes**.
6. Restart the OIG Managed Server for the above changes to take effect:

   ```
   cd $WORKDIR/kubernetes/domain-lifecycle
   ```

   ```
   ./restartServer.sh -s oim_server1 -d <domain_uid> -n <domain_namespace>
   ```

   For example:

   ```
   ./restartServer.sh -s oim_server1 -d governancedomain -n oigns
   ```

   The output will look similar to the following:

   ```
   [<DATE>][INFO] Initiating restart of 'oim_server1' by deleting server pod 'governancedomain-oim-server1'.
   [<DATE>][INFO] Server restart succeeded !
   ```

7. Check oim-server1 has a READY status of 1/1 before continuing:

   ```
   kubectl get pods -n <domain_namespace> | grep oim-server1
   ```

   For example:

   ```
   kubectl get pods -n oigns | grep oim-server1
   ```

   The output will look similar to the following:

   ```
   governancedomain-oim-server1                1/1     Running   0          8m
   ```

10.5.3 Using the Design Console Client

To use the Design Console with an Oracle Identity Governance (OIG) deployed on Kubernetes, you can choose one of the following options:

- Using On-Premises Design Console

- Using a Container Image for Design Console

10.5.3.1 Using On-Premises Design Console

To use on-premises Design Console with Oracle Identity Governance (OIG), perform the following steps:

1. Install Design Console on an on-premises machine.
2. If your OIG entry point uses SSL, then import the Certificate Authority (CA) certificate(s) that signed the certificate into the java truststore used by the Design Console. For example:

   ```
   keytool -import -trustcacerts -alias dc -file <certificate> -keystore $JAVA_HOME/jre/lib/security/cacerts
   ```

   Where <certificate> is the CA certificate(s).
3. Follow Logging in to the Design Console.

10.5.3.2 Using a Container Image for Design Console

The Design Console can be run from a container using X Windows emulation.

To use a container image for Design Console with Podman or Docker, perform the following steps:

> **Note**
>
> The example below use podman. Unless stated the podman command can be replaced with docker.

1. On the parent machine where the Design Console is to be displayed, run xhost +.
2. Find which worker node the oim-server1 pod is running. For example:

   ```
   kubectl get pods -n <domain_namespace> -o wide | grep <domainUID>-oim-server1
   ```

   For example:

   ```
   kubectl get pods -n oigns -o wide | grep governancedomain-oim-server1
   ```

   The output will look similar to the following:

   ```
   governancedomain-oim-server1                1/1     Running   0          31m   10.244.2.98   worker-node2
   ```
3. On the worker node returned above, for example worker-node2, execute the following command to find the OIG container image name:

   ```
   sudo podman images
   ```

The output will be similar to the following:

```
REPOSITORY                                                TAG                          IMAGE ID      CREATED       SIZE
container-registry.oracle.com/middleware/oig_cpu  14.1.2.1.0-jdk17-ol8-<YYDDMM>  7cde9673ba56  5 days ago    4.43 GB
```

4. Run the following command to start a container to run Design Console:

```
podman run -u root --name oigdcbase -it <image> bash
```

For example:

```
podman run -u root -it --name oigdcbase container-registry.oracle.com/middleware/oig_cpu:14.1.2.1.0-jdk17-ol8-<YYDDMM> bash
```

This will take you into a bash shell inside the container:

```
bash-4.2#
```

5. Inside the bash shell for the container, run the following command to set the proxy to the internet:

```
export https_proxy=http://proxy.example.com:80
```

6. Run the following command to install the required packages:

```
yum install libXext libXrender libXtst
```

7. If using SSL, copy the Certificate Authority (CA) certificate(s) for your OIG URL entry point, to the worker node where the `oigdcbase` image is to be created. Run the following command outside the container:

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

```
podman cp <certificate> <container_name>:/u01/jdk/jre/lib/security/<certificate>
```

For example:

```
podman cp ca.crt oigdcbase:/u01/jdk/jre/lib/security/ca.crt
```

8. Inside the bash shell for the container, run the following command to import the CA certificate:

```
/u01/jdk/bin/keytool -import -trustcacerts -alias dc -file /u01/jdk/jre/lib/security/<certificate> \
-keystore /u01/jdk/jre/lib/security/cacerts
```

For example:

```
/u01/jdk/bin/keytool -import -trustcacerts -alias dc -file /u01/jdk/jre/lib/security/ca.crt \
-keystore /u01/jdk/jre/lib/security/cacerts
```

9. Outside the container, run the following command to create a new Design Console image from the container:

   ```
   podman commit <container_name> <design_console_image_name>
   ```

   For example:

   ```
   podman commit oigdcbase oigdc
   ```

10. Exit the container bash session:

    ```
    exit
    ```

11. Start a new container using the Design Console image:

    ```
    podman run --name oigdc -it oigdc /bin/bash
    ```

    This will take you into a bash shell for the container:

    ```
    bash-4.2#
    ```

12. In the bash shell for the container, run the following to export the DISPLAY:

    ```
    export DISPLAY=<parent_machine_hostname:1>
    ```

13. Start the Design Console from inside the container:

    ```
    cd idm/designconsole
    ```

    ```
    sh xlclient.sh
    ```

    The Design Console login should be displayed.

10.5.4 Logging in to the Design Console

To login to the Design Console, perform the following steps:

1. Launch the Design Console and in the Oracle Identity Manager Design Console login page enter the following details and click **Login**:
   - **Server URL**: http(s)://${HOSTNAME}:${PORT}
   - **User ID**: xelsysadm
   - **Password**: <password>
2. If successful the Design Console will be displayed.

Part III

Administering Oracle Identity Governance on Kubernetes

Administer Oracle Identity Governance (OIG) on Kubernetes.

This section contains the following chapters:

11
Scaling OIG Pods

As Oracle Identity Governance (OIG) domains use the WebLogic Kubernetes Operator, domain life cycle operations are managed using the WebLogic Kubernetes Operator itself.

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

For more detailed information refer to Domain Life Cycle in the WebLogic Kubernetes Operator documentation.

This chapter includes the following topics:

11.1 Viewing Existing OIG Instances

The default Oracle Identity Governance (OIG) deployment starts the Administration Server (AdminServer), one OIG Managed Server (oim_server1) and one SOA Managed Server (soa_server1).

The deployment also creates, but doesn't start, four extra OIG Managed Servers (oim-server2 to oim-server5) and four more SOA Managed Servers (soa_server2 to soa_server5).

All these servers are visible in the WebLogic Remote Console by navigating to **Environment** > **Servers**.

Run the following command to view the pods in the OIG deployment:

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

For example:

```
kubectl get pods -n oigns
```

The output should look similar to the following:

```
NAME                                READY   STATUS      RESTARTS   AGE
governancedomain-adminserver        1/1     Running     0          23h
governancedomain-oim-server1        1/1     Running     0          23h
governancedomain-soa-server1        1/1     Running     0          23h
```

11.2 Scaling Up OIG Instances

The number of Oracle Identity Governance (OIG) managed servers running, or SOA managed servers running, is dependent on the replicas parameter configured for the oim_cluster and soa_cluster respectively.

To start more OIG servers perform the following steps:

1. Run the following command to edit the cluster:
   - For OIG managed servers:

     ```
     kubectl edit cluster <domainUID>-oim-cluster -n <domain_namespace>
     ```

     For example:

     ```
     kubectl edit cluster governancedomain-oim-cluster -n oigns
     ```

   - For SOA Managed servers:

     ```
     kubectl edit cluster <domainUID>-soa-cluster -n <domain_namespace>
     ```

     For example:

     ```
     kubectl edit cluster governancedomain-soa-cluster -n oigns
     ```

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

2. In the edit session, search for spec:, and then look for the replicas parameter under clusterName: <cluster>.
   By default the replicas parameter, for both OIG managed servers and SOA managed servers, is set to "1" hence one OIG managed server and one SOA managed server is started (oim_server1 and soa-server1 respectively):
   - For oim_cluster:

     ```
     spec:
       clusterName: oim_cluster
       replicas: 1
       serverPod:
         env:
         - name: USER_MEM_ARGS
     ```

```
        value: -Djava.security.egd=file:/dev/./urandom -Xms8192m -Xmx8192m
    ...
```

- For soa_cluster:

```
spec:
  clusterName: soa_cluster
  replicas: 1
  serverPod:
    env:
    - name: USER_MEM_ARGS
      value: '-Xms8192m -Xmx8192m '
...
```

3. To start more OIG managed servers or SOA managed servers, increase the replicas value as desired.
   In the example below, two more OIG managed servers (oim-server2 and oim-server3) will be started by setting replicas to "3" for the oim_cluster:

```
spec:
  clusterName: oim_cluster
  replicas: 3
  serverPod:
    env:
    - name: USER_MEM_ARGS
      value: -Djava.security.egd=file:/dev/./urandom -Xms8192m -Xmx8192m
...
```

4. Save the file and exit (:wq!).
   The output will look similar to the following:

```
cluster.weblogic.oracle/governancedomain-oim-cluster edited
```

5. Run the following command to view the pods:

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

   For example:

```
kubectl get pods -n oigns
```

   The output will look similar to the following:

```
NAME                                READY   STATUS      RESTARTS   AGE
governancedomain-adminserver        1/1     Running     0          23h
governancedomain-oim-server1        1/1     Running     0          23h
governancedomain-oim-server2        0/1     Running     0          7s
governancedomain-oim-server3        0/1     Running     0          7s
governancedomain-soa-server1        1/1     Running     0          23h
```

   Two new pods (governancedomain-oim-server2 and governancedomain-oim-server3) are started, but currently have a READY status of 0/1. This means oim_server2 and oim_server3 are not currently running but are in the process of starting.

The servers will take several minutes to start so keep executing the command until READY shows 1/1:

> **Note**
>
> Alternatively, you can run kubectl get pods -n oigns -w to watch updates to the status of the pods.

```
NAME                                READY   STATUS      RESTARTS   AGE
governancedomain-adminserver        1/1     Running     0          23h
governancedomain-oim-server1        1/1     Running     0          23h
governancedomain-oim-server2        1/1     Running     0          5m27s
governancedomain-oim-server3        1/1     Running     0          5m27s
governancedomain-soa-server1        1/1     Running     0          23h
```

To check what is happening during server startup when READY is 0/1, run the following command to view the log of the pod that is starting:

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

For example:

```
kubectl logs governancedomain-oim-server2 -n oigns
```

11.3 Scaling Down OIG Instances

Scaling down Oracle Identity Governance (OIG) servers is performed in exactly the same way as in Scaling Up OIG Instances except the replicaCount is reduced to the required number of servers.

To stop one or more OIG servers, perform the following steps:

1. Run the following command to edit the cluster:
   - For OIG managed servers:

     ```
     kubectl edit cluster <domainUID>-oim-cluster -n <domain_namespace>
     ```

     For example:

     ```
     kubectl edit cluster governancedomain-oim-cluster -n oigns
     ```

   - For SOA managed servers:

     ```
     kubectl edit cluster <domainUID>-soa-cluster -n <domain_namespace>
     ```

     For example:

     ```
     kubectl edit cluster governancedomain-soa-cluster -n oigns
     ```

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

2. In the edit session, search for spec:, and then look for the replicas parameter under clusterName: <cluster>.
   In the example below replicas is set to "3", hence three OIG managed servers are started (oim_server1 - oim_server3):

```
...
spec:
  clusterName: oim_cluster
  replicas: 3
  serverPod:
    env:
    - name: USER_MEM_ARGS
      value: -Djava.security.egd=file:/dev/./urandom -Xms8192m -Xmx8192m
...
```

3. To stop OIG servers, decrease the replicas value as desired. In the example below, two managed servers will be stopped by setting replicas to "1":

```
...
spec:
  clusterName: oim_cluster
  replicas: 1
  serverPod:
    env:
    - name: USER_MEM_ARGS
      value: -Djava.security.egd=file:/dev/./urandom -Xms8192m -Xmx8192m
...
```

4. Save the file and exit (:wq!).
   The output will look similar to the following:

```
cluster.weblogic.oracle/governancedomain-oim-cluster edited
```

5. Run the following command to view the pods:

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

   For example:

```
kubectl get pods -n oigns
```

   The output will look similar to the following:

```
NAME                                READY   STATUS      RESTARTS   AGE
governancedomain-adminserver        1/1     Running     0          23h
governancedomain-oim-server1        1/1     Running     0          23h
governancedomain-oim-server2        1/1     Running     0          7m30s
```

```
governancedomain-oim-server3                1/1   Terminating   0          7m30s
governancedomain-soa-server1                1/1   Running       0          23h
```

One pod now has a STATUS of Terminating (governancedomain-oim-server3).

The server will take a minute or two to stop. Once terminated the other pod (governancedomain-oim-server2) will move to Terminating and then stop.

The servers will take several minutes to stop so keep executing the command until the pods have disappeared:

```
NAME                                READY   STATUS    RESTARTS   AGE
governancedomain-adminserver        1/1     Running   0          23h
governancedomain-oim-server1        1/1     Running   0          23h
governancedomain-soa-server1        1/1     Running   0          23h
```

11.4 Stopping and Starting the Domain

Stopping the Domain

Stopping the Oracle Identity Governance (OIG) domain shuts down all the OIG servers and the Administration Server in one operation.

To stop the OIG domain:

1. Run the following kubectl command to edit the domain:

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

   For example:

   ```
   kubectl edit domain governancedomain -n oigns
   ```

2. In the edit session, search for serverStartPolicy: IfNeeded under the domain spec:

   ```
   ...
     volumeMounts:
     - mountPath: /u01/oracle/user_projects
       name: weblogic-domain-storage-volume
     volumes:
     - name: weblogic-domain-storage-volume
       persistentVolumeClaim:
         claimName: governancedomain-domain-pvc
   serverStartPolicy: IfNeeded
   webLogicCredentialsSecret:
     name: governance-domain-credentials
   ...
   ```

3. Change serverStartPolicy: IfNeeded to Never as follows:

   ```
   ...
     volumeMounts:
     - mountPath: /u01/oracle/user_projects
       name: weblogic-domain-storage-volume
   ```

```
  volumes:
  - name: weblogic-domain-storage-volume
    persistentVolumeClaim:
      claimName: governancedomain-domain-pvc
 serverStartPolicy: IfNeeded
 webLogicCredentialsSecret:
   name: governance-domain-credentials
 ...
```

4. Save the file and exit (:wq!).
5. Run the following command to view the pods:

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

   For example:

   ```
   kubectl get pods -n oigns
   ```

   The output will look similar to the following:

   ```
   NAME                                READY   STATUS        RESTARTS   AGE
   governancedomain-adminserver        1/1     Terminating   0          23h
   governancedomain-oim-server1        1/1     Terminating   0          23h
   governancedomain-soa-server1        1/1     Terminating   0          23h
   ```

   The Administration Server pods and OIG server pods will move to a STATUS of Terminating. After a few minutes, run the command again and the pods should have disappeared.
6. To start the Administration Server and Managed Servers up again, repeat the previous steps but change serverStartPolicy: Never to IfNeeded as follows:

   ```
   ...
     volumeMounts:
     - mountPath: /u01/oracle/user_projects
       name: weblogic-domain-storage-volume
     volumes:
     - name: weblogic-domain-storage-volume
       persistentVolumeClaim:
         claimName: governancedomain-domain-pvc
   serverStartPolicy: Never
   webLogicCredentialsSecret:
     name: governance-domain-credentials
   ...
   ```

7. Run the following command to view the pods:

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

   For example:

   ```
   kubectl get pods -n oigns
   ```

The output will look similar to the following:

```
NAME                                READY   STATUS     RESTARTS   AGE
goverancedomain-introspector-jwqxw          1/1     Running    0          10s
```

The introspect job will start, followed by the Administration Server pod, and then the OIG server pods. This process will take several minutes, so keep executing the command until all the pods are running with READY status 1/1:

> **Note**
>
> Alternatively, you can run `kubectl get pods -n oigns -w` to watch updates to the status of the pods.

```
NAME                                READY   STATUS     RESTARTS   AGE
governancedomain-adminserver                1/1     Running    0          6m57s
governancedomain-oim-server1                1/1     Running    0          4m33s
governancedomain-soa-server1                1/1     Running    0          4m33s
```

Starting the Domain

Starting the Oracle Identity Governance (OIG) domain starts all the OIG servers and the Administration Server in one operation.

To start the OIG domain:

1. Run the following kubectl command to edit the domain:

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

   For example:

   ```
   kubectl edit domain governancedomain -n oigns
   ```

2. In the edit session, search for `serverStartPolicy: Never` under the domain spec:

   ```
   ...
     volumeMounts:
     - mountPath: /u01/oracle/user_projects
       name: weblogic-domain-storage-volume
     volumes:
     - name: weblogic-domain-storage-volume
       persistentVolumeClaim:
         claimName: governancedomain-domain-pvc
   serverStartPolicy: Never
   webLogicCredentialsSecret:
     name: governance-domain-credentials
   ...
   ```

3. Change `serverStartPolicy: Never` to `IfNeeded` as follows:

```
...
  volumeMounts:
  - mountPath: /u01/oracle/user_projects
    name: weblogic-domain-storage-volume
  volumes:
  - name: weblogic-domain-storage-volume
    persistentVolumeClaim:
      claimName: governancedomain-domain-pvc
 serverStartPolicy: IfNeeded
 webLogicCredentialsSecret:
  name: governance-domain-credentials
 ...
```

4. Save the file and exit (`:wq!`).
5. Run the following command to view the pods:

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

For example:

```
kubectl get pods -n oigns
```

The output will look similar to the following:

```
NAME                                 READY   STATUS     RESTARTS   AGE
goverancedomain-introspector-jwqxw             1/1     Running    0          10s
```

The introspect job will start, followed by the Administration Server pod, and then the OIG server pods. This process will take several minutes, so keep executing the command until all the pods are running with `READY` status `1/1`:

> **Note**
>
> Alternatively, you can run `kubectl get pods -n oigns -w` to watch updates to the status of the pods.

```
NAME                                 READY   STATUS     RESTARTS   AGE
governancedomain-adminserver                   1/1     Running    0          6m57s
governancedomain-oim-server1                   1/1     Running    0          4m33s
governancedomain-soa-server1                   1/1     Running    0          4m33s
```

11.5 Domain Life Cycle Scripts

The WebLogic Kubernetes Operator provides sample scripts to start up or shut down a specific Managed Server or cluster in a deployed domain, or the entire deployed domain.

> **Note**
>
> Prior to running these scripts, you must have previously created and deployed the domain.

The scripts are located in the `$WORKDIR/kubernetes/domain-lifecycle` directory.

For more information, see Sample Lifecycle Management Scripts.

12

WLST Administration Operations

This chapter contains the following topics:

12.1 Connecting to OIG via WLST

In order to use WLST to administer the Oracle Identity Governance (OIG) domain, use must use a helper pod.

1. Check to see if the helper pod exists by running:

   ```
   kubectl get pods -n <domain_namespace> | grep helper
   ```

   For example:

   ```
   kubectl get pods -n oigns | grep helper
   ```

   The output should look similar to the following:

   ```
   helper                      1/1     Running   0          26h
   ```

   If the helper pod doesn't exist then run the following:

   - If using Oracle Container Registry or your own container registry for the OIG container image:

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

     For example:

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

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

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

  For example:

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

  The output will look similar to the following:

  ```
  pod/helper created
  ```

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

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

   For example:

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

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

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

3. Inside the helper pod, connect to WLST using the following command:

   ```
   cd $ORACLE_HOME/oracle_common/common/bin
   ```

   ```
   ./wlst.sh
   ```

   The output will look similar to the following:

   ```
   Initializing WebLogic Scripting Tool (WLST) ...

   Jython scans all the jar files it can find at first startup. Depending on the system, this process may take a few minutes to complete, and WLST may not return a prompt right away.

   Welcome to WebLogic Server Administration Scripting Shell

   Type help() for help on available commands

   wls:/offline>
   ```

4. To access t3 for the Administration Server connect as follows:

   ```
   connect('weblogic','<password>','t3://governancedomain-adminserver:7001')
   ```

The output will look similar to the following:

```
Connecting to t3://governancedomain-adminserver:7001 with userid weblogic ...
Successfully connected to Admin Server "AdminServer" that belongs to domain "governancedomain".

Warning: An insecure protocol was used to connect to the server.
To ensure on-the-wire security, the SSL port or Admin port should be used instead.

wls:/governancedomain/serverConfig/>
```

Or to access t3 for the OIG Cluster service, connect as follows:

```
connect('weblogic','<password>','t3://governancedomain-cluster-oim-cluster:14000')
```

The output will look similar to the following:

```
Connecting to t3://governancedomain-cluster-oim-cluster:14000 with userid weblogic ...
Successfully connected to managed Server "oim_server1" that belongs to domain "governancedomain".

Warning: An insecure protocol was used to connect to the server.
To ensure on-the-wire security, the SSL port or Admin port should be used instead.

wls:/governancedomain/serverConfig/>
```

12.2 Sample WLST Operations

The following are some sample WLST operations that can be performed against the Oracle Identity Governance(OIG) domain.

For a full list of WLST operations, see WebLogic Server WLST Online and Offline Command Reference.

Display Servers

1. Run the following commands to display the server:

```
wls:/governancedomain/serverConfig/> cd('/Servers')



wls:/governancedomain/serverConfig/Servers> ls()
```

The output will look similar to the following:

```
dr--   AdminServer
dr--   oim_server1
dr--   oim_server2
dr--   oim_server3
dr--   oim_server4
dr--   oim_server5
dr--   soa_server1
dr--   soa_server2
dr--   soa_server3
```

```
dr--   soa_server4
dr--   soa_server5

wls:/governancedomain/serverConfig/Servers>
```

12.3 Performing WLST Administration via SSL

The following steps show how to perform WLST administration via SSL:

1. By default the SSL port is not enabled for the Administration Server or Oracle Identity Governance (OIG) managed servers. To configure the SSL port for the Administration Server and Managed Servers:
   a. Login to WebLogic Remote Console.
   b. Click **Edit Tree** and in the left-hand navigation menu, navigate to **Environment** > **Servers** > <server_name> and click on the **General** tab.
   c. Check the SSL Listen Port Enabled button and provide the SSL Port (For AdminServer: 7002 and for oim_server1): 14101
   d. Click **Save**.
   e. Click the **Shopping Cart** and select **Commit Changes**.

   > **Note**
   >
   > If configuring the OIG managed servers for SSL you must enable SSL on the same port for all servers (oim_server1 through oim_server5).

2. Create a myscripts directory as follows:

```
cd $WORKDIR/kubernetes


mkdir myscripts


cd myscripts
```

3. Create a sample yaml template file in the myscripts directory called <domain_uid>-adminserver-ssl.yaml to create a Kubernetes service for the Administration Server:

   > **Note**
   >
   > Update the domainName, domainUID and namespace based on your environment. For example:

```
apiVersion: v1
kind: Service
metadata:
  labels:
    serviceType: SERVER
    weblogic.domainName: governancedomain
```

```
    weblogic.domainUID: governancedomain
    weblogic.resourceVersion: domain-v2
    weblogic.serverName: AdminServer
  name: governancedomain-adminserver-ssl
  namespace: oigns
spec:
  clusterIP: None
  ports:
  - name: default
    port: 7002
    protocol: TCP
    targetPort: 7002
  selector:
    weblogic.createdByOperator: "true"
    weblogic.domainUID: governancedomain
    weblogic.serverName: AdminServer
  type: ClusterIP
```

4. Create a <domain_uid>-oimcluster-ssl.yaml for the OIG managed server:

```
apiVersion: v1
kind: Service
metadata:
  labels:
    serviceType: SERVER
    weblogic.domainName: governancedomain
    weblogic.domainUID: governancedomain
    weblogic.resourceVersion: domain-v2
  name: governancedomain-cluster-oim-cluster-ssl
  namespace: oigns
spec:
  clusterIP: None
  ports:
  - name: default
    port: 14101
    protocol: TCP
    targetPort: 14101
  selector:
    weblogic.clusterName: oim_cluster
    weblogic.createdByOperator: "true"
    weblogic.domainUID: governancedomain
  type: ClusterIP
```

5. Apply the template using the following command for the AdminServer:

```
kubectl apply -f <domain_uid>-adminserver-ssl.yaml
```

For example:

```
kubectl apply -f governancedomain-adminserver-ssl.yaml
```

The output will look similar to the following:

```
service/governancedomain-adminserverssl created
```

6. Apply the template using the following command for the OIG managed server:

   ```
   kubectl apply -f governancedomain-oim-cluster-ssl.yaml
   ```

   For example:

   ```
   kubectl apply -f governancedomain-oimcluster-ssl.yaml
   ```

   The output will look similar to the following:

   ```
   service/governancedomain-oimcluster-ssl created
   ```

7. Validate that the Kubernetes services to access SSL ports are created successfully:

   ```
   kubectl get svc -n <domain_namespace> |grep ssl
   ```

   For example:

   ```
   kubectl get svc -n oigns |grep ssl
   ```

   The output will look similar to the following:

   ```
   governancedomain-adminserver-ssl          ClusterIP   None          <none>       7002/TCP                 74s
   governancedomain-cluster-oim-cluster-ssl  ClusterIP   None          <none>       14101/TCP                21s
   ```

8. Inside the bash shell of the running helper pod, run the following:

   ```
   export WLST_PROPERTIES="-Dweblogic.security.SSL.ignoreHostnameVerification=true -
   Dweblogic.security.TrustKeyStore=DemoTrust"


   cd /u01/oracle/oracle_common/common/bin


   ./wlst.sh
   ```

   The output will look similar to the following:

   ```
   Initializing WebLogic Scripting Tool (WLST) ...

   Welcome to WebLogic Server Administration Scripting Shell

   Type help() for help on available commands
   wls:/offline>
   ```

   To connect to the Administration Server t3s service:

   ```
   connect('weblogic','<password>','t3s://governancedomain-adminserver-ssl:7002')
   ```

The output will look similar to the following:

```
Connecting to t3s://governancedomain-adminserver-ssl:7002 with userid weblogic ...
<DATE> <Info> <Security> <BEA-090905> <Disabling the CryptoJ JCE Provider self-integrity check for
better startup performance. To enable this check, specify -
Dweblogic.security.allowCryptoJDefaultJCEVerification=true.>
<DATE> <Info> <Security> <BEA-090906> <Changing the default Random Number Generator in RSA
CryptoJ from ECDRBG128 to HMACDRBG. To disable this change, specify -
Dweblogic.security.allowCryptoJDefaultPRNG=true.>
<DATE> <Info> <Security> <BEA-090909> <Using the configured custom SSL Hostname Verifier
implementation: weblogic.security.utils.SSLWLSHostnameVerifier$NullHostnameVerifier.>
Successfully connected to Admin Server "AdminServer" that belongs to domain "governancedomain".

wls:/governancedomain/serverConfig/>
```

To connect to the OIG Managed Server t3s service:

```
connect('weblogic','<password>','t3s://governancedomain-cluster-oim-cluster-ssl:14101')
```

The output will look similar to the following:

```
Connecting to t3s://governancedomain-cluster-oim-cluster-ssl:14101 with userid weblogic ...
<DATE> <Info> <Security> <BEA-090905> <Disabling the CryptoJ JCE Provider self-integrity check for
better startup performance. To enable this check, specify -
Dweblogic.security.allowCryptoJDefaultJCEVerification=true.>
<DATE> <Info> <Security> <BEA-090906> <Changing the default Random Number Generator in RSA
CryptoJ from ECDRBG128 to HMACDRBG. To disable this change, specify -
Dweblogic.security.allowCryptoJDefaultPRNG=true.>
<DATE> <Info> <Security> <BEA-090909> <Using the configured custom SSL Hostname Verifier
implementation: weblogic.security.utils.SSLWLSHostnameVerifier$NullHostnameVerifier.>
Successfully connected to managed Server "oim_server1" that belongs to domain "governancedomain".

wls:/governancedomain/serverConfig/>
```

13

Logging and Visualization

This chapter describes how to publish WebLogic Kubernetes Operator and WebLogic server logs into Elasticsearch, and interact with them in Kibana.

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

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

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

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

This chapter includes the following topics:

13.1 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.

13.2 Creating the Logstash Pod

Topics in the section include:

13.2.1 Variables Used in This Section

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

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

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

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

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

13.2.2 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 oigns --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 oigns --from-literal
password=<ELK_PASSWORD>
```

The output will look similar to the following:

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

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

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

> **Note**
>
> Before executing the command below, you must first have a user account on hub.docker.com.

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

For example:

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

The output will look similar to the following:

```
secret/dockercred created
```

13.2.3 Finding Required Domain Details

The YAML files for ELK require certain domain values to be added.

1. Run the following command to get the mountPath of your domain:

```
kubectl describe domains <domain_uid> -n <domain_namespace> | grep "Mount Path"
```

For example:

```
kubectl describe domains governancedomain -n oigns | grep "Mount Path"
```

If you deployed OIG using WLST, the output will look similar to the following:

```
Mount Path: /u01/oracle/user_projects/domains
```

If you deployed OIG using WDT, the output will look similar to the following:

```
Mount Path: /u01/oracle/user_projects
```

2. Run the following command to get the Domain Home and Log Home of your domain:

```
kubectl describe domains <domain_uid> -n <domain_namespace> | egrep "Domain Home: | Log Home:"
```

For example:

```
kubectl describe domains governancedomain -n oigns | egrep "Domain Home: | Log Home:"
```

The output will look similar to the following:

```
Domain Home:                /u01/oracle/user_projects/domains/governancedomain
Http Access Log In Log Home:    true
Log Home:                   /u01/oracle/user_projects/domains/logs/governancedomain
```

3. Run the following command to get the OIG domain persistence volume details:

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

For example:

```
kubectl get pv -n oigns
```

The output will look similar to the following:

```
NAME                    CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM                       STORAGECLASS                REASON   AGE
governancedomain-domain-pv  10Gi       RWX            Retain           Bound    oigns/governancedomain-domain-
pvc   governancedomain-oim-storage-class          28h
```

Make note of the CLAIM value. In the example above the value is governancedomain-domain-pvc.

13.2.4 Creating the Configmap

Perform the following steps to create the Kubernetes ConfigMap for ELK:

1. Copy the elk.crt file to the $WORKDIR/kubernetes/elasticsearch-and-kibana directory.

2. Navigate to the $WORKDIR/kubernetes/elasticsearch-and-kibana directory and run the following:

   ```
   kubectl create configmap elk-cert --from-file=elk.crt -n <namespace>
   ```

   For example:

   ```
   kubectl create configmap elk-cert --from-file=elk.crt -n oigns
   ```

   The output will look similar to the following:

   ```
   configmap/elk-cert created
   ```

3. Create a logstash_cm.yaml file in the $WORKDIR/kubernetes/elasticsearch-and-kibana directory as follows:

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: oig-logstash-configmap
     namespace: <ELKNS>
   data:
     logstash.yml: |
       #http.host: "0.0.0.0"
     logstash-config.conf: |
       input {
         file {
           path => "<Log Home>/servers/AdminServer/logs/AdminServer*.log*"
           tags => "Adminserver_log"
           start_position => beginning
         }
         file {
           path => "<Log Home>/**/logs/soa_server*.log*"
           tags => "soaserver_log"
           start_position => beginning
         }
         file {
           path => "<Log Home>/**/logs/oim_server*.log*"
           tags => "Oimserver_log"
           start_position => beginning
         }
         file {
           path => "<Domain Home>/servers/AdminServer/logs/AdminServer-diagnostic.log*"
           tags => "Adminserver_diagnostic"
           start_position => beginning
         }
         file {
           path => "<Domain Home>/servers/**/logs/soa_server*-diagnostic.log*"
           tags => "Soa_diagnostic"
           start_position => beginning
         }
         file {
           path => "<Domain Home>/servers/**/logs/oim_server*-diagnostic.log*"
           tags => "Oimserver_diagnostic"
           start_position => beginning
   ```

```
    }
    file {
      path => "<Domain Home>/servers/**/logs/access*.log*"
      tags => "Access_logs"
      start_position => beginning
    }
  }
  filter {
    grok {
      match => [ "message", "<%{DATA:log_timestamp}> <%{WORD:log_level}> <%{WORD:thread}> <%
{HOSTNAME:hostname}> <%{HOSTNAME:servername}> <%{DATA:timer}> <<%{DATA:kernel}>> <>
<%{DATA:uuid}> <%{NUMBER:timestamp}> <%{DATA:misc}    > <%{DATA:log_number}> <%
{DATA:log_message}>" ]
    }
  if "_grokparsefailure" in [tags] {
     mutate {
        remove_tag => [ "_grokparsefailure" ]
     }
  }
  }
  output {
    elasticsearch {
  hosts => ["<ELK_HOSTS>"]
  cacert => '/usr/share/logstash/config/certs/elk.crt'
  index => "oiglogs-000001"
  ssl => <ELK_SSL>
  ssl_certificate_verification => false
  user => "<ELK_USER>"
  password => "${ELASTICSEARCH_PASSWORD}"
  api_key => "${ELASTICSEARCH_PASSWORD}"
    }
  }
```

Change the values in the above file as follows:

- Change the <ELKNS>, <ELK_HOSTS>, <ELK_SSL>, and <ELK_USER> to match the values in Variables Used in This Section.
- Change <Log Home> and <Domain Home> to match the Log Home and Domain Home returned in Finding Required Domain Details.
- If using API KEY for your ELK authentication, delete the user and password lines.
- If using a password for ELK authentication, delete the api_key line.
- If no authentication is used for ELK, delete the user, password, and api_key lines.

For example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: oig-logstash-configmap
  namespace: oigns
data:
  logstash.yml: |
    #http.host: "0.0.0.0"
  logstash-config.conf: |
```

```
  input {
    file {
      path => "/u01/oracle/user_projects/domains/logs/governancedomain/servers/AdminServer/logs/
AdminServer*.log*"
      tags => "Adminserver_log"
      start_position => beginning
    }
    file {
      path => "/u01/oracle/user_projects/domains/logs/governancedomain/**/logs/soa_server*.log*"
      tags => "soaserver_log"
      start_position => beginning
    }
    file {
      path => "/u01/oracle/user_projects/domains/logs/governancedomain/**/logs/oim_server*.log*"
      tags => "Oimserver_log"
      start_position => beginning
    }
    file {
      path => "/u01/oracle/user_projects/domains/governancedomain/servers/AdminServer/logs/AdminServer-
diagnostic.log*"
      tags => "Adminserver_diagnostic"
      start_position => beginning
    }
    file {
      path => "/u01/oracle/user_projects/domains/governancedomain/servers/**/logs/soa_server*-
diagnostic.log*"
      tags => "Soa_diagnostic"
      start_position => beginning
    }
    file {
      path => "/u01/oracle/user_projects/domains/governancedomain/servers/**/logs/oim_server*-
diagnostic.log*"
      tags => "Oimserver_diagnostic"
      start_position => beginning
    }
    file {
      path => "/u01/oracle/user_projects/domains/governancedomain/servers/**/logs/access*.log*"
      tags => "Access_logs"
      start_position => beginning
    }
  }
  filter {
    grok {
      match => [ "message", "<%{DATA:log_timestamp}> <%{WORD:log_level}> <%{WORD:thread}> <%
{HOSTNAME:hostname}> <%{HOSTNAME:servername}> <%{DATA:timer}> <<%{DATA:kernel}>> <>
<%{DATA:uuid}> <%{NUMBER:timestamp}> <%{DATA:misc}    > <%{DATA:log_number}> <%
{DATA:log_message}>" ]
    }
  if "_grokparsefailure" in [tags] {
      mutate {
         remove_tag => [ "_grokparsefailure" ]
      }
  }
  }
  output {
    elasticsearch {
```

```
hosts => ["https://elasticsearch.example.com:9200"]
cacert => '/usr/share/logstash/config/certs/elk.crt'
index => "oiglogs-000001"
ssl => true
ssl_certificate_verification => false
user => "logstash_internal"
password => "${ELASTICSEARCH_PASSWORD}"
  }
}
```

4. Run the following command to create the ConfigMap:

```
kubectl apply -f logstash_cm.yaml
```

The output will look similar to the following:

```
configmap/oig-logstash-configmap created
```

13.2.5 Enabling Logstash

Perform the following steps to enable logstash:

1. Navigate to the $WORKDIR/kubernetes/elasticsearch-and-kibana directory and create a logstash.yaml file as follows:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: oig-logstash
  namespace: <ELKNS>
spec:
  selector:
    matchLabels:
      k8s-app: logstash
  template: # create pods using pod definition in this template
    metadata:
      labels:
        k8s-app: logstash
    spec:
      imagePullSecrets:
      - name: dockercred
      containers:
      - command:
        - logstash
        image: logstash:<ELK_VER>
        imagePullPolicy: IfNotPresent
        name: oig-logstash
        env:
        - name: ELASTICSEARCH_PASSWORD
          valueFrom:
            secretKeyRef:
              name: elasticsearch-pw-elastic
              key: password
        resources:
        ports:
```

```
  - containerPort: 5044
    name: logstash
  volumeMounts:
  - mountPath: <mountPath>
    name: weblogic-domain-storage-volume
  - name: shared-logs
    mountPath: /shared-logs
  - mountPath: /usr/share/logstash/pipeline/
    name: oig-logstash-pipeline
  - mountPath: /usr/share/logstash/config/logstash.yml
    subPath: logstash.yml
    name: config-volume
  - mountPath: /usr/share/logstash/config/certs
    name: elk-cert
volumes:
- configMap:
    defaultMode: 420
    items:
    - key: elk.crt
      path: elk.crt
    name: elk-cert
  name: elk-cert
- configMap:
    defaultMode: 420
    items:
    - key: logstash-config.conf
      path: logstash-config.conf
    name: oig-logstash-configmap
  name: oig-logstash-pipeline
- configMap:
    defaultMode: 420
    items:
    - key: logstash.yml
      path: logstash.yml
    name: oig-logstash-configmap
  name: config-volume
- name: weblogic-domain-storage-volume
  persistentVolumeClaim:
    claimName: governancedomain-domain-pvc
- name: shared-logs
  emptyDir: {}
```

- Change the *<ELK_VER>*, *<ELK_SSL>* to match the values for your environment.
- Change <mountPath> to match the mountPath returned in Finding Required Domain Details.
- Change the claimName value to match the claimName returned earlier
- If your Kubernetes environment does not allow access to the internet to pull the logstash image, you must load the logstash image in your own container registry and change image: logstash:<ELK_VER> to the location of the image in your container registry, for example container-registry.example.com/logstash:8.3.1

For example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: oig-logstash
  namespace: oigns
spec:
  selector:
    matchLabels:
      k8s-app: logstash
  template: # create pods using pod definition in this template
    metadata:
      labels:
        k8s-app: logstash
    spec:
      imagePullSecrets:
      - name: dockercred
      containers:
      - command:
        - logstash
        image: logstash:8.3.1
        imagePullPolicy: IfNotPresent
        name: oig-logstash
        env:
        - name: ELASTICSEARCH_PASSWORD
          valueFrom:
            secretKeyRef:
              name: elasticsearch-pw-elastic
              key: password
        resources:
        ports:
        - containerPort: 5044
          name: logstash
        volumeMounts:
        - mountPath: /u01/oracle/user_projects
          name: weblogic-domain-storage-volume
        - name: shared-logs
          mountPath: /shared-logs
        - mountPath: /usr/share/logstash/pipeline/
          name: oig-logstash-pipeline
        - mountPath: /usr/share/logstash/config/logstash.yml
          subPath: logstash.yml
          name: config-volume
        - mountPath: /usr/share/logstash/config/certs
          name: elk-cert
      volumes:
      - configMap:
          defaultMode: 420
          items:
          - key: elk.crt
            path: elk.crt
          name: elk-cert
        name: elk-cert
      - configMap:
          defaultMode: 420
```

```
    items:
    - key: logstash-config.conf
      path: logstash-config.conf
    name: oig-logstash-configmap
  name: oig-logstash-pipeline
- configMap:
    defaultMode: 420
    items:
    - key: logstash.yml
      path: logstash.yml
    name: oig-logstash-configmap
  name: config-volume
- name: weblogic-domain-storage-volume
  persistentVolumeClaim:
    claimName: governancedomain-domain-pvc
- name: shared-logs
  emptyDir: {}
```

2. Deploy the logstash pod by executing the following command:

```
kubectl create -f $WORKDIR/kubernetes/elasticsearch-and-kibana/logstash.yaml
```

The output will look similar to the following:

```
deployment.apps/oig-logstash created
```

13.3 Verifying the Pods

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

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

For example:

```
kubectl get pods -n oigns
```

The output should look similar to the following:

```
NAME                                READY   STATUS    RESTARTS   AGE
governancedomain-adminserver        1/1     Running   0          90m
governancedomain-oim-server1        1/1     Running   0          88m
governancedomain-soa-server1        1/1     Running   0          88m
oig-logstash-77fbbc66f8-lsvcw       1/1     Running   0          3m25s
```

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

```
kubectl logs -f oig-logstash-<pod> -n oigns
```

Most errors occur due to misconfiguration of the logstash_cm.yaml or logstash.yaml. This is usually because of an incorrect value set, or the certificate was not pasted with the correct indentation.
If the pod has errors, delete the pod and ConfigMap as follows:

```
kubectl delete -f $WORKDIR/kubernetes/elasticsearch-and-kibana/logstash.yaml
```

```
kubectl delete -f $WORKDIR/kubernetes/elasticsearch-and-kibana/logstash_cm.yaml
```

Once you have resolved the issue in the yaml files, run the commands outlined earlier to recreate the ConfigMap and logstash pod.

13.4 Verifying and Accessing the Kibana Console

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

Kibana Version 7.8.X or Higher

1. Access the Kibana console with http://<hostname>:<port>/app/kibana and login with your username and password.
2. From the Navigation menu, navigate to **Management** > **Kibana** > **Index Patterns**.
3. In the **Create Index Pattern** page enter oiglogs* 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 OIG 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: oiglogs*
   - 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 oiglogs* to view the log file entries.

14

Monitoring an Oracle Identity Governance Domain

Using the WebLogic Monitoring Exporter you can scrape runtime information from a running Oracle Identity Governance (OIG) domain and monitor using Prometheus and Grafana.

To set up monitoring, see Monitor the Oracle Identity Governance Instance Using Prometheus and Grafana.

For more information on WebLogic Monitoring Exporter, see WebLogic Monitoring Exporter.

15

Kubernetes Horizontal Pod Autoscaler

Kubernetes Horizontal Pod Autoscaler (HPA) allows automatic scaling (up and down) of the Oracle Identity Governance (OIG) servers. If load increases then extra OIG servers will be started as required, up to the value configuredManagedServerCount defined when the domain was created. Similarly, if load decreases, OIG servers will be automatically shutdown.

For more information on HPA, see Horizontal Pod Autoscaling.

The instructions below show you how to configure and run an HPA to scale an OIG cluster (governancedomain-oim-cluster), based on CPU utilization or memory resource metrics. If required, you can also perform the following for the governancedomain-soa-cluster.

> **Note**
>
> If you enable HPA and then decide you want to start,stop, or scale OIG servers manually as per Scaling OIG Pods, it is recommended to delete HPA beforehand as per Deleting HPA.

This chapter includes the following topics:

15.1 Prerequisite Configurations

In order to use HPA, Oracle Identity Governance (OIG) must have been created with the required resources parameter. For OIG domains created with WLST scripts, this is as per Setting the OIG Server Memory Parameters. For OIG domains created with WDT models, the values should be set by default. For example:

For example:

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

```
requests:
  cpu: 1000m
  memory: 4Gi
```

If you created the OIG domain without setting these parameters, then you can update the domain using the following steps:

1. Run the following command to edit the cluster:

   ```
   kubectl edit cluster <cluster> -n <namespace>
   ```

   For example:

   ```
   kubectl edit cluster governancedomain-oim-cluster -n oigns
   ```

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

2. In the edit session, search for spec:, and then look for the replicas parameter under `clusterName: oim_cluster`. Change the entry so it looks as follows:

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

3. Save the file and exit (`:wq!`).
   The output will look similar to the following:

   ```
   cluster.weblogic.oracle/governancedomain-oim-cluster edited
   ```

   The OIG managed server pods will then automatically be restarted.

15.2 Deploying the Kubernetes Metrics Server

Before deploying HPA you must deploy the Kubernetes Metrics Server.

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

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

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

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

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

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

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

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

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

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

   The output will look similar to the following:

   ```
   serviceaccount/metrics-server created
   clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created
   clusterrole.rbac.authorization.k8s.io/system:metrics-server created
   rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created
   clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator created
   clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created
   service/metrics-server created
   deployment.apps/metrics-server created
   apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io created
   ```

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

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

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

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

15.3 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
```

15.4 Deploying HPA

The steps below show how to configure and run an HPA to scale Oracle Identity Governance (OIG), based on the CPU or memory utilization resource metrics.

The default OIG deployment creates the cluster governancedomain-oim-cluster which starts one OIG managed server (oim_server1). The deployment also creates, but doesn't start, four extra OIG Managed Servers (oig-server2 to oig-server5).

In the following example an HPA resource is created, cluster resource governancedomain-oim-cluster. This resource will autoscale OIG managed server from a minimum of 1 cluster member up to 5 cluster members. Scaling up will occur when the average CPU is consistently over 70%. Scaling down will occur when the average CPU is consistently below 70%.

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

```
#
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: governancedomain-oim-cluster-hpa
  namespace: oigns
spec:
  scaleTargetRef:
    apiVersion: weblogic.oracle/v1
    kind: Cluster
    name: governancedomain-oim-cluster
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 60
    scaleUp:
      stabilizationWindowSeconds: 60
  minReplicas: 1
  maxReplicas: 5
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
```

where:

- governancedomain is the <domainUID>
- oigns is the <domain_namespace>.
- minReplicas and maxReplicas should match your current domain setting.

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

2. Run the following command to create the autoscaler:

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

   The output will look similar to the following:

   ```
   horizontalpodautoscaler.autoscaling/governancedomain-oim-cluster-hpa created
   ```

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

   ```
   kubectl get hpa -n oigns
   ```

   The output will look similar to the following:

   ```
   NAME                        REFERENCE                          TARGETS   MINPODS   MAXPODS
   REPLICAS   AGE
   governancedomain-oim-cluster-hpa   Cluster/governancedomain-oim-cluster   16%/70%   1         5         1
   20s
   ```

   In the example above, this shows that CPU is currently running at 16% for the governancedomain-oim-cluster-hpa.

15.5 Verifying HPA

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

1. Check the current status of the Oracle Identity Governance (OIG) servers:

   ```
   kubectl get pods -n oigns
   ```

   The output will look similar to the following:

   ```
   NAME                                READY   STATUS      RESTARTS   AGE
   governancedomain-adminserver                1/1     Running     0          20m
   governancedomain-oim-server1                1/1     Running     0          17m
   governancedomain-soa-server1                1/1     Running     0          17m
   ```

In the above example only governancedomain-oim-server1 is running.

2. To test HPA can scale up the WebLogic cluster governancedomain-oim-cluster, run the following commands:

```
kubectl exec --stdin --tty governancedomain-oim-server1 -n oigns -- /bin/bash
```

This will take you inside a bash shell inside the oim_server1 pod:

```
[oracle@governancedomain-oim-server1 oracle]$
```

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

```
[oracle@governancedomain-oim-server1 oracle]$ dd if=/dev/zero of=/dev/null
```

This command will continue to run in the foreground.

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

```
kubectl get hpa -n oigns
```

The output will look similar to the following:

```
NAME                          REFERENCE                          TARGETS    MINPODS   MAXPODS
REPLICAS   AGE
governancedomain-oim-cluster-hpa   Cluster/governancedomain-oim-cluster   386%/70%   1         5         1
2m47s
```

In the above example the CPU has increased to 386%. As this is above the 70% limit, the autoscaler increases the replicas on the Cluster resource, and the operator responds by starting additional cluster members.

5. Run the following to see if any more OIG Managed Servers are started:

```
kubectl get pods -n oigns
```

The output will look similar to the following:

```
NAME                                READY   STATUS      RESTARTS      AGE
governancedomain-adminserver        1/1     Running     0             30m
governancedomain-oim-server1        1/1     Running     0             27m
governancedomain-oim-server2        1/1     Running     0             10m
governancedomain-oim-server3        1/1     Running     0             10m
governancedomain-oim-server4        1/1     Running     0             10m
governancedomain-oim-server5        1/1     Running     0             10m
governancedomain-soa-server1        1/1     Running     0             27m
```

In the example above four more OIG managed servers have been started (oig-server2 - oig-server5).

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

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

```
[oracle@governancedomain-oim-server1 oracle]$ dd if=/dev/zero of=/dev/null
^C
[oracle@governancedomain-oim-server1 oracle]$ exit
```

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

```
kubectl get hpa -n oigns
```

The output will look similar to the following:

```
NAME                        REFERENCE                          TARGETS   MINPODS   MAXPODS
REPLICAS   AGE
governancedomain-oim-cluster-hpa   Cluster/governancedomain-oim-cluster   33%/70%   1         5         5
37m
```

In the above example CPU has dropped to 33%. As this is below the 70% threshold, you should see the autoscaler scale down the servers:

```
kubectl get pods -n oigns
```

The output will look similar to the following:

```
NAME                                READY   STATUS        RESTARTS      AGE
governancedomain-adminserver        1/1     Running       0             43m
governancedomain-oim-server1        1/1     Running       0             40m
governancedomain-oim-server2        1/1     Running       0             13m
governancedomain-oim-server3        1/1     Running       0             13m
governancedomain-oim-server4        1/1     Running       0             13m
governancedomain-oim-server5        0/1     Terminating   0             13m
governancedomain-soa-server1        1/1     Running       0             40m
```

Eventually, all the servers except oim-server1 will disappear:

```
NAME                                READY   STATUS        RESTARTS      AGE
governancedomain-adminserver        1/1     Running       0             44m
governancedomain-oim-server1        1/1     Running       0             41m
governancedomain-soa-server1        1/1     Running       0             41m
```

15.6 Deleting HPA

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

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

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

The output will look similar to the following:

```
horizontalpodautoscaler.autoscaling "governancedomain-oim-cluster-hpa" deleted
```

15.7 Other Considerations for HPA

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

- If HPA is deployed and you need to upgrade the Oracle Identity Governance (OIG) container image, then you must delete the HPA before upgrading. To delete the HPA, see Deleting HPA. Once the upgrade is successful you can deploy HPA again.
- If you choose to start/stop an OIG managed server manually as per Scaling OIG Pods, then it is recommended to delete the HPA before doing so.

16

Patching and Upgrading

This chapter includes the following topics:

- Patching and Upgrading Within 14.1.2
- Upgrading from Oracle Identity Governance 12.2.1.4 to 14.1.2

16.1 Patching and Upgrading Within 14.1.2

Learn how to patch or upgrade the Oracle Identity Governance (OIG) image used by an OIG 14.1.2 container.

This section contains the following topics:

- Patching a Container Image
- Upgrading WebLogic Kubernetes Operator

16.1.1 Patching a Container Image

The instructions in this section relate to patching or upgrading an existing 14.1.2.1.0 Oracle Identity Governance (OIG) deployment with a new OIG container image.

> **Note**
>
> Administrators should be aware of the following:
>
> - If you are not using Oracle Container Registry or your own container registry, then you must first load the new container image on all nodes in your Kubernetes cluster.
> - If you have Kubernetes Horizontal Pod Autoscaler (HPA) enabled, you must disable HPA before performing the steps below. See, Deleting HPA.

> **Note**
>
> The instructions in this section relate to patching an OIG container image to the July 25 container image or later. If you are patching to a version earlier than July 25, you only need to follow Patching the Container Image and Verifying the OIG Deployment is Using the New Image .

Patching the container image involves the following:

- Stopping the Domain
- Patching the Database Schemas
- Patching the Container Image

- Starting the OIG Domain
- Verifying the OIG Deployment is Using the New Image

Stopping the Domain

Before patching you must shutdown the OIG Domain. For details on how to do this, see Stopping and Starting the Domain.

Patching the Database Schemas

Once the image has been updated you must patch the schemas in the database.

1. Check to see if the helper pod exists by running:

   ```
   kubectl get pods -n <domain_namespace> | grep helper
   ```

   For example:

   ```
   kubectl get pods -n oigns | grep helper
   ```

   The output should look similar to the following:

   ```
   helper                       1/1     Running   0          26h
   ```

   If the helper pod exists delete the pod with following command:

   ```
   kubectl delete pod helper -n <namespace>
   ```

   For example:

   ```
   kubectl delete pod helper -n oigns
   ```

2. Create a new helper pod using the new image.
   If using Oracle Container Registry or your own container registry for the Oracle Identity Governance (OIG) container image:

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

   For example:

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

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

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

For example:

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

The output will look similar to the following:

```
pod/helper created
```

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

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

For example:

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

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

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

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

```
export DB_HOST=<db_host.domain>

export DB_PORT=<db_port>

export DB_SERVICE=<service_name>

export RCUPREFIX=<rcu_schema_prefix>

export RCU_SCHEMA_PWD=<rcu_schema_pwd>

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

cat /tmp/pwd.txt
```

Where:

- <db_host.domain> is the database server hostname.
- <db_port> is the database listener port.
- <service_name> is the database service name.

- <rcu_schema_prefix> is the RCU schema prefix you want to set.
- <rcu_schema_pwd> is the password you want to set for the <rcu_schema_prefix>.
- <db_pwd> is the SYS password for the database.

For example:

```
export DB_HOST=mydatabasehost.example.com

export DB_PORT=1521

export DB_SERVICE=orcl.example.com

export RCUPREFIX=OIGK8S

export RCU_SCHEMA_PWD=<password>

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

cat /tmp/pwd.txt
```

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

5. In the helper bash shell run the following command to patch the schemas:

```
/u01/oracle/oracle_common/modules/thirdparty/org.apache.ant/apache-ant/bin/ant \
-f /u01/oracle/idm/server/setup/deploy-files/automation.xml \
run-patched-sql-files \
-logger org.apache.tools.ant.NoBannerLogger \
-logfile /u01/oracle/idm/server/bin/patch_oim_wls.log \
-DoperationsDB.host="$DB_HOST" \
-DoperationsDB.port="$DB_PORT" \
-DoperationsDB.serviceName="$DB_SERVICE" \
-DoperationsDB.user="$RCUPREFIX"_OIM \
-DOIM.DBPassword="$RCU_SCHEMA_PWD" \
-Dojdbc=/u01/oracle/oracle_common/modules/oracle.jdbc/ojdbc11.jar
```

The output will look similar to the following:

```
Buildfile: /u01/oracle/idm/server/setup/deploy-files/automation.xml
```

6. Inside the helper pod, verify the database was patched successfully by viewing the patch_oim_wls.log:

```
cat /u01/oracle/idm/server/bin/patch_oim_wls.log
```

The output should look similar to below:

```
...
run-patched-sql-files:
     [sql] Executing resource: /u01/oracle/idm/server/db/oim/oracle/StoredProcedures/Recon/
OIM_SP_ReconBlkAccountChglog.sql
     [sql] Executing resource: /u01/oracle/idm/server/db/oim/oracle/Upgrade/oim14cBP/list/
oim14c_dml_pty_insert_self_assignment_allowed.sql
     [sql] Executing resource: /u01/oracle/idm/server/db/oim/oracle/StoredProcedures/API/
oim_role_mgmt_pkg_body.sql
     [sql] Executing resource: /u01/oracle/idm/server/db/oim/oracle/StoredProcedures/API/
oim_usr_mgmt_pkg_body.sql
     [sql] Executing resource: /u01/oracle/idm/server/db/oim/oracle/StoredProcedures/DBDiagnostics/
oim_db_diagnostics_pkg_body.sql
     [sql] 5 of 5 SQL statements executed successfully
BUILD SUCCESSFUL
Total time: 1 second
```

Patching the Container Image

To update the domain:

1. Run the following command to set the `image` parameter to the location of the new image:

   ```
   kubectl patch domain <domainUID> -n <domain_namespace> --type merge  -p '{"spec":
   {"image":"<repository>:<new_tag>"}}'
   ```

   For example:

   - If using Oracle Container Registry or your own container registry for your OIG container image:

     ```
     kubectl patch domain governancedomain -n oigns --type merge  -p '{"spec":{"image":"container-
     registry.oracle.com/middleware/oig_cpu:<new_tag>"}}'
     ```

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

     ```
     kubectl patch domain governancedomain -n oigns --type merge  -p '{"spec":{"image":"oracle/
     oig:<new_tag>"}}'
     ```

     The output will look similar to the following:

     ```
     domain.weblogic.oracle/governancedomain patched
     ```

Starting the OIG Domain

To start the OIG domain, seeStopping and Starting the Domain.

Verifying the OIG Deployment is Using the New Image

Once the upgrade is successful, you can run the following command to show the image is used by the pods:

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

For example:

```
kubectl describe pod governancedomain-oim-server1 -n oigns
```

The new image should be displayed in the following section:

```
...
Containers:
  weblogic-server:
    Container ID:  cri-o://220fa83d079e079ac183c00f884b10ea30a794527dbb65e6964a035d450384f8
    Image:         container-registry.oracle.com/middleware/oig_cpu:<new>
    Image ID:      container-registry.oracle.com/middleware/
oig_cpu@sha256:cdf51b6aa47cd05573bc53244681b193fb4e2f6db56e50d2251b9416bc68ebc0
    Port:          14100/TCP
    Host Port:     0/TCP
    Command:
...
```

16.1.2 Upgrading WebLogic Kubernetes Operator

The instructions in this section relate to upgrading the WebLogic Kubernetes Operator used by an Oracle Identity Governance (OIG) deployment.

> **Note**
>
> This applies to WebLogic Kubernetes Operator in the 4.X release family as additional versions are released.

To upgrade the WebLogic Kubernetes Operator used by the OIG deployment, perform the following steps:

1. On the Kubernetes administrative host, download the new WebLogic Kubernetes Operator source code from the operator github project:

   ```
   mkdir <workdir>/weblogic-kubernetes-operator-4.X.X
   ```

   ```
   cd <workdir>/weblogic-kubernetes-operator-4.X.X
   ```

   ```
   git clone https://github.com/oracle/weblogic-kubernetes-operator.git --branch v4.X.X
   ```

For example:

```
mkdir /OIGK8S/weblogic-kubernetes-operator-4.X.X
```

```
cd /OIGK8S/weblogic-kubernetes-operator-4.X.X
```

```
git clone https://github.com/oracle/weblogic-kubernetes-operator.git --branch v4.X.X
```

2. Run the following helm commands to upgrade the operator::

```
cd <workdir>/weblogic-kubernetes-operator-4.X.X/weblogic-kubernetes-operator
```

```
helm upgrade --reuse-values \
--set image=ghcr.io/oracle/weblogic-kubernetes-operator:4.X.X \
--namespace <sample-kubernetes-operator-ns> \
--wait weblogic-kubernetes-operator \
kubernetes/charts/weblogic-operator
```

For example:

```
cd /OIGK8S/weblogic-kubernetes-operator-4.X.X/weblogic-kubernetes-operator
```

```
helm upgrade --reuse-values \
--set image=ghcr.io/oracle/weblogic-kubernetes-operator:4.X.X \
--namespace opns \
--wait weblogic-kubernetes-operator \
kubernetes/charts/weblogic-operator
```

The output will look similar to the following:

```
Release "weblogic-kubernetes-operator" has been upgraded. Happy Helming!
NAME: weblogic-kubernetes-operator
LAST DEPLOYED: <DATE>
NAMESPACE: opns
STATUS: deployed
REVISION: 2
TEST SUITE: None
```

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

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

For example:

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

The output will look similar to the following:

```
NAME                                        READY   STATUS    RESTARTS   AGE
pod/weblogic-operator-b7d6df78c-mfrc4       1/1     Running   0          40s
pod/weblogic-operator-webhook-7996b8b58b-frtwp   1/1     Running   0          42s

NAME                                  TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)             AGE
service/weblogic-operator-webhook-svc   ClusterIP   10.106.51.57   <none>        8083/TCP,8084/TCP   42s

NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/weblogic-operator      1/1     1            1           6d
deployment.apps/weblogic-operator-webhook   1/1     1            1           42s

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/weblogic-operator-5884685f4f      0         0         0       6d
replicaset.apps/weblogic-operator-b7d6df78c       1         1         1       40s
replicaset.apps/weblogic-operator-webhook-7996b8b58b   1         1         1       42s
```

16.2 Upgrading from Oracle Identity Governance 12.2.1.4 to 14.1.2

The instructions in this section are for upgrading an existing Oracle Identity Governance (OIG) 12.2.1.4 deployment on Kubernetes to OIG 14.1.2.1.0.

This section contains the following topics:

16.2.1 Upgrade Prerequisite Steps

Before upgrading Oracle Identity Governance (OIG) from 12c to 14c, you must meet the following prerequisites.

OIG 12c Prerequisites

It is recommended to be on the latest OIG 12c container image, and supported WebLogic Kubernetes Operator, before upgrading to 14c. For further details on latest versions and supported operator versions, see, Oracle Identity Governance 12c on Kubernetes Release Notes.

Kubernetes Prerequisties

- The Kubernetes cluster and container engine must meet the minimum version requirements outlined in document ID 2723908.1 on My Oracle Support.
- Your <persistent_volume> must have enough free storage to back up the contents of the persistent volume and to allow for the upgrade. An upgrade can take around 3GB of storage space, so it's advisable to have at least 5GB spare before performing the upgrade.

Gathering Variables

You must gather the following information for your existing OIG 12c deployment. The values for these variables will be used later in Creating the domainUpgradeResponse.txt File, Creating the OIGDomainConfigResponse.txt File, and Creating the domain-upgrade-pod.yaml .

Table 16-1 List of Variables

Variable	Description	Sample Value
%NAMESPACE%	The domain namespace used by the OIG 12c deployment.	oigns
%DOMAIN_UID%	This is the domain uid used by the OIG 12c deployment. To find the domain name, run: kubectl get domain -n <namespace>	governancedomain
%DOMAIN_MOUNT_PATH%	The mount path used by the OIG 12c deployment. To find the mount path, run: kubectl describe domains <domainUID> -n <namespace> | grep "Mount Path"	• For WDT domains:/u01/oracle/user_projects • For WLST domains:/u01/oracle/user_projects/domains
%DOMAIN_HOME%	The domain home location used by the OIG 12c deployment. To find the mount path, run: kubectl describe domains <domainUID> -n <namespace> | grep "Domain Home:"	/u01/oracle/user_projects/domains/governancedomain
%DOMAIN_ROOT_DIR%	For WLST created domains, this is the *%DOMAIN_MOUNT_PATH%/*. For WDT created domains, this is the *%DOMAIN_MOUNT_PATH%/domains* directory.	/u01/oracle/user_projects/domains

Table 16-1 (Cont.) List of Variables

Variable	Description	Sample Value
%CONNECTION_STRING%	The connection string for the database where the OIG 12c schemas reside, in the format: <host.domain>:<db_port>/<db_service>	mydatabasehost.example.com:1521/orcl.example.com
%RCU_PREFIX%	The RCU schema prefix for the OIG 12c deployment.	OIGK8S
%RCU_SCHEMA_PWD%	The password for *%RCUPREFIX%*.	<password>
%SYS_USERNAME%	The SYS username for the database where the OIG 12c schemas reside.	sys
%SYS_USERNAME_PWD%	The password for *%SYS_USERNAME%*.	<password>
%DOMAIN_PVC_NAME%	The persistent volume claim (PVC) for the OIG 12c deployment. To find the PVC, run: kubectl get pvc -n <namespace>	governancedomain-domain-pvc
%RCU_CREDENTIALS_SECRET_NAME%	The RCU secret for the OIG 12c deployment. To find the RCU secret, run: kubectl get secrets -n <namespace> | grep rcu	governancedomain-rcu-credentials
%WEBLOGIC_IMAGE%	The location of the OIG 14c container image.	container-registry.oracle.com/middleware/oig_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD>
%WEBLOGIC_IMAGE_PULL_POLICY%	The image pull policy you used in the OIG 12c deployment. To find the image pull policy, run: kubectl describe domains <domainUID> -n <namespace> | grep "Image Pull Policy"	IfNotPresent

16.2.2 Creating the domainUpgradeResponse.txt File

Run the following steps to create the domainUpgradeResponse.txt file:

1. Create a working directory for the upgrade scripts and navigate to it:

   mkdir <workdir>/upgradescripts

   cd <workdir>/upgradescripts

For example:

```
mkdir /OIG12CUPG/upgradescripts


cd /OIG12CUPG/upgradescripts
```

2. Create a domainUpgradeResponse.txt file and replace the environment variables listed with the corresponding values collected in Upgrade Prerequisite Steps:

```
[GENERAL]

fileFormatVersion = 3

#================================================================
[UAWLSINTERNAL.UAWLS]
pluginInstance = 1

# Specifies the WebLogic Server domain directory:
UASVR.path = %DOMAIN_HOME%

#================================================================
[MDS.SCHEMA_UPGRADE]
pluginInstance = 2
MDS.databaseConnectionString = %CONNECTION_STRING%
MDS.schemaConnectionString = %CONNECTION_STRING%
MDS.schemaUserName = %RCUPREFIX%_MDS
MDS.cleartextSchemaPassword = %SCHEMA_PASSWORD%
MDS.dbaUserName = %SYS_USERNAME% as sysdba
MDS.cleartextDbaPassword = %SYS_SCHEMA_PASSWORD%

#================================================================
[SOA.SOA1]
pluginInstance = 3
dependsOnPluginInstance = 2
SOAINFRA.databaseConnectionString = %CONNECTION_STRING%
SOAINFRA.schemaUserName = %RCUPREFIX%_SOAINFRA
SOAINFRA.cleartextSchemaPassword = %SCHEMA_PASSWORD%
SOAINFRA.dbaUserName = %SYS_USERNAME% as sysdba
SOAINFRA.cleartextDbaPassword = %SYS_SCHEMA_PASSWORD%

#================================================================
[OPSS.OPSS_SCHEMA_PLUGIN]
pluginInstance = 4
OPSS.databaseConnectionString = %CONNECTION_STRING%
OPSS.schemaUserName = %RCUPREFIX%_OPSS
OPSS.cleartextSchemaPassword = %SCHEMA_PASSWORD%
OPSS.dbaUserName = %SYS_USERNAME% as sysdba
OPSS.cleartextDbaPassword = %SYS_SCHEMA_PASSWORD%

#================================================================
[IAU.AUDIT_SCHEMA_PLUGIN]
pluginInstance = 5
IAU.databaseConnectionString = %CONNECTION_STRING%
IAU.schemaConnectionString = %CONNECTION_STRING%
IAU.schemaUserName = %RCUPREFIX%_IAU
```

```
IAU.cleartextSchemaPassword = %SCHEMA_PASSWORD%
IAU.dbaUserName = %SYS_USERNAME% as sysdba
IAU.cleartextDbaPassword = %SYS_SCHEMA_PASSWORD%

#================================================================
[FMWCONFIG.CIE_SCHEMA_PLUGIN]
pluginInstance = 6
STB.databaseConnectionString = %CONNECTION_STRING%
STB.schemaConnectionString = %CONNECTION_STRING%
STB.schemaUserName = %RCUPREFIX%_STB
STB.cleartextSchemaPassword = %SCHEMA_PASSWORD%
STB.dbaUserName = %SYS_USERNAME% as sysdba
STB.cleartextDbaPassword = %SYS_SCHEMA_PASSWORD%

#================================================================
[WLS.WLS]
pluginInstance = 7
WLS.databaseConnectionString = %CONNECTION_STRING%
WLS.schemaConnectionString = %CONNECTION_STRING%
WLS.schemaUserName = %RCUPREFIX%_WLS
WLS.cleartextSchemaPassword = %SCHEMA_PASSWORD%
WLS.dbaUserName = %SYS_USERNAME% as sysdba
WLS.cleartextDbaPassword = %SYS_SCHEMA_PASSWORD%

# WLS_RUNTIME entries
WLS_RUNTIME.databaseConnectionString = %CONNECTION_STRING%
WLS_RUNTIME.schemaUserName = %RCUPREFIX%_WLS_RUNTIME
WLS_RUNTIME.cleartextSchemaPassword = %SCHEMA_PASSWORD%
WLS_RUNTIME.dbaUserName = %SYS_USERNAME% as sysdba
WLS_RUNTIME.cleartextDbaPassword = %SYS_SCHEMA_PASSWORD%

#================================================================
[UCSUMS.UCSUMS_SCHEMA_PLUGIN]
pluginInstance = 8
UMS.databaseConnectionString = %CONNECTION_STRING%
UMS.schemaUserName = %RCUPREFIX%_UMS
UMS.cleartextSchemaPassword = %SCHEMA_PASSWORD%
UMS.dbaUserName = %SYS_USERNAME% as sysdba
UMS.cleartextDbaPassword = %SYS_SCHEMA_PASSWORD%

#================================================================
[OIM.OIM1]
pluginInstance = 9
dependsOnPluginInstance = 3
OIM.databaseConnectionString = %CONNECTION_STRING%
OIM.schemaUserName = %RCUPREFIX%_OIM
OIM.cleartextSchemaPassword = %SCHEMA_PASSWORD%
OIM.dbaUserName = %SYS_USERNAME% as sysdba
OIM.cleartextDbaPassword = %SYS_SCHEMA_PASSWORD%
```

For example:

```
[GENERAL]

fileFormatVersion = 3
```

```
#===============================================================
[UAWLSINTERNAL.UAWLS]
pluginInstance = 1

# Specifies the WebLogic Server domain directory:
UASVR.path = /u01/oracle/user_projects/domains/governancedomain

#===============================================================
[MDS.SCHEMA_UPGRADE]
pluginInstance = 2
MDS.databaseConnectionString = mydatabasehost.example.com:1521/orcl.example.com
MDS.schemaConnectionString = mydatabasehost.example.com:1521/orcl.example.com
MDS.schemaUserName = OIGK8S_MDS
MDS.cleartextSchemaPassword = <password>
MDS.dbaUserName = sys as sysdba
MDS.cleartextDbaPassword = <password>

#===============================================================
[SOA.SOA1]
pluginInstance = 3
dependsOnPluginInstance = 2
SOAINFRA.databaseConnectionString = mydatabasehost.example.com:1521/orcl.example.com
SOAINFRA.schemaUserName = OIGK8S_SOAINFRA
SOAINFRA.cleartextSchemaPassword = <password>
SOAINFRA.dbaUserName = sys as sysdba
SOAINFRA.cleartextDbaPassword = <password>

#===============================================================
[OPSS.OPSS_SCHEMA_PLUGIN]
pluginInstance = 4
OPSS.databaseConnectionString = mydatabasehost.example.com:1521/orcl.example.com
OPSS.schemaUserName = OIGK8S_OPSS
OPSS.cleartextSchemaPassword = <password>
OPSS.dbaUserName = sys as sysdba
OPSS.cleartextDbaPassword = <password>

#===============================================================
[IAU.AUDIT_SCHEMA_PLUGIN]
pluginInstance = 5
IAU.databaseConnectionString = mydatabasehost.example.com:1521/orcl.example.com
IAU.schemaConnectionString = mydatabasehost.example.com:1521/orcl.example.com
IAU.schemaUserName = OIGK8S_IAU
IAU.cleartextSchemaPassword = <password>
IAU.dbaUserName = sys as sysdba
IAU.cleartextDbaPassword = <password>

#===============================================================
[FMWCONFIG.CIE_SCHEMA_PLUGIN]
pluginInstance = 6
STB.databaseConnectionString = mydatabasehost.example.com:1521/orcl.example.com
STB.schemaConnectionString = mydatabasehost.example.com:1521/orcl.example.com
STB.schemaUserName = OIGK8S_STB
STB.cleartextSchemaPassword = <password>
STB.dbaUserName = sys as sysdba
STB.cleartextDbaPassword = <password>
```

```
#=============================================================
[WLS.WLS]
pluginInstance = 7
WLS.databaseConnectionString = mydatabasehost.example.com:1521/orcl.example.com
WLS.schemaConnectionString = mydatabasehost.example.com:1521/orcl.example.com
WLS.schemaUserName = OIGK8S_WLS
WLS.cleartextSchemaPassword = <password>
WLS.dbaUserName = sys as sysdba
WLS.cleartextDbaPassword = <password>

# WLS_RUNTIME entries
WLS_RUNTIME.databaseConnectionString = mydatabasehost.example.com:1521/orcl.example.com
WLS_RUNTIME.schemaUserName = OIGK8S_WLS_RUNTIME
WLS_RUNTIME.cleartextSchemaPassword = <password>
WLS_RUNTIME.dbaUserName = sys as sysdba
WLS_RUNTIME.cleartextDbaPassword = <password>

#=============================================================
[UCSUMS.UCSUMS_SCHEMA_PLUGIN]
pluginInstance = 8
UMS.databaseConnectionString = mydatabasehost.example.com:1521/orcl.example.com
UMS.schemaUserName = OIGK8S_UMS
UMS.cleartextSchemaPassword = <password>
UMS.dbaUserName = sys as sysdba
UMS.cleartextDbaPassword = <password>

#=============================================================
[OIM.OIM1]
pluginInstance = 9
dependsOnPluginInstance = 3
OIM.databaseConnectionString = mydatabasehost.example.com:1521/orcl.example.com
OIM.schemaUserName = OIGK8S_OIM
OIM.cleartextSchemaPassword = <password>
OIM.dbaUserName = sys as sysdba
OIM.cleartextDbaPassword = <password>
```

16.2.3 Creating the OIGDomainConfigResponse.txt File

Run the following steps to create the OIGDomainConfigResponse.txt file:

1. In the <workdir>/upgradescripts create an OIGDomainConfigResponse.txt file and replace the %DOMAIN_HOME% environment variable with the value collected in Upgrade Prerequisite Steps:

```
[GENERAL]
# This is the file format version number.  Do not change the next line.
fileFormatVersion = 3

# The next section contains information for accessing a WebLogic Server domain.
[UAWLSINTERNAL.UAWLS]
# The following number uniquely identifies this instance of an
# upgrade plugin.  Do not change it.
pluginInstance = 1
```

```
# Specifies the WebLogic Server domain directory:
UASVR.path = %DOMAIN_HOME%

# The next section contains the information for performing a mid-tier
# upgrade on Oracle Identity Manager, as described in the Upgrade
# Descriptor file located at
#   /u01/oracle/idm/plugins/upgrade/oim.xml
# Do not change the next line.
[OIM.OIMCONFIGPLUGIN]
# The following number uniquely identifies this instance of an
# upgrade plugin.  Do not change it.
pluginInstance = 17

# The next section contains the information for performing a mid-tier
# upgrade on Oracle JRF, as described in the Upgrade
# Descriptor file located at
#   /u01/oracle/oracle_common/plugins/upgrade/jrfua.xml
# Do not change the next line.
[JRF.JRF_CONFIG_PLUGIN]
# The following number uniquely identifies this instance of an
# upgrade plugin.  Do not change it.
pluginInstance = 6

# The next section contains the information for performing a mid-tier
# upgrade on System Components Infrastructure, as described in the Upgrade
# Descriptor file located at
#   /u01/oracle/oracle_common/plugins/upgrade/syscomp.xml
# Do not change the next line.
[CAM.SYSCOMP]
# The following number uniquely identifies this instance of an
# upgrade plugin.  Do not change it.
pluginInstance = 11

# The next section contains the information for performing a mid-tier
# upgrade on Oracle Web Services Manager, as described in the Upgrade
# Descriptor file located at
#   /u01/oracle/oracle_common/plugins/upgrade/wsm.xml
# Do not change the next line.
[WSM.WSMPLUGIN]
# The following number uniquely identifies this instance of an
# upgrade plugin.  Do not change it.
pluginInstance = 10

# The next section contains the information for performing a mid-tier
# upgrade on User Messaging Service, as described in the Upgrade
# Descriptor file located at
#   /u01/oracle/oracle_common/plugins/upgrade/usermessaging.xml
# Do not change the next line.
[UCSUMS.UCSUMS_CONFIGURATION_PLUGIN]
# The following number uniquely identifies this instance of an
# upgrade plugin.  Do not change it.
pluginInstance = 9
# Specifies a choice from a collection of values,
# "Is this correct?"
REMOTE_COPY_CHOICE.choose = REMOTE_COPY_CHOICE3
```

For example:

```
[GENERAL]
# This is the file format version number.  Do not change the next line.
fileFormatVersion = 3

# The next section contains information for accessing a WebLogic Server domain.
[UAWLSINTERNAL.UAWLS]
# The following number uniquely identifies this instance of an
# upgrade plugin.  Do not change it.
pluginInstance = 1

# Specifies the WebLogic Server domain directory:
UASVR.path = /u01/oracle/user_projects/domains/governancedomain

# The next section contains the information for performing a mid-tier
# upgrade on Oracle Identity Manager, as described in the Upgrade
# Descriptor file located at
#   /u01/oracle/idm/plugins/upgrade/oim.xml
# Do not change the next line.
[OIM.OIMCONFIGPLUGIN]
# The following number uniquely identifies this instance of an
# upgrade plugin.  Do not change it.
pluginInstance = 17

# The next section contains the information for performing a mid-tier
# upgrade on Oracle JRF, as described in the Upgrade
# Descriptor file located at
#   /u01/oracle/oracle_common/plugins/upgrade/jrfua.xml
# Do not change the next line.
[JRF.JRF_CONFIG_PLUGIN]
# The following number uniquely identifies this instance of an
# upgrade plugin.  Do not change it.
pluginInstance = 6

# The next section contains the information for performing a mid-tier
# upgrade on System Components Infrastructure, as described in the Upgrade
# Descriptor file located at
#   /u01/oracle/oracle_common/plugins/upgrade/syscomp.xml
# Do not change the next line.
[CAM.SYSCOMP]
# The following number uniquely identifies this instance of an
# upgrade plugin.  Do not change it.
pluginInstance = 11

# The next section contains the information for performing a mid-tier
# upgrade on Oracle Web Services Manager, as described in the Upgrade
# Descriptor file located at
#   /u01/oracle/oracle_common/plugins/upgrade/wsm.xml
# Do not change the next line.
[WSM.WSMPLUGIN]
# The following number uniquely identifies this instance of an
# upgrade plugin.  Do not change it.
pluginInstance = 10

# The next section contains the information for performing a mid-tier
```

```
# upgrade on User Messaging Service, as described in the Upgrade
# Descriptor file located at
#   /u01/oracle/oracle_common/plugins/upgrade/usermessaging.xml
# Do not change the next line.
[UCSUMS.UCSUMS_CONFIGURATION_PLUGIN]
# The following number uniquely identifies this instance of an
# upgrade plugin.  Do not change it.
pluginInstance = 9
# Specifies a choice from a collection of values,
# "Is this correct?"
REMOTE_COPY_CHOICE.choose = REMOTE_COPY_CHOICE3
```

16.2.4 Creating the domain-upgrade-pod.yaml

Run the following steps to create the domain-upgrade-pod.yaml file:

1. In the <workdir>/upgradescripts directory create a domain-upgrade-pod.yaml and replace the environment variables listed, with the corresponding values collected in Upgrade Prerequisite Steps:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    weblogic.domainUID: %DOMAIN_UID%
    weblogic.domainName: %DOMAIN_UID%
    app: %DOMAIN_UID%-domain-upgrade
  name: %DOMAIN_UID%-domain-upgrade
  namespace: %NAMESPACE%
spec:
  containers:
  - args:
    - sleep
    - infinity
    image: %WEBLOGIC_IMAGE%
    imagePullPolicy: %WEBLOGIC_IMAGE_PULL_POLICY%
    name: %DOMAIN_UID%-domain-upgrade
    volumeMounts:
    - mountPath: /u01/scripts
      name: domain-upgrade-cm-volume
    - mountPath: %DOMAIN_MOUNT_PATH%
      name: domain-storage-volume
    - mountPath: /weblogic-operator/rcu-secrets
      name: rcu-credentials-volume
    env:
    - name: DOMAIN_UID
      value: "%DOMAIN_UID%"
    - name: DOMAIN_ROOT_DIR
      value: "%DOMAIN_ROOT_DIR%"
    - name: DOMAIN_HOME_DIR
      value: "%DOMAIN_HOME%"
    - name: DOMAIN_NAME
      value: "%DOMAIN_UID%"
    - name: CONNECTION_STRING
      value: "%CONNECTION_STRING%"
```

```
    - name: RCUPREFIX
      value: "%RCUPREFIX%"
    - name: DOMAIN_TYPE
      value: "OIG"
    - name: SECURE_ENABLED
      value: "false"
  volumes:
  - name: domain-upgrade-cm-volume
    configMap:
      name: %DOMAIN_UID%-domain-upgrade-pod-cm
  - name: domain-storage-volume
    persistentVolumeClaim:
      claimName: %DOMAIN_PVC_NAME%
  - name: rcu-credentials-volume
    secret:
      secretName: %RCU_CREDENTIALS_SECRET_NAME%
```

For example:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    weblogic.domainUID: governancedomain
    weblogic.domainName: governancedomain
    app: governancedomain-domain-upgrade
  name: governancedomain-domain-upgrade
  namespace: oigns
spec:
  containers:
  - args:
    - sleep
    - infinity
    image: container-registry.oracle.com/middleware/oig_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD>
    imagePullPolicy: IfNotPresent
    name: governancedomain-domain-upgrade
    volumeMounts:
    - mountPath: /u01/scripts
      name: domain-upgrade-cm-volume
    - mountPath: /u01/oracle/user_projects
      name: domain-storage-volume
    - mountPath: /weblogic-operator/rcu-secrets
      name: rcu-credentials-volume
    env:
    - name: DOMAIN_UID
      value: "governancedomain"
    - name: DOMAIN_ROOT_DIR
      value: "/u01/oracle/user_projects/domains"
    - name: DOMAIN_HOME_DIR
      value: "/u01/oracle/user_projects/domains/governancedomain"
    - name: DOMAIN_NAME
      value: "governancedomain"
    - name: CONNECTION_STRING
      value: "mydatabasehost.example.com:1521/orcl.example.com"
```

```
  - name: RCUPREFIX
    value: "OIGK8S"
  - name: DOMAIN_TYPE
    value: "OIG"
  - name: SECURE_ENABLED
    value: "false"
 volumes:
 - name: domain-upgrade-cm-volume
   configMap:
     name: governancedomain-domain-upgrade-pod-cm
 - name: domain-storage-volume
   persistentVolumeClaim:
     claimName: governancedomain-domain-pvc
 - name: rcu-credentials-volume
   secret:
     secretName: governancedomain-rcu-credentials
```

16.2.5 Shutting Down the OIG Domain

Run the following commands to shutdown the Oracle Identity Governance (OIG) 12c deployment:

1. Shut down the OIG deployment using the following command:

   ```
   kubectl patch domain %DOMAIN_UID% -n %NAMESPACE% --type=merge --patch "{\"spec\": {\"serverStartPolicy\": \"Never\"}}"
   ```

   For example:

   ```
   kubectl patch domain governancedomain -n oigns --type=merge --patch "{\"spec\": {\"serverStartPolicy\": \"Never\"}}"
   ```

   The output will look similar to the following:

   ```
   domain.weblogic.oracle/governancedomain patched
   ```

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

   ```
   kubectl get pods -n %NAMESPACE%
   ```

   For example:

   ```
   kubectl get pods -n oigns
   ```

   The output will look similar to the following:

   ```
   NAME                          READY   STATUS        RESTARTS   AGE
   governancedomain-adminserver  1/1     Running       0          22h
   governancedomain-soa-server1  1/1     Running       0          22h
   governancedomain-oim-server1  0/1     Terminating   0          22h
   ```

The Administration Server pods and Managed Server pods will move to a STATUS of Terminating.

After a few minutes, run the command again and make sure the pods should have disappeared before continuing.

3. If a helper pod exists, then delete it:

```
kubectl delete pod helper -n %NAMESPACE%
```

For example:

```
kubectl delete pod helper -n oigns
```

16.2.6 Backing Up the Database and Persistent Volume

You must take a backup of the Oracle Database used by Oracle Identity Governance (OIG) 12c, and the persistent volume.

Backing Up the Oracle Database

Take a backup of the Oracle Database used by OIG 12c, using your usual Oracle Database backup procedure.

Backing Up the Persistent Volume

Take a backup of the persistent volume directory:

```
sudo cp -rp <persistent_volume>/governancedomainpv <persistent_volume>/governancedomain_bkp12c
```

For example:

```
sudo cp -rp /nfs_volumes/oig/governancedomainpv /nfs_volumes/oig/governancedomainpv_bkp12c
```

16.2.7 Creating an Upgrade ConfigMap

Create a ConfigMap for the upgrade by performing the following:

1. Run the following command to create the ConfigMap:

```
kubectl create configmap %DOMAIN_UID%-domain-upgrade-pod-cm -n %NAMESPACE% \
--from-file <workdir>/upgradescripts --dry-run=client -o yaml
```

For example:

```
kubectl create configmap governancedomain-domain-upgrade-pod-cm -n oigns \
--from-file /OIG12CUPG/upgradescripts --dry-run=client -o yaml | kubectl apply -f -
```

The output should look similar to the following:

```
configmap/governancedomain-domain-upgrade-pod-cm created
```

16.2.8 Performing the Upgrade

To perform the upgrade you must create an upgrade pod and run several upgrade commands.

> **Note**
>
> If the upgrade fails, see Restoring After a Failed Upgrade.

1. Run the following command to create the domain-upgrade-pod:

   ```
   kubectl apply -f <workdir>/upgradescripts/domain-upgrade-pod.yaml
   ```

   For example:

   ```
   kubectl apply -f /OIG12CUPG/upgradescripts/domain-upgrade-pod.yaml
   ```

   The output should look similar to the following:

   ```
   pod/governancedomain-domain-upgrade created
   ```

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

   ```
   kubectl get pods -n %NAMESPACE%
   ```

   For example:

   ```
   kubectl get pods -n oigns
   ```

   The output will look similar to the following:

   ```
   NAME                                READY   STATUS    RESTARTS   AGE
   governancedomain-domain-upgrade     1/1     Running   0          2m3s
   ```

   It may take a few minutes until the pod is in a status of READY 1/1.

3. If you are running Kubernetes 1.30 or higher, set the following environment variable:

   ```
   export KUBECTL_REMOTE_COMMAND_WEBSOCKETS=false
   ```

4. Run the following command to enter a bash shell in the domain-upgrade pod:

   ```
   kubectl exec -it %DOMAIN_UID%-domain-upgrade -n %NAMESPACE% -- /bin/bash
   ```

For example:

```
kubectl exec -it governancedomain-domain-upgrade -n oigns -- /bin/bash
```

This will take you into a bash shell in the domain-upgrade pod:

```
[oracle@governancedomain-domain-upgrade oracle]$
```

5. Inside the upgrade pod, navigate to the /u01/scriptsdirectory:

```
cd /u01/scripts
```

6. Run the following command to run the Upgrade Assistant:

```
$ORACLE_HOME/oracle_common/upgrade/bin/ua -response /u01/scripts/domainUpgradeResponse.txt -logLevel TRACE -logDir /tmp
```

The output should look similar to the following:

```
Oracle Fusion Middleware Upgrade Assistant 14.1.2.0.0
Log file is located at: /tmp/ua<DATE>.log
Reading installer inventory, this will take a few moments...
...completed reading installer inventory.
Using response file /u01/scripts/domainUpgradeResponse.txt for input
 Oracle Metadata Services schema examine is in progress
 Oracle Platform Security Services schema examine is in progress
 Oracle Audit Services schema examine is in progress
 Common Infrastructure Services schema examine is in progress
 Common Infrastructure Services schema examine finished with status: ready for upgrade
 Oracle Metadata Services schema examine finished with status: ready for upgrade
 Oracle WebLogicServer schema examine is in progress
 User Messaging Service schema examine is in progress
 Oracle WebLogicServer schema examine finished with status: ready for upgrade
 Oracle SOA schema examine is in progress
 Oracle Platform Security Services schema examine finished with status: ready for upgrade
 User Messaging Service schema examine finished with status: ready for upgrade
 Oracle SOA schema examine finished with status: ready for upgrade
 Oracle Identity Manager schema examine is in progress
 Oracle Audit Services schema examine finished with status: ready for upgrade
 Oracle Identity Manager schema examine finished with status: ready for upgrade
Schema Version Registry saved to: /tmp/ua2025-03-13-15-34-59PM.xml
 Oracle Metadata Services schema upgrade is in progress
 Common Infrastructure Services schema upgrade is in progress
 Oracle Platform Security Services schema upgrade is in progress
 Oracle Audit Services schema upgrade is in progress
 Common Infrastructure Services schema upgrade finished with status: succeeded
 User Messaging Service schema upgrade is in progress
 Oracle Audit Services schema upgrade finished with status: succeeded
 Oracle WebLogicServer schema upgrade is in progress
 Oracle Metadata Services schema upgrade finished with status: succeeded
 Oracle SOA schema upgrade is in progress
 Oracle WebLogicServer schema upgrade finished with status: succeeded
 Oracle Platform Security Services schema upgrade finished with status: succeeded
 User Messaging Service schema upgrade finished with status: succeeded
```

```
 Oracle SOA schema upgrade finished with status: succeeded
 Oracle Identity Manager schema upgrade is in progress
UPGRADE PATH : [12.2.1.4.0, 14.1.2.1.0]
 Oracle Identity Manager schema upgrade finished with status: succeeded

Oracle SOA
1. The Upgrade Assistant has successfully upgraded all active instances. You can now close the Upgrade
Assistant.
2. The automated upgrade of closed instances will continue in the background after the Upgrade Assistant is
exited and until the SOA server is started,at which point the upgrade will stop.  You can schedule the upgrade of
any remaining closed instances for a time when the SOA server is less busy.
 Close the Upgrade Assistant and use the instance data administration scripts to administer and monitor the
overall progress of this automated upgrade. For more information see "Administering and Monitoring the
Upgrade of SOA Instance Data" in Upgrading SOA Suite and Business Process Management.

[oracle@governancedomain-domain-upgrade scripts]$
```

7. Enter a wlst prompt inside the domain-upgrade pod:

   ```
   $ORACLE_HOME/oracle_common/common/bin/wlst.sh
   ```

   The output will look similar to the following:

   ```
   Initializing WebLogic Scripting Tool (WLST) ...

   Jython scans all the jar files it can find at first startup. Depending on the system, this process may take a few
   minutes to complete, and WLST may not return a prompt right away.

   Welcome to WebLogic Server Administration Scripting Shell

   Type help() for help on available commands

   wls:/offline>
   ```

8. Run the following at the wlst prompt to perform domain reconfiguration:

   a. Set the domainHome:

      ```
      domainHome='/u01/oracle/user_projects/domains/%DOMAIN_UID%'
      ```

      For example:

      ```
      domainHome='/u01/oracle/user_projects/domains/governancedomain'
      ```

   b. Read the domanHome:

      ```
      readDomainForUpgrade(domainHome)
      ```

      The output will look similar to the following:

      ```
      wls:/offline/governancedomain>
      ```

c. Update the domain:

updateDomain()

> **Note**
>
> This command can take approximately 10 minutes to complete.

The output will look similar to the following:

```
INFO: JPS Config: /u01/oracle/user_projects/domains/governancedomain/config/fmwconfig/jps-config-jse.xml
INFO: JPS Config: /u01/oracle/user_projects/domains/governancedomain/config/fmwconfig/jps-config.xml
WARNING: Bootstrap services are used by OPSS internally and clients should never need to directly read/write bootstrap credentials. If required, use Wlst or configuration management interfaces.
INFO: JPS Config: /u01/oracle/user_projects/domains/governancedomain/config/fmwconfig/jps-config-jse.xml
INFO: JPS Config: /u01/oracle/user_projects/domains/governancedomain/config/fmwconfig/jps-config.xml
INFO: No ADFLoggerExtension registration, skipping.
INFO: ADFLoggerExtensionODL setup complete
WARNING: MDS-11019: The default CharSet US-ASCII is not a unicode character set. File names with non-ASCII characters may not operate as expected. Check locale settings.
INFO: MDS_Repository type being ignored with insufficient uniqueness in /oracle/mds/ (MDS)
INFO: MDS_Partition type being ignored with insufficient uniqueness in /oracle/mds//oracle (MDS_Repository)
INFO: Property for read store in parallel: oracle.security.jps.az.runtime.readstore.threads = null
INFO: MDS_Partition type being ignored with insufficient uniqueness in /oracle/mds//oracle (MDS_Repository)
```

d. Close the domain:

closeDomain()

9. Run the following post upgrade steps at the wls:/offline> prompt:

a. Set the domainHome:

domainHome='/u01/oracle/user_projects/domains/*%DOMAIN_UID%*'

For example:

domainHome='/u01/oracle/user_projects/domains/*governancedomain*'

b. Set the RCUPREFIX and RCU_SCHEMA_PWD password:

wlsRuntimeUser='*%RCUPREFIX%*_WLS_RUNTIME'

schemaPassword='*%RCU_SCHEMA_PWD%*'

For example:

```
wlsRuntimeUser='OIGK8S_WLS_RUNTIME'

schemaPassword='<password>'
```

c. Read the domain:

```
readDomain(domainHome)
```

d. Change to the following directory:

```
cd('/JdbcSystemResource/WLSRuntimeSchemaDataSource/JdbcResource/
WLSRuntimeSchemaDataSource/JdbcDriverParams/NO_NAME_0')
```

e. Set the CONNECTION_STRING:

```
dbUrl="jdbc:oracle:thin:@%CONNECTION_STRING%"
```

For example:

```
dbUrl="jdbc:oracle:thin:@mydatabasehost.example.com:1521/orcl.example.com"
```

f. Run the following to update the domain:

```
cmo.setUrl(dbUrl)

cmo.setDriverName('oracle.jdbc.OracleDriver')

set('PasswordEncrypted', schemaPassword)

cd('Properties/NO_NAME_0/Property/user')

cmo.setValue(wlsRuntimeUser)

cd('/')

updateDomain()
```

No output will be returned to the screen and you will just be returned to the prompt.

g. Close the domain and exit:

```
closeDomain()

exit()
```

10. Inside bash shell of the domain-upgrade pod. run the following to perform the configuration upgrade:

```
$ORACLE_HOME/oracle_common/upgrade/bin/ua -configUpgrade -response /u01/scripts/
OIGDomainConfigResponse.txt -logLevel TRACE -logDir /tmp
```

The output will look similar to the following:

```
Oracle Fusion Middleware Upgrade Assistant 14.1.2.0.0
Log file is located at: /tmp/ua2025-03-13-15-45-36PM.log
Reading installer inventory, this will take a few moments...
...completed reading installer inventory.
Using response file /u01/scripts/OIGDomainConfigResponse.txt for input
 Oracle Identity Manager configuration examine is in progress
UPGRADE PATH : [12.2.1.4.0, 14.1.2.1.0]
 Oracle Identity Manager configuration examine finished with status: upgrade not necessary
 Oracle JRF configuration examine is in progress
 Oracle JRF configuration examine finished with status: upgrade not necessary
 System Components Infrastructure configuration examine is in progress
 System Components Infrastructure configuration examine finished with status: upgrade not necessary
 Oracle Web Services Manager configuration examine is in progress
java.lang.Throwable: STARTUP - DIAGNOSTIC STACK
      at
oracle.adf.share.platform.AdfServerPlatformUtil$SingletonHolder.internalLogSystemProperties(AdfServerPlatfo
rmUtil.java:746)
      at
oracle.adf.share.platform.AdfServerPlatformUtil$SingletonHolder.logSystemProperties(AdfServerPlatformUtil.j
ava:723)
      at
oracle.adf.share.platform.AdfServerPlatformUtil$SingletonHolder.<clinit>(AdfServerPlatformUtil.java:485)
      at oracle.adf.share.platform.AdfServerPlatformUtil.<clinit>(AdfServerPlatformUtil.java:297)
      at oracle.adf.share.mt.util.CloudHelper$SingletonHolder.<clinit>(CloudHelper.java:37)
      at oracle.adf.share.mt.util.CloudHelper.isMultitenancySupportable(CloudHelper.java:89)
      at oracle.adf.share.mt.util.MultiPartitionStorage.initValueStorage(MultiPartitionStorage.java:21)
      at oracle.adf.share.mt.util.MultiTenantStorage.<init>(MultiTenantStorage.java:47)
      at oracle.adf.share.mt.util.MultiPartitionStorage.<init>(MultiPartitionStorage.java:15)
      at oracle.mds.internal.util.MDSTimerMTS$4.<init>(MDSTimerMTS.java:277)
      at oracle.mds.internal.util.MDSTimerMTS.<clinit>(MDSTimerMTS.java:275)
      at oracle.mds.internal.util.MDSTimer.createTimer(MDSTimer.java:67)
      at oracle.mds.persistence.stores.db.DBMetadataStore.<clinit>(DBMetadataStore.java:2290)
      at oracle.wsm.repository.mds.MDSInstanceFactory.getMDSInstance(MDSInstanceFactory.java:352)
      at oracle.wsm.lifecycle.common.util.PluginBeanCreator.getBean(PluginBeanCreator.java:87)
      at oracle.wsm.lifecycle.upgrade.impl.UpgradePluginExecutor.<init>(UpgradePluginExecutor.java:186)
      at oracle.wsm.lifecycle.upgrade.impl.WSMUpgradePlugin.examine(WSMUpgradePlugin.java:316)
      at oracle.ias.update.plugin.Plugin.examine(Plugin.java:689)
      at oracle.ias.update.plan.PlanStep.examine(PlanStep.java:734)
      at oracle.ias.update.PhaseProcessor$ExamineProcessor.runStepPhase(PhaseProcessor.java:900)
      at oracle.ias.update.PhaseProcessor.runStep(PhaseProcessor.java:481)
      at oracle.ias.update.PhaseProcessor$ExtendedRunnable.run(PhaseProcessor.java:1735)
      at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1136)
      at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:635)
      at java.base/java.lang.Thread.run(Thread.java:842)
STARTUP - SYSTEM PROPERTIES
APPSERVER_TYPE=wls
common.components.home=/u01/oracle/oracle_common
```

file.encoding=ANSI_X3.4-1968
file.separator=/
ice.pilots.html4.ignoreNonGenericFonts=true
java.class.path=/u01/oracle/oracle_common/upgrade/jlib/ua.jar:/u01/oracle/oracle_common/upgrade/jlib/SchemaVersion.jar:/u01/oracle/oracle_common/modules/features/com.oracle.db.jdbc-no-dms.jar:/u01/oracle/oracle_common/modules/datadirect/fmwgenerictoken.jar:/u01/oracle/oracle_common/modules/datadirect/wlsqlserver.jar:/u01/oracle/oracle_common/modules/datadirect/wldb2.jar:/u01/oracle/oracle_common/modules/mysql-connector-java-commercial-5.1.22/mysql-connector-java-commercial-5.1.22-bin.jar:/u01/oracle/wlserver/common/derby/lib/derbyclient.jar:/u01/oracle/oracle_common/modules/oracle.bali.jewt/jewt4.jar:/u01/oracle/oracle_common/modules/oracle.bali.jewt/olaf2.jar:/u01/oracle/oracle_common/modules/oracle.odl/ojdl.jar:/u01/oracle/oracle_common/modules/oracle.dms/dms.jar:/u01/oracle/oracle_common/modules/oracle.bali.share/share.jar:/u01/oracle/jlib/ojmisc.jar:/u01/oracle/oracle_common/modules/oracle.ldap/ldapjclnt.jar:/u01/oracle/oracle_common/modules/oracle.help/help-share.jar:/u01/oracle/oracle_common/modules/oracle.help/ohj.jar:/u01/oracle/oracle_common/modules/oracle.help/oracle_ice.jar:/u01/oracle/jlib/oraclepki.jar:/u01/oracle/oracle_common/modules/oracle.nlsrtl/orai18n-mapping.jar:/u01/oracle/oracle_common/modules/oracle.jrf/jrf-api.jar:/u01/oracle/wlserver/server/lib/weblogic.jar:/u01/oracle/wlserver/modules/org.slf4j.slf4j-nop.jar:/u01/oracle/wlserver/modules/wlstt3client.jar:/u01/oracle/oracle_common/jlib/wizardCommonResources.jar:/u01/oracle/oracle_common/jlib/rcucommon.jar:/u01/oracle/oracle_common/modules/features/rcuapi_lib.jar:/u01/oracle/oracle_common/modules/features/cieCfg_wls_lib.jar:/u01/oracle/oracle_common/modules/features/cieCfg_cam_lib.jar:/u01/oracle/oui/modules/gdr-external.jar:/u01/oracle/oracle_common/modules/oracle.jps/jps-manifest.jar:/u01/oracle/oui/modules/private/xml-apis-1.4.01.jar
java.class.version=61.0
java.home=/u01/jdk
java.io.tmpdir=/tmp
java.library.path=/u01/oracle/oracle_common/lib:/u01/oracle/oracle_common/adr::/usr/java/packages/lib:/usr/lib64:/lib64:/lib:/usr/lib
java.runtime.name=Java(TM) SE Runtime Environment
java.runtime.version=17.0.12+8-LTS-286
java.security.egd=file:/dev/urandom
java.specification.name=Java Platform API Specification
java.specification.vendor=Oracle Corporation
java.specification.version=17
java.vendor=Oracle Corporation
java.vendor.url=https://java.oracle.com/
java.vendor.url.bug=https://bugreport.java.com/bugreport/
java.version=17.0.12
java.version.date=2024-07-16
java.vm.compressedOopsMode=32-bit
java.vm.info=mixed mode, sharing
java.vm.name=Java HotSpot(TM) 64-Bit Server VM
java.vm.specification.name=Java Virtual Machine Specification
java.vm.specification.vendor=Oracle Corporation
java.vm.specification.version=17
java.vm.vendor=Oracle Corporation
java.vm.version=17.0.12+8-LTS-286
jdk.debug=release
line.separator=

native.encoding=ANSI_X3.4-1968
oracle.domain.config.dir=/u01/oracle/user_projects/domains/governancedomain/config/fmwconfig
oracle.security.jps.config=/u01/oracle/user_projects/domains/governancedomain/config/fmwconfig/jps-config-jse.xml
oracle.webservice.policy.config=/u01/oracle/user_projects/domains/governancedomain/config/fmwconfig
org.jboss.logging.provider=jdk
org.xml.sax.driver=com.sun.org.apache.xerces.internal.parsers.SAXParser
os.arch=amd64

```
os.name=Linux
os.version=5.15.0-302.167.6.el8uek.x86_64
path.separator=:
sun.arch.data.model=64
sun.boot.library.path=/u01/jdk/lib
sun.cpu.endian=little
sun.io.unicode.encoding=UnicodeLittle
sun.java.command=oracle.ias.update.UpgradeDriver -configUpgrade -response /u01/scripts/
OIGDomainConfigResponse.txt -logLevel TRACE -logDir /tmp
sun.java.launcher=SUN_STANDARD
sun.jnu.encoding=ANSI_X3.4-1968
sun.lang.ClassLoader.allowArraySyntax=true
sun.management.compiler=HotSpot 64-Bit Tiered Compilers
sun.stderr.encoding=ANSI_X3.4-1968
sun.stdout.encoding=ANSI_X3.4-1968
ua.home=/u01/oracle/oracle_common
ua.wl.home=/u01/oracle/wlserver
user.country=US
user.dir=/u01/scripts
user.home=/home/oracle
user.language=en
user.name=oracle
user.timezone=UTC

STARTUP - INITED STATE
isODLAvailable=true, isDMSAvailable=true, isDMS4LoggingAvailable=true, isJRFAvailable=true,
isMDSAvailable=true, isDFWAvailable=true, isJEE=false, platName=jse, isWebLogic=false,
isWebLogicCore=false, isWebSphere=false, isWebSphereND=false, isWebSphereAS=false, isJBoss=false,
isGlassfish=false, isTomcat=false, isGrizzly=false, isMicroServicePlatform=false, isOciNative=false,
breezeVer=, breezeStat=SUCCESS - !isClassAvailable(com/oracle/breeze/Breeze)
 Oracle Web Services Manager configuration examine finished with status: ready for upgrade
 User Messaging Service configuration examine is in progress
 User Messaging Service configuration examine finished with status: ready for upgrade
 Oracle Web Services Manager configuration upgrade is in progress
 Oracle Web Services Manager configuration upgrade finished with status: succeeded
 User Messaging Service configuration upgrade is in progress
 User Messaging Service configuration upgrade finished with status: succeeded
```

> **Note**
>
> The java.lang.Throwable: STARTUP - DIAGNOSTIC STACK message can be ignored.

11. Exit the domain-upgrade pod:

```
exit
```

16.2.9 Updating the OIG Container Image to 14c

You must update the deployment to use the Oracle Identity Governance (OIG) 14c container image:

> **Note**
>
> If the container image upgrade fails, see Restoring After a Failed Upgrade.

1. Run the following command to update the deployment with the OIG 14c container image:

   ```
   kubectl patch domain %DOMAIN_UID% -n %NAMESPACE% --type merge  -p '{"spec":
   {"image":"%WEBLOGIC_IMAGE%"}}'
   ```

   For example:

   - If using Oracle Container Registry or your own container registry for your OIG container image:

     ```
     kubectl patch domain governancedomain -n oigns --type merge  -p '{"spec":{"image":"container-
     registry.oracle.com/middleware/oig_cpu:<new_tag>"}}'
     ```

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

     ```
     kubectl patch domain governancedomain -n oigns --type merge  -p '{"spec":{"image":"oracle/
     oig:14.1.2.1.0"}}'
     ```

     The output will look similar to the following:

     ```
     domain.weblogic.oracle/governancedomain patched
     ```

16.2.10 Updating the WebLogic Kubernetes Operator

You must update the deployment to use a WebLogic Kubernetes Operator version supported with 14c. The current supported version is 4.2.10.

1. Run the following command to see the current version of the WebLogic Kubernetes Operator:

   ```
   helm list -n opns
   ```

   The output will look similar to the following:

   ```
   NAME                    NAMESPACE      REVISION       UPDATED   STATUS
   CHART                          APP VERSION
   weblogic-kubernetes-operator    opns          1          <DATE>    deployed       weblogic-operator-4.1.8-
   RELEASE-MARKER  4.1.8-RELEASE-MARKER
   ```

   In the above example the version is 4.1.8 and therefore the operator must be upgraded.

2. To upgrade the WebLogic Kubernetes Operator to 4.2.10, see Upgrading WebLogic Kubernetes Operator.
3. Once the operator is upgraded, continue with Starting the OIG 14c Deployment.

16.2.11 Starting the OIG 14c Deployment

Start the Oracle Identity Governance (OIG) 14c deployment.

1. Run the following command to start the OIG domain:

   ```
   kubectl patch domain.v9.weblogic.oracle "%DOMAIN_UID%" -n "%NAMESPACE%" \
   --type=merge --patch "{\"spec\": {\"serverStartPolicy\": \"IfNeeded\"}}"
   ```

   For example:

   ```
   kubectl patch domain.v9.weblogic.oracle "governancedomain" -n "oigns" \
   --type=merge --patch "{\"spec\": {\"serverStartPolicy\": \"IfNeeded\"}}"
   ```

   The output will look similar to the following:

   ```
   domain.weblogic.oracle/governancedomain patched
   ```

2. Run the following command to view the pods:

   ```
   kubectl get pods -n %NAMESPACE%
   ```

   For example:

   ```
   kubectl get pods -n oigns
   ```

   The output will look similar to the following:

   ```
   NAME                                READY   STATUS      RESTARTS   AGE
   governancedomain-introspector-jwqxw             1/1     Running     0          10s
   ```

   The introspect job will start, followed by the Administration Server pod, and then the OIG server pods. This process will take several minutes, so keep executing the command until all the pods are running with READY status 1/1:

   > **Note**
   >
   > Alternatively, you can add the watch flag, -w, which allows you watch the status of the pods as they change.

   ```
   NAME                          READY   STATUS    RESTARTS   AGE
   governancedomain-adminserver   1/1     Running   0          16m
   governancedomain-soa-server1   1/1     Running   0          13m
   governancedomain-oim-server1   1/1     Running   0          13m
   ```

3. Once everything is running, check the consoles are accessible as per Validating the Domain URLs.

4. Once you are confident the upgrade is successful, delete the domain-upgrade pod and ConfigMap as follows:

   kubectl delete pod *%DOMAIN_UID%*-domain-upgrade -n *%NAMESPACE%*

   kubectl delete configmap *%DOMAIN_UID%*-domain-upgrade-pod-cm -n *%NAMESPACE%*

   For example:

   kubectl delete pod governancedomain-domain-upgrade -n oigns

   kubectl delete configmap governancedomain-domain-upgrade-pod-cm -n oigns

16.2.12 Upgrading the Ingress

In order to access the Oracle Identity Governance (OIG) 14c domain via WebLogic Remote Console, you must upgrade the ingress.

1. Download the latest code repository to a new directory and set the $WORKDIR to the new directory structure. See, Setting Up the Code Repository for OIG.

   > **Note**
   >
   > Make sure not to delete the original OIG 12c code repository as you will need the values.yaml used to create the original ingress.

2. Navigate to the following directory:

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

3. Make a copy of the values.yaml:

   cp values.yaml $WORKDIR/

4. Copy over the values.yaml from the original OIG 12c code repository. For example:

   cp /OIGK8S/fmw-kubernetes/OracleIdentityGovernance/kubernetes/charts/ingress-per-domain \
   $WORKDIR/kubernetes/charts/ingress-per-domain

5. Upgrade the governancedomain-nginx with the following commands:

   cd $WORKDIR

   helm upgrade governancedomain-nginx kubernetes/charts/ingress-per-domain/ --namespace *%NAMESPACE%* \
   --values kubernetes/charts/ingress-per-domain/values.yaml --reuse-values

For example:

```
helm upgrade governancedomain-nginx kubernetes/charts/ingress-per-domain/ --namespace oigns \
--values kubernetes/charts/ingress-per-domain/values.yaml --reuse-values
```

The output will look similar to the following:

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

6. Check you can connect to the OIG 14c domain using the WebLogic Remote Console.

> **Note**
>
> For more information about installing and configuring the console, see Getting Started Using Administration Console.

16.2.13 Restoring After a Failed Upgrade

If the upgrade fails at any point, you can restore back to the Oracle Identity Governance 12c deployment using the following steps:

1. Shut down the OIG 14c deployment using the following command:

```
kubectl patch domain <domain> -n <domain_namespace> --type=merge --patch "{\"spec\":
{\"serverStartPolicy\": \"Never\"}}"
```

For example:

```
kubectl patch domain governancedomain -n oigns --type=merge --patch "{\"spec\": {\"serverStartPolicy\":
\"Never\"}}"
```

The output will look similar to the following:

```
domain.weblogic.oracle/governancedomain patched
```

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

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

For example:

```
kubectl get pods -n oigns
```

The output will look similar to the following:

```
NAME                          READY   STATUS       RESTARTS   AGE
governancedomain-adminserver   1/1     Terminating   0          16m
governancedomain-soa-server1   1/1     Terminating   0          13m
governancedomain-oim-server1   1/1     Terminating   0          5m22s
```

The Administration Server pods and Managed Server pods will move to a STATUS of Terminating.

After a few minutes, run the command again and make sure the pods should have disappeared before continuing.

3. Restore the persistent volume from the backup taken before the upgrade:

```
sudo cp -rp <persistent_volume>/governancedomainpv <persistent_volume>/governancedomain_bkp14c

sudo rm -rf <persistent_volume>/governancedomainpv

sudo cp -rp <persistent_volume>/governancedomainpv_bkp12c <persistent_volume>/governancedomain
```

For example:

```
sudo cp -rp /nfs_volumes/oig/governancedomainpv /nfs_volumes/oig/governancedomain_bkp14c

sudo rm -rf /nfs_volumes/oig/governancedomainpv

sudo cp -rp /nfs_volumes/oig/governancedomainpv_bkp12c /nfs_volumes/oig/governancedomain
```

4. Restore the Oracle Database from the backup taken before the upgrade.
5. Run the following command to update the deployment with the OIG 12c container image used previously:

```
kubectl patch domain %DOMAIN_UID% -n %NAMESPACE% --type merge  -p '{"spec":
{"image":"%WEBLOGIC_IMAGE%"}}'
```

For example:

- If using Oracle Container Registry or your own container registry for your OIG container image:

```
kubectl patch domain governancedomain -n oigns \
--type merge  -p '{"spec":{"image":"container-registry.oracle.com/middleware/oig_cpu:12.2.1.4-jdk8-ol8-
<YYMMDD>"}}'
```

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

```
kubectl patch domain governancedomain -n oigns \
--type merge  -p '{"spec":{"image":"oracle/oig:12.2.1.4.0"}}'
```

The output will look similar to the following:

```
domain.weblogic.oracle/governancedomain patched
```

6. Run the following command to start the OIG domain:

```
kubectl patch domain.v9.weblogic.oracle "%DOMAIN_UID%" -n "%NAMESPACE%" \
--type=merge --patch "{\"spec\": {\"serverStartPolicy\": \"IfNeeded\"}}"
```

For example:

```
kubectl patch domain.v9.weblogic.oracle "governancedomain" -n "oigns" \
--type=merge --patch "{\"spec\": {\"serverStartPolicy\": \"IfNeeded\"}}"
```

The output will look similar to the following:

```
domain.weblogic.oracle/governancedomain patched
```

7. Downgrade the WebLogic Kubernetes Operator to a version supported by OIG 12c. Follow the instructions at Updating the WebLogic Kubernetes Operator, but use a supported operator for OIG 12c.
8. Run the following command to view the pods:

```
kubectl get pods -n %NAMESPACE%
```

For example:

```
kubectl get pods -n oigns
```

The output will look similar to the following:

```
NAME                                READY   STATUS      RESTARTS   AGE
governancedomain-introspector-jwqxw             1/1     Running     0          10s
```

The introspect job will start, followed by the Administration Server pod, and then the OIG server pods. This process will take several minutes, so keep executing the command until all the pods are running with READY status 1/1:

> **Note**
>
> Alternatively, you can add the watch flag, -w, which allows you watch the status of the pods as they change.

```
NAME                             READY   STATUS    RESTARTS   AGE
governancedomain-adminserver     1/1     Running   0          16m
governancedomain-soa-server1     1/1     Running   0          13m
governancedomain-oim-server1     1/1     Running   0          5m22s
```

9. Once everything is running, check the consoles are accessible as per Validating the Domain URLs.

17

General Troubleshooting

This chapter includes the following topics:

17.1 Viewing Pod Logs

To view logs for a pod use the following command:

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

For example:

```
kubectl logs governancedomain-oim-server1 -n oigns
```

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

17.2 Viewing Pod Descriptions

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

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

For example:

```
kubectl describe pod governancedomain-oim-server1 -n oigns
```

The output will look similar to the following:

```
Name:           governancedomain-oim-server1
Namespace:      oigns
Priority:       0
Service Account:  default
Node:           worker-2/100.105.211.87
Start Time:     <DATE>
Labels:         weblogic.clusterName=oim_cluster
                weblogic.clusterObservedGeneration=2
```

```
                weblogic.createdByOperator=true
                weblogic.domainName=governancedomain
                weblogic.domainObservedGeneration=3
                weblogic.domainUID=governancedomain
                weblogic.operatorVersion=4.2.10
                weblogic.serverName=oim_server1
Annotations:    prometheus.io/path: /wls-exporter/metrics
                prometheus.io/port: 14000
                prometheus.io/scrape: true
                weblogic.sha256: ca63e88a762186c5b4ca394be1c7c3b3ca0f0474c91bb3c8b6a928546aeafd35
Status:         Running
SeccompProfile: RuntimeDefault
IP:             10.244.2.251
IPs:
  IP:           10.244.2.251
Controlled By:  Domain/governancedomain
Init Containers:
  compat-connector-init:
    Container ID:  cri-o://dbd6a9ef07c3b9cbaa30685e6382f587a4be5461f1ada4750a947e00349e5c2e
    Image:         container-registry.oracle.com/middleware/oig_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD>
    Image ID:      container-registry.oracle.com/middleware/
oam_cpu@sha256:b52e106cdb9b522e9a7a971888481af3869542ef433ba3ea9cd6d47b323c781b
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/bash
      -c
      mkdir -p  /u01/oracle/user_projects/domains/ConnectorDefaultDirectory
      mkdir -p  /u01/oracle/user_projects/domains/wdt-logs
    State:         Terminated
      Reason:      Completed
      Exit Code:   0
      Started:     <DATE>
      Finished:    <DATE>
    Ready:         True
    Restart Count: 0
    Limits:
      cpu:     2
      memory:  8Gi
    Requests:
      cpu:       1
      memory:    4Gi
    Environment:  <none>
    Mounts:
      /u01/oracle/user_projects from weblogic-domain-storage-volume (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-qxpt4 (ro)
Containers:
  weblogic-server:
    Container ID:  cri-o://80b0b0d9241ae88e57a7ceb5d81d0716f1b812d70d5e7c1b1ee148495559a8b3
    Image:         container-registry.oracle.com/middleware/oig_cpu:14.1.2.1.0-jdk17-ol8-<YYMMDD>
    Image ID:      container-registry.oracle.com/middleware/
oig_cpu@sha256:b52e106cdb9b522e9a7a971888481af3869542ef433ba3ea9cd6d47b323c781b
    Ports:         14002/TCP, 14000/TCP
    Host Ports:    0/TCP, 0/TCP
    Command:
      /weblogic-operator/scripts/startServer.sh
```

```
    State:          Running
      Started:      <DATE>
    Last State:     Terminated
      Reason:       Error
      Exit Code:    137
      Started:      <DATE>
      Finished:     <DATE>
    Ready:          True
    Restart Count:  1
    Limits:
      cpu:     2
      memory:  8Gi
    Requests:
      cpu:      1
      memory:   4Gi
    Liveness:   exec [/weblogic-operator/scripts/livenessProbe.sh] delay=30s timeout=5s period=45s #success=1
#failure=1
    Readiness:  http-get http://:14000/weblogic/ready delay=30s timeout=5s period=5s #success=1 #failure=1
    Environment Variables from:
      governancedomain-rcu-credentials  Secret  Optional: false
    Environment:
      USER_MEM_ARGS:                    -Djava.security.egd=file:/dev/./urandom -Xms8192m -Xmx8192m
      JAVA_OPTIONS:                     -Dweblogic.StdoutDebugEnabled=false
      WLSDEPLOY_LOG_DIRECTORY:          /u01/oracle/user_projects/domains/wdt-logs
      FRONTENDHOST:                     example.com
      FRONTENDPORT:                     14000
      WLSDEPLOY_PROPERTIES:             -Dwdt.config.disable.rcu.drop.schema=true
      DOMAIN_NAME:                      governancedomain
      DOMAIN_HOME:                      /u01/oracle/user_projects/domains/governancedomain
      ADMIN_NAME:                       AdminServer
      ADMIN_PORT:                       7001
      SERVER_NAME:                      oim_server1
      DOMAIN_UID:                       governancedomain
      NODEMGR_HOME:                     /u01/nodemanager
      LOG_HOME:                         /u01/oracle/user_projects/domains/logs/governancedomain
      SERVER_OUT_IN_POD_LOG:            true
      SERVICE_NAME:                     governancedomain-oim-server1
      AS_SERVICE_NAME:                  governancedomain-adminserver
      ADMIN_USERNAME:
      ADMIN_PASSWORD:
      LOCAL_ADMIN_PORT:                 14000
      LOCAL_ADMIN_PROTOCOL:             t3
      SHUTDOWN_TYPE:                    Graceful
      SHUTDOWN_TIMEOUT:                 30
      SHUTDOWN_IGNORE_SESSIONS:         false
      REPLACE_VARIABLES_IN_JAVA_OPTIONS:  false
      DYNAMIC_CONFIG_OVERRIDE:          true
      DOMAIN_SOURCE_TYPE:               PersistentVolume
    Mounts:
      /u01/oracle/user_projects from weblogic-domain-storage-volume (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-qxpt4 (ro)
      /weblogic-operator/debug from weblogic-domain-debug-cm-volume (ro)
      /weblogic-operator/introspector from weblogic-domain-introspect-cm-volume (rw)
      /weblogic-operator/scripts from weblogic-scripts-cm-volume (ro)
  Conditions:
    Type                Status
```

```
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  weblogic-scripts-cm-volume:
    Type:      ConfigMap (a volume populated by a ConfigMap)
    Name:      weblogic-scripts-cm
    Optional:  false
  weblogic-domain-debug-cm-volume:
    Type:      ConfigMap (a volume populated by a ConfigMap)
    Name:      governancedomain-weblogic-domain-debug-cm
    Optional:  true
  weblogic-domain-introspect-cm-volume:
    Type:      ConfigMap (a volume populated by a ConfigMap)
    Name:      governancedomain-weblogic-domain-introspect-cm
    Optional:  false
  weblogic-domain-storage-volume:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  governancedomain-domain-pvc
    ReadOnly:   false
  kube-api-access-qxpt4:
    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:
```

17.3 Known Issues

This section contains information about known issues.

Domain Creation Failure With WLST

The instructions in this section relate to problems creating Oracle Identity Governance (OIG) domains using WLST. See, Creating OIG Domains Using WLST Offline Scripts.

If the OIG domain creation fails, run the following to diagnose the issue:

```
kubectl logs <domain_job> -n <domain_namespace>
```

For example:

```
kubectl logs governancedomain-create-fmw-infra-sample-domain-job-c6vfb -n oigns
```

Also run:

```
kubectl describe pod <domain_job> -n <domain_namespace>
```

For example:

```
kubectl describe pod governancedomain-create-fmw-infra-sample-domain-job-c6vfb -n oigns
```

Using the output you should be able to diagnose the problem and resolve the issue.

If any of the above commands return the following error:

```
Failed to start container "create-fmw-infra-sample-domain-job": Error response from daemon: error while creating
mount source path
'/nfs_volumes/oig/governancedomainpv ': mkdir /nfs_volumes/oig/governancedomainpv : permission denied
```

Then there is a permissions error on the directory for the PV and PVC and the following should be checked:

- The directory has 777 permissions: `chmod -R 777 <persistent_volume>/governancedomainpv`.
- If it does have the permissions, check if an oracle user exists and the uid is 1000 and gid is 0.
  Create the oracle user if it doesn't exist and set the uid to 1000 and gid to 0.
- Edit the `$WORKDIR/kubernetes/create-weblogic-domain-pv-pvc/create-pv-pvc-inputs.yaml` and add a slash to the end of the directory for the `weblogicDomainStoragePath` parameter:

  ```
  weblogicDomainStoragePath: /nfs_volumes/oig/governancedomainpv/
  ```

Once you have diagnosed the problem, clean down the failed domain creation by following:

- Deleting the OIG Domain
- Deleting RCU Schemas
- Deleting Persistent Volume Contents

Then follow the instructions again in Creating OIG Domains Using WLST Offline Scripts

Domain Creation Failure With WDT Models

The instructions in this section relate to problems creating OIG domains using WDT models. See, Creating OIG Domains Using WDT Models.

If the domain creation fails while creating domain resources using the `domain.yaml` file, run the following steps to diagnose the issue:

1. Check the domain events, by running the following command:

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

   For example:

   ```
   kubectl describe domain governancedomain -n oigns
   ```

   Using the output, you should be able to diagnose the problem and resolve the issue.

2. If the introspect job fails due to validation errors, then you can recreate the domain resources using the commands:

   ```
   kubectl delete -f domain.yaml
   ```

   ```
   kubectl create -f domain.yaml
   ```

3. If the domain creation fails because of database issues:

   a. Create a helper pod:

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

   b. Once you have diagnosed the problem, clean down the failed domain creation by following:

      - Deleting the OIG Domain
      - Deleting RCU Schemas
      - Deleting Persistent Volume Contents

   c. Execute the steps in Creating OIG Domains Using WDT Models again.

   > **Note**
   >
   > You might need to recreate the domain creation image depending upon the errors. Domain creation logs are stored in <persistent_volume>/domains/wdt-logs.

4. If there is any issues bringing up the administration server, or OIG managed server pods, you can run the following to check the logs:

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

   For example:

   ```
   kubectl logs governancedomain-adminserver -n oigns
   ```

   If the above does not give any information you can also run:

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

   For example:

   ```
   kubectl describe pod governancedomain-adminserver -n oigns
   ```

For more details related to debugging issues, refer to Domain Debugging.

Pods Restarting Due to LivenessProbe

If the server pods keep restarting due to livenessProbe or readinessProbe failure, then make the following changes in the oim-cluster and soa-cluster respectively.

If the restart is due to resource (CPU in this case) limit issue, then the CPU parameter need to be adjusted as needed:

```
kubectl edit cluster <cluster> -n <domain_namespace>
```

For example:

```
kubectl edit cluster governancedomain-oim-cluster -n oigns
```

In the edit session change the CPU parameter as follows:

```
spec:
  clusterName: oim_cluster
  replicas: 1
  serverPod:
    env:
    - name: USER_MEM_ARGS
      value: '-Djava.security.egd=file:/dev/./urandom -Xms4096m -Xmx10240m -
Dweblogic.rjvm.allowUnknownHost=true '
    livenessProbe:
      failureThreshold: 3
      periodSeconds: 60
      timeoutSeconds: 60
    readinessProbe:
      failureThreshold: 3
      periodSeconds: 60
      timeoutSeconds: 60
    resources:
      limits:
        cpu: 1700m
        memory: 6Gi
      requests:
        cpu: 500m
        memory: 4Gi
```

Patch Domain Failures

The instructions in this section relate to problems patching a deployment with a new image as per Patching a Container Image.

If the OIG domain patching fails when running patch_oig_domain.sh, run the following to diagnose the issue:

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

For example:

```
kubectl describe domain governancedomain -n oigns
```

Using the output you should be able to diagnose the problem and resolve the issue.

If the domain is already patched successfully and the script failed at the last step of waiting for pods to come up with the new image, then you do not need to rerun the script again after issue resolution. The pods will come up automatically once you resolve the underlying issue.

If the script is stuck at the following message for a long time:

```
[INFO] Waiting for weblogic pods to be ready..This may take several minutes, do not close the window. Check log /scratch/OIGK8Slatest/fmw-kubernetes/OracleIdentityGovernance/kubernetes/domain-lifecycle/log/oim_patch_log-<DATE>/monitor_weblogic_pods.log for progress
```

Run the following command to diagnose the issue:

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

For example:

```
kubectl get pods -n oigns
```

Run the following to check the logs of the Admininstration Server, SOA server or OIM server pods, as there may be an issue that is not allowing the domain pods to start properly:

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

If the above does not glean any information you can also run:

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

Further diagnostic logs can also be found under the $WORKDIR/kubernetes/domain-lifecycle.

Once any issue is resolved the pods will come up automatically without the need to rerun the script.

18

Deleting an OIG Deployment

This chapter explains how to delete the Oracle Identity Governance (OIG) domain and other Kubernetes objects used by the OIG domain.

The instructions in this chapter should only be followed if you need to remove a certain part of the domain because of a deployment failure, or if you need clear the domain down completely for some other reason. If you are unsure consult Oracle Support.

This chapter includes the following topics:

18.1 Deleting the OIG Domain

The steps to delete an Oracle Identity Governance (OIG) domain depends on whether the domain was created with WLST or WDT.

Deleting WLST OIG Domains

1. Navigate to the $WORKDIR/kubernetes/delete-domain directory:

   ```
   cd $WORKDIR/kubernetes/delete-domain
   ```

2. Run the following command to delete the domain:

   ```
   ./delete-weblogic-domain-resources.sh -d <domain_uid>
   ```

   For example:

   ```
   ./delete-weblogic-domain-resources.sh -d governancedomain
   ```

Deleting WDT OIG Domains

1. Run the following command to delete the domain and clusters:

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

2. Navigate to the $WORKDIR/kubernetes/delete-domain directory:

   ```
   cd $WORKDIR/kubernetes/delete-domain
   ```

3. Run the following command to remove other domain objects:

```
./delete-weblogic-domain-resources.sh -d <domain_uid>
```

For example:

```
./delete-weblogic-domain-resources.sh -d governancedomain
```

18.2 Deleting RCU Schemas

To delete the RCU schemas, perform the following steps:

1. Check to see if the helper pod exists by running:

```
kubectl get pods -n <domain_namespace> | grep helper
```

For example:

```
kubectl get pods -n oigns | grep helper
```

The output should look similar to the following:

```
helper                          1/1     Running   0          26h
```

If the helper pod doesn't exist, run the following:

- If using Oracle Container Registry or your own container registry for the Oracle Identity Governance (OIG) container image:

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

For example:

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

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

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

For example:

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

The output will look similar to the following:

```
pod/helper created
```

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

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

For example:

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

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

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

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

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

export RCUPREFIX=<rcu_schema_prefix>

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

Where:

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

For example:

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

export RCUPREFIX=OIGK8S

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

cat /tmp/pwd.txt
```

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

4. In the helper bash shell, drop the RCU schemas as follows:

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

5. Exit the helper bash shell by issuing the command exit.

18.3 Deleting Persistent Volume Contents

Perform the following step to delete the persistent volume contents:

```
rm -rf <persistent_volume>/governancedomainpv/*
```

For example:

```
rm -rf /nfs_volumes/oig/governancedomainpv/*
```

18.4 Deleting the WebLogic Kubernetes Operator

To delete the WebLogic Kubernetes Operator, perform the following steps:

1. Run the following command to remove the operator:

   ```
   helm delete weblogic-kubernetes-operator -n opns
   ```

2. Delete the label from the OIG namespace::

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

   For example:

   ```
   kubectl label namespaces oigns weblogic-operator-
   ```

3. Delete the service account for the operator:

   ```
   kubectl delete serviceaccount <sample-kubernetes-operator-sa> -n <domain_namespace>
   ```

   For example:

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

4. Delete the operator namespace:

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

For example:

```
kubectl delete namespace opns
```

18.5 Deleting the Ingress

Perform the following steps to delete the ingress and ingress controller:

1. To delete the ingress:

   ```
   helm delete governancedomain-nginx -n <domain_namespace>
   ```

   For example:

   ```
   helm delete governancedomain-nginx -n oigns
   ```

2. To delete the ingress controller:

   ```
   helm delete nginx-ingress -n <domain_namespace>
   ```

   For example:

   ```
   helm delete nginx-ingress -n mynginxns
   ```

3. Delete the namespace using the following command:

   ```
   kubectl delete namespace <domain_namespace>
   ```

   For example:

   ```
   kubectl delete namespace mynginxns
   ```

18.6 Deleting the OIG Namespace

Perform the following step to delete the Oracle Identity Governance (OIG) namespace:

1. Delete the helper pod if it is running:

   ```
   kubectl delete pod helper -n <domain_namepace>
   ```

   For example:

   ```
   kubectl delete pod helper -n oigns
   ```

2. Check to make sure all Kubernetes in the namespace are deleted:

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

For example:

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

If any objects remain, delete them manually.

3. Delete the namespace using the following command:

```
kubectl delete namespace <domain_namespace>
```

For example:

```
kubectl delete namespace oigns
```