Oracle® Fusion Middleware

Oracle SOA Suite on Kubernetes

14c (14.1.2.0.0)
F99931-02

Oracle Fusion Middleware Oracle SOA Suite on Kubernetes, 14c (14.1.2.0.0)

F99931-02

Contents

1 Abstract

This guide describes how to provision and manage Oracle SOA Suite instances on Kubernetes.

2

Preface

The WebLogic Kubernetes Operator (the “operator”) supports deployment of Oracle SOA Suite components such as Oracle Service-Oriented Architecture (SOA), Oracle Service Bus, and Oracle Enterprise Scheduler (ESS).

Topics:

Audience

This guide is intended for users who want to create and manage Oracle SOA Suite instances on Kubernetes.

Related Resources

For more information, see these Oracle resources:

- Oracle Cloud at `http://cloud.oracle.com`.
- Fusion Middleware Documentation

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `https://www.oracle.com/corporate/accessibility/`.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `https://support.oracle.com/portal/` or visit `Oracle Accessibility Learning and Support` if you are hearing impaired.

Conventions

The following text conventions are used in this document.

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

3

Oracle SOA Suite

The WebLogic Kubernetes Operator (the “operator”) supports deployment of Oracle SOA Suite components such as Oracle Service-Oriented Architecture (SOA), Oracle Service Bus, and Oracle Enterprise Scheduler (ESS).

Currently the operator supports these domain types:

- `soa`: Deploys a SOA domain with Oracle Enterprise Scheduler (ESS)
- `osb`: Deploys an Oracle Service Bus domain
- `soaosb`: Deploys a domain with SOA, Oracle Service Bus, and Oracle Enterprise Scheduler (ESS)

In this release, Oracle SOA Suite domains are supported using the “domain on a persistent volume” model only, where the domain home is located in a persistent volume (PV).

The operator has several key features to assist you with deploying and managing Oracle SOA Suite domains in a Kubernetes environment. You can:

- Create Oracle SOA Suite instances in a Kubernetes persistent volume (PV). This PV can reside in an NFS file system or other Kubernetes volume types.
- Start servers based on declarative startup parameters and desired states.
- Expose the Oracle SOA Suite services and composites for external access.
- Scale Oracle SOA Suite domains by starting and stopping Managed Servers on demand, or by integrating with a REST API.
- Publish operator and WebLogic Server logs to Elasticsearch and interact with them in Kibana.
- Monitor the Oracle SOA Suite instance using Prometheus and Grafana.

Topics in this section:

- Recent changes and known issues
- Pricing and licensing
- Limitations
- About this documentation
- Additional reading

Recent changes and known issues

See the Release Notes for recent changes and known issues for Oracle SOA Suite domains deployment on Kubernetes.

Pricing and licensing

See here for pricing and licensing details.

Limitations

See here for limitations in this release.

About this documentation

This documentation includes sections targeted to different audiences. To help you find what you are looking for more easily, please consult this table of contents:

- Quick Start explains how to quickly get an Oracle SOA Suite domain instance running using default settings. Note that this is only for development and test purposes.
- Install Guide and Administration Guide provide detailed information about all aspects of using the Kubernetes operator including:
  - Installing and configuring the operator.
  - Using the operator to create and manage Oracle SOA Suite domains.
  - Configuring Kubernetes load balancers.
  - Configuring custom SSL certificates.
  - Configuring Elasticsearch and Kibana to access the operator and WebLogic Server log files.
  - Deploying composite applications for Oracle SOA Suite and Oracle Service Bus.
  - Patching an Oracle SOA Suite Docker image.
  - Removing domains.
  - And much more!

Additional reading

Oracle SOA Suite domains deployment on Kubernetes leverages the WebLogic Kubernetes Operator framework.

- To develop an understanding of the operator, including design, architecture, domain life cycle management, and configuration overrides, review the operator documentation.
- To learn more about the Oracle SOA Suite architecture and components, see Understanding Oracle SOA Suite.
- To review the known issues and common questions for Oracle SOA Suite domains deployment on Kubernetes, see the Frequently Asked Questions

4 Release Notes

Review the release notes for Oracle SOA Suite on Kubernetes.

Date	Version	Change
December 2024	14.1.2.0.0 GitHub release version 24.4.3	First release of Oracle SOA Suite on Kubernetes 14.1.2.0.0.

5
Install Guide

Install the WebLogic Kubernetes Operator and prepare and deploy Oracle SOA Suite domains.

- Requirements and pricing

  Understand the system requirements, limitations, licensing, and pricing for deploying and running Oracle SOA Suite domains with the WebLogic Kubernetes Operator, including the SOA cluster sizing.
- Prepare your environment

  Prepare for creating Oracle SOA Suite domains, including required secrets creation, persistent volume and volume claim creation, database creation, and database schema creation.
- Create Oracle SOA Suite domains

  Create an Oracle SOA Suite domain home on an existing PV or PVC, and create the domain resource YAML file for deploying the generated Oracle SOA Suite domain.

Requirements and Pricing

This section provides information about the system requirements, limitations, licensing, and pricing for deploying and running Oracle SOA Suite domains with the WebLogic Kubernetes Operator.

Topics in this section:

- System requirements for Oracle SOA Suite domains
- Limitations
- Pricing and Licensing

System requirements for Oracle SOA Suite domains

- WebLogic Kubernetes Operator 4.2.9. See operator releases 4.2.9.
- Review Operator Prerequisites for supported Kubernetes, Docker, Flannel and Calico versions.
- You must have the cluster-admin role to install the operator. The operator does not need the cluster-admin role at runtime. For more information, see the role-based access control (RBAC) documentation.
- Helm 3.10.2+ (check with `helm version --client --short`).
- Container images based on Oracle Linux 8 and Oracle Linux 9 with JDK 17 and JDK 21 are supported.
- Additionally, see the Oracle SOA Suite documentation for other requirements, such as database version.

See here for resource sizing information for Oracle SOA Suite domains set up on a Kubernetes cluster.

Limitations

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

- In this release, Oracle SOA Suite 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” and “model in image” models are not supported. Also, “WebLogic Deploy Tooling (WDT)” based deployments are currently not supported.
- Only configured clusters are supported. Dynamic clusters are not supported for Oracle SOA Suite 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. Mixed clusters (configured servers targeted to a dynamic cluster) are not supported.
- The WebLogic Logging Exporter project has been archived. Users are encouraged to use Fluentd or Logstash.
- The WebLogic Monitoring Exporter currently supports WebLogic MBean trees only. Support for JRF and Oracle SOA Suite MBeans is not available. Also, a metrics dashboard specific to Oracle SOA Suite is not available. Instead, use the WebLogic Server dashboard to monitor the Oracle SOA Suite server metrics in Grafana.
- Some features such as multicast, multitenancy, production redeployment, and Node Manager (although it is used internally for the liveness probe and to start WebLogic Server instances) are not supported in this release.
- Features such as Java Messaging Service whole server migration and consensus leasing are not supported in this release.
- Enabling or disabling the memory resiliency for Oracle Service Bus using the Enterprise Manager Console is not supported in this release.

For up-to-date information about the features of WebLogic Server that are supported in Kubernetes environments, see My Oracle Support Doc ID 2349228.1.

Pricing and licensing

The WebLogic Kubernetes Operator and Oracle Linux are open source and free; WebLogic Server requires licenses in any environment. All WebLogic Server licenses are suitable for deploying WebLogic to containers and Kubernetes, including free single desktop Oracle Technology Network (OTN) developer licenses. See the following sections for more detailed information:

- Oracle SOA Suite
  Oracle SOA Suite is licensed as an option to Oracle WebLogic Suite. Valid licenses are needed in at least one of the following combinations:
  - WebLogic Suite and Oracle SOA Suite
  - WebLogic Suite and Oracle Service Bus
  - WebLogic Suite and Oracle BPEL Engine

  For more information, see the Fusion Middleware Licensing Information User Manual - Oracle SOA Suite.
- Oracle Linux
  Oracle Linux is under open source license and is completely free to download and use.

  Note that Oracle SOA Suite licenses that include support do not include customer entitlements for direct access to Oracle Linux support or Unbreakable Linux Network (to directly access the standalone Oracle Linux patches). The latest Oracle Linux patches are included with the latest Oracle SOA Suite images.

- Oracle Java
  Oracle support for Java is included with Oracle SOA Suite licenses when Java is used for running WebLogic and Coherence servers or clients.

  For more information, see the Fusion Middleware Licensing Information User Manual.
- Oracle SOA Suite images
  Oracle provides two different types of Oracle SOA Suite images:
  - Critical Patch Update (CPU) images: Images with the latest Oracle SOA Suite, Fusion Middleware Infrastructure, Coherence PSUs, and other fixes released by the Critical Patch Update (CPU) program. CPU images are intended for production use.
  - General Availability (GA) images: Images that are not intended for production use and do not include Oracle SOA Suite, WebLogic, Fusion Middleware Infrastructure, or Coherence PSUs.

  All Oracle SOA Suite licenses, including free Oracle Technology Network (OTN) developer licenses, include access to the latest General Availability (GA) Oracle SOA Suite images, which bundle Java SE.

  Customers with access to Oracle SOA Suite support additionally have:
  - Access to Critical Patch Update (CPU) Oracle SOA Suite images, which bundle Java SE.
  - Access to Oracle SOA Suite patches.
  - Oracle support for Oracle SOA Suite images.
- WebLogic Kubernetes Operator
  The WebLogic Kubernetes Operator is open source and free, licensed under the Universal Permissive license (UPL), Version 1.0. For support details, see Get help.
- Additional references
  Supported Virtualization and Partitioning Technologies for Oracle Fusion Middleware (search for keyword ‘Kubernetes’)

  Running and Licensing Oracle Programs in Containers and Kubernetes

Prepare Your Environment

To prepare your Oracle SOA Suite in Kubernetes environment, complete the following steps.

Note:

Refer to the troubleshooting page to troubleshoot issues during the domain deployment process.

- Set up your Kubernetes cluster
- Install Helm
- Pull images from Oracle Container Registry
- Set up the code repository to deploy Oracle SOA Suite domains
- Obtain the Oracle SOA Suite Docker image

- Install the WebLogic Kubernetes Operator
- Prepare the environment for Oracle SOA Suite domains
  - a. Create a namespace for an Oracle SOA Suite domain
  - b. Create a persistent storage for an Oracle SOA Suite domain
  - c. Create a Kubernetes secret with domain credentials
  - d. Create a Kubernetes secret with the RCU credentials
  - e. Configure access to your database
  - f. Run the Repository Creation Utility to set up your database schemas
- Create Oracle SOA Suite domain

Set up Your Kubernetes Cluster

Refer the official Kubernetes set up documentation to set up a production grade Kubernetes cluster.

Supported Environments

Oracle Cloud Native Environment Cluster

Oracle Cloud Native Environment is a fully integrated suite for the development and management of cloud-native applications. Based on Open Container Initiative (OCI) and Cloud Native Computing Foundation (CNCF) standards, Oracle Cloud Native Environment delivers a simplified framework for installations, updates, upgrades, and configuration of key features for orchestrating microservices.

Deploy Oracle Cloud Native Environment with the Kubernetes module, following instructions from Oracle Cloud Native Environment: Get Started.

To make it easier to set up and manage OCNE Cluster, we use a range of OpenTofu scripts. See OpenTofu for Oracle Cloud Native Environment for samples.

Oracle Cloud Infrastructure

The operator and Oracle SOA Suite domain are supported on Oracle Cloud Infrastructure using Oracle Container Engine for Kubernetes, or in a cluster running Oracle Linux Container Services for use with Kubernetes on Oracle Cloud Infrastructure Compute.

Create the Oracle Container Engine for Kubernetes using the Oracle Cloud Infrastructure Console or by some other means.

Refer Deploy Oracle SOA Suite on Oracle Kubernetes Engine (OKE) for samples to create Oracle Container Engine for Kubernetes.

To make it easier to set up and manage OKE Cluster, we use a range of OpenTofu scripts. See Sample OpenTofu Scripts to Create an OKE Cluster for samples.

Oracle Private Cloud Appliance (PCA)
Oracle Private Cloud Appliance Container Engine for Kubernetes (OKE) is a scalable, highly available service that can be used to deploy any containerized application to the cloud. See here for details.

Deploy Oracle SOA Suite on Oracle Kubernetes Engine (OKE)

This section describes the steps required to provision a Kubernetes cluster on Oracle Kubernetes Engine with a database for the SOA Suite schemas and a file storage mountpath to store the SOA Suite domain files, and Oracle SOA Suite in Kubernetes.

Prerequisites

To deploy Oracle SOA Suite on Container Engine for Kubernetes, ensure you have available resources and quota for:

- One file storage systems.
- One mount target.
- One Database, either an on-premise Database or Oracle Base Database Service or Oracle Single Instance Database using Database Operator.
- One Kubernetes cluster and a node pool with the required number of nodes as per your requirement.

Note:

- Refer the Domain Resource Sizing and Requirements and Pricing and choose the node pool shape needed for an Oracle SOA Suite domains, one OCPU will not be sufficient.
- Default cluster block volume size may be inadequate. Refer this documentation for instructions on resizing the volume. It is recommended to allocate a minimum of 300 GB for each node.

Create a Kubernetes cluster on OKE

You can create a Kubernetes cluster using Container Engine for Kubernetes (OKE). See here for details.

To ensure high availability, Container Engine for Kubernetes:

- Creates the Kubernetes Control Plane on multiple Oracle-managed control plane nodes (distributing the control plane nodes across different availability domains in a region, where supported).
- Creates worker nodes in each of the fault domains in an availability domain (distributing the worker nodes as evenly as possible across the fault domains, subject to any other infrastructure restrictions).

Preparing for OKE

Before you start creating the Container Engine for Kubernetes, refer the Preparing for Container Engine for Kubernetes and find out if you have met some of the below requirements:

- Access to an Oracle Cloud Infrastructure tenancy.
- Check the service limits for the components listed in Prerequisites in your Oracle Cloud Infrastructure tenancy and, if necessary, request a service limit increase.
- Belong to tenancy's Administrators group and also have appropriate Container Engine for Kubernetes permissions

- Access to perform Kubernetes operations on a cluster.

Create a compartment

Within your tenancy, there must already be a compartment to contain the necessary network resources (such as a VCN, subnets, internet gateway, route table, security lists). If such a compartment does not exist already, you will have to create it. Note that the network resources can reside in the root compartment. However, if you expect multiple teams to create clusters, best practice is to create a separate compartment for each team.

Refer Managing Compartments and Network Resource Configuration for Cluster Creation and Deployment for more details.

Create compartment policies

To create and/or manage clusters, you must belong to one of the following:

- The tenancy's Administrators group
- A group to which a policy grants the appropriate Container Engine for Kubernetes permissions.

See Policy Configuration for Cluster Creation and Deployment for details.

Create an OKE Cluster

Create a new Kubernetes clusters using Container Engine for Kubernetes to create new Kubernetes clusters. You can create clusters using the Console, the CLI, and the API. See Creating a Cluster for details.

Create node pool

When you create a new cluster using the Console, can create managed node pools using the Console. See here to create new managed node pools using the Console, the CLI, or the API. If your worker nodes are configured as private, you will need to establish a bastion host to access them. Refer Managing Bastions.

Set up access to your Cluster

Container Engine for Kubernetes creates a Kubernetes kubeconfig configuration file that you use to access the cluster using kubectl. Refer Setting Up Cluster Access and create access via Cloud Shell or Local access.

Install tools

Once you have setup the cluster access, verify or install the below versions of the tools required for deploying Oracle SOA Suite domain: - **kubectl** (>= 1.24) : See here for the installation instructions. - **Helm** (>= 3.10.2): Helm is a Kubernetes deployment package manager. See here to install helm locally.

Create storage for domain home

You can use the File Storage service to provision persistent volume claims which will be used for domain home. Refer Provisioning PVCs on the File Storage Service for details in the OCI documentation. Sample files are available for provisioning a PVC using the CSI Volume Plug in. You can update the `fss-dyn-st-class.yaml` and `fss-dyn-claim.yaml` files with the relevant parameters to provision the PVC for the domain. See WebLogic Kubernetes Operator documentation for updating the permissions of shared directory to 1000:0 for domain home.

Create an ingress controller

When you create clusters using Container Engine for Kubernetes, you can set up:

- The OCI native ingress controller.
- A third-party ingress controller such as Nginx or Traefik ingress controller.

See Managing Ingress Controllers.See Setup a Load Balancer for installation details on third-party ingress controller. Note to set the `service.type` to `LoadBalancer`, so that OCI provisions the Load Balancer.

Create an Oracle SOA Suite domain

See Prepare Your Environment for preparing the environment for Oracle SOA Suite domains.

Perform the following steps:

1. Create a Kubernetes secret to enable pulling the Oracle SOA Suite image from the registry.

   ```
     $ kubectl -n DOMAIN_NAMESPACE create secret docker-registry image-secret
   \
        --docker-server=container-registry.oracle.com \
        --docker-username=YOUR_REGISTRY_USERNAME \
        --docker-password=YOUR_REGISTRY_PASSWORD \
        --docker-email=YOUR_REGISTRY_EMAIL
   ```

   Replace DOMAIN_NAMESPACE, YOUR_REGISTRY_USERNAME, YOUR_REGISTRY_PASSWORD, and YOUR_REGISTRY_EMAIL with the values you use to access the registry.
2. While creating a persistent storage, ensure to set the `weblogicDomainStorageType` to NFS and `weblogicDomainStoragePath` to the address obtained in Create Storage for Domain home.

To create the domain, see Create Oracle SOA Suite Domains Manually for details on creation of an Oracle SOA Suite domain. Note that, the default timeout value of 600s may not be sufficient for creating the domain on OKE, hence pass a sufficient timeout value greater than 600 with -t.

Configure ingress controller to access Oracle SOA Suite domain services

Refer OCI documentation, in case you have set up OCI native ingress controller. See Setup a Load Balancer for creating the ingress resources on third-party ingress controllers.

Sample OpenTofu Scripts to Create an OKE Cluster

The sample script creates:

A new Virtual Cloud Network (VCN) for the cluster.

Two LoadBalancer subnets with security lists.

Three Worker subnets with security lists.

A Kubernetes Cluster with one Node Pool.

A `kubeconfig` file to allow access using `kubectl`.

A Mount target and two filesystems based on FSS for PV usage, one for the domain home and the other for the database volume.

Nodes and network settings will be configured to allow SSH access, and the cluster networking policies will allow NodePort services to be exposed. This cluster can be used for testing and development purposes only. The provided samples of OpenTofu scripts should not be considered for creating production clusters, without more of a review.

All OCI Container Engine masters are Highly Available (HA) and fronted by load balancers.

Prerequisites

Before using OpenTofu scripts, ensure the following:

- Have an existing tenancy with enough compute and networking resources available for the desired cluster.
- Have an Identity and Access Management policy in place within that tenancy to allow the OCI Container Engine for Kubernetes service to manage tenancy resources.
- Have a user defined within that tenancy.
- Have an API key defined for use with the OCI API, as documented here.
- Have an SSH key pair for configuring SSH access to the nodes in the cluster.
- Install kubectl, kubectl version must be within one minor version(older or newer) of the kubernetes version running on the control plane nodes. See here for the installation instructions.
- You must have downloaded, installed, and configured OCI CLI for use. See here for the installation instructions.

Copy the oci.props.template file to oci.props and add all the following required values:

- `user.ocid`: OCID for the tenancy user - can be obtained from the user settings in the OCI console.
- `tfvars.filename`: File name for the generated tfvar file.
- `okeclustername`: The name for OCI Container Engine for Kubernetes cluster.
- `tenancy.ocid`: OCID for the target tenancy.
- `region`: name of region in the target tenancy.
- `compartment.ocid`: OCID for the target compartment. To find the OCID of the compartment - https://docs.oracle.com/en-us/iaas/Content/GSG/Tasks/contactingsupport_topic-Finding_the_OCID_of_a_Compartment.htm
- `compartment.name`: Name for the target compartment.
- `ociapi.pubkey.fingerprint`: Fingerprint of the OCI user's public key.
- `ocipk.path`: API Private Key local path to the private key for the API key pair.
- `vcn.cidr.prefix`: Prefix for VCN CIDR. Use when creating subnets. Examine the target compartment find a CIDR that is available.
- `vcn.cidr`: Full CIDR for the VCN. Must be unique within the compartment, first 2 octets must match the vcn_cidr_prefix.
- `nodepool.shape`: A valid OCI VM Shape for the cluster nodes.
- `k8s.version`: Kubernetes version.
- `nodepool.ssh.pubkey`: SSH public key (key contents as a string).

- `nodepool.imagename`: A valid image OCID for Node Pool creation. To find the OCID of the image - https://docs.oracle.com/en-us/iaas/Content/ContEng/Reference/contengimagesshapes.htm#images__oke-images
- `tofu.installdir`: Location to install tofu binaries.

Optional step to modify the shape of the node, edit `node-pool.tf`.

```
node_shape_config {
        #Optional
        memory_in_gbs = 48.0
        ocpus = 4.0
  }
```

Optional step to add more nodes to the cluster (by default 2 worker nodes are created), modify `vcn.tf` to add worker subnets:

```
resource "oci_core_subnet" "oke-subnet-worker-3" {
  availability_domain =
data.oci_identity_availability_domains.ADs.availability_domains[2]["name"]
  cidr_block          = "${var.vcn_cidr_prefix}.12.0/24"
  display_name        = "${var.cluster_name}-WorkerSubnet03"
  dns_label           = "workers03"
  compartment_id      = var.compartment_ocid
  vcn_id              = oci_core_virtual_network.oke-vcn.id
  security_list_ids   = [oci_core_security_list.oke-worker-security-list.id]
  route_table_id      = oci_core_virtual_network.oke-
vcn.default_route_table_id
  dhcp_options_id     = oci_core_virtual_network.oke-
vcn.default_dhcp_options_id
}
```

Add the corresponding egress_security_rules and ingress_security_rules for the worker subnets:

```
 egress_security_rules {
    destination = "${var.vcn_cidr_prefix}.12.0/24"
    protocol    = "all"
    stateless   = true
  }


  ingress_security_rules {
    stateless = true
    protocol  = "all"
    source    = "${var.vcn_cidr_prefix}.12.0/24"
  }
```

Modify `node-pool.tf` subnet_ids to add new worker subnets to the pool.

```
subnet_ids = [oci_core_subnet.oke-subnet-worker-1.id, oci_core_subnet.oke-
subnet-worker-2.id,
        oci_core_subnet.oke-subnet-worker-3.id]
```

To run the script, use the command:

```
$ kubernetes/oke/samples/opentofu/oke.create.sh oci.props
```

The script collects the values from oci.props file and performs the following steps:

- Creates a new tfvars file based on the values from the provided `oci.props` file.
- Downloads and installs all needed binaries for OpenTofu, OpenTofu OCI Provider, based on OS system (macOS or Linux).
- Applies the configuration, creates OKE Cluster using OpenTofu and generates kubeconfig file based on the `okeclustername` property defined in `oci.props` file.

Output of the oke.create.sh script If there are errors in the configuration, output is as follows:

```
If you ever set or change modules or backend configuration for tofu,
rerun this command to reinitialize your working directory. If you forget,
other
commands will detect it and remind you to do so if necessary.
\u2577
\u2502 Error: Reference to undeclared resource
\u2502
\u2502   on node-pool.tf line 12, in resource "oci_containerengine_node_pool"
"fmw_node_pool":
\u2502   12:   subnet_ids = [oci_core_subnet.oke-subnet-worker-1.id,
oci_core_subnet.oke-subnet-worker-2.id, oci_core_subnet.oke-subnet-
worker-3.id, oci_core_subnet.oke-subnet-worker-4.id, oci_core_subnet.oke-
subnet-worker-5.id]
\u2502
\u2502 A managed resource "oci_core_subnet" "oke-subnet-worker-5" has not
been declared in the root module.
\u2575
\u2577
\u2502 Error: Reference to undeclared resource
\u2502
\u2502   on node-pool.tf line 12, in resource "oci_containerengine_node_pool"
"fmw_node_pool":
\u2502   12:   subnet_ids = [oci_core_subnet.oke-subnet-worker-1.id,
oci_core_subnet.oke-subnet-worker-2.id, oci_core_subnet.oke-subnet-
worker-3.id, oci_core_subnet.oke-subnet-worker-4.id, oci_core_subnet.oke-
subnet-worker-5.id]
\u2502
\u2502 A managed resource "oci_core_subnet" "oke-subnet-worker-5" has not
been declared in the root module.
\u2575
```

If the cluster is created successfully, the following output displays:

```
cluster_id = "ocid1.cluster.oc1.phx.axxxxxxx.....jlxa"
fmw1_export_path = "/fmw1"
fmw1_fs_ocid = "ocid1.filesystem.oc1.phx.aaaxxxxxx....fuzaaaaa"
fmw2_export_path = "/fmw2"
fmw2_fs_ocid = "ocid1.filesystem.oc1.phx.aaaaxxxxx....xxxxxxaaaa"
fmw_mount_target_ip = "10.1.11.43"
Confirm access to cluster...
```

```
NotReady
echo '[ERROR] Some Nodes in the Cluster are not in the Ready Status , sleep
10s more ...
Status is Ready Iter [1/100]
- able to access cluster
okecluster6 cluster is up and running
```

To access the cluster, set the KUBECONFIG environment variable or use the --kubeconfig option with the kubeconfig file generated based on the okeclustername property defined in oci.props file. For example: okeclustername6_kubeconfig

```
$ kubectl get nodes --kubeconfig=okecluster6_kubeconfig
NAME           STATUS    ROLES    AGE     VERSION
10.1.10.2     Ready     node     2d11h   v1.29.1
10.1.11.3     Ready     node     2d11h   v1.29.1
```

To add new nodes to the cluster after its created, make changes in vcn.tf and node-pool.tf files and run the below commands.

```
${tofu.installdir}/tofu plan -var-file=<tfvars.filename>
${tofu.installdir}/tofu apply -var-file=<tfvars.filename>
```

To delete the cluster, run oke.delete.sh script. It reads the oci.props file from the current directory and deletes the cluster.

```
$ kubernetes/oke/samples/opentofu/oke.delete.sh oci.props
```

Deploy SOA on a OKE cluster using Helmfile

Refer to OracleSOASuite/helm-charts/README.md for detailed instructions.

OCI NFS volume static provisioning

For SOA usage, the file system can be directly referenced in the values.yaml file under the domain.storage section. Example:

```
storage:
     capacity: 10Gi
     reclaimPolicy: Retain
     type: nfs
     path: /fmw1   #Export Path
     nfs:
      server: 10.0.10.156 #Mount Target IP Address
```

For DB usage, here is an example of persistent volume(PV) definition using a File System(FS) and a Mount target IP:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: fmw2-pv-db
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  csi:
    driver: fss.csi.oraclecloud.com
    volumeHandle: "<OCID of the file system>:<Mount Target IP Address>:/
<Export Path>"
```

Update the values.yaml file under oracledb.persistence section to use the created PV as a volume. For example:

```
   persistence:
     storageClass: ""
     size: 10Gi
     accessMode: "ReadWriteOnce"
     volumeName: "fmw2-pv-db"
```

Note:

- Example volumeHandle in the above configuration file :

  ```
  volumeHandle:"ocid1.filesystem.oc1.phx.aaaaanoxxxxx....aaaa:10.0.10
  .156:/fmw2"
  ```

- Obtain the `volumeHandle` details from the console output after the cluster is created successfully.
- Whenever a mount target is provisioned in OCI, the `Reported Size (GiB)` values are very large. This is visible on the mount target page when logged in to the OCI console. Applications will fail to install if the results of a space requirements check show too much available disk space. So in the OCI Console, click the little "Pencil" icon besides the **Reported Size** parameter of the Mount Target to specify, in gigabytes (GiB), the maximum capacity reported by file systems exported through this mount target. This setting does not limit the actual amount of data you can store. See here for setting a File System's Reported Size.

OpenTofu for Oracle Cloud Native Environment

This repository is the top level for a modularized method for deploying OCNE and its sub-components into OCI using OpenTofu. Each sub-module focuses on a specific portion of an OCNE deployment and allows users to select specific configurations to deploy.

This module helps you to create the following resources:

- OCNE API Server: The OCNE API Server orchestrates OCNE agents running on Control Plane and Worker nodes to perform installation of Kubernetes and other OCNE modules.
- Control Plane Nodes: The compute instances for the Control Plane Nodes of the Kubernetes cluster.
- Worker Nodes: The compute instances for the Worker Nodes of the Kubernetes cluster.
- Mount target: A mount target and two file systems using FSS for persistent volume (PV) usage: one for the domain home and the other for the database volume.

- Bastion Server: A Bastion server acts as a secure gateway for accessing private machines in a network.

High-Level Deployment Options

This module supports several out-of-the-box common deployment scenarios. These scenarios are listed here to avoid having to duplicated them in each of the relevant module descriptions below:

- OCNE API Server on a dedicated compute instance.
- Passing in a default network to build the deployment.
- Allowing these modules to create and configure a new network.
- Use OpenSSL to generate and distribute certificates to each node.

Prerequisites

To use these OpenTofu scripts, ensure that you maintain the following:

- An existing tenancy with enough compute and networking resources available for the desired cluster.
- Identity and Access Management policy in place within that tenancy to allow the OCI Container Engine for Kubernetes service to manage tenancy resources.
- User defined within that tenancy.
- An API key defined for use with the OCI API, as documented here.
- SSH key pair for configuring SSH access to the nodes in the cluster.
- Download, install, and configure OCI CLI for use. See here for the installation instructions.

Overview of terraform.tfvars.template file variables

Name	Description
tenancy_id	The OCID of your tenancy. To get the value, see Where to Get the Tenancy's OCID and User's OCID.
compartment_id	The OCID of the compartment.
user_id	The OCID of the user that is used by OpenTofu to create OCI resources. To get the value, see Where to Get the Tenancy's OCID and User's OCID.
fingerprint	Fingerprint for the key pair. To get the value, see How to Get the Key's Fingerprint.
api_private_key_path	The path to the private key used by the OCI user to authenticate with OCI APIs. For details on how to create and configure keys see How to Generate an API Signing Key and How to Upload the Public Key.
region	The OCI region where resources are created. To get the value, See Regions and Availability Domains.
availability_domain_id	The ID of the availability domain inside the `region` to create the deployment.
prefix	A unique prefix to attach to the name of all OCNE resources that are created as a part of the deployment.

Name	Description
ssh_private_key_path	The SSH private key path that goes with the SSH public key that is used when accessing compute resources that are created as part of this deployment. To generate the keys, see Generating an SSH Key Pair for Oracle Compute Cloud Service Instances.
ssh_public_key_path	The SSH public key path to use when configuring access to any compute resources created as part of this deployment. To generate the keys, see Generating an SSH Key Pair for Oracle Compute Cloud Service Instances.
control_plane_node_count	The number of Kubernetes control plane nodes to deploy. To view the recommended worker node count, see Kubernetes High Availability Requirements.
worker_node_count	The number of Kubernetes worker nodes to deploy. To view the recommended worker node count, see Kubernetes High Availability Requirements.
os_version	The version of Oracle Linux to use as the base image for all compute resources that are part of this deployment.
environment_name	The name of the OCNE Environment that is created by this module to deploy module instances into. For more details, see Creating an Environment.
kubernetes_name	The name of the instance of the OCNE Kubernetes module that is installed as part of this deployment. For more details, see Creating a Kubernetes Module.
ocne_version	The version and release of OCNE to deploy. For more details on the versions, see OCNE Release Notes. To install the latest patch version of `<major.minor>`, please set the value to `<major.minor>` or set the value to `<major.minor.patch>` to install a specific patch version.
config_file_path	The path to the OCNE configuration file. For more details on the configuration file, see OCNE configuration file.

Deploying Environment

Refer Set up the Code Repository to Deploy Oracle SOA Suite Domains and set up the deployment scripts.

Copy the provided `oci.props.template` file available at `$WORKDIR/ocne/samples/opentofu` to `oci.props` and add all the required values:

To run the script, use the command:

```
$ $WORKDIR/ocne/samples/opentofu/ocne.create.sh oci.props
```

The script collects the values from `oci.props` file and performs the following steps:

- Creates a new tfvars file based on the values from the provided `oci.props` file.

- Downloads and installs all the necessary binaries for OpenTofu, yq and jq tools.
- Applies the configuration, creates OCNE environment using OpenTofu, and generates kubeconfig file.

If the OCNE environment is created successfully, the following output is displayed by `ocne.create.sh`

```
api_private_key_path = "/home/user1/.oci/oci_api_key.pem"
apiserver_ip = "10.0.0.33"
availability_domain_id = "PJzM:PHX-AD-2"
bastion_private_key_path = "/home/user1/.ssh/id_rsa"
bastion_public_ip = "129.xxx.xxx.xx"
bastion_user = "opc"
compartment_id = "ocid1.compartment.oc1..aaaaaaaaq6xxxxxxx....4a"
config_file_path = ""
container_registry = "container-registry.oracle.com/olcne"
control_plane_node_count = 1
control_plane_nodes = tolist([
  "10.0.0.155",
])
environment_name = "myenvironment"
extra_cas = tolist([])
fingerprint = "a6:c8:xx:xx:xx:xx..:XX"
fmw1_export_path = "/fmw1"
fmw1_fs_ocid = "ocid1.filesystem.oc1.phx.aaaaaaaaaaj7vxxxxxx...aa"
fmw2_export_path = "/fmw2"
fmw2_fs_ocid = "ocid1.filesystem.oc1.phx.aaaaaaaaaaxxxxxx...aaaa"
freeform_tags = tomap({})
image_ocid = "ocid1.image.oc1.phx.aaaaaaaahgrsxxxxx...cq"
instance_shape = tomap({
  "memory" = "32"
  "ocpus" = "2"
  "shape" = "VM.Standard.E4.Flex"
})
key_ocid = ""
kube_apiserver_endpoint_ip = "10.0.0.207"
kube_apiserver_port = "6443"
kube_apiserver_virtual_ip = ""
kubernetes_name = "mycluster"
load_balancer_ip = "10.0.0.207"
load_balancer_policy = "LEAST_CONNECTIONS"
load_balancer_shape = tomap({
  "flex_max" = "50"
  "flex_min" = "10"
  "shape" = "flexible"
})
mount_target_ip = "10.0.0.128"
ocne_secret_name = "vk1-ocne_keys"
ocne_vault_client_token = ""
ocne_version = "1.9"
os_version = "8"
prefix = "vk1"
provision_mode = "OCNE"
provision_modes_map = {
  "provision_mode_infrastucture" = "Infrastructure"
  "provision_mode_ocne" = "OCNE"
}
proxy = ""
region = "us-phoenix-1"
secret_name = "vk1-vault_keys"
ssh_private_key_path = "/home/user1/.ssh/id_rsa"
standalone_api_server = true
subnet_id = "ocid1.subnet.oc1.phx.aaaaaaaas77xxxxxxx...wq"
tenancy_id = "ocid1.tenancy.oc1..aaaaaaaxxxxx......bmffq"
use_vault = false
user_id = "ocid1.user.oc1..aaaaaaxxxxxxx.....sya"
vault_ha_storage_bucket = ""
vault_instances = []
vault_namespace = ""
vault_ocid = ""
vault_pool_size = 1
vault_storage_bucket = ""
vault_uri = ""
vault_version = "1.3.4"
vcn_id = "ocid1.vcn.oc1.phx.amaaaaaxxxxxx....kq"
worker_node_count = 1
worker_nodes = [
  "10.0.0.76",
]
yum_repo_url = "http://yum.oracle.com/repo/OracleLinux/OL8/olcne16/x86_64"
kubeconfig file successfully created in /home/user1/ocne_env/opentofu.
```

To delete the OCNE environment, run the `ocne.delete.sh` script. It reads the `oci.props` file from the current directory and deletes the cluster.

```
$ $WORKDIR/ocne/samples/opentofu/ocne.delete.sh oci.props
```

Installing the Oracle Cloud Infrastructure Cloud Controller Manager Module(OCI-CCM) in an OCNE environment configured using Tofu

The Oracle Cloud Infrastructure Cloud Controller Manager module is used to provision Oracle Cloud Infrastructure storage. See here for detailed installation steps.

Create an Oracle Cloud Infrastructure Cloud Controller Manager module and associate it with the Kubernetes module named mycluster using the `--oci-ccm-kubernetes-module` option. In this example, the Oracle Cloud Infrastructure Cloud Controller Manager module is named `myoci`.

After successfully creating the OCNE environment using Tofu configurations, you can retrieve the vcn and subnet details from the console output. Log in to the platform API server node (OCNE Operator node) and perform these steps. For example:

Once you've successfully created the OCNE environment using Tofu configurations, you can obtain the vcn and subnet details from the console output. Alternatively, you can run tofu and get the vcn and subnet IDs from the tofu output. Next, log in to the platform API server node (OCNE Operator node) and follow these steps. For example:

```
# The path to the node certificate(IP refers to the operator/platform API 
server node).
export OLCNE_SM_CA_PATH=/home/opc/.olcne/certificates/10.0.0.137\:8091/ca.cert

# The path to the Certificate Authority certificate.
```

```
export OLCNE_SM_CERT_PATH=/home/opc/.olcne/certificates/10.0.0.137\:8091/
node.cert

# The path to the key for the node's certificate.
export OLCNE_SM_KEY_PATH=/home/opc/.olcne/certificates/10.0.0.137\:8091/
node.key

olcnectl module create \
--environment-name myenvironment \
--module oci-ccm \
--name myoci \
--oci-ccm-kubernetes-module mycluster \
--oci-region us-ashburn-1 \
--oci-tenancy ocid1.tenancy.oc1..unique_ID \
--oci-compartment ocid1.compartment.oc1..unique_ID \
--oci-user ocid1.user.oc1..unique_ID \
--oci-fingerprint b5:52:... \
--oci-private-key-file /home/opc/.oci/oci_api_key.pem \
--oci-vcn ocid1.vcn.oc1..unique_ID \
--oci-lb-subnet1 ocid1.subnet.oc1..unique_ID
```

Use the olcnectl module install command to install the Oracle Cloud Infrastructure Cloud Controller Manager module. For example:

```
olcnectl module install \
--environment-name myenvironment \
--name myoci
```

Verify the Oracle Cloud Infrastructure Cloud Controller Manager module is deployed using the olcnectl module instances. For example:

```
olcnectl module instances \
--environment-name myenvironment

INSTANCE        MODULE          STATE
10.0.0.24:8090  node            installed
10.0.0.199:8090 node            installed
mycluster       kubernetes      installed
myoci           oci-ccm         installed


[opc@vk-api-server-001 ~]$ kubectl get nodes --
kubeconfig=kubeconfig.myenvironment.mycluster
NAME                    STATUS   ROLES           AGE   VERSION
vk-control-plane-001    Ready    control-plane   17m   v1.29.3+3.el8
vk-worker-001           Ready    <none>          17m   v1.29.3+3.el8
```

Using Object Storage for State Files

Using Object Storage statefile requires that you create an AWS S3 Compatible API Key on OCI. This can be done from both the OCI UI and CLI. For more details visit Using Object Storage for State Files. To get started, rename `state_backend.tf.example` to `state_backend.tf` and fill out the appropriate variables. Variable definitions for the S3 Backend can be found in the OpenTofu S3 Backend Documentation.

Deploying SOA on a OCNE cluster using Helmfile

Refer to OracleSOASuite/helm-charts/README.md for detailed instructions.

OCI NFS Volume Static Provisioning:

For SOA usage, the file system can be directly referenced in the values.yaml file under the domain.storage section. Example:

```
   storage:
     capacity: 10Gi
     reclaimPolicy: Retain
     type: nfs
     path: /fmw1   #Export Path
     nfs:
      server: 10.0.10.156 #Mount Target IP Address
```

For DB usage, here is an example of persistent volume(PV) definition using a File System(FS) and a Mount target IP:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: fmw2-pv-db
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  csi:
    driver: fss.csi.oraclecloud.com
    volumeHandle: "<OCID of the file system>:<Mount Target IP Address>:/
<Export Path>"
```

Update the values.yaml file under oracledb.persistence section to use the created PV as a volume. Example:

```
   persistence:
     storageClass: ""
     size: 10Gi
     accessMode: "ReadWriteOnce"
     volumeName: "fmw2-pv-db"
```

Note:

- Example volumeHandle in the above config file:

  ```
  volumeHandle:
  
  "ocid1.filesystem.oc1.phx.aaaaanoxxxxx....aaaa:10.0.10.156:/fmw2"
  ```

- Obtain the `volumeHandle` details from the console output after the cluster is created successfully.
- Whenever a mount target is provisioned in OCI, its Reported Size (GiB) values are very large. This is visible on the mount target page when logged in to the OCI console. Applications will fail to install if the results of a space requirements check show too much available disk space. So in the OCI Console, click the little "Pencil" icon besides the **Reported Size** parameter of the Mount Target to specify, in gigabytes (GiB), the maximum capacity reported by file systems exported through this mount target. This setting does not limit the actual amount of data you can store. See here for setting a File System's Reported Size.

Install Helm

The operator uses Helm to create and deploy the necessary resources and then run the operator in a Kubernetes cluster. For Helm installation and usage information, see here.

Pull images from Oracle Container Registry

Obtain the required access to pull images from Oracle Container Registry.

1. For first time users, to pull an image from the Oracle Container Registry, navigate to https://container-registry.oracle.com and log in using the Oracle Single Sign-On (SSO) authentication service. If you do not already have an SSO account, you can create an Oracle Account here. Use the web interface to accept the Oracle Standard Terms and Restrictions for the Oracle software images that you intend to deploy. Your acceptance of these terms are stored in a database that links the software images to your Oracle Single Sign-On login credentials.
2. To verify the access, log in to the Oracle Container Registry (container-registry.oracle.com) from your Podman or Docker client and pull the image.

   ```
   podman login container-registry.oracle.com
   podman pull <container-registry.oracle.com/....>
   ```

Set up the Code Repository to Deploy Oracle SOA Suite Domains

Oracle SOA Suite domain deployment on Kubernetes leverages the WebLogic Kubernetes Operator infrastructure.

To deploy an Oracle SOA Suite domain, you must set up the deployment scripts.

1. Create a working directory to set up the source code:

   ```
   $ mkdir $HOME/soa_14.1.2
   $ cd $HOME/soa_14.1.2
   ```

2. Download the WebLogic Kubernetes Operator source code and Oracle SOA Suite Kubernetes deployment scripts from the SOA repository. Required artifacts are available at OracleSOASuite/kubernetes.

   ```
   $ git clone https://github.com/oracle/fmw-kubernetes.git
   $ export WORKDIR=$HOME/soa_14.1.2/fmw-kubernetes/OracleSOASuite/kubernetes
   ```

Obtain the Oracle SOA Suite Docker Image

The Oracle SOA Suite image is prebuilt by Oracle for 14.1.2.0.0. The Oracle Container Registry hosts container images based on both Oracle Linux 8 (ol8) and 9 (ol9) for both JDK 17 and JDK 21. These are the only images supported for production deployments.

> **Note:**
>
> The default Oracle SOA Suite image name used for Oracle SOA Suite domains deployment is `soasuite:release-version`. The image obtained must be tagged accordingly using the podman tag command. If you want to use a different name for the image, make sure to update the new image tag name in the `create-domain-inputs.yaml` file and also in other instances where the `soasuite:release-version` image name is used.

Obtain the Oracle SOA Suite images by downloading from the Oracle Container Registry.

1. Log in to Oracle Container Registry, navigate to Middleware > soasuite and accept the license agreement if not already done.
2. Log in to the Oracle Container Registry (container-registry.oracle.com) from your podman:

   ```
   $ podman login container-registry.oracle.com
   ```

3. Pull the required image. For example:

   ```
   podman pull container-registry.oracle.com/middleware/soasuite:14.1.2.0-<17
   or 21>-<ol8 or ol9>-<tag>
   ```

Install the WebLogic Kubernetes Operator

The WebLogic Kubernetes Operator supports the deployment of Oracle SOA Suite domains in the Kubernetes environment.

Follow the steps in this document to install the operator.

Optionally, you can follow these steps to send the contents of the operator’s logs to Elasticsearch.

In the following example commands to install the WebLogic Kubernetes Operator, `opns` is the namespace and `op-sa` is the service account created for the operator:

```
kubectl create namespace opns
kubectl create serviceaccount -n opns op-sa
helm repo add weblogic-operator https://oracle.github.io/weblogic-kubernetes-
operator/charts --force-update
helm install weblogic-kubernetes-operator weblogic-operator/weblogic-
operator  --version 4.2.9 --namespace opns  --set serviceAccount=op-sa --set
"javaLoggingLevel=FINE" --wait
```

This Helm release deploys the operator with the default behavior of managing Oracle SOA Suite domains in all Kubernetes namespaces with the label `weblogic-operator=enabled`.

Prepare the environment for Oracle SOA Suite domains

Create a namespace for an Oracle SOA Suite domain

Create a Kubernetes namespace (for example, soans) for the domain unless you intend to use the default namespace. Label the namespace with weblogic-operator=enabled to manage the domain. Use the new namespace in the remaining steps in this section. For details, see Prepare to run a domain.

```
$ kubectl create namespace soans
$ kubectl label namespace soans weblogic-operator=enabled
```

Oracle SOA Suite image prebuilt by Oracle have an oracle user with UID 1000 with the default group set to root. To ensure that `weblogicDomainStoragePath` path is having this ownership of `1000:0`, you can refer the utility script, pv-pvc-helper.sh provided as part of the lifecycle scripts to change the ownership and permissions of the shared directory on the persistent storage.

For example, you can change the ownership and permissions of weblogicDomainStoragePath for the created PVC soainfra-domain-pvc in soans domain namespace with below steps:

Launch the helper pod with the PVC `soainfra-domain-pvc` in `soans` namespace and mount path `/shared`:

```
$ pv-pvc-helper.sh -n soans -c soainfra-domain-pvc -m /shared -r
```

After the helper pod is created, use the following command to get a shell to the running pod container.

```
$ kubectl -n soans exec -it pvhelper -- /bin/sh
```

After you get a shell to the running Pod container, change the directory to /shared, and you can change the ownership or permissions using the appropriate chown or chmod commands. For example,

```
$ chown 1000:0 /shared/. && find /shared/. -maxdepth 1 ! -name '.snapshot' ! -
name '.' -print0 | xargs -r -0 chown -R 1000:0
```

Create a persistent storage for an Oracle SOA Suite domain

In the Kubernetes namespace you created, create the PV and PVC for the domain by running the create-pv-pvc.sh script. Follow the instructions for using the script to create a dedicated PV and PVC for the Oracle SOA Suite domain.

1. Review the configuration parameters for PV creation here. Based on your requirements, update the values in the create-pv-pvc-inputs.yaml file located at ${WORKDIR}/create-weblogic-domain-pv-pvc/. Sample configuration parameter values for an Oracle SOA Suite domain are:
   - baseName: `domain`
   - domainUID: `soainfra`
   - namespace: `soans`
   - weblogicDomainStorageType: `HOST_PATH`
   - weblogicDomainStoragePath: `/scratch/k8s_dir/SOA`
2. Ensure that the path for the weblogicDomainStoragePath property exists and have the ownership for 1000:0. If not, you need to create it as follows:

   ```
   $ sudo mkdir /scratch/k8s_dir/SOA
   $ sudo chown -R 1000:0 /scratch/k8s_dir/SOA
   ```

3. Run the create-pv-pvc.sh script:

   ```
   $ cd ${WORKDIR}/create-weblogic-domain-pv-pvc
   $ ./create-pv-pvc.sh -i create-pv-pvc-inputs.yaml -o output_soainfra
   ```

4. The create-pv-pvc.sh script will create a subdirectory pv-pvcs under the given /path/to/output-directory directory and creates two YAML configuration files for PV and PVC. Apply these two YAML files to create the PV and PVC Kubernetes resources using the kubectl create -f command:

   ```
   $ kubectl create -f output_soainfra/pv-pvcs/soainfra-domain-pv.yaml
   $ kubectl create -f output_soainfra/pv-pvcs/soainfra-domain-pvc.yaml
   ```

Create a Kubernetes secret with domain credentials

Create the Kubernetes secrets `username` and `password` of the administrative account in the same Kubernetes namespace as the domain:

```
$ cd ${WORKDIR}/create-weblogic-domain-credentials
  $ ./create-weblogic-credentials.sh -u weblogic -p Welcome1 -n soans -d
soainfra -s soainfra-domain-credentials
```

For more details, see this document..

You can check the secret with the

```
kubectl get secret
```

command.

For example:

```
$ kubectl get secret soainfra-domain-credentials -o yaml -n soans
  apiVersion: v1
  data:
    password: T3JhZG9jX2RiMQ==
    sys_password: T3JhZG9jX2RiMQ==
    sys_username: c3lz
    username: U09BMQ==
  kind: Secret
  metadata:
    creationTimestamp: "2020-06-25T14:08:16Z"
    labels:
      weblogic.domainName: soainfra
      weblogic.domainUID: soainfra
    managedFields:
    - apiVersion: v1
      fieldsType: FieldsV1
      fieldsV1:
        f:data:
          .: {}
          f:password: {}
          f:sys_password: {}
          f:sys_username: {}
          f:username: {}
        f:metadata:
          f:labels:
            .: {}
            f:weblogic.domainName: {}
            f:weblogic.domainUID: {}
        f:type: {}
      manager: kubectl
      operation: Update
      time: "2020-06-25T14:08:16Z"
    name: soainfra-rcu-credentials
    namespace: soans
    resourceVersion: "265386"
    selfLink: /api/v1/namespaces/soans/secrets/soainfra-rcu-credentials
    uid: 2d93941c-656b-43a4-8af2-78ca8be0f293
  type: Opaque
```

Create a Kubernetes secret with the RCU credentials

You also need to create a Kubernetes secret containing the credentials for the database schemas. When you create your domain, it will obtain the RCU credentials from this secret. Use the provided sample script to create the secret:

```
$ cd ${WORKDIR}/create-rcu-credentials
$ ./create-rcu-credentials.sh \
  -u SOA1 \
  -p Oradoc_db1 \
  -a sys \
  -q Oradoc_db1 \
  -d soainfra \
  -n soans \
  -s soainfra-rcu-credentials
```

The parameter values are:

- -u username for schema owner (regular user), required.
- -p password for schema owner (regular user), required.
- -a username for SYSDBA user, required.
- -q password for SYSDBA user, required.
- -d domainUID. Example: soainfra
- -n namespace. Example: soans
- -s secretName. Example: soainfra-rcu-credentials

You can confirm the secret was created as expected with the

```
kubectl get secret
```

command.

For example:

```
$ kubectl get secret soainfra-rcu-credentials -o yaml -n soans
  apiVersion: v1
  data:
    password: T3JhZG9jX2RiMQ==
    sys_password: T3JhZG9jX2RiMQ==
    sys_username: c3lz
    username: U09BMQ==
  kind: Secret
  metadata:
    creationTimestamp: "2020-06-25T14:08:16Z"
    labels:
      weblogic.domainName: soainfra
      weblogic.domainUID: soainfra
    managedFields:
    - apiVersion: v1
      fieldsType: FieldsV1
      fieldsV1:
        f:data:
          .: {}
          f:password: {}
          f:sys_password: {}
          f:sys_username: {}
          f:username: {}
        f:metadata:
          f:labels:
            .: {}
            f:weblogic.domainName: {}
            f:weblogic.domainUID: {}
        f:type: {}
      manager: kubectl
      operation: Update
      time: "2020-06-25T14:08:16Z"
    name: soainfra-rcu-credentials
    namespace: soans
    resourceVersion: "265386"
    selfLink: /api/v1/namespaces/soans/secrets/soainfra-rcu-credentials
    uid: 2d93941c-656b-43a4-8af2-78ca8be0f293
  type: Opaque
```

Create Oracle database

Oracle SOA Suite domains require a database with the necessary schemas installed in them. The Repository Creation Utility (RCU) allows you to create those schemas. You must set up the database before you create your domain. There are no additional requirements added by running Oracle SOA Suite in Kubernetes; the same existing requirements apply.

Refer here to create Single Instance Container Database managed by Oracle Database Operator.

Once you have the required database created or have the access details, continue with next steps to set up the necessary schemas in your database.

Run the Repository Creation Utility to set up your database schemas

Create schemas

The RCU pod requires that you create a secret in the same namespace as the RCU pod that contains the database's SYSDBA username and password in its sys_username and sys_password fields, and also contains a password of your choice for RCU schemas in its password field.

For example:

```
kubectl -n default create secret generic oracle-rcu-secret \
  --from-literal='sys_username=sys' \
  --from-literal='sys_password=MY_SYS_PASSWORD' \
  --from-literal='password=MY_RCU_SCHEMA_PASSWORD'
```

To create the database schemas for Oracle SOA Suite, run the create-rcu-schema.sh script.

For example:

```
cd ${WORKDIR}/create-rcu-schema
./create-rcu-schema.sh  -h
usage: ./create-rcu-schema.sh -s <schemaPrefix> [-t <schemaType>] [-d
<dburl>] [-n <namespace>] [-c <credentialsSecretName>] [-p <docker-store>] [-
i <image>] [-u <imagePullPolicy>] [-o <rcuOutputDir>] [-r <customVariables>]
[-l <timeoutLimit>] [-b <databaseType>] [-e <edition>] [-h]
  -s RCU Schema Prefix (required)
  -t RCU Schema Type (optional)
      (supported values: osb,soa,soaosb, default: soa)
  -d RCU Oracle Database URL (optional)
      (default: oracle-db.default.svc.cluster.local:1521/devpdb.k8s)
  -n Namespace for RCU pod (optional)
      (default: default)
  -c Name of credentials secret (optional).
       (default: oracle-rcu-secret)
       Must contain SYSDBA username at key 'sys_username',
       SYSDBA password at key 'sys_password',
       and RCU schema owner password at key 'password'.
  -p OracleSOASuite ImagePullSecret (optional)
      (default: none)
  -i OracleSOASuite Image (optional)
      (default: soasuite:release-version)
  -u OracleSOASuite ImagePullPolicy (optional)
      (default: IfNotPresent)
  -o Output directory for the generated YAML file. (optional)
      (default: rcuoutput)
  -r Comma-separated custom variables in the format variablename=value.
(optional).
      (default: none)
  -l Timeout limit in seconds. (optional).
      (default: 300)
  -b Type of database to which you are connecting (optional). Supported
values: ORACLE,EBR
      (default: ORACLE)
  -e The edition name. This parameter is only valid if you specify type of
database (-b) as EBR. (optional).
      (default: 'ORA$BASE')
  -h Help

NOTE: The c, p, i, u, and o arguments are ignored if an rcu pod is already
running in the namespace.


./create-rcu-schema.sh \
  -s SOA1 \
  -t soaosb \
  -d oracle-db.default.svc.cluster.local:1521/devpdb.k8s \
  -n default \
  -c oracle-rcu-secret \
  -b EBR \
  -i soasuite:release-version \
  -r SOA_PROFILE_TYPE=SMALL,HEALTHCARE_INTEGRATION=NO
```

For Oracle SOA Suite domains, the create-rcu-schema.sh script supports:

- domain types: `soa`, `osb`, and `soaosb`. You must specify one of these using the `-t` flag.
- Type of database to which you are connecting: ORACLE, EBR. You can specify one of these using -b flag. Default is ORACLE. Use EBR for 14.1.2.
- For Oracle SOA Suite you must specify the Oracle SOA schema profile type using the `-r` flag. For example, `-r SOA_PROFILE_TYPE=SMALL`. Supported values for `SOA_PROFILE_TYPE` are `SMALL`, `MED`, and `LARGE`.

Note:

To use the `LARGE` schema profile type, make sure that the partitioning feature is enabled in the Oracle Database.

Make sure that you maintain the association between the database schemas and the matching domain just like you did in a non-Kubernetes environment. There is no specific functionality provided to help with this.

Drop schemas

If you want to drop a schema, you can use the drop-rcu-schema.sh script.

For example:

```
cd ${WORKDIR}/create-rcu-schema

./drop-rcu-schema.sh -h
usage: ./drop-rcu-schema.sh -s <schemaPrefix> [-t <schemaType>] [-d <dburl>]
[-n <namespace>] [-c <credentialsSecretName>] [-p <docker-store>] [-i
<image>] [-u <imagePullPolicy>] [-o <rcuOutputDir>] [-r <customVariables>] [-
b <databaseType>] [-e <edition>] [-h]
  -s RCU Schema Prefix (required)
  -t RCU Schema Type (optional)
      (supported values: osb,soa,soaosb, default: soa)
  -d RCU Oracle Database URL (optional)
      (default: oracle-db.default.svc.cluster.local:1521/devpdb.k8s)
  -n Namespace for RCU pod (optional)
      (default: default)
  -c Name of credentials secret (optional).
       (default: oracle-rcu-secret)
       Must contain SYSDBA username at key 'sys_username',
       SYSDBA password at key 'sys_password',
       and RCU schema owner password at key 'password'.
  -p OracleSOASuite ImagePullSecret (optional)
      (default: none)
  -i OracleSOASuite Image (optional)
      (default: soasuite:release-version)
  -u OracleSOASuite ImagePullPolicy (optional)
      (default: IfNotPresent)
  -o Output directory for the generated YAML file. (optional)
      (default: rcuoutput)
  -r Comma-separated custom variables in the format variablename=value.
(optional).
      (default: none)
  -b Type of database to which you are connecting (optional). Supported
values: ORACLE,EBR
      (default: ORACLE)
  -e The edition name. This parameter is only valid if you specify type of
database (-b) as EBR. (optional).
      (default: 'ORA$BASE')
  -h Help

NOTE: The c, p, i, u, and o arguments are ignored if an rcu pod is already
running in the namespace.


./drop-rcu-schema.sh \
  -s SOA1 \
  -t soaosb \
  -d oracle-db.default.svc.cluster.local:1521/devpdb.k8s \
  -n default \
  -c oracle-rcu-secret \
  -b EBR \
  -r SOA_PROFILE_TYPE=SMALL,HEALTHCARE_INTEGRATION=NO
```

For Oracle SOA Suite domains, the drop-rcu-schema.sh script supports:

- Domain types: soa, osb, and soaosb. You must specify one of these using the -t flag.
- Type of database to which you are connecting: ORACLE, EBR. You can specify one of these using -b flag. Default is ORACLE. Use EBR for 14.1.2.
- For Oracle SOA Suite, you must specify the Oracle SOA schema profile type using the -r flag. For example, -r SOA_PROFILE_TYPE=SMALL. Supported values for SOA_PROFILE_TYPE are SMALL, MED, and LARGE.

Create an Oracle SOA Suite domain

Now that you have your Docker images and you have created your RCU schemas, you are ready to create your domain.

To continue, follow the instructions in Create Oracle SOA Suite domains.

Create Oracle SOA Suite Domains Manually

The SOA deployment scripts demonstrate the creation of an Oracle SOA Suite 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.

Topics in this section:

- Prerequisites
- Prepare to use the create domain script
- Configuration parameters
- Run the create domain script
- Start the domain
- Verify the results
- Verify the domain
- Verify the pods
- Verify the services

Prerequisites

Before you begin, complete the following steps:

1. Review the Domain resource documentation.
2. Review the requirements and limitations.
3. Ensure that you have executed all the preliminary steps in Prepare your environment.
4. Ensure that the database and the WebLogic Kubernetes Operator are running.

Prepare to use the create domain script

The sample scripts for Oracle SOA Suite domain deployment are available at `${WORKDIR}/create-soa-domain`. You must edit `create-domain-inputs.yaml` (or a copy of it) to provide the details for your domain. Refer to the configuration parameters below to understand the information that you must provide in this file.

Supported deployment modes:

- Development mode (set productionModeEnabled to false)
- Production mode (set productionModeEnabled to true and secureEnabled to false)
- Secured production mode (set productionModeEnabled to true and secureEnabled to true)

Refer Using Secured Production Mode for details.

Configuration parameters

The following parameters can be provided in the inputs file.

Parameter	Definition	Default
sslEnabled	Boolean value that indicates whether SSL must be enabled for each WebLogic Server instance. To enable end-to-end SSL access during load balancer setup, set sslEnabled to true and also, set appropriate value for the javaOptions property as detailed in this table. In 14.1.2.0.0, if secureEnabled is set to true, sslEnabled will be set to true by default.	false
adminPort	Port number for the Administration Server inside the Kubernetes cluster.	7001
adminServerSSLPort	SSL port number of the Administration Server inside the Kubernetes cluster.	7002
adminNodePort	Port number of the Administration Server outside the Kubernetes cluster.	30701
adminServerName	Name of the Administration Server.	AdminServer
configuredManagedServerCount	Number of Managed Server instances to generate for the domain.	5
soaClusterName	Name of the SOA WebLogic Server cluster instance to generate for the domain. By default, the cluster name is soa_cluster. This configuration parameter is applicable only for soa and soaosb domain types.	soa_cluster

Parameter	Definition	Default
osbClusterName	Name of the Oracle Service Bus WebLogic Server cluster instance to generate for the domain. By default, the cluster name is osb_cluster. This configuration parameter is applicable only for osb and soaosb domain types.	osb_cluster
createDomainFilesDir	Directory on the host machine to locate all the files to create a WebLogic Server domain, including the script that is specified in the createDomainScriptName parameter. 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 Server domain. 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 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 by the createDomainScriptName parameter) 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

Parameter	Definition	Default
createDomainScriptName	Script that the create domain script uses to create a WebLogic Server 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 by the createDomainScriptsMountPath parameter. If you need to provide your own scripts to create the domain home, instead of using the built-in 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
domainHome	Home directory of the SOA domain. If not specified, the value is derived from the domainUID as /shared/domains/<domainUID>.	/u01/oracle/user_projects/domains/soainfra
domainPVMountPath	Mount path of the domain persistent	volume./u01/oracle/user_projects
domainPVMountPath	Mount path of the domain persistent volume.	/u01/oracle/user_projects
domainType	Type of the domain. Mandatory input for Oracle SOA Suite domains. You must provide one of the supported domain type values: soa (deploys a SOA domain with Enterprise Scheduler (ESS)), osb (deploys an Oracle Service Bus domain), and soaosb (deploys a domain with SOA, Oracle Service Bus, and Enterprise Scheduler (ESS)).	soa
exposeAdminNodePort	Boolean value indicating if the Administration Server is exposed outside of the Kubernetes cluster.	false
exposeAdminT3Channel	Boolean value indicating if the T3 administrative channel is exposed outside the Kubernetes cluster.	false
httpAccessLogInLogHome	Boolean value indicating if server HTTP access log files should be written to the same directory as logHome. If false, server HTTP access log files will be written to the directory specified in the WebLogic Server domain home configuration.	true
image	SOA Suite Docker image. The operator requires Oracle SOA Suite 14.1.2.0. Refer to Obtain the Oracle SOA Suite Docker image for details on how to obtain or create the image.	soasuite:release-version

Parameter	Definition	Default
imagePullPolicy	Oracle SOA Suite Docker image pull policy. Valid values are IfNotPresent, Always, Never.	IfNotPresent
imagePullSecretName	Name of the Kubernetes secret to access the Docker Store to pull the WebLogic Server Docker image. The presence of the secret will be validated when this parameter is specified.	
includeServerOutInPodLog	Boolean value indicating whether to include the server .out to the pod's stdout.	true
initialManagedServerReplicas	Number of Managed Servers to initially start for the domain.	1
javaOptions	Java options for initiating the Administration Server and Managed Servers. A Java option can have references to one or more of the following predefined variables to obtain WebLogic Server domain information: $(DOMAIN_NAME), $(DOMAIN_HOME), $(ADMIN_NAME), $(ADMIN_PORT), and $(SERVER_NAME). If sslEnabled is set to true, add -Dweblogic.ssl.Enabled=true -Dweblogic.security.SSL.ignoreHostnameVerification=true to allow the Managed Servers to connect to the Administration Server while booting up. In this environment, the demo certificate generated by the WebLogic Server contains a host name that is different from the runtime container's host name.	-Dweblogic.StdoutDebugEnabled=false
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/soainfra
soaManagedServerNameBase	Base string used to generate Managed Server names in the SOA cluster. The default value is soa_server. This configuration parameter is applicable only for soa and soaosb domain types.	soa_server
osbManagedServerNameBase	Base string used to generate Managed Server names in the Oracle Service Bus cluster. The default value is osb_server. This configuration parameter is applicable only for osb and soaosb domain types.	osb_server

Parameter	Definition	Default
soaManagedServerPort	Port number for each Managed Server in the SOA cluster. This configuration parameter is applicable only for soa and soaosb domain types.	7003
soaManagedServerSSLPort	SSL port number for each Managed Server in the SOA cluster. This configuration parameter is applicable only for soa and soaosb domain types.	7004
soaAdministrationPort	Administration port number for each Managed Server in the SOA cluster. This configuration parameter is applicable only for soa and soaosb domain types. This value will be used if secureEnabled is set to true.	9004
osbAdministrationPort	Administration port number for each Managed Server in the OSB cluster. This configuration parameter is applicable only for \`osb\` and \`soaosb\` domain types. This value will be used if secureEnabled is set to true.	9007
osbManagedServerSSLPort	SSL port number for each Managed Server in the Oracle Service Bus cluster. This configuration parameter is applicable only for osb and soaosb domain types.	8003
namespace	Kubernetes namespace in which to create the domain.	soans
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.	soainfra-domain-pvc
productionModeEnabled	Boolean value indicating if production mode is enabled for the domain.	true
secureEnabled	Boolean value indicating if secure mode is enabled for the domain. This value has significance only with Oracle SOA Suite 14.1.2.0.0.	false
serverStartPolicy	Determines which WebLogic Server instances will be started. Valid values are Never, IfNeeded, or AdminOnly.	IfNeeded
t3ChannelPort	Port for the T3 channel of the NetworkAccessPoint.	30012

Parameter	Definition	Default
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.	soainfra-domain-credentials
weblogicImagePullSecretName	Name of the Kubernetes secret for the Docker Store, 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. Refer to the Kubernetes documentation on Managing Compute Resources for Containers for details.	Resource requests and resource limits are not specified.
rcuSchemaPrefix	The schema prefix to use in the database. For example SOA1. You may wish to make this the same as the domainUID in order to simplify matching domains to their RCU schemas.	SOA1
rcuDatabaseURL	The database URL.	oracle-db.default.svc.cluster.local:1521/devpdb.k8s
rcuCredentialsSecret	The Kubernetes secret containing the database credentials.	soainfra-rcu-credentials
persistentStore	The persistent store for 'JMS servers' and 'Transaction log store' in the domain. Valid values are jdbc, file.	jdbc

Note that 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, soaClusterName, and soaManagedServerNameBase etc. 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 Oracle SOA Suite domain home and associated Kubernetes resources for the domain. 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.

Run the create domain script

Run the create domain script, specifying your inputs file and an output directory to store the generated artifacts:

```
./create-domain.sh \
  -i create-domain-inputs.yaml \
  -o <path to output-directory>
```

The script will perform the following steps:

- Create a directory for the generated Kubernetes YAML files for this domain if it does not already exist. The path name is <path to output-directory>/weblogic-domains/<domainUID>. If the directory already exists, its contents must be removed before using this script.
- Create a Kubernetes job that will start up a utility Oracle SOA Suite container and run offline WLST scripts to create the domain on the shared storage.
- Run and wait for the job to finish.
- Create a Kubernetes domain YAML file, domain.yaml, in the “output” directory that was created above.
- Create a convenient utility script, delete-domain-job.yaml, to clean up the domain home created by the create script.

Start the domain

The domain.yaml created by create-domain.sh script above has details about the Oracle SOA Suite Domain and Cluster Kubernetes resources. You can create Oracle SOA Suite Domain using the kubectl create -f or kubectl apply -f command:

```
```
kubectl apply -f <path to output-directory>/weblogic-domains/<domainUID>/
domain.yaml
```
```

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

- An Administration Server named AdminServer listening on port 7001.
- A configured cluster named soa_cluster of size 5.
- Managed Server, named soa_server1 listening on port 7003.
- Log files that are located in /shared/logs/<domainUID>.
- SOA Infra, SOA Composer, and WorklistApp applications deployed.

Note:

Refer to the troubleshooting page to troubleshoot issues during the domain creation.

Verify the results

The create domain script verifies that the domain was created, and reports failure if there is an error. However, it may be desirable to manually verify the domain, even if just to gain familiarity with the various Kubernetes objects that were created by the script.

Generated YAML files with the default inputs

Sample content of the generated `domain.yaml` for `soaosb` domainType that creates SOA and Oracle Service Bus clusters.

```
cat output/weblogic-domains/soainfra/domain.yaml
# Copyright (c) 2020, 2023, 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 a Domain resource.
#
apiVersion: "weblogic.oracle/v9"
kind: Domain
metadata:
  name: soainfra
  namespace: soans
  labels:
    weblogic.domainUID: soainfra
spec:
  # The WebLogic Domain Home
  domainHome: /u01/oracle/user_projects/domains/soainfra

  # 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: "soasuite:14.1.2.0"

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

  # Identify which secret contains the credentials for pulling an image
  #imagePullSecrets:
  #- name:

  # Identify which secret contains the WebLogic Admin credentials (note that
there is an example of
  # how to create that secret at the end of this topic)
  webLogicCredentialsSecret:
    name: soainfra-domain-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/soainfra
  # 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:
    # an (optional) 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 "
    volumes:
    - name: weblogic-domain-storage-volume
      persistentVolumeClaim:
        claimName: soainfra-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

  # 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: soainfra-osb-cluster
  - name: soainfra-soa-cluster

---
# This is an example of how to define a Cluster resource.
apiVersion: "weblogic.oracle/v1"
kind: Cluster
metadata:
  name: soainfra-osb-cluster
  # Update this with the namespace your domain will run in:
  namespace: soans
  labels:
    # Update this with the `domainUID` of your domain:
    weblogic.domainUID: soainfra
spec:
  clusterName: osb_cluster
  serverService:
    precreateService: true
  serverPod:
    env:
    # This parameter can be used to pass in new system properties; use the
space delimiter to append multiple values.
    # Do not change this value, only append new values to it.
    - name: K8S_REFCONF_OVERRIDES
      value: "-Doracle.sb.tracking.resiliency.MemoryMetricEnabled=false "
  replicas: 1
  # 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: soainfra-soa-cluster
  # Update this with the namespace your domain will run in:
  namespace: soans
  labels:
    # Update this with the `domainUID` of your domain:
    weblogic.domainUID: soainfra
spec:
  clusterName: soa_cluster
  serverService:
    precreateService: true
  serverPod:
    env:
    # This parameter can be used to pass in new system properties; use the
space delimiter to append multiple values.
    # Do not change this value, only append new values to it.
    - name: K8S_REFCONF_OVERRIDES
      value: "-Doracle.soa.tracking.resiliency.MemoryMetricEnabled=false "
  replicas: 1
  # The number of managed servers to start for unlisted clusters
  # replicas: 1
```

Verify the domain

To confirm that the domain was created, enter the following command:

```
kubectl describe domain DOMAINUID -n NAMESPACE
```

Replace DOMAINUID with the domainUID and NAMESPACE with the actual namespace.

Sample domain description:

```
kubectl describe domain soainfra -n soans
bash-4.2# kubectl describe domain soainfra -n soans
Name:         soainfra
Namespace:    soans
Labels:       weblogic.domainUID=soainfra
Annotations:  <none>
API Version:  weblogic.oracle/v9
Kind:         Domain
Metadata:
  Creation Timestamp:  2023-02-03T13:12:56Z
  Generation:          1
  Managed Fields:
    API Version:  weblogic.oracle/v9
    Fields Type:  FieldsV1
    fieldsV1:
      f:metadata:
        f:labels:
          .:
          f:weblogic.domainUID:
      f:spec:
        .:
        f:clusters:
        f:dataHome:
        f:domainHome:
        f:domainHomeSourceType:
        f:failureRetryIntervalSeconds:
        f:failureRetryLimitMinutes:
        f:httpAccessLogInLogHome:
        f:image:
        f:imagePullPolicy:
        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-create
    Operation:    Update
    Time:         2023-02-03T13:12:56Z
    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:            2023-02-03T13:21:29Z
  Resource Version:  39889912
  UID:               141ec231-dbcf-4afe-8912-e142278f9a79
Spec:
  Clusters:
    Name:                             soainfra-osb-cluster
    Name:                             soainfra-soa-cluster
  Data Home:
  Domain Home:                        /u01/oracle/user_projects/domains/soainfra
  Domain Home Source Type:            PersistentVolume
  Failure Retry Interval Seconds:     120
  Failure Retry Limit Minutes:        1440
  Http Access Log In Log Home:        true
  Image:                              soasuite:14.1.2.0
  Image Pull Policy:                  IfNotPresent
  Include Server Out In Pod Log:      true
  Log Home:                           /u01/oracle/user_projects/domains/logs/
soainfra
  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
      Name:        weblogic-domain-storage-volume
    Volumes:
      Name:  weblogic-domain-storage-volume
      Persistent Volume Claim:
        Claim Name:     soainfra-domain-pvc
  Server Start Policy:  IfNeeded
  Web Logic Credentials Secret:
    Name:  soainfra-domain-credentials
Status:
  Clusters:
    Cluster Name:  osb_cluster
    Conditions:
      Last Transition Time:  2023-02-03T13:21:25.985058Z
      Status:                True
      Type:                  Available
      Last Transition Time:  2023-02-03T13:21:25.985176Z
      Status:                True
      Type:                  Completed
    Label Selector:
weblogic.domainUID=soainfra,weblogic.clusterName=osb_cluster
    Maximum Replicas:        5
    Minimum Replicas:        0
    Observed Generation:     1
    Ready Replicas:          1
    Replicas:                1
    Replicas Goal:           1
    Cluster Name:            soa_cluster
    Conditions:
      Last Transition Time:  2023-02-03T13:21:25.985247Z
      Status:                True
      Type:                  Available
      Last Transition Time:  2023-02-03T13:21:25.985324Z
      Status:                True
      Type:                  Completed
    Label Selector:
weblogic.domainUID=soainfra,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:  2023-02-03T13:21:25.984952Z
    Status:                True
    Type:                  Available
    Last Transition Time:  2023-02-03T13:21:29.921800Z
    Status:                True
    Type:                  Completed
  Observed Generation:     1
  Servers:
    Health:
      Activation Time:  2023-02-03T13:17:40.865000Z
      Overall Health:   ok
      Subsystems:
        Subsystem Name:  ServerRuntime
        Symptoms:
    Node Name:     devhost
    Pod Phase:     Running
    Pod Ready:     True
    Server Name:   AdminServer
    State:         RUNNING
```

```
State Goal:    RUNNING
Cluster Name:  osb_cluster
Health:
  Activation Time:  2023-02-03T13:20:24.163000Z
  Overall Health:   ok
  Subsystems:
    Subsystem Name:  ServerRuntime
    Symptoms:
Node Name:     devhost
Pod Phase:     Running
Pod Ready:     True
Server Name:   osb_server1
State:         RUNNING
State Goal:    RUNNING
Cluster Name:  osb_cluster
Server Name:   osb_server2
State:         SHUTDOWN
State Goal:    SHUTDOWN
Cluster Name:  osb_cluster
Server Name:   osb_server3
State:         SHUTDOWN
State Goal:    SHUTDOWN
Cluster Name:  osb_cluster
Server Name:   osb_server4
State:         SHUTDOWN
State Goal:    SHUTDOWN
Cluster Name:  osb_cluster
Server Name:   osb_server5
State:         SHUTDOWN
State Goal:    SHUTDOWN
Cluster Name:  soa_cluster
Health:
  Activation Time:  2023-02-03T13:21:18.284000Z
  Overall Health:   ok
  Subsystems:
    Subsystem Name:  ServerRuntime
    Symptoms:
Node Name:     devhost
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:      2023-02-03T13:13:28.055432Z
Events:            <none>
```

In the `Status` section of the output, the available servers and clusters are listed. Note that if this command is issued very soon after the script finishes, there may be no servers available yet, or perhaps only the Administration Server but no Managed Servers. The operator will start up the Administration Server first and wait for it to become ready before starting the Managed Servers.

Verify the pods

Enter the following command to see the pods running the servers:

```
kubectl get pods -n NAMESPACE
```

Here is an example of the output of this command. You can verify that an Administration Server and a Managed Server for each cluster (SOA and Oracle Service Bus) are running for soaosb domain type.

```
kubectl get pods -n soans
NAME                                                READY   STATUS
RESTARTS   AGE
soainfra-adminserver                         1/1     Running     0
53m
soainfra-osb-server1                         1/1     Running     0
50m
soainfra-soa-server1                         1/1     Running     0
50m
```

Verify the services

Enter the following command to see the services for the domain:

```
kubectl get services -n NAMESPACE
```

Here is an example of the output of this command. You can verify that services for Administration Server and Managed Servers (for SOA and Oracle Service Bus clusters) are created for soaosb domain type.

Sample list of services:

```
kubectl get services -n soans
NAME                            TYPE          CLUSTER-IP        EXTERNAL-IP
PORT(S)                        AGE
soainfra-adminserver            ClusterIP     None              <none>
30012/TCP,7001/TCP,7002/TCP    54m
soainfra-cluster-osb-cluster    ClusterIP     10.100.138.57     <none>
8002/TCP,8003/TCP              51m
soainfra-cluster-soa-cluster    ClusterIP     10.99.117.240     <none>
7003/TCP,7004/TCP              51m
soainfra-osb-server1            ClusterIP     None              <none>
```

```
8002/TCP,8003/TCP              51m
soainfra-osb-server2             ClusterIP      10.108.50.145    <none>
8002/TCP,8003/TCP              51m
soainfra-osb-server3             ClusterIP      10.108.71.8      <none>
8002/TCP,8003/TCP              51m
soainfra-osb-server4             ClusterIP      10.100.1.144     <none>
8002/TCP,8003/TCP              51m
soainfra-osb-server5             ClusterIP      10.108.57.147    <none>
8002/TCP,8003/TCP              51m
soainfra-soa-server1             ClusterIP      None             <none>
7003/TCP,7004/TCP              51m
soainfra-soa-server2             ClusterIP      10.97.165.179    <none>
7003/TCP,7004/TCP              51m
soainfra-soa-server3             ClusterIP      10.98.160.126    <none>
7003/TCP,7004/TCP              51m
soainfra-soa-server4             ClusterIP      10.105.164.133   <none>
7003/TCP,7004/TCP              51m
soainfra-soa-server5             ClusterIP      10.109.168.179   <none>
7003/TCP,7004/TCP              51m
```

Create Oracle SOA Suite Domain Using Helmfile

The Helmfile performs the following tasks using the Helm charts from local or remote charts:

- Installs the Oracle Database required for the domain:
  - Installs the certificate manager required for Oracle Database Operator.
  - Installs the Oracle Database Operator.
  - Creates the Oracle Single Instance Database.
- Installs the WebLogic Kubernetes Operator.
- Deploys the Oracle SOA Suite domain.
- Installs the ingress-based load balancers such as Traefik and NGINX.
- Sets up the path-based routing ingresses for application URL access.
- Triggers various events during deployment using Helmfile Hooks:
  - Labeling for using default domain namespace management of WebLogic Kubernetes Operator.
  - Wait for the domain to be up and running.
  - Collect domain pod logs after successful domain deployment.
  - Back-up domain home during deleting the domain deployment.

The helmfile charts for Oracle SOA Suite domain deployment are available at `${WORKDIR}/helm-charts`. Update the values available at `${WORKDIR}/helm-charts/values.yaml` of Helm charts required for deploying Oracle SOA Suite on Kubernetes.

Refer to the README.md for details and configuration parameters before running the helmfile.

6 Administration Guide

Administer Oracle SOA Suite domains in Kubernetes.

- **Setup a Load Balancer**

  Configure different load balancers for Oracle SOA Suite domains.
- **Enable Additional URL Access**

  Extend an existing ingress to enable additional application URL access for Oracle SOA Suite domains.
- **Configure SSL Certificates**

  Create and configure custom SSL certificates for Oracle SOA Suite domains.
- Replace Demonstration CA Signed Certificates

  Replace demonstration CA certificates
- **Monitor a Domain and Publish Logs**

  Monitor an Oracle SOA Suite domain and publish the WebLogic Server logs to Elasticsearch.
- **Expose the T3/T3S Protocol**

  Create a T3/T3S channel and the corresponding Kubernetes service to expose the T3/T3S protocol for the Administration Server and Managed Servers in an Oracle SOA Suite domain.
- **Deploy Composite Applications**

  Deploy composite applications for Oracle SOA Suite and Oracle Service Bus domains.
- **Persist Adapter Customizations**

  Persist the customizations done for Oracle SOA Suite adapters.
- **Perform WLST Operations**

  Perform WLST administration operations using a helper pod running in the same Kubernetes cluster as the Oracle SOA Suite domain.

Setup a Load Balancer

The WebLogic Kubernetes Operator supports ingress-based load balancers such as Traefik and NGINX (kubernetes/ingress-nginx).

- Traefik

  Configure the ingress-based Traefik load balancer for Oracle SOA Suite domains.
- NGINX

  Configure the ingress-based NGINX load balancer for Oracle SOA Suite domains.

Traefix

This section provides information about how to install and configure the ingress-based Traefik load balancer (version 2.2.1 or later for production deployments) to load balance Oracle SOA Suite domain clusters. You can configure Traefik for non-SSL, SSL termination, and end-to-end SSL access of the application URL. But for secure domain only end-to-end SSL configuration is applicable.

Follow these steps to set up Traefik as a load balancer for an Oracle SOA Suite domain in a Kubernetes cluster:

1. Install the Traefik (ingress-based) load balancer
2. Create an Ingress for the domain
3. Verify domain application URL access
4. Uninstall the Traefik ingress
5. Uninstall Traefik

Install the Traefik (ingress-based) load balancer

1. Use Helm to install the Traefik (ingress-based) load balancer. Use the values.yaml file in the sample but set kubernetes.namespaces specifically.

```
$ cd ${WORKDIR}
     $ kubectl create namespace traefik
     $ helm repo add traefik https://helm.traefik.io/traefik --force-update
```

Sample output:

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

2. Install Traefix.

```
$ helm install traefik  traefik/traefik \
          --namespace traefik \
          --values charts/traefik/values.yaml \
          --set  "kubernetes.namespaces={traefik}" \
          --set "service.type=NodePort" --wait
```

Sample output:

```
LAST DEPLOYED: Sun Sep 13 21:32:00 2020
        NAMESPACE: traefik
        STATUS: deployed
        REVISION: 1
        TEST SUITE: None
```

A sample values.yaml for deployment of Traefik:

```
image:
    pullPolicy: IfNotPresent
    ingressRoute:
```

```
    dashboard:
       enabled: true
       # Additional ingressRoute annotations (e.g. for kubernetes.io/
ingress.class)
       annotations: {}
       # Additional ingressRoute labels (e.g. for filtering IngressRoute
by custom labels)
       labels: {}
    providers:
    kubernetesCRD:
       enabled: true
    kubernetesIngress:
       enabled: true
       # IP used for Kubernetes Ingress endpoints
    ports:
    traefik:
       port: 9000
       expose: true
       # The exposed port for this service
       exposedPort: 9000
       # The port protocol (TCP/UDP)
       protocol: TCP
    web:
       port: 8000
       # hostPort: 8000
       expose: true
       exposedPort: 30305
       nodePort: 30305
       # The port protocol (TCP/UDP)
       protocol: TCP
       # Use nodeport if set. This is useful if you have configured
Traefik in a
       # LoadBalancer
       # nodePort: 32080
       # Port Redirections
       # Added in 2.2, you can make permanent redirects via entrypoints.
       # https://docs.traefik.io/routing/entrypoints/#redirection
       # redirectTo: websecure
    websecure:
       port: 8443
    #    # hostPort: 8443
       expose: true
       exposedPort: 30443
       # The port protocol (TCP/UDP)
       protocol: TCP
       nodePort: 30443
```

3. Verify the Traefik status and find the port number of the SSL and non-SSL services:

   ```
   $ kubectl get all -n traefik
   ```

   Sample output:

   ```
   NAME                           READY   STATUS    RESTARTS   AGE
       pod/traefik-5fc4947cf9-fbl9r   1/1     Running   5          7d17h
   ```

```
    NAME                TYPE        CLUSTER-IP       EXTERNAL-IP
PORT(S)                                              AGE
    service/traefik   NodePort    10.100.195.70    <none>        9000:31288/
TCP,30305:30305/TCP,30443:30443/TCP    7d17h

    NAME                       READY   UP-TO-DATE   AVAILABLE    AGE
    deployment.apps/traefik    1/1     1            1            7d17h

    NAME                                  DESIRED    CURRENT    READY    AGE
    replicaset.apps/traefik-5fc4947cf9    1          1          1        7d17h
```

4. Access the Traefik dashboard through the URL http://<MASTERNODE-HOSTNAME>:31288, with the HTTP host traefik.example.com:

   ```
   $ curl -H "host: <MASTERNODE-HOSTNAME>" http://<MASTERNODE-HOSTNAME>:31288/
   dashboard/
   ```

   Note:

   Make sure that you specify a fully qualified node name for <MASTERNODE-HOSTNAME>

5. Configure Traefik to manage ingresses created in this namespace, where traefik is the Traefik namespace and soans is the namespace of the domain:

   ```
   $ helm upgrade traefik traefik/traefik --namespace traefik     --reuse-
   values \
       --set "kubernetes.namespaces={traefik,soans}"
   ```

   Sample output:

   ```
   Release "traefik" has been upgraded. Happy Helming!
       NAME: traefik
       LAST DEPLOYED: Sun Sep 13 21:32:12 2020
       NAMESPACE: traefik
       STATUS: deployed
       REVISION: 2
       TEST SUITE: None
   ```

Create an Ingress for the domain

Create an ingress for the domain in the domain namespace by using the sample Helm chart. Here path-based routing is used for ingress. Sample values for default configuration are shown in the file ${WORKDIR}/charts/ingress-per-domain/values.yaml. By default, type is TRAEFIK, sslType is NONSSL, wlsDomain.secureEnabled is false, and domainType is soa. These values can be overridden by passing values through the command line or can be edited in the sample file values.yaml based on the type of configuration (NONSSL, SSL, and E2ESSL). If needed, you can update the ingress YAML file to define more path rules (in section spec.rules.host.http.paths) based on the domain application URLs that need to be accessed. The template YAML file for the Traefik (ingress-based) load balancer is located at ${WORKDIR}/charts/ingress-per-domain/templates/traefik-ingress.yaml.

Note:

Click here to view all the configuration parameters.

1. Choose an appropriate LOADBALANCER_HOSTNAME for accessing the Oracle SOA Suite domain application URLs.

   ```
   $ export LOADBALANCER_HOSTNAME=<LOADBALANCER_HOSTNAME>
   ```

   For example, if you are executing the commands from a master node terminal, where the master hostname is LOADBALANCER_HOSTNAME:

   ```
   $ export LOADBALANCER_HOSTNAME=$(hostname -f)
   ```

2. Install ingress-per-domain using Helm for NONSSL configuration:

   ```
    $ cd ${WORKDIR}
    $ helm install soa-traefik-ingress  \
        charts/ingress-per-domain \
        --namespace soans \
        --values charts/ingress-per-domain/values.yaml \
        --set "traefik.hostname=${LOADBALANCER_HOSTNAME}"
   ```

3. For secured access (SSL termination and E2ESSL) to the Oracle SOA Suite application, create a certificate, and generate a Kubernetes secret:

   ```
    $ openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /tmp/
   tls1.key -out /tmp/tls1.crt -subj "/CN=*"
    $ kubectl -n soans create secret tls soainfra-tls-cert --key /tmp/
   tls1.key --cert /tmp/tls1.crt
   ```

4. Create the Traefik TLSStore custom resource.

   In case of SSL termination, Traefik should be configured to use the user-defined SSL certificate. If the user-defined SSL certificate is not configured, Traefik will create a default SSL certificate. To configure a user-defined SSL certificate for Traefik, use the TLSStore custom resource. The Kubernetes secret created with the SSL certificate should be referenced in the TLSStore object. Run the following command to create the TLSStore:

   ```
   $ cat <<EOF | kubectl apply -f -
   apiVersion: traefik.containo.us/v1alpha1
   kind: TLSStore
   metadata:
     name: default
     namespace: soans
   spec:
     defaultCertificate:
       secretName:  soainfra-tls-cert
   EOF
   ```

5. Install ingress-per-domain using Helm for SSL configuration. The Kubernetes secret name should be updated in the template file. The template file also contains the following annotations:

   ```
    traefik.ingress.kubernetes.io/router.entrypoints: websecure
    traefik.ingress.kubernetes.io/router.tls: "true"
    traefik.ingress.kubernetes.io/router.middlewares: soans-wls-proxy-
   ssl@kubernetescrd
   ```

   The entry point for SSL termination access and the Middleware name should be updated in the annotation. The Middleware name should be in the form <namespace>-<middleware name>@kubernetescrd.

   ```
    $ cd ${WORKDIR}
    $ helm install soa-traefik-ingress  \
        charts/ingress-per-domain \
        --namespace soans \
        --values charts/ingress-per-domain/values.yaml \
        --set "traefik.hostname=${LOADBALANCER_HOSTNAME}" \
        --set sslType=SSL
   ```

6. Install ingress-per-domain using Helm for E2ESSL configuration for non-secure domain.

   To use the E2ESSL configuration, you must have created the Oracle SOA Suite domain with sslEnabled set to true. See Create Oracle SOA Suite domains for details.

   ```
    $ cd ${WORKDIR}$ helm install soa-traefik-ingress  \
               charts/ingress-per-domain \
           --namespace soans \
           --values charts/ingress-per-domain/values.yaml \
           --set sslType=E2ESSL --set wlsDomain.secureEnabled=true
   ```

   Sample output:

   ```
   NAME: soa-traefik-ingress
    LAST DEPLOYED: Fri Apr  9 09:47:27 2021
    NAMESPACE: soans
    STATUS: deployed
    REVISION: 1
    TEST SUITE: None
   ```

7. For NONSSL access to the Oracle SOA Suite application, get the details of the services by the ingress:

   ```
    $ kubectl describe ingress soainfra-traefik -n soans
   ```

   See all services supported by the above deployed ingress:

   ```
   Name:             soainfra-traefik
     Namespace:        soans
     Address:
     Default backend:  default-http-backend:80 (<error: endpoints "default-
   http-backend" not found>)
     Rules:
   ```

```
     Host                                                      Path  Backends
     ----                                                      ----  --------
     www.example.com

                                                              /em
           soainfra-adminserver:7001 (10.244.0.45:7001)
                                                              /weblogic/
ready            soainfra-adminserver:7001 (10.244.0.45:7001)

           soainfra-cluster-soa-cluster:7003
(10.244.0.46:8011,10.244.0.47:7003)
                                                               /soa-
infra                 soainfra-cluster-soa-cluster:7003
(10.244.0.46:8011,10.244.0.47:7003)
                                                               /soa/
composer              soainfra-cluster-soa-cluster:7003
(10.244.0.46:8011,10.244.0.47:7003)
                                                               /integration/
worklistapp   soainfra-cluster-soa-cluster:7003
(10.244.0.46:8011,10.244.0.47:7003)
    Annotations:                                                kubernetes.io/
ingress.class: traefik
    Events:                                                     <none>
```

8. For SSL access to the Oracle SOA Suite application, get the details of the services by the above deployed ingress:

```
$ kubectl describe ingress soainfra-traefik -n soans
```

See all services supported by the above deployed ingress:

```
```
  Name: soainfra-traefik
  Namespace: soans
  Address:
  Default backend: default-http-backend:80 (<error: endpoints "default-
http-backend" not found>)
  TLS:
  soainfra-tls-cert terminates www.example.com
  Rules:
  Host   Path Backends
  ----   ---- --------
  www.example.com

                          /em   soainfra-
adminserver:7001 ()
                          /weblogic/ready   soainfra-
adminserver:7001 ()
                                                    soainfra-cluster-soa-
cluster:7003 ()
                          /soa-infra   soainfra-cluster-soa-
cluster:7003 ()
                          /soa/composer   soainfra-cluster-soa-
cluster:7003 ()
                          /integration/worklistapp soainfra-cluster-soa-
```
```

```
cluster:7003 ()
  Annotations:              kubernetes.io/ingress.class: traefik
                          meta.helm.sh/release-name: soa-traefik-ingress
                          meta.helm.sh/release-namespace: soans
                          traefik.ingress.kubernetes.io/
router.entrypoints: websecure
                          traefik.ingress.kubernetes.io/
router.middlewares: soans-wls-proxy-ssl@kubernetescrd
                          traefik.ingress.kubernetes.io/router.tls: true
  Events:                   <none>
  ```
```

9. For E2ESSL access to the Oracle SOA Suite application, get the details of the services by the above deployed ingress:

```
$ kubectl describe IngressRouteTCP soainfra-traefik -n soans
```

See all services supported by the above deployed ingress:

```
 ```
  Name: soa-cluster-routetcp
  Namespace: soans
  Labels: app.kubernetes.io/managed-by=Helm
  Annotations: meta.helm.sh/release-name: soa-traefik-ingress
                meta.helm.sh/release-namespace: soans
  API Version: traefik.containo.us/v1alpha1
  Kind: IngressRouteTCP
  Metadata:
  Creation Timestamp: 2021-04-09T09:47:27Z
  Generation: 1
  Managed Fields:
  API Version: traefik.containo.us/v1alpha1
  Fields Type: FieldsV1
  fieldsV1:
   f:metadata:
    f:annotations:
      .:
      f:meta.helm.sh/release-name:
      f:meta.helm.sh/release-namespace:
    f:labels:
      .:
      f:app.kubernetes.io/managed-by:
   f:spec:
    .:
    f:entryPoints:
    f:routes:
    f:tls:
      .:
      f:passthrough:
   Manager: Go-http-client
   Operation: Update
   Time: 2021-04-09T09:47:27Z
   Resource Version: 548305
   Self Link: /apis/traefik.containo.us/v1alpha1/namespaces/soans/
ingressroutetcps/soa-cluster-routetcp
```

```
      UID:                 933e524c-b773-474b-a87f-560d69f08d4b
      Spec:
      Entry Points:
      websecure
      Routes:
        Match:  HostSNI(`HostName`)
        Services:
         Termination Delay:  400
        Name:                soainfra-adminserver
        Port:                7002
        Weight:              3
       Tls:
       Passthrough:  true
     Events:            <none>
     ```
```

10. To confirm that the load balancer noticed the new ingress and is successfully routing to the domain server pods, you can send a request to the URL for the "WebLogic ReadyApp framework", which should return an HTTP 200 status code, as follows:

```
 $ curl -v http://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_PORT}/weblogic/
ready
 *   Trying 149.87.129.203...
 > GET http://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_PORT}/weblogic/ready
HTTP/1.1
 > User-Agent: curl/7.29.0
 > Accept: */*
 > Proxy-Connection: Keep-Alive
 > host: ${LOADBALANCER_HOSTNAME}
 >
 < HTTP/1.1 200 OK
 < Date: Sat, 14 Mar 2020 08:35:03 GMT
 < Vary: Accept-Encoding
 < Content-Length: 0
 < Proxy-Connection: Keep-Alive
 <
 * Connection #0 to host localhost left intact
```

Verify domain application URL access

For NONSSL configuration

After setting up the Traefik (ingress-based) load balancer, verify that the domain application URLs are accessible through the non-SSL load balancer port 30305 for HTTP access. The sample URLs for Oracle SOA Suite domain of type soa are:

```
http://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_NON_SSLPORT}/weblogic/ready
    http://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_NON_SSLPORT}/em
    http://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_NON_SSLPORT}/soa-infra
    http://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_NON_SSLPORT}/soa/composer
    http://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_NON_SSLPORT}/integration/
worklistapp
```

For SSL configuration
```

After setting up the Traefik (ingress-based) load balancer, verify that the domain applications are accessible through the SSL load balancer port 30443 for HTTPS access. The sample URLs for Oracle SOA Suite domain of type soa are:

```
https://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_SSLPORT}/weblogic/ready
    https://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_SSLPORT}/em
    https://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_SSLPORT}/soa-infra
    https://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_SSLPORT}/soa/composer
    https://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_SSLPORT}/integration/
worklistapp
```

Uninstall the Traefik ingress

Uninstall and delete the ingress deployment:

```
$ helm delete soa-traefik-ingress  -n soans
```

Uninstall Traefik

```
$ helm delete traefik -n traefik
```

NGINX

This section provides information about how to install and configure the ingress-based NGINX load balancer to load balance Oracle SOA Suite domain clusters. You can configure NGINX for non-SSL, SSL termination, and end-to-end SSL access of the application URL. But for secure domain, only end-to-end SSL configuration is applicable.

Follow these steps to set up NGINX as a load balancer for an Oracle SOA Suite domain in a Kubernetes cluster:

1. See the official installation document for prerequisites.
2. Install the NGINX load balancer for non-SSL and SSL termination configuration
3. Generate secret for SSL access
4. Install NGINX load balancer for end-to-end SSL configuration
5. Configure NGINX to manage ingresses
6. Verify domain application URL access
7. Uninstall NGINX ingress
8. Uninstall NGINX

To get repository information, enter the following Helm commands:

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

Install the NGINX load balancer for non-SSL and SSL termination configuration

Deploy the `ingress-nginx` controller by using Helm on the domain namespace:

```
 $ helm install nginx-ingress -n soans \
        --set controller.service.type=NodePort \
        --set controller.admissionWebhooks.enabled=false \
        ingress-nginx/ingress-nginx
```

Here is the sample output:

```
NAME: nginx-ingress
  LAST DEPLOYED: Thu May  5 13:27:30 2022
  NAMESPACE: soans
  STATUS: deployed
  REVISION: 1
  TEST SUITE: None
  NOTES:
  The ingress-nginx controller has been installed.
  Get the application URL by running these commands:
  export HTTP_NODE_PORT=$(kubectl --namespace soans get services -o
jsonpath="{.spec.ports[0].nodePort}" nginx-ingress-ingress-nginx-controller)
  export HTTPS_NODE_PORT=$(kubectl --namespace soans get services -o
jsonpath="{.spec.ports[1].nodePort}" nginx-ingress-ingress-nginx-controller)
  export NODE_IP=$(kubectl --namespace soans 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:
      name: example
      namespace: foo
    spec:
      ingressClassName: nginx
      rules:
      - host: www.example.com
        http:
        paths:
          - pathType: Prefix
            backend:
              service:
                name: exampleService
                port:
                  number: 80
          path: /
   # This section is only required if TLS is to be enabled for the ingress
   tls:
    - hosts:
        - www.example.com
      secretName: example-tls

   If TLS is enabled for the ingress, a secret containing the certificate and
key must also be provided:
  apiVersion: v1
  kind: Secret
  metadata:
    name: example-tls
    namespace: foo
  data:
   tls.crt: <base64 encoded cert>
   tls.key: <base64 encoded key>
  type: kubernetes.io/tls
```

Generate secret for SSL access

For secured access (SSL and E2ESSL) to the Oracle SOA Suite application, create a certificate and generate secrets:

```
$ openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /tmp/tls1.key -
out /tmp/tls1.crt -subj "/CN=domain1.org"


$ kubectl -n soans create secret tls soainfra-tls-cert --key /tmp/tls1.key --
cert /tmp/tls1.crt
```

> **Note:**
>
> The value of `CN` is the host on which this ingress is to be deployed and secret name should be <domainUID>-tls-cert.

Install NGINX load balancer for end-to-end SSL configuration

1. Deploy the ingress-nginx controller by using Helm on the domain namespace:

```
  $ helm install nginx-ingress -n soans \
        --set controller.extraArgs.default-ssl-certificate=soans/soainfra-
tls-cert \
        --set controller.service.type=NodePort \
        --set controller.admissionWebhooks.enabled=false \
        --set controller.extraArgs.enable-ssl-passthrough=true  \
         ingress-nginx/ingress-nginx
```

   Sample output:

```
    NAME: nginx-ingress
    LAST DEPLOYED: Thu May  5 12:21:50 2022
    NAMESPACE: soans
    STATUS: deployed
    REVISION: 1
    TEST SUITE: None
    NOTES:
    The ingress-nginx controller has been installed.
    Get the application URL by running these commands:
    export HTTP_NODE_PORT=$(kubectl --namespace soans get services -o
jsonpath="{.spec.ports[0].nodePort}" nginx-ingress-ingress-nginx-
controller)
```

```
    export HTTPS_NODE_PORT=$(kubectl --namespace soans get services -o
jsonpath="{.spec.ports[1].nodePort}" nginx-ingress-ingress-nginx-
controller)
    export NODE_IP=$(kubectl --namespace soans 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:
        name: example
        namespace: foo
      spec:
        ingressClassName: nginx
        rules:
        - host: www.example.com
          http:
          paths:
            - pathType: Prefix
              backend:
                service:
                  name: exampleService
                  port:
                    number: 80
            path: /
     # This section is only required if TLS is to be enabled for the
Ingress
     tls:
      - hosts:
          - www.example.com
        secretName: example-tls

     If TLS is enabled for the Ingress, a Secret containing the
certificate and key must also be provided:
    apiVersion: v1
    kind: Secret
    metadata:
      name: example-tls
      namespace: foo
    data:
     tls.crt: <base64 encoded cert>
     tls.key: <base64 encoded key>
    type: kubernetes.io/tls
```

2. Check the status of the deployed ingress controller:

```
kubectl --namespace soans get services | grep ingress-nginx-controller
```

Sample output:

```
 nginx-ingress-ingress-nginx-controller   NodePort    10.106.186.235
<none>        80:32125/TCP,443:31376/TCP   19m
```

Configure NGINX to manage ingresses

1. Choose an appropriate `LOADBALANCER_HOSTNAME` for accessing the Oracle SOA Suite domain application URLs.

   ```
   export LOADBALANCER_HOSTNAME=<LOADBALANCER_HOSTNAME>
   ```

   For example, if you are executing the commands from a master node terminal, where the master hostname is `LOADBALANCER_HOSTNAME`:

   ```
   export LOADBALANCER_HOSTNAME=$(hostname -f)
   ```

2. Create an ingress for the domain in the domain namespace by using the sample Helm chart. The path-based routing is used for ingress. Sample values for default configuration are shown in the file `${WORKDIR}/charts/ingress-per-domain/values.yaml`. By default, type is `TRAEFIK` , sslType is `NONSSL`, wlsDomain.secureEnabled is `false` and domainType is `soa`. These values can be overridden by passing values through the command line or can be edited in the sample file `values.yaml`. If needed, you can update the ingress YAML file to define more path rules (in the section `spec.rules.host.http.paths`) based on the domain application URLs that need to be accessed. Update the template YAML file for the NGINX load balancer located at `${WORKDIR}/charts/ingress-per-domain/templates/nginx-ingress.yaml`.

   Note:

   See here for all the configuration parameters.

   ```
    $ cd ${WORKDIR}
    $ helm install soa-nginx-ingress  charts/ingress-per-domain \
        --namespace soans \
        --values charts/ingress-per-domain/values.yaml \
        --set "nginx.hostname=${LOADBALANCER_HOSTNAME}" \
        --set type=NGINX
   ```

   Sample output:

   ```
   NAME: soa-nginx-ingress
   LAST DEPLOYED: Fri Jul 24 09:34:03 2020
   NAMESPACE: soans
   STATUS: deployed
   REVISION: 1
   TEST SUITE: None
   ```

3. Install `ingress-per-domain` using Helm for SSL termination configuration:

```
$ cd ${WORKDIR}
$ helm install soa-nginx-ingress  charts/ingress-per-domain \
    --namespace soans \
    --values charts/ingress-per-domain/values.yaml \
    --set "nginx.hostname=${LOADBALANCER_HOSTNAME}" \
    --set type=NGINX --set sslType=SSL
```

4. Install `ingress-per-domain` using Helm for E2ESSL configuration for non-secure domain.

   Note:

   To use the `E2ESSL` configuration, you must have created the Oracle SOA Suite domain with `sslEnabled` set to `true`. See Create an Oracle SOA Suite domain.

```
$ cd ${WORKDIR}
$ helm install soa-nginx-ingress  charts/ingress-per-domain \
    --namespace soans \
    --values charts/ingress-per-domain/values.yaml \
    --set type=NGINX --set sslType=E2ESSL
```

   Sample output:

```
NAME: soa-nginx-ingress
LAST DEPLOYED: Fri Jul 24 09:34:03 2020
NAMESPACE: soans
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

5. Install `ingress-per-domain` using Helm for E2ESSL configuration for secure domain.

   Note:

   To use the E2ESSL configuration for secure domain, you must have created the Oracle SOA Suite domain with sslEnabled set to true and secureEnabled set to true. See Create an Oracle SOA Suite domain.

```
$ cd ${WORKDIR}    $ helm install soa-nginx-ingress  charts/ingress-per-
domain \
                --namespace soans \
                --values charts/ingress-per-domain/values.yaml \
                --set type=NGINX --set sslType=E2ESSL --set
          wlsDomain.secureEnabled=true
```

6. For **NONSSL access** to the Oracle SOA Suite application, get the details of the services by the ingress:

```
kubectl describe ingress soainfra-nginx -n soans
```

   Sample output of the services supported by the above deployed ingress:

```
Name:             soainfra-nginx
Namespace:        soans
Address:          100.111.150.225
Default backend:  default-http-backend:80 (<error: endpoints "default-http-
backend" not found>)
Rules:
Host                                                       Path  Backends
----                                                       ----  --------
domain1.org


                                                         /em
     soainfra-adminserver:7001 (10.244.0.45:7001)
                                                         /weblogic/
ready            soainfra-adminserver:7001 (10.244.0.45:7001)
                                                         /
     soainfra-cluster-soa-cluster:7003 (10.244.0.46:8011,10.244.0.47:7003)
                                                         /soa-
infra                 soainfra-cluster-soa-cluster:7003
(10.244.0.46:8011,10.244.0.47:7003)
                                                         /soa/
composer              soainfra-cluster-soa-cluster:7003
(10.244.0.46:8011,10.244.0.47:7003)
                                                         /integration/
worklistapp   soainfra-cluster-soa-cluster:7003
(10.244.0.46:8011,10.244.0.47:7003)
Annotations:                                                 <none>
Events:
Type    Reason  Age    From                       Message
----    ------  ----   ----                       -------
Normal  CREATE  2m32s  nginx-ingress-controller  Ingress soans/soainfra-
nginx
Normal  UPDATE  94s    nginx-ingress-controller  Ingress soans/soainfra-
nginx
```

7. For SSL access to the Oracle SOA Suite application, get the details of the services by the above deployed ingress:

```
 $ kubectl describe ingress soainfra-nginx -n soans
```

   Sample output of the services supported by the above deployed ingress:

```
 Name:             soainfra-nginx
 Namespace:        soans
 Address:          100.111.150.225
 Default backend:  default-http-backend:80 (<error: endpoints "default-
http-backend" not found>)
 TLS:
    soainfra-tls-cert terminates domain1.org
  Rules:
     Host                                                     Path  Backends
     ----                                                     ----  --------
      domain1.org

                                                       /em
       soainfra-adminserver:7001 (10.244.0.45:7001)
                                                       /weblogic/
 ready           soainfra-adminserver:7001 (10.244.0.45:7001)
                                                       /
       soainfra-cluster-soa-cluster:7003 (10.244.0.46:8011,10.244.0.47:7003)
                                                       /soa-
 infra                soainfra-cluster-soa-cluster:7003
 (10.244.0.46:8011,10.244.0.47:7003)
                                                       /soa/
 composer             soainfra-cluster-soa-cluster:7003
 (10.244.0.46:8011,10.244.0.47:7003)
                                                       /integration/
 worklistapp   soainfra-cluster-soa-cluster:7003
 (10.244.0.46:8011,10.244.0.47:7003)
  Annotations:                                               kubernetes.io/
 ingress.class: nginx

 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/ingress.allow-http: false
  Events:
    Type    Reason  Age    From                      Message
    ----    ------  ----   ----                      -------
    Normal  CREATE  3m47s  nginx-ingress-controller  Ingress soans/soainfra-
 nginx
    Normal  UPDATE  3m25s  nginx-ingress-controller  Ingress soans/soainfra-
 nginx
```

8. For E2ESSL access to the Oracle SOA Suite application, get the details of the services by the above deployed ingress:

```
 $  kubectl describe ingress  soainfra-nginx-e2essl -n soans
```

Sample output of the services supported by the above deployed ingress:

```
 Name:             soainfra-nginx-e2essl-admin
 Namespace:        soans
 Address:
 Default backend:  default-http-backend:80 (<error: endpoints "default-
http-backend" not found>)
 TLS:
  soainfra-tls-cert terminates admin.org
 Rules:
   Host        Path  Backends
```

```
    ----        ----  --------
    admin.org
                          soainfra-adminserver-nginx-ssl:7002
 (10.244.0.247:7002)
   Annotations:  kubernetes.io/ingress.class: nginx
                 meta.helm.sh/release-name: soa-nginx-ingress
                 meta.helm.sh/release-namespace: soans
                 nginx.ingress.kubernetes.io/ssl-passthrough: true
  Events:
  Type    Reason  Age   From                      Message
  ----    ------  ----  ----                      --------
  Normal  Sync    4s    nginx-ingress-controller  Scheduled for sync

  Name:             soainfra-nginx-e2essl-soa
  Namespace:        soans
  Address:
  Default backend:  default-http-backend:80 (<error: endpoints "default-
 http-backend" not found>)
  TLS:
   soainfra-tls-cert terminates soa.org
  Rules:
   Host        Path  Backends
   ----        ----  --------
   soa.org
                  /   soainfra-cluster-soa-cluster:7004 (10.244.0.249:7004)
  Annotations:  kubernetes.io/ingress.class: nginx
                meta.helm.sh/release-name: soa-nginx-ingress
                meta.helm.sh/release-namespace: soans
                nginx.ingress.kubernetes.io/ssl-passthrough: true
 Events:
    Type    Reason  Age   From                      Message
   ----    ------  ----  ----                      --------
  Normal  Sync    4s    nginx-ingress-controller  Scheduled for sync
```

Verify domain application URL access

NONSSL configuration

1. Get the `LOADBALANCER_NON_SSLPORT` NodePort of NGINX using the command:

```
LOADBALANCER_NON_SSLPORT=$(kubectl --namespace soans  get services -o
jsonpath="{.spec.ports[0].nodePort}" nginx-ingress-ingress-nginx-
controller)
echo ${LOADBALANCER_NON_SSLPORT}
```

2. Verify that the Oracle SOA Suite domain application URLs are accessible through the `LOADBALANCER_NON_SSLPORT`:

```
http://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_NON_SSLPORT}/weblogic/ready
http://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_NON_SSLPORT}/em
http://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_NON_SSLPORT}/soa-infra
http://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_NON_SSLPORT}/soa/composer
http://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_NON_SSLPORT}/integration/
worklistapp
```

SSL configuration

1. Get the `LOADBALANCER_SSLPORT` NodePort of NGINX using the command:

   ```
   LOADBALANCER_SSLPORT=$(kubectl --namespace soans  get services -o
   jsonpath="{.spec.ports[1].nodePort}" nginx-ingress-ingress-nginx-
   controller)
   echo ${LOADBALANCER_SSLPORT}
   ```

2. Verify that the Oracle SOA Suite domain application URLs are accessible through the `LOADBALANCER_SSLPORT`:

   ```
   https://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_SSLPORT}/weblogic/ready
   https://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_SSLPORT}/em
   https://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_SSLPORT}/soa-infra
   https://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_SSLPORT}/soa/composer
   https://${LOADBALANCER_HOSTNAME}:${LOADBALANCER_SSLPORT}/integration/
   worklistapp
   ```

E2ESSL configuration

1. To access the SOA Suite domain application URLs from a remote browser, update the browser host config file `/etc/hosts` (In Windows, `C:\Windows\System32\Drivers\etc\hosts`) with the IP address of the host on which the ingress is deployed with below entries:

   ```
   X.X.X.X  admin.org
   X.X.X.X  soa.org
   X.X.X.X  osb.org
   ```

   Note:

   - The value of X.X.X.X is the host IP address on which this ingress is deployed.
   - If you are behind any corporate proxy, make sure to update the browser proxy settings appropriately to access the host names updated `/etc/hosts` file.

2. Get the `LOADBALANCER_SSLPORT` NodePort of NGINX using the command:

   ```
   LOADBALANCER_SSLPORT=$(kubectl --namespace soans  get services -o
   jsonpath="{.spec.ports[1].nodePort}" nginx-ingress-ingress-nginx-
   controller)
   echo ${LOADBALANCER_SSLPORT}
   ```

3. Verify that the Oracle SOA Suite domain application URLs are accessible through `LOADBALANCER_SSLPORT`:

   ```
   https://admin.org:${LOADBALANCER_SSLPORT}/weblogic/ready
   https://admin.org:${LOADBALANCER_SSLPORT}/em
   https://soa.org:${LOADBALANCER_SSLPORT}/soa-infra
   https://soa.org:${LOADBALANCER_SSLPORT}/soa/composer
   ```

```
https://soa.org:${LOADBALANCER_SSLPORT}/integration/worklistapp
```

Note:

This is the default host name. If you have updated the host name in `values.yaml`, then use the updated values.

Uninstall NGINX ingress

Uninstall and delete the `ingress-nginx` deployment:

```
helm delete soa-nginx-ingress  -n soans
```

Uninstall NGINX

```
helm delete nginx-ingress -n soans
```

Enable Additional URL Access

This section provides information about how to extend an existing ingress (Non-SSL and SSL termination) to enable additional application URL access for Oracle SOA Suite domains.

The ingress per domain created in the steps in Setup a Load Balancer exposes the application paths defined in template YAML files present at ${WORKDIR}/charts/ingress-per-domain/templates/.

To extend an existing ingress with additional application URL access:

1. Update the template YAML file at ${WORKDIR}/charts/ingress-per-domain/templates/ to define additional path rules.
   For example, to extend an existing NGINX-based ingress with additional paths /path1 and /path2 of an Oracle SOA Suite cluster, update nginx-ingress-nonssl.yaml, nginx-ingress-ssl.yaml, or nginx-ingress-e2essl.yaml accordingly with additional paths:

```
# Copyright (c) 2020, 2022, Oracle and/or its affiliates.
# Licensed under the Universal Permissive License v 1.0 as shown at
https://oss.oracle.com/licenses/upl.
{{- if eq .Values.type "NGINX" }}
{{- if (eq .Values.sslType "NONSSL") }}
---
apiVersion: networking.k8s.io/v1
kind: Ingress
.
.
spec:
  rules:
  - host: '{{ .Values.nginx.hostname }}'
    http:
      paths:
      # Add new paths -- start
      - path: /path1
        backend:
            service:
              name: '{{ .Values.wlsDomain.domainUID }}-cluster-
{{ .Values.wlsDomain.soaClusterName | lower | replace "_" "-" }}'
              port:
                number: {{ .Values.wlsDomain.soaManagedServerPort  }}
      - path: /path2
        backend:
          service:
            name: '{{ .Values.wlsDomain.domainUID }}-cluster-
{{ .Values.wlsDomain.soaClusterName | lower | replace "_" "-" }}'
            port:
              number: {{ .Values.wlsDomain.soaManagedServerPort  }}
      # Add new paths -- end
      - path: /em
        backend:
.
.
{{- end }}
```

2. Get the Helm release name for the ingress installed in your domain namespace:

   ```
   helm ls -n <domain_namespace>
   ```

   For example, in the `soans` namespace:

   ```
   helm ls -n soans
   ```

   Sample output, showing the Helm release name for a NGINX-based ingress as `soa-nginx-ingress`:

   ```
   NAME                    NAMESPACE       REVISION
   UPDATED                                 STATUS        CHART
   APP VERSION
   soa-nginx-ingress       soans            1         2021-02-17
   13:42:03.252742314 +0000 UTC   deployed  ingress-per-domain-0.1.0     1.0
   ```

3. To extend the existing ingress per domain with additional paths defined in the template YAML, use the `helm upgrade` command:

   ```
   cd ${WORKDIR}
   helm upgrade <helm_release_for_ingress> \
       charts/ingress-per-domain \
       --namespace <domain_namespace> \
       --reuse-values
   ```

   Note:

   `helm_release_for_ingress` is the ingress name used in the corresponding helm install command for the ingress installation.

Sample command for a NGINX-based ingress soa-nginx-ingress in the soans namespace:

```
cd ${WORKDIR}
helm upgrade soa-nginx-ingress \
    charts/ingress-per-domain \
    --namespace soans \
    --reuse-values
```

This will upgrade the existing ingress to pick up the additional paths updated in the template YAML.

4. Verify that additional paths are updated into the existing ingress.
   a. Get the existing ingress deployed in the domain namespace:

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

      For example, in the soans namespace:

      ```
      kubectl get ingress -n soans
      ```

      Sample output, showing the existing ingress as `soainfra-nginx`:

      ```
      NAME                 CLASS    HOSTS         ADDRESS          PORTS     AGE
      soainfra-nginx     <none>   domain1.org  10.109.211.160   80, 443   xxd
      ```

   b. Describe the ingress object and verify that new paths are available and pointing to desired backends.
      Sample command and output, showing path and backend details for /path1 and /path2:

      ```
      kubectl describe ingress soainfra-nginx -n soans|grep path
                                                     /path1
      soainfra-cluster-soa-cluster:7003 (172.17.0.19:7003,172.17.0.20:7003)
                                                     /path2
      soainfra-cluster-soa-cluster:8011 (172.17.0.19:7003,172.17.0.20:7003)
      ```

Configure SSL Certificates

Secure Socket Layer (SSL) provides a secured communication for data sent over unsecured networks. In an SSL termination scenario, you can configure SSL between the client browser and the load balancer in your Oracle SOA Suite instance to ensure that applications are accessed securely.

In an SSL end-to-end scenario, an Oracle SOA Suite domain is configured to use a self-signed SSL certificate that was generated during domain creation. Clients will typically receive a message indicating that the signing CA for the certificate is unknown and not trusted.

- Create custom SSL certificates in an SSL end-to-end scenario
- Create custom SSL certificates in an SSL termination at a load balancer

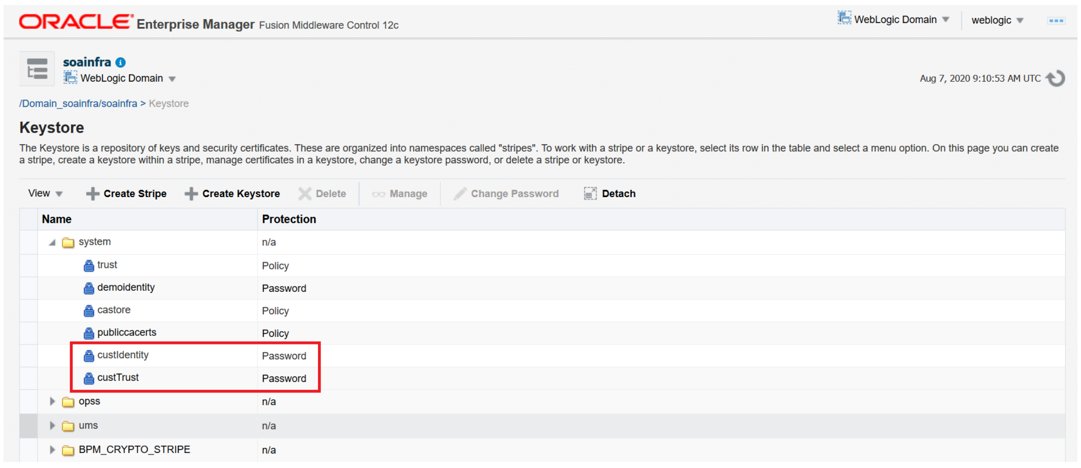
Create custom SSL certificates in an SSL end-to-end scenario

These steps describe how to replace the identity and trust keystore of an Oracle SOA Suite domain with a custom identity and custom trust keystore and register with digital certificates procured from any third party authority.

In this documentation, the registered domain is mydomain.com and the CA signed certificates are taken from mydomain.

Create a custom identity and custom trust keystore and generate a certificate signing request (CSR)
To create a custom identity and custom trust keystore and generate a CSR:

1. Log in to the Enterprise Manager (EM) Console and access the Keystores page by opening WebLogic Domain > Security > Keystore.
2. Under the system stripe, click Create Keystore to create a new keystore.
3. Provide the following details for custom identity:
   Keystore Name: custIdentity Protection: Select the Password option. Keystore Password: Enter the password. Confirm Password: Confirm the password.
4. Click Create Keystore to create another new keystore.
5. Provide the following details for custom trust:
   - **Keystore Name**: custTrust
   - **Protection**: Select the Password option.
   - **Keystore Password**: Enter the password.
   - **Confirm Password**: Confirm the password.


ORACLE Enterprise Manager Fusion Middleware Control 12c
WebLogic Domain
weblogic
soainfra
WebLogic Domain
Aug 7, 2020 9:10:53 AM UTC
/Domain_soainfra/soainfra > Keystore
View
Create Stripe
Create Keystore
Delete
Manage
Change Password
Detach

Keystore

The Keystore is a repository of keys and security certificates. These are organized into namespaces called "stripes". To work with a stripe or a keystore, select its row in the table and select a menu option. On this page you can create a stripe, create a keystore within a stripe, manage certificates in a keystore, change a keystore password, or delete a stripe or keystore.

Name	Protection
system	n/a
trust	Policy
demoidentity	Password
castore	Policy
publiccacerts	Policy
custIdentity	Password
custTrust	Password
opss	n/a
ums	n/a
BPM_CRYPTO_STRIPE	n/a

6. Click Manage on the custIdentity keystore name and provide the password that you specified previously.
7. Click Generate Keypair to create a new key pair, and provide the following details for custIdentity with alias as custIdentity and password:
   - Alias Name: custIdentity
   - Common Name: Common name, for example, soak8s.mydomain.com (Registered domain name)
   - Organizational Unit: Name of the organizational unit

   - Organization: Organization name
   - Enter City, State, and Country names
   - Key Type: RSA
   - Key Size: 2048
   - Password: Enter the password
8. Click OK to generate the keypair.
9. Select the newly created keypair and click Generate CSR.
10. Export the created CSR, share it with Certificate Authority, such as digicert CA, and get root, intermediate, and signed certificates. The certificate is generated for the domain name you used in the Common Name field.

It is not mandatory to create identity and trust keystore under the `system` stripe that comes with default provisioning. You can create a new custom stripe and create identity and trust keystores under it.

Share the CSR with CA to get CA-signed certificate

1. Select the new keypair under the custIdentity and click Generate CSR.
2. Export the created CSR and share it with the Certificate Authority and get root, intermediate, and signed certificates. The certificate is generated for the domain name you used in the Common Name field.
3. Download the certificates shared in the zip file from the CA. The zip file contains one of the following:
   - the three certificates individually - root, intermediate, and signed certificates
   - root and intermediate certificates in one chain and signed certificate separately
4. Double-click the certificate chain for root and intermediate certificates. You can see the full chain when you click on the certification path.
5. Extract the root and intermediate certificates individually by going to the certification path, select the certificate to be extracted (root or intermediate) and click View Certificate.
6. On the View Certificates pop-up, select the Details tab and click Copy to File.
7. In the Certificate Export wizard, click Next, then select Base 64 encoded X.509 (CER), and then click Next. Export the certificate.
8. Name the exported certificate as root and intermediate certificates respectively.

Import CA certificates

Certificate Authority (CA) certificates must be imported in the following order: first the signed server certificate, then the intermediate certificate, and then the root certificate.

To import CA certificates:

1. Use WLST commands to import the certificate chain in the identity keystore (custIdentity):
   a. Combine the three certificates into a single text file called chain.pem in the following order: signed server certificate, followed by intermediate certificate, followed by root certificate:

      ```
      -----BEGIN CERTIFICATE-----
      <signed server certificate>
      -----END CERTIFICATE-----
      ```

```
-----BEGIN CERTIFICATE-----
<intermediate certificate>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<root certificate>
-----END CERTIFICATE-----
```

b. Place the chain.pem in /tmp from where you will be executing the kubectl commands (for example, on the master node).

c. Enter the following command to change the file ownership to 1000:1000 user/group:

```
sudo chown 1000:1000 /tmp/chain.pem
```

d. Copy /tmp/chain.pem into the Administration Server pod (for example, soainfra-adminserver):

```
kubectl cp /tmp/chain.pem soans/soainfra-adminserver:/tmp/chain.pem
```

e. Exec into the Administration Server pod to perform all operations:

```
kubectl exec -it soainfra-adminserver -n soans -- bash
```

f. Start WLST and access the Oracle Platform Security Services (OPSS) key store service:

```
cd /u01/oracle/oracle_common/common/bin/
./wlst.sh
:
:
wls:/offline> connect("weblogic","Welcome1","t3://soainfra-
adminserver:7001")
:
:
wls:/soainfra/serverConfig/> svc =
getOpssService(name='KeyStoreService')
```

g. Use the WLST importKeyStoreCertificate command to import chain.pem:

```
svc.importKeyStoreCertificate(appStripe='stripe', name='keystore',
password='password', alias='alias', keypassword='keypassword',
type='entrytype',filepath='absolute_file_path')
```

For example:

```
wls:/soainfra/serverConfig/>
svc.importKeyStoreCertificate(appStripe='system', name='custIdentity',
password=welcome1, alias='custIdentity', keypassword='welcome1',
type='CertificateChain', filepath='/tmp/chain.pem')
```

h. Exit WLST:

```
exit()
```

2. Use Oracle Enterprise Manager to import the certificate chain into the trust keystore (custTrust):
   a. Log in to the Enterprise Manager Console and access the Keystores page by opening **WebLogic domain** > **Security** > **Keystore**.
   b. Select the trust keystore (custTrust) and click **Manage**.
   c. Click Import Certificate and import the certificates in this order:
      - The signed server certificate as a trusted certificate (alias mySignedCert)
      - The intermediate certificate from CA as a trusted certificate (alias myInterCA)
      - The root certificate from CA as a trusted certificate (alias myRootCA)

Synchronize the local keystore with the security store

Synchronize keystores to synchronize information between the domain home and the Oracle Platform Security Services (OPSS) store in the database.

To synchronize keystores:

1. Exec into the Administration server pod (for example, soainfra-adminserver).

```
kubectl exec -it soainfra-adminserver -n soans -- bash
```

2. Start WLST and access the Oracle Platform Security Services (OPSS) keystore service.

```
cd /u01/oracle/oracle_common/common/bin/
./wlst.sh
:
:
wls:/offline> connect("weblogic","Welcome1","t3://soainfra-
adminserver:7001")
:
:
wls:/soainfra/serverConfig/> svc = getOpssService(name='KeyStoreService')
```

3. Enter the following commands to synchronize the custom identity and custom trust keystores:

Note:

This step is necessary only if you are using the system stripe. You do not need to synchronize the keystores if you are using a custom stripe.

```
wls:/soainfra/serverConfig/> svc.listKeyStoreAliases(appStripe="system",
name="custIdentity", password=" ****", type="*")
wls:/soainfra/serverConfig/>
syncKeyStores(appStripe='system',keystoreFormat='KSS')
wls:/soainfra/serverConfig/> svc.listKeyStoreAliases (appStripe="system",
name="myKSSTrust", password="****", type="*")
wls:/soainfra/serverConfig/>
syncKeyStores(appStripe='system',keystoreFormat='KSS')
```

Update the WebLogic keystores with custom identity and trust

To update the WebLogic keystores with custom identity and custom trust:

1. In the WebLogic Remote Console, connect to the Administration server and then navigate to Edit Tree > Environment > Servers > AdminServer > Security > Keystores tab.
2. Change the Keystores to Custom Identity and Custom Trust and Save.
3. Provide the values for Custom Identity:
   - **Custom Identity Key Store File Name**:

     ```
     kss://system/custidentity
     ```

   - **Custom Identity Key Store Type**:

     ```
     KSS
     ```

   - **Custom Identity Key Store Pass Phrase**: enter password given while creating the

     ```
     custIdentity
     ```

     keystore.
4. Provide the values for Custom Trust:
   - **Custom Trust Key Store File Name**: kss://system/custTrust
   - **Custom Trust Key Store Type**: KSS
   - **Custom Trust Key Store Pass Phrase**: enter password given while creating the custTrust keystore.
5. Click **Save** and then **Commit** changes.
6. Open the SSL tab and provide the following details:
   - **Server Private Key Alias**: custIdentity (this is the alias given while creating the key pair in the custIdentity keystore.)
   - **Server Private Key Pass Phrase**: enter password given while creating the key pair under the custIdentity keystore.
7. In the Advanced section, change Hostname Verification to **None**.
8. Click **Save** and **Commit** changes.
9. Repeat steps 1 to 7 for all Managed Servers.
10. Restart the domain.
11. Once the servers are up and running, you can check if the SSL URLs show the updated certificates.

For more details, refer Administering Oracle SOA Cloud Service and Administering Oracle Fusion Middleware.

Create custom SSL certificates in an SSL termination at a load balancer

This section provides references to configure a custom SSL certificate at a load balancer.

There are multiple CA vendors in the marketplace today, each offering different levels of service at varying price points. Research and choose a CA vendor that meets your service-level and budget requirements.

For a CA vendor to issue you a CA-issued SSL certificate, you must provide the following information:

- Your custom domain name.
- Public information associated with the domain confirming you as the owner.
- Email address associated with the custom domain for verification.

Create a Certificate Signing Request (CSR) for your load balancer and submit the CSR to the CA vendor. After receiving the CA-issued certificate, refer to Administering Oracle SOA Cloud Service to import the CA-issued SSL certificate to the load balancer. If you are using openssl to create the certificates, you can refer to Manually Generate a Certificate Signing Request (CSR) Using OpenSSL to submit the CSR to the CA vendor.

Replace Demonstration CA Signed Certificates

Oracle highly recommends to use third-party Certificate Authority (CA) signed certificates or domain CA signed certificates when you deploy applications to a production environment. By default, any certificates created using the OPSS keystore service in the domain are signed using the demonstration CA.

Note:

These demonstration certificates should not be used in a production environment. The private key of the demonstration certificate is available to all installations of WebLogic Server, therefore each installation can generate a demonstration signed CA certificate using the same key. Hence, you cannot trust these certificates.

For more details, see here.

Replacing Demo CA Certificates With Domain CA Signed Certificates

A domain CA is a self-signed certificate that acts like a CA for a domain. Unlike a demonstration CA, the private key used in a domain CA certificate is unique to each domain, and provides more security.

You can create a domain CA certificate and replace all the demonstration CA certificates in a domain as described in Replacing Demo CA Certificates With Domain CA Signed Certificates.

Alternatively, you can run the helper script `custom-keystore.sh` available at `${WORKDIR}/custom-keystore` to configure custom keystore for Oracle SOA Suite domains. For more details, refer README.

Replacing Demo CA Certificates With Third-Party CA Signed Certificates

A third-party CA validates identities and issues certificates. To get the certificate, you must create a Certificate Request and submit it to the CA. The CA will authenticate the certificate requestor and create a digital certificate based on the request. To replace demonstration certificates with third-party CA signed certificates, see Replacing Demo CA Certificates With Third-Party CA Signed Certificates and Configure SSL certificates.

Monitor a Domain and Publish Logs

After the Oracle SOA Suite domain is set up, you can:

- Monitor the Oracle SOA Suite instance using Prometheus and Grafana
- Publish WebLogic Server logs into Elasticsearch
- Publish SOA server diagnostics logs into Elasticsearch

Monitor the Oracle SOA Suite instance using Prometheus and Grafana

Using the WebLogic Monitoring Exporter you can scrape runtime information from a running Oracle SOA Suite instance and monitor them using Prometheus and Grafana.

Set up monitoring, follow these steps to set up monitoring for an Oracle SOA Suite instance. For more details on WebLogic Monitoring Exporter, see here.

Publish WebLogic Server logs into Elasticsearch

WebLogic Server logs can be published to Elasticsearch using Fluentd. See Fluentd configuration steps.

Publish SOA server diagnostics logs into Elasticsearch

This section shows you how to publish diagnostics logs to Elasticsearch and view them in Kibana. For publishing operator logs, see this sample.

Prerequisites
If you have not already set up Elasticsearch and Kibana for logs collection, refer to this document and complete the setup.

Publish to Elasticsearch
The diagnostics or other logs can be pushed to Elasticsearch server using logstash pod. The logstash pod should have access to the shared domain home or the log location. In case of the Oracle SOA Suite domain, the persistent volume of the domain home can be used in the logstash pod. To create the logstash pod, follow these steps:

1. Get the domain home persistence volume claim details of the domain home of the Oracle SOA Suite domain. The following command lists the persistent volume claim details in the namespace - `soans`. In the example below, the persistent volume claim is `soainfra-domain-pvc`:

   ```
   kubectl get pvc -n soans
   ```

   Sample output:

   ```
   NAME                  STATUS   VOLUME              CAPACITY   ACCESS
   MODES   STORAGECLASS                    AGE
   soainfra-domain-pvc   Bound    soainfra-domain-pv   10Gi
   RWX            soainfra-domain-storage-class   xxd
   ```

2. Create the logstash configuration file (`logstash.conf`). Below is a sample logstash configuration to push diagnostic logs of all servers available at DOMAIN_HOME/servers/<server_name>/logs/-diagnostic.log:

```
input
{

  file
{

    path => "/u01/oracle/user_projects/domains/soainfra/servers/**/logs/*-
diagnostic.log"
    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}>" ]

  }

}

output
{

  elasticsearch
{

    hosts =>
["elasticsearch.default.svc.cluster.local:9200"]

  }

}
```

3. Copy the `logstash.conf` into `/u01/oracle/user_projects/domains` so that it can be used for logstash deployment, using the Administration Server pod (for example `soainfra-adminserver` pod in namespace `soans`):

```
kubectl cp logstash.conf  soans/soainfra-adminserver:/u01/oracle/
user_projects/domains --namespace soans
```

4. Create a deployment YAML (`logstash.yaml`) for the logstash pod using the domain home persistence volume claim. Make sure to point the logstash configuration file to the correct location (for example, copy logstash.conf to /u01/oracle/user_projects/domains/logstash.conf) and also the correct domain home persistence volume claim. Below is a sample logstash deployment YAML:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: logstash-soa
  namespace: soans
spec:
  selector:
    matchLabels:
      app: logstash-soa
  template: # create pods using pod definition in this template
    metadata:
      labels:
        app: logstash-soa
    spec:
      volumes:
      - name: soainfra-domain-storage-volume
        persistentVolumeClaim:
          claimName: soainfra-domain-pvc
      - name: shared-logs
        emptyDir: {}
      containers:
      - name: logstash
        image: logstash:6.6.0
        command: ["/bin/sh"]
        args: ["/usr/share/logstash/bin/logstash", "-f", "/u01/oracle/
user_projects/domains/logstash.conf"]
        imagePullPolicy: IfNotPresent
        volumeMounts:
        - mountPath: /u01/oracle/user_projects
          name: soainfra-domain-storage-volume
        - name: shared-logs
          mountPath: /shared-logs
        ports:
        - containerPort: 5044
          name: logstash
```

5. Deploy logstash to start publish logs to Elasticsearch:

```
kubectl create -f  logstash.yaml
```

6. Now, you can view the diagnostics logs using Kibana with index pattern “logstash-*”.

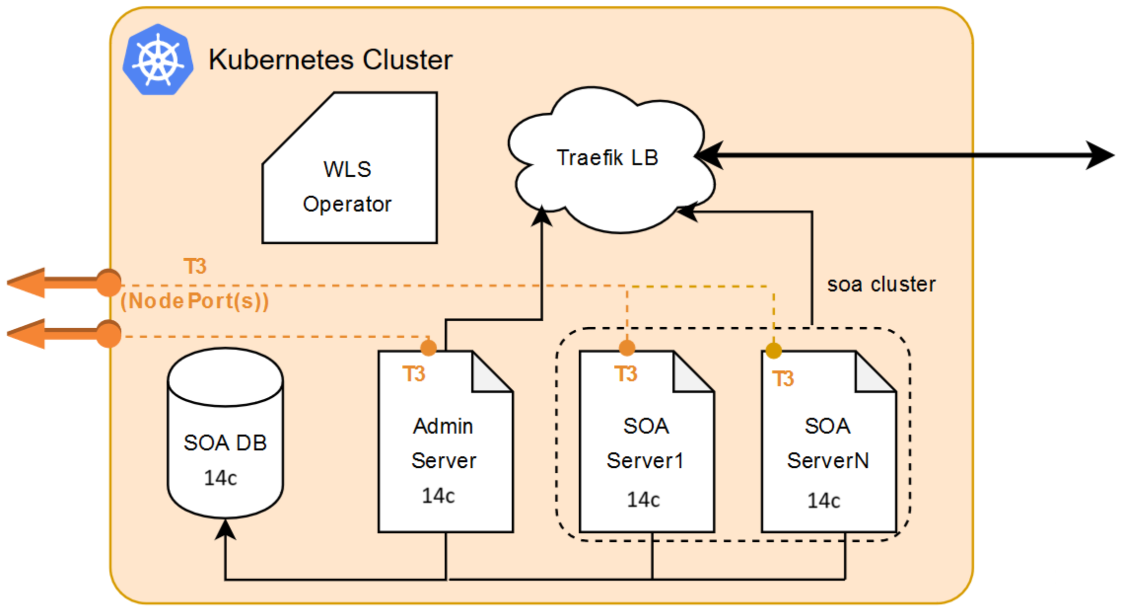
Expose the T3/T3S Protocol

You can create T3/T3S channels and the corresponding Kubernetes service to expose the T3/T3S protocol for the Administration Server and Managed Servers in an Oracle SOA Suite domain.

> **Note:**
>
> Oracle strongly recommends that you do not expose non-HTTPS traffic (T3/T3s/LDAP/IIOP/IIOPs) outside of the external firewall. You can control this access using a combination of network channels and firewalls.

The WebLogic Kubernetes Operator provides an option to expose a T3 channel for the Administration Server using the exposeAdminT3Channel setting during domain creation, then the matching T3 service can be used to connect. By default, when exposeAdminT3Channel is set, the WebLogic Kubernetes Operator environment exposes the NodePort for the T3 channel of the NetworkAccessPoint at 30012 (use t3ChannelPort to configure the port to a different value).

If you miss enabling exposeAdminT3Channel during domain creation, follow these steps to create a T3 channel for Administration Server manually.



- Expose a T3/T3S Channel for the Administration Server
- Expose T3/T3S for Managed Servers
- Remove T3/T3S Configuration

Expose a T3/T3S Channel for the Administration Server

To create a custom T3/T3S channel for the Administration Server that has a listen port **listen_port** and a paired public port **public_port**:

1. Create `t3_admin_config.py` with the following content:

```
admin_pod_name = sys.argv[1]
admin_port = sys.argv[2]
user_name = sys.argv[3]
```

```
password = sys.argv[4]
listen_port = sys.argv[5]
public_port = sys.argv[6]
public_address = sys.argv[7]
AdminServerName = sys.argv[8]
channelType = sys.argv[9]
print('custom admin_pod_name : [%s]' % admin_pod_name);
print('custom admin_port : [%s]' % admin_port);
print('custom user_name : [%s]' % user_name);
print('custom password : ********');
print('public address : [%s]' % public_address);
print('channel listen port : [%s]' % listen_port);
print('channel public listen port : [%s]' % public_port);
connect(user_name, password, 't3://' + admin_pod_name + ':' + admin_port)
edit()
startEdit()
cd('/')
cd('Servers/%s/' % AdminServerName )
if channelType == 't3':
   create('T3Channel_AS','NetworkAccessPoint')
   cd('NetworkAccessPoints/T3Channel_AS')
   set('Protocol','t3')
   set('ListenPort',int(listen_port))
   set('PublicPort',int(public_port))
   set('PublicAddress', public_address)
   print('Channel T3Channel_AS added')
elif channelType == 't3s':
   create('T3SChannel_AS','NetworkAccessPoint')
   cd('NetworkAccessPoints/T3SChannel_AS')
   set('Protocol','t3s')
   set('ListenPort',int(listen_port))
   set('PublicPort',int(public_port))
   set('PublicAddress', public_address)
   set('HttpEnabledForThisProtocol', true)
   set('OutboundEnabled', false)
   set('Enabled', true)
   set('TwoWaySSLEnabled', true)
   set('ClientCertificateEnforced', false)
else:
   print('channelType [%s] not supported',channelType)
activate()
disconnect()
```

2. Copy `t3_admin_config.py` into the domain home (for example, `/u01/oracle/user_projects/domains/soainfra`) of the Administration Server pod (for example, `soainfra-adminserver` in `soans` namespace).

```
 $ kubectl cp t3_admin_config.py soans/soainfra-adminserver:/u01/oracle/
user_projects/domains/soainfra
```

3. Run wlst.sh t3_admin_config.py by using exec into the Administration Server pod with the following parameters:
   - admin_pod_name: soainfra-adminserver # *Administration Server pod*
   - admin_port: 7001

- user_name: weblogic
- password: Welcome1 # *weblogic password*
- listen_port: 30014 # *New port for T3 Administration Server*
- public_port: 30014 # *Kubernetes NodePort which will be used to expose T3 port externally*
- public_address:
- AdminServerName: AdminServer # *Give administration Server name*
- channelType: t3 # *t3 or t3s protocol channel*

```
kubectl exec -it <Administration Server pod> -n <namespace> -- /u01/oracle/
oracle_common/common/bin/wlst.sh  <domain_home>/t3_admin_config.py
<Administration Server pod>  <Administration Server port>  weblogic
<password for weblogic> <t3 port on Administration Server> <t3 nodeport>
<master_ip> <AdminServerName> <channelType t3 or t3s>
```

For example:

```
kubectl exec -it soainfra-adminserver -n soans -- /u01/oracle/
oracle_common/common/bin/wlst.sh /u01/oracle/user_projects/domains/
soainfra/t3_admin_config.py soainfra-adminserver  7001 weblogic Welcome1
30014 30014 xxx.xxx.xxx.xxx AdminServer t3
```

4. Create `t3_admin_svc.yaml` with the following contents to expose T3 at NodePort `30014` for domainName and domainUID as `soainfra` and domain deployed in `soans` namespace:

> **Note:**
>
> For T3S, replace NodePort 30014 with the appropriate value used with `public_port` while creating the T3S channel using `wlst.sh` in the previous step.

```
apiVersion: v1
kind: Service
metadata:
   name: soainfra-adminserver-t3-external
   namespace: soans
   labels:
     weblogic.serverName: AdminServer
     weblogic.domainName: soainfra
     weblogic.domainUID: soainfra
spec:
  type: NodePort
  selector:
    weblogic.domainName: soainfra
    weblogic.domainUID: soainfra
    weblogic.serverName: AdminServer
  ports:
  - name: t3adminport
    protocol: TCP
    port: 30014
    targetPort: 30014
    nodePort: 30014
```

5. Create the NodePort service for port 30014:

   ```
   kubectl create -f t3_admin_svc.yaml
   ```

6. Verify that you can access T3 for the Administration Server with the following URL:

   ```
   t3://<master_ip>:30014
   ```

7. Similarly, you can access T3S as follows:

   a. First get the certificates from the Administration Server to be used for secured (T3S) connection from the client. You can export the certificate from the Administration Server with WLST commands. For example, to export the default demoidentity:

      Note:

      If you are using the custom SSL certificate, replace the steps accordingly.

      ```
      kubectl exec -it soainfra-adminserver -n soans -- bash
      /u01/oracle/oracle_common/common/bin/wlst.sh
      connect('weblogic','Welcome1','t3://soainfra-adminserver:7001')
      svc = getOpssService(name='KeyStoreService')
      svc.exportKeyStoreCertificate(appStripe='system', name='demoidentity',
      password='DemoIdentityKeyStorePassPhrase', alias='DemoIdentity',
      type='Certificate', filepath='/tmp/cert.txt/')
      ```

      These steps download the certificate at `/tmp/cert.txt`.

   b. Use the same certificates from the client side and connect using `t3s`. For example:

      ```
      export JAVA_HOME=/u01/jdk
      keytool -import -v -trustcacerts -alias soadomain -file cert.txt -
      keystore $JAVA_HOME/lib/security/cacerts -keypass changeit -storepass
      changeit
      export WLST_PROPERTIES="-
      Dweblogic.security.SSL.ignoreHostnameVerification=true"
      cd $ORACLE_HOME/oracle_common/common/bin
      ./wlst.sh
        Initializing WebLogic Scripting Tool (WLST) ...
        Welcome to WebLogic Server Administration Scripting Shell
        Type help() for help on available commands
      wls:/offline> connect('weblogic','Welcome1','t3s://<Master IP
      address>:30014')
      ```

Expose T3/T3S for Managed Servers

To create a custom T3/T3S channel for all Managed Servers, with a listen port **listen_port** and a paired public port **public_port**:

1. Create `t3_ms_config.py` with the following content:

```
admin_pod_name = sys.argv[1]
admin_port = sys.argv[2]
user_name = sys.argv[3]
password = sys.argv[4]
listen_port = sys.argv[5]
public_port = sys.argv[6]
public_address = sys.argv[7]
managedNameBase = sys.argv[8]
ms_count = sys.argv[9]
channelType = sys.argv[10]
print('custom host : [%s]' % admin_pod_name);
print('custom port : [%s]' % admin_port);
print('custom user_name : [%s]' % user_name);
print('custom password : ********');
print('public address : [%s]' % public_address);
print('channel listen port : [%s]' % listen_port);
print('channel public listen port : [%s]' % public_port);

connect(user_name, password, 't3://' + admin_pod_name + ':' + admin_port)

edit()
startEdit()
for index in range(0, int(ms_count)):
  cd('/')
  msIndex = index+1
  cd('/')
  name = '%s%s' % (managedNameBase, msIndex)
  cd('Servers/%s/' % name )
  if channelType == 't3':
    create('T3Channel_MS','NetworkAccessPoint')
    cd('NetworkAccessPoints/T3Channel_MS')
    set('Protocol','t3')
    set('ListenPort',int(listen_port))
    set('PublicPort',int(public_port))
    set('PublicAddress', public_address)
    print('Channel T3Channel_MS added ...for ' + name)
  elif channelType == 't3s':
    create('T3SChannel_MS','NetworkAccessPoint')
    cd('NetworkAccessPoints/T3SChannel_MS')
    set('Protocol','t3s')
    set('ListenPort',int(listen_port))
    set('PublicPort',int(public_port))
    set('PublicAddress', public_address)
    set('HttpEnabledForThisProtocol', true)
    set('OutboundEnabled', false)
    set('Enabled', true)
    set('TwoWaySSLEnabled', true)
    set('ClientCertificateEnforced', false)
    print('Channel T3SChannel_MS added ...for ' + name)
  else:
    print('Protocol [%s] not supported' % channelType)
activate()
disconnect()
```

2. Copy `t3_ms_config.py` into the domain home (for example, `/u01/oracle/user_projects/domains/soainfra`) of the Administration Server pod (for example, `soainfra-adminserver` in `soans` namespace).

   ```
   kubectl cp t3_ms_config.py soans/soainfra-adminserver:/u01/oracle/
   user_projects/domains/soainfra
   ```

3. Run `wlst.sh t3_ms_config.p`y by exec into the Administration Server pod with the following parameters:

   - admin_pod_name: soainfra-adminserver # *Administration Server pod*
   - admin_port: 7001
   - user_name: weblogic
   - password: Welcome1 # *weblogic password*
   - listen_port: 30016 # *New port for T3 Managed Servers*
   - public_port: 30016 # *Kubernetes NodePort which will be used to expose T3 port externally*
   - public_address:
   - managedNameBase: soa_server # *Give Managed Server base name. For osb_cluster this will be osb_server*
   - ms_count: 5 # *Number of configured Managed Servers*
   - channelType: t3 # *channelType is t3 or t3s*

   ```
   admin_pod_name: soainfra-adminserver # Administration Server pod
   admin_port:  7001 user_name: weblogic password: Welcome1 # weblogic
   password listen_port: 30016 # New port for T3 Managed Serverspublic_port:
   30016 # Kubernetes NodePort which will be used to expose T3 port
             externallypublic_address: managedNameBase: soa_server # Give
   Managed Server base name. For osb_cluster this will
             be osb_serverms_count: 5 # Number of configured Managed
   ServerschannelType: t3 # channelType is t3 or t3s
   ```

   For example:

   ```
   kubectl exec -it soainfra-adminserver -n soans -- /u01/oracle/
   oracle_common/common/bin/wlst.sh /u01/oracle/user_projects/domains/
   soainfra/t3_ms_config.py soainfra-adminserver  7001 weblogic Welcome1
   30016 30016 xxx.xxx.xxx.xxx soa_server 5 t3
   ```

4. Create `t3_ms_svc.yaml` with the following contents to expose T3 at Managed Server port `30016` for domainName, domainUID as soainfra, and clusterName as soa_cluster for the SOA cluster. Similarly, you can create the Kubernetes service with clusterName as osb_cluster for an Oracle Service Bus cluster:

> **Note:**
>
> For T3S, replace NodePort 30016 with the appropriate value used with public_port while creating the T3S channel using `wlst.sh` in the previous step.

```
apiVersion: v1
kind: Service
metadata:
   name: soainfra-soa-cluster-t3-external
   namespace: soans
   labels:
     weblogic.clusterName: soa_cluster
     weblogic.domainName: soainfra
     weblogic.domainUID: soainfra
spec:
  type: NodePort
  selector:
    weblogic.domainName: soainfra
    weblogic.domainUID: soainfra
    weblogic.clusterName: soa_cluster
  ports:
  - name: t3soaport
    protocol: TCP
    port: 30016
    targetPort: 30016
    nodePort: 30016
```

5. Create the NodePort service for port 30016:

   ```
   kubectl create -f t3_ms_svc.yaml
   ```

6. Verify that you can access T3 for the Managed Server with the following URL:

   ```
   t3://<master_ip>:30016
   ```

7. Similarly, you can access T3S as follows:

   a. First get the certificates from the Administration Server to be used for secured (t3s) connection from client. You can export the certificate from the Administration Server with wlst commands. Sample commands to export the default demoidentity:

   > **Note:**
   >
   > In case you are using the custom SSL certificate, replaces the steps accordingly

   ```
   kubectl exec -it soainfra-adminserver -n soans -- bash
   /u01/oracle/oracle_common/common/bin/wlst.sh
   connect('weblogic','Welcome1','t3://soainfra-adminserver:7001')
   svc = getOpssService(name='KeyStoreService')
   svc.exportKeyStoreCertificate(appStripe='system', name='demoidentity',
   ```

```
password='DemoIdentityKeyStorePassPhrase', alias='DemoIdentity',
type='Certificate', filepath='/tmp/cert.txt/')
```

The above steps download the certificate at `/tmp/cert.txt`.

b. Use the same certificates from the client side and connect using t3s. For example:

```
export JAVA_HOME=/u01/jdk
keytool -import -v -trustcacerts -alias soadomain -file cert.txt -
keystore $JAVA_HOME/lib/security/cacerts -keypass changeit -storepass
changeit
export WLST_PROPERTIES="-
Dweblogic.security.SSL.ignoreHostnameVerification=true"
cd $ORACLE_HOME/oracle_common/common/bin
./wlst.sh
  Initializing WebLogic Scripting Tool (WLST) ...
  Welcome to WebLogic Server Administration Scripting Shell
  Type help() for help on available commands
wls:/offline> connect('weblogic','Welcome1','t3s://<Master IP
address>:30016')
```

Remove T3/T3S Configuration

1. For administration server, do the following:

   a. Create `t3_admin_delete.py` with the following content:

```
admin_pod_name = sys.argv[1]
admin_port = sys.argv[2]
user_name = sys.argv[3]
password = sys.argv[4]
AdminServerName = sys.argv[5]
channelType = sys.argv[6]
print('custom admin_pod_name : [%s]' % admin_pod_name);
print('custom admin_port : [%s]' % admin_port);
print('custom user_name : [%s]' % user_name);
print('custom password : ********');
connect(user_name, password, 't3://' + admin_pod_name + ':' +
admin_port)
edit()
startEdit()
cd('/')
cd('Servers/%s/' % AdminServerName )
if channelType == 't3':
   delete('T3Channel_AS','NetworkAccessPoint')
elif channelType == 't3s':
   delete('T3SChannel_AS','NetworkAccessPoint')
else:
   print('channelType [%s] not supported',channelType)
activate()
disconnect()
```

b. Copy `t3_admin_delete.py` into the domain home (for example, `/u01/oracle/user_projects/domains/soainfra`) of the Administration Server pod (for example, `soainfra-adminserver` in `soans namespace`).

```
kubectl cp t3_admin_delete.py soans/soainfra-adminserver:/u01/oracle/
user_projects/domains/soainfra
```

c. Run `wlst.sh t3_admin_delete.py` by exec into the Administration Server pod with the following parameters:

- admin_pod_name: soainfra-adminserver # *Administration Server pod*
- admin_port: 7001
- user_name: weblogic
- password: Welcome1 # *weblogic password*
- AdminServerName: AdminServer # *Give administration Server name*
- channelType: t3 # *T3 channel*

```
kubectl exec -it <Administration Server pod> -n <namespace> -- /u01/
oracle/oracle_common/common/bin/wlst.sh  <domain_home>/
t3_admin_delete.py <Administration Server pod>  <Administration Server
port>  weblogic <password for weblogic> <AdminServerName> <protocol t3
or t3s>
```

For example:

```
kubectl exec -it soainfra-adminserver -n soans -- /u01/oracle/
oracle_common/common/bin/wlst.sh /u01/oracle/user_projects/domains/
soainfra/t3_admin_delete.py soainfra-adminserver 7001 weblogic Welcome1
AdminServer t3
```

d. Delete the NodePort service for port `30014`:

```
kubectl delete -f t3_admin_svc.yaml
```

2. For managed servers, do the following:

   These steps delete the custom T3/T3S channel created by Expose T3/T3S for Managed Servers for all Managed Servers.

   a. Create `t3_ms_delete.py` with the following content:

```
admin_pod_name = sys.argv[1]
admin_port = sys.argv[2]
user_name = sys.argv[3]
password = sys.argv[4]
managedNameBase = sys.argv[5]
ms_count = sys.argv[6]
channelType = sys.argv[7]
print('custom host : [%s]' % admin_pod_name);
print('custom port : [%s]' % admin_port);
print('custom user_name : [%s]' % user_name);
print('custom password : ********');
connect(user_name, password, 't3://' + admin_pod_name + ':' +
admin_port)
edit()
startEdit()
for index in range(0, int(ms_count)):
  cd('/')
  msIndex = index+1
  cd('/')
  name = '%s%s' % (managedNameBase, msIndex)
  cd('Servers/%s/' % name )
  if channelType == 't3':
    delete('T3Channel_MS','NetworkAccessPoint')
  elif channelType == 't3s':
    delete('T3SChannel_MS','NetworkAccessPoint')
  else:
    print('Protocol [%s] not supported' % channelType)
activate()
disconnect()
```

b. Copy `t3_ms_delete.py` into the domain home (for example, `/u01/oracle/user_projects/domains/soainfra`) of the Administration Server pod (for example, `soainfra-adminserver` in `soans` namespace).

```
kubectl cp t3_ms_delete.py soans/soainfra-adminserver:/u01/oracle/
user_projects/domains/soainfra
```

c. Run `wlst.sh t3_ms_delete.py` by exec into the Administration Server pod with the following parameters:

- admin_pod_name: soainfra-adminserver # *Administration Server pod*
- admin_port: 7001
- user_name: weblogic
- password: Welcome1 # *weblogic password*
- managedNameBase: soa_server # *Give Managed Server base name. For osb_cluster this will be osb_server*
- ms_count: 5 # *Number of configured Managed Servers*
- channelType: t3 # *channelType is t3 or t3s*

```
kubectl exec -it <Administration Server pod> -n <namespace> -- /u01/
oracle/oracle_common/common/bin/wlst.sh  <domain_home>/t3_ms_delete.py
<Administration Server pod>  <Administration Server port>  weblogic
<password for weblogic> <t3 port on Managed Server> <t3 nodeport>
<master_ip> <managedNameBase> <ms_count> <channelType t3 or t3s>
```

For example:

```
kubectl exec -it soainfra-adminserver -n soans -- /u01/oracle/
oracle_common/common/bin/wlst.sh /u01/oracle/user_projects/domains/
soainfra/t3_ms_delete.py soainfra-adminserver 7001 weblogic Welcome1
soa_server 5 t3
```

d. Delete the NodePort service for port `30016` (or the NodePort used while creating the Kubernetes service):

```
kubectl delete -f t3_ms_svc.yaml
```

Deploy Composite Applications

Learn how to deploy the composite applications for Oracle SOA Suite and Oracle Service Bus domains.

- Deploy Using JDeveloper

  Deploy Oracle SOA Suite and Oracle Service Bus composite applications from Oracle JDeveloper to Oracle SOA Suite in the WebLogic Kubernetes Operator environment.
- Deploy Using Maven and Ant

  Deploy Oracle SOA Suite and Oracle Service Bus composite applications using the Maven and Ant based approach in an Oracle SOA Suite deployment.
- Deploy Using Composites in a Persistent Volume or Image

  Deploy Oracle SOA Suite and Oracle Service Bus composite applications artifacts in a persistent volume or in an image.

Deploy Using JDeveloper

Learn how to deploy Oracle SOA Suite and Oracle Service Bus composite applications from Oracle JDeveloper to an Oracle SOA Suite instance in the WebLogic Kubernetes Operator environment.

> **Note:**
>
> Use JDeveloper for development and test environments only. For a production environment, you should deploy using Application Control and WLST methods.

Deploy Oracle SOA Suite and Oracle Service Bus composite applications to Oracle SOA Suite from JDeveloper

Prerequisites

Secure domain

To deploy Oracle SOA Suite and Oracle Service Bus composite applications from Oracle JDeveloper in a secure domain we need to perform deployments using internal Kubernetes services. For more details, refer Deploy Oracle SOA Suite and Oracle Service Bus applications using JDeveloper to Oracle SOA Suite on Kubernetes.

Non-secure development

1. To deploy Oracle SOA Suite and Oracle Service Bus composite applications from Oracle JDeveloper, the Administration Server must be configured to expose a T3 channel. The WebLogic Kubernetes Operator provides an option to expose a T3 channel for the Administration Server using the `exposeAdminT3Channel` setting during domain creation, then the matching T3 service can be used to connect. By default, when `exposeAdminT3Channel` is set, the WebLogic Kubernetes Operator environment exposes

the `NodePort` for the T3 channel of the `NetworkAccessPoint` at `30012` (use `t3ChannelPort` to configure the port to a different value).

2. If you miss enabling exposeAdminT3Channel during domain creation, follow Expose a T3/T3S Channel for the Administration Server to expose a T3 channel manually.
3. Get the Kubernetes cluster master address and verify the T3 port that will be used for creating application server connections. Use the following command to get the T3 port:

   ```
   kubectl get service <domainUID>-<AdministrationServerName>-external -n
   <namespace>-o jsonpath='{.spec.ports[0].nodePort}'
   ```

   For example:

   ```
   kubectl get service soainfra-adminserver-external -n  soans -o
   jsonpath='{.spec.ports[0].nodePort}'
   ```

4. JDeveloper needs to access the Servers during deployment. In the WebLogic Kubernetes Operator environment, Administration and Managed Servers are pods and cannot be accessed directly by JDeveloper. As a workaround, you must configure the reachability of the Managed Servers:

   Note:

   The Managed Server T3 port is not exposed by default and opening this will have a security risk as the authentication method here is based on a userid/password. It is not recommended to do this on production instances.

   a. Decide on an external IP address to be used to configure access to the Managed Servers. Master or worker node IP address can be used to configure Managed Server reachability. In these steps, the Kubernetes cluster master IP is used for demonstration.
   b. Get the pod names of the Administration Server and Managed Servers (that is, `<domainUID>-<server name>`), which will be used to map in `/etc/hosts`.
   c. Update `/etc/hosts` (or in Windows, `C:\Windows\System32\Drivers\etc\hosts`) on the host where JDeveloper is running with the entries below, where:

      ```
      <Master IP> <Administration Server pod name>
      <Master IP> <Managed Server1 pod name>
      <Master IP> <Managed Server2 pod name>
      ```

      Sample /etc/hosts entries looks as follows, where `X.X.X.X` is the master node IP address:

      ```
      X.X.X.X soainfra-adminserver
      X.X.X.X soainfra-soa-server1
      X.X.X.X soainfra-soa-server2
      ```

d. Get the Kubernetes service name of the Oracle SOA Suite cluster to access externally with the master IP (or external IP):

```
kubectl get service <domainUID>-cluster-<soa-cluster> -n <namespace>
```

For example:

```
kubectl get service soainfra-cluster-soa-cluster -n soans
```

e. Create a Kubernetes service to expose the Oracle SOA Suite cluster service (<domainUID>-cluster-<soa-cluster>) externally with same port as the Managed Server:

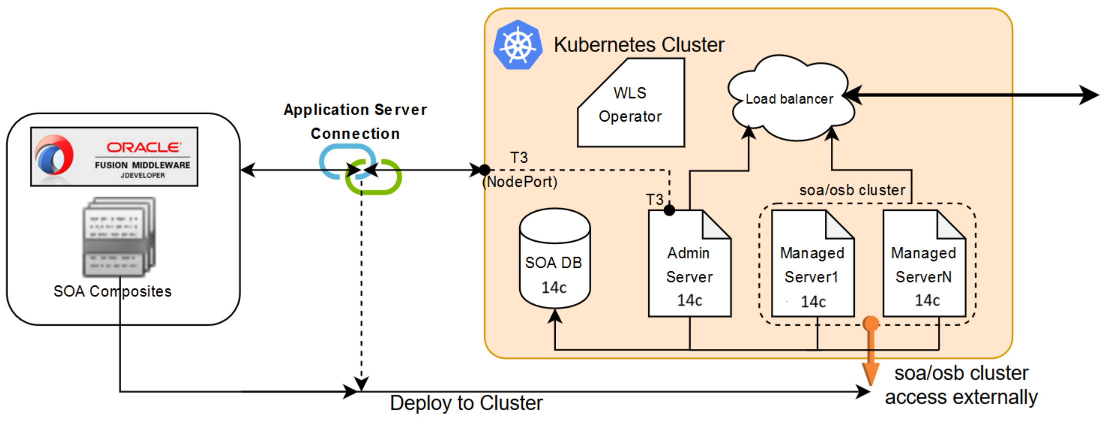
```
kubectl expose service  <domainUID>-cluster-<soa-cluster> --name
<domainUID>-<soa-cluster>-ext --external-ip=<Master IP> -n <namespace>
```

For example:

```
kubectl expose service  soainfra-cluster-soa-cluster --name soainfra-
cluster-soa-cluster-ext --external-ip=X.X.X.X -n soans
```

Note:

In a production environment, exposing the SOA cluster service with an external IP address is not recommended, as it can cause message drops on the SOA Managed Servers.

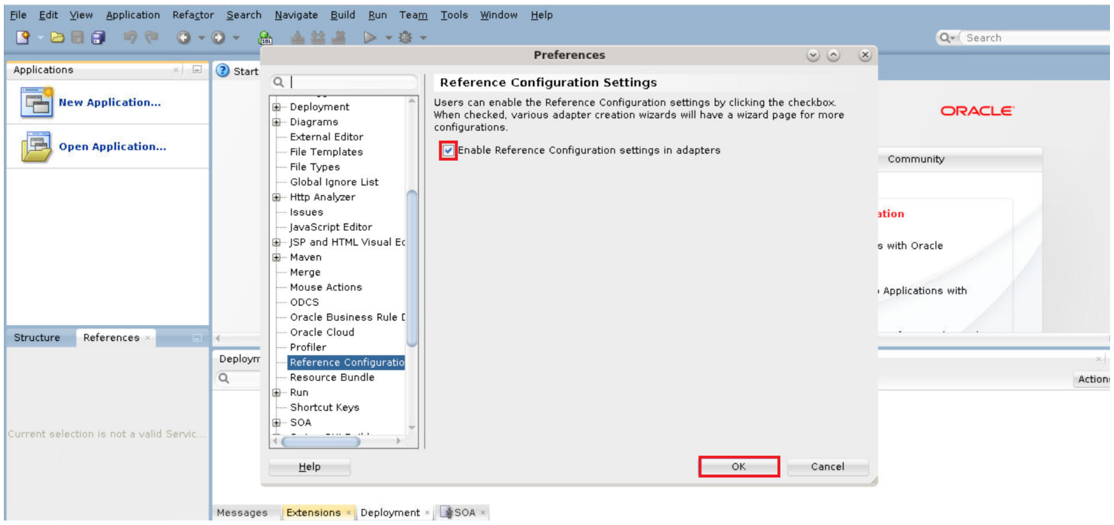


After you have completed the preceding prerequisite steps, perform the following steps:

1. Create an Application Server Connection in JDeveloper
2. Deploy SOA Composite Applications Using JDeveloper
3. Create an Application Server Connection in JDeveloper

Create an Application Server Connection in JDeveloper

Oracle SOA Suite in the WebLogic Kubernetes Operator environment is deployed in a *Reference Configuration domain*. If a SOA project is developed in Classic mode JDeveloper displays a Mismatch notification in the Deploy Composite Wizard. By default, JDeveloper is in Classic mode. To develop SOA projects in Reference Configuration mode, you must manually enable this feature in JDeveloper: a. From the File menu, select **Tools**, then **Preferences**. b. Select **Reference Configuration Settings**. c. Select **Enable Reference Configuration settings in adapters**.

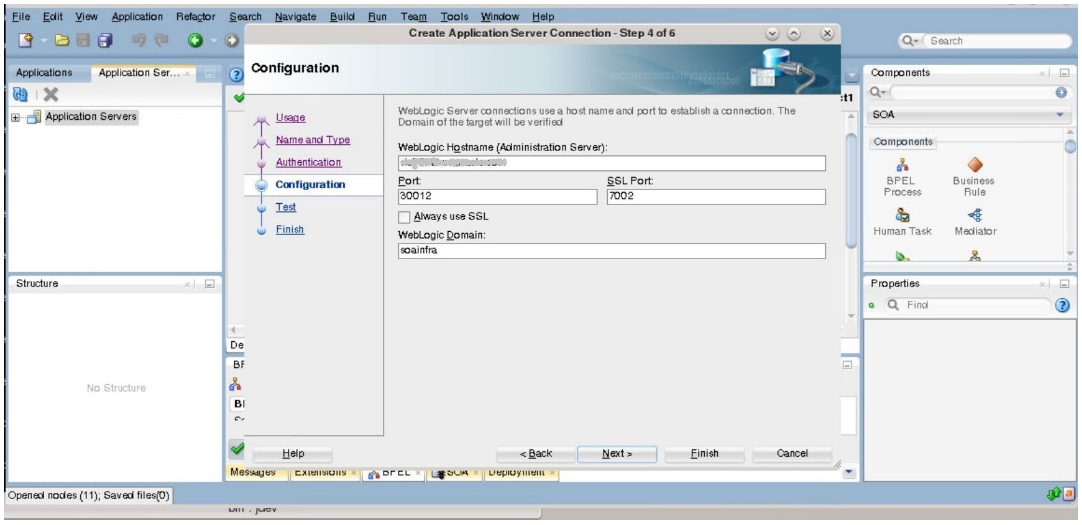


Reference Configuration Settings

Users can enable the Reference Configuration settings by clicking the checkbox. When checked, various adapter creation wizards will have a wizard page for more configurations.

[✓] Enable Reference Configuration settings in adapters

1. Create a new application server connection (for example `wls-k8s-op-connection`) in JDeveloper:



WebLogic Server connections use a host name and port to establish a connection. The Domain of the target will be verified.

WebLogic Hostname (Administration Server): [redacted]

Port: 30012

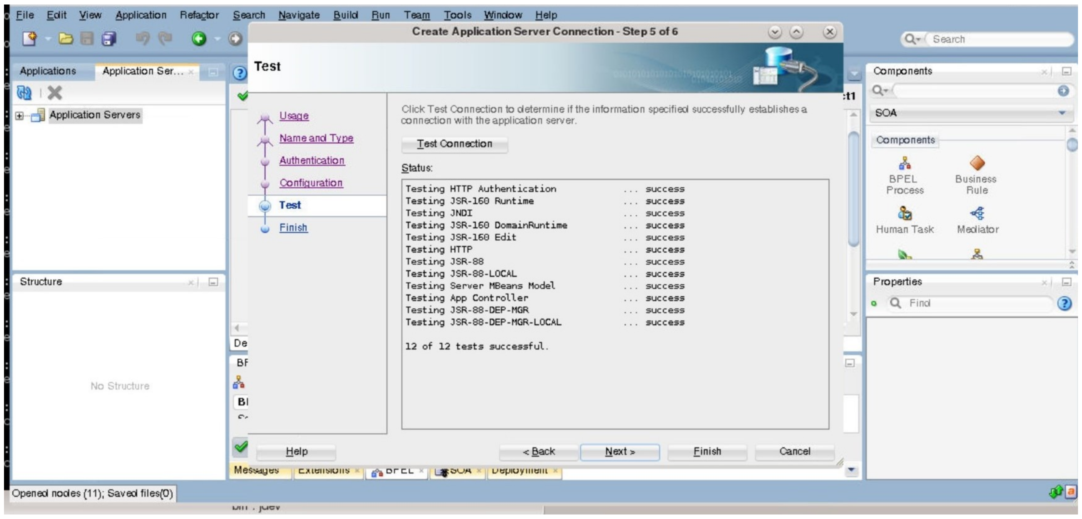
SSL Port: 7002

[ ] Always use SSL

WebLogic Domain: soainfra

2. In the configuration page, provide the WebLogic Hostname
3. Update the port details by following the steps in Prerequisites.
4. Enter the WebLogic Domain value (domainUID).

5. Test the connection to verify it is successful.



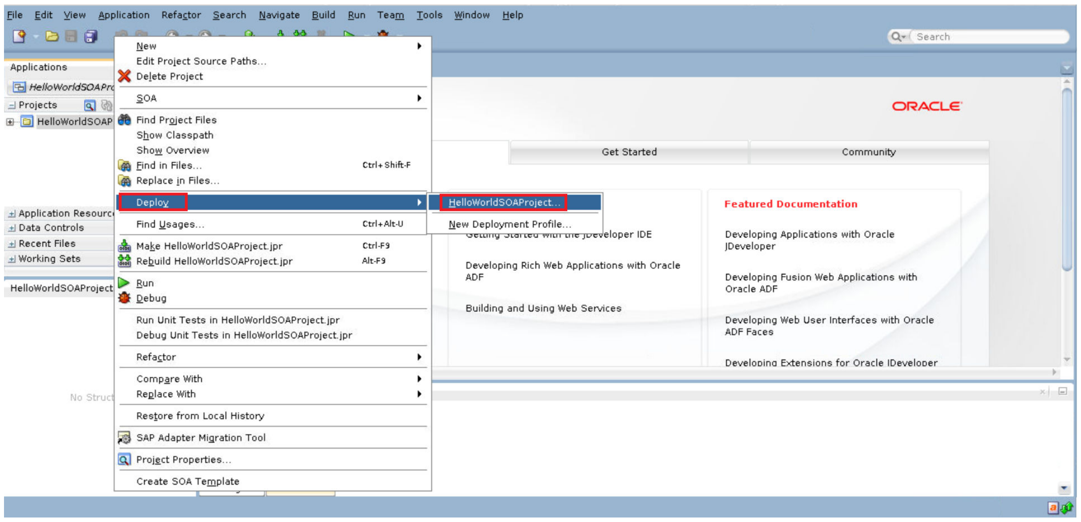
Click Test Connection to determine if the information specified successfully establishes a connection with the application server.

Status:

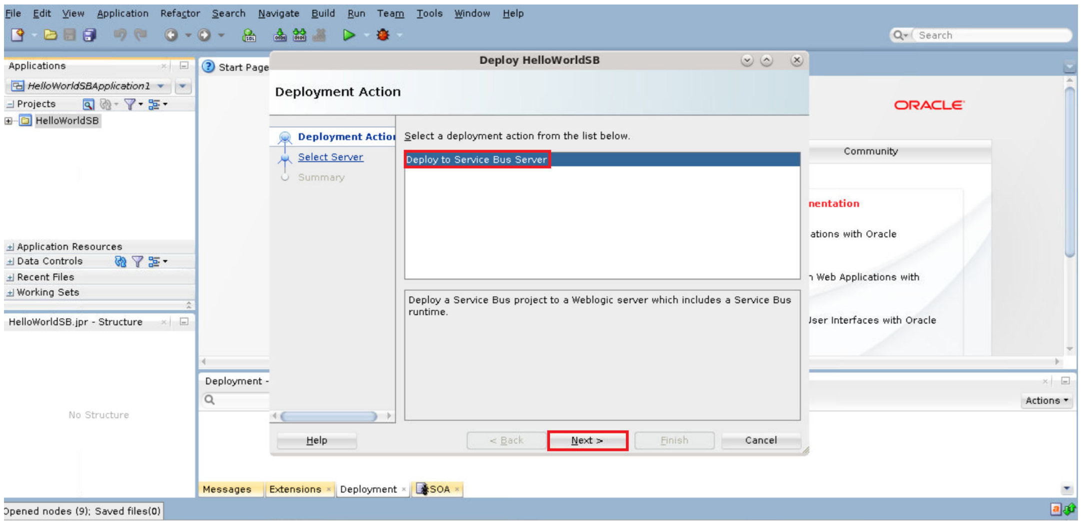
```
Testing HTTP Authentication         ... success
Testing JSR-160 Runtime             ... success
Testing JNDI                        ... success
Testing JSR-160 DomainRuntime       ... success
Testing JSR-160 Edit                ... success
Testing HTTP                        ... success
Testing JSR-88                      ... success
Testing JSR-88-LOCAL                ... success
Testing Server MBeans Model         ... success
Testing App Controller              ... success
Testing JSR-88-DEP-MGR              ... success
Testing JSR-88-DEP-MGR-LOCAL        ... success

12 of 12 tests successful.
```

Deploy SOA Composite Applications Using JDeveloper

1. In JDeveloper, right-click the SOA project you want to deploy and select **Deploy** to display the deployment wizard.


New
Edit Project Source Paths...
Delete Project
SOA
Find Project Files
Show Classpath
Show Overview
Find in Files... Ctrl+Shift-F
Replace in Files...
Deploy
Find Usages... Ctrl+Alt-U
Make HelloWorldSOAProject.jpr Ctrl-F9
Rebuild HelloWorldSOAProject.jpr Alt-F9
Run
Debug
Run Unit Tests in HelloWorldSOAProject.jpr
Debug Unit Tests in HelloWorldSOAProject.jpr
Refactor
Compare With
Replace With
Restore from Local History
SAP Adapter Migration Tool
Project Properties...
Create SOA Template
HelloWorldSOAProject...
New Deployment Profile...

2. In the Deployment Action page, select **Deploy to Application Server** and click **Next**.

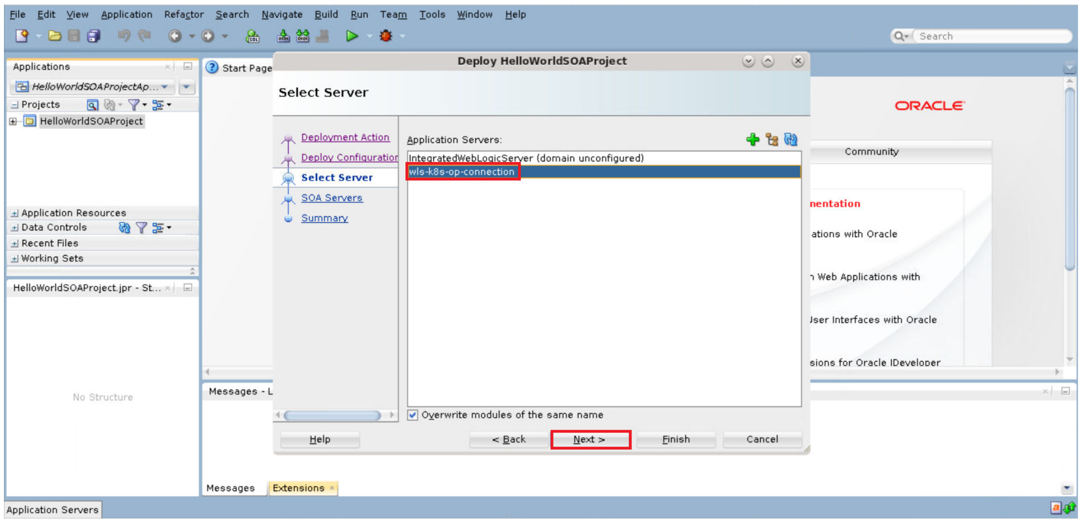


Select a deployment action from the list below.

Deploy to Service Bus Server

Deploy a Service Bus project to a Weblogic server which includes a Service Bus runtime.

3. In the Deployment Configuration page, select the appropriate options and click **Next**.



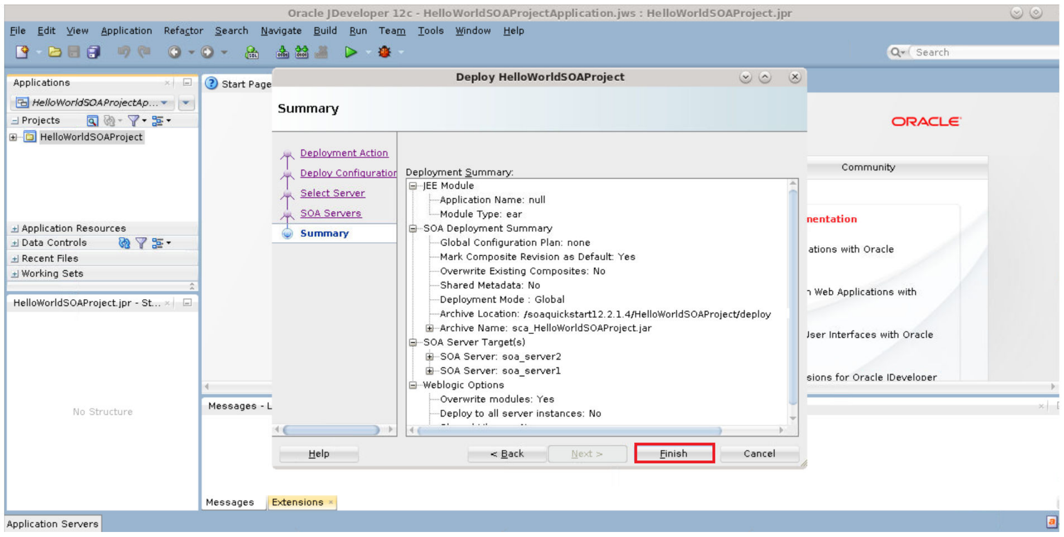
Application Servers:

IntegratedWebLogicServer (domain unconfigured)

wls-k8s-op-connection

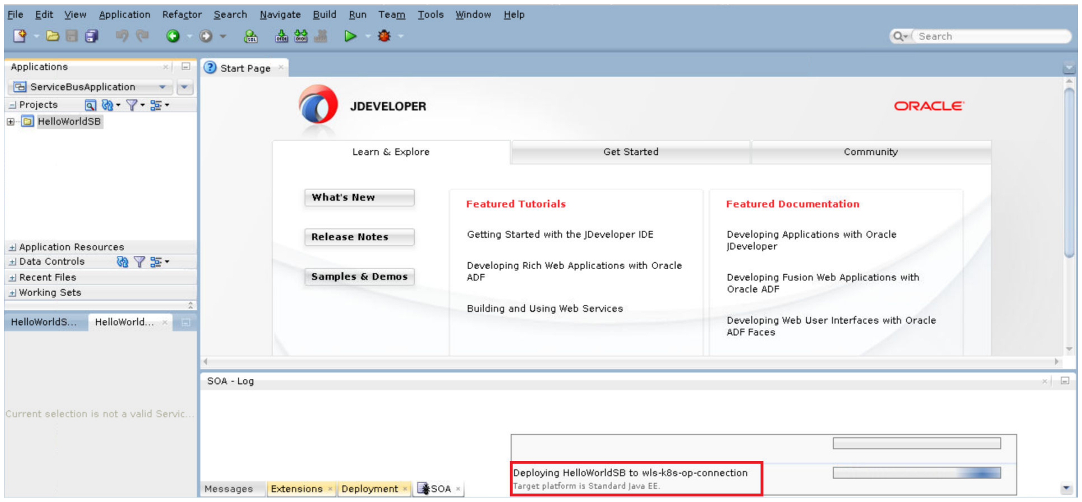
[✓] Overwrite modules of the same name

4. In the Select server page, select the application server connection (`wls-k8s-op-connection`) that was created earlier and click **Next**.
5. If the Prerequisites were configured correctly, the lookup discovers the Managed Servers for deploying the composite.
6. Using the application server connection, the Managed Servers (Oracle SOA Suite cluster) are listed on the SOA Servers page. Select the Oracle SOA Suite cluster and click **Next**.
7. On the Summary page, click **Finish** to start deploying the composites to the Oracle SOA Suite cluster.

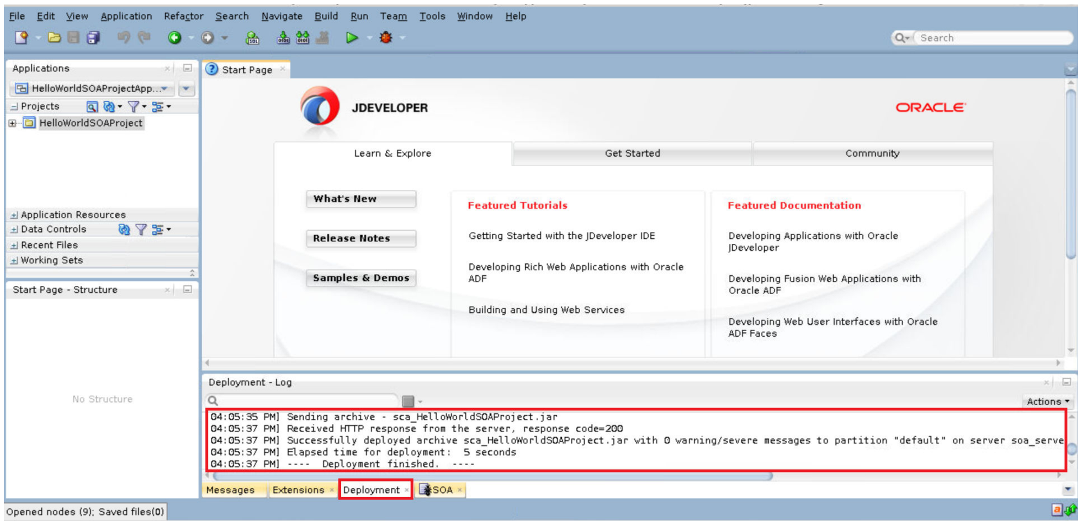

Deploy HelloWorldSOAProject

Summary

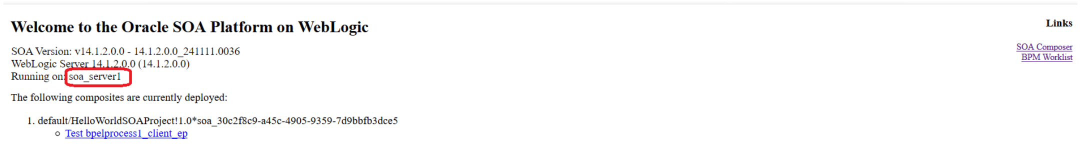
```
Deployment Summary:
JEE Module
    Application Name: null
    Module Type: ear
SOA Deployment Summary
    Global Configuration Plan: none
    Mark Composite Revision as Default: Yes
    Overwrite Existing Composites: No
    Shared Metadata: No
    Deployment Mode : Global
    Archive Location: /soaquickstart12.2.1.4/HelloWorldSOAProject/deploy
    Archive Name: sca_HelloWorldSOAProject.jar
SOA Server Target(s)
    SOA Server: soa_server2
    SOA Server: soa_server1
Weblogic Options
    Overwrite modules: Yes
    Deploy to all server instances: No
```


```
Deploying HelloWorldSB to wls-k8s-op-connection
Target platform is Standard Java EE.
```

8. Verify logs on JDeveloper to confirm successful deployment.


```
04:05:35 PM] Sending archive - sca_HelloWorldSOAProject.jar
04:05:37 PM] Received HTTP response from the server, response code=200
04:05:37 PM] Successfully deployed archive sca_HelloWorldSOAProject.jar with 0 warning/severe messages to partition "default" on server soa_serve
04:05:37 PM] Elapsed time for deployment:  5 seconds
04:05:37 PM] ----  Deployment finished.  ----
```

9. Enter the soa-infra URLs in a browser to confirm the composites are deployed on both servers of the Oracle SOA Suite cluster.



Welcome to the Oracle SOA Platform on WebLogic

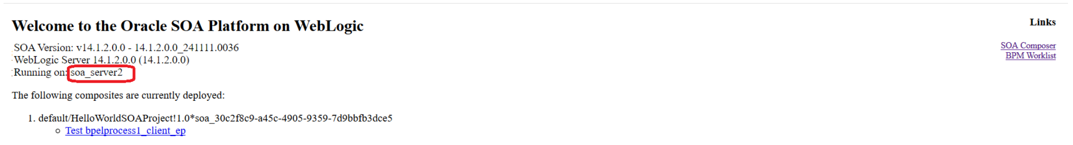
SOA Version: v14.1.2.0.0 - 14.1.2.0.0_241111.0036
WebLogic Server 14.1.2.0.0 (14.1.2.0.0)
Running on: soa_server1

The following composites are currently deployed:

1. default/HelloWorldSOAProject!1.0*soa_30c2f8c9-a45c-4905-9359-7d9bbfb3dce5
   - Test bpelprocess1_client_ep

Links

SOA Composer
BPM Worklist



Welcome to the Oracle SOA Platform on WebLogic

SOA Version: v14.1.2.0.0 - 14.1.2.0.0_241111.0036
WebLogic Server 14.1.2.0.0 (14.1.2.0.0)
Running on: soa_server2

The following composites are currently deployed:

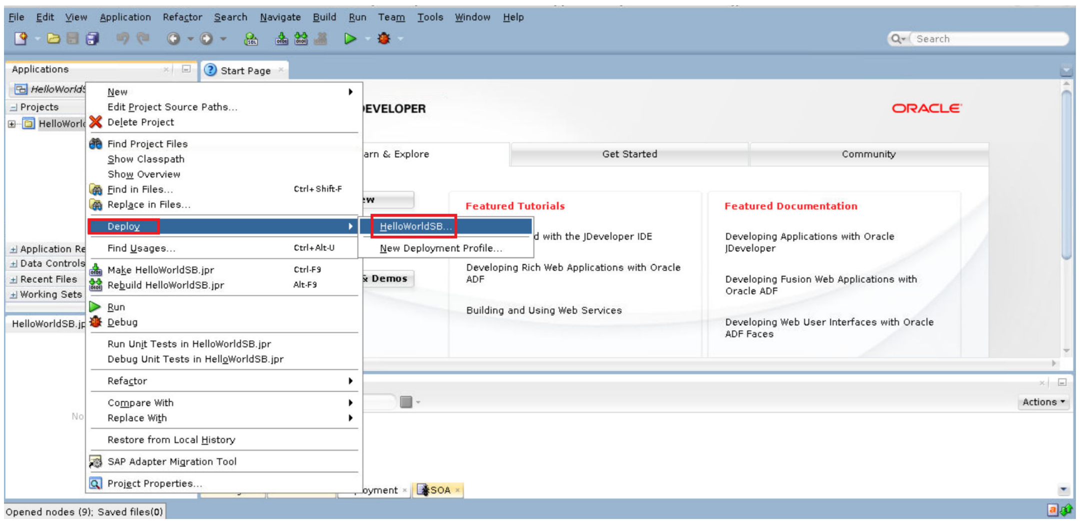
1. default/HelloWorldSOAProject!1.0*soa_30c2f8c9-a45c-4905-9359-7d9bbfb3dce5
   - Test bpelprocess1_client_ep

Links

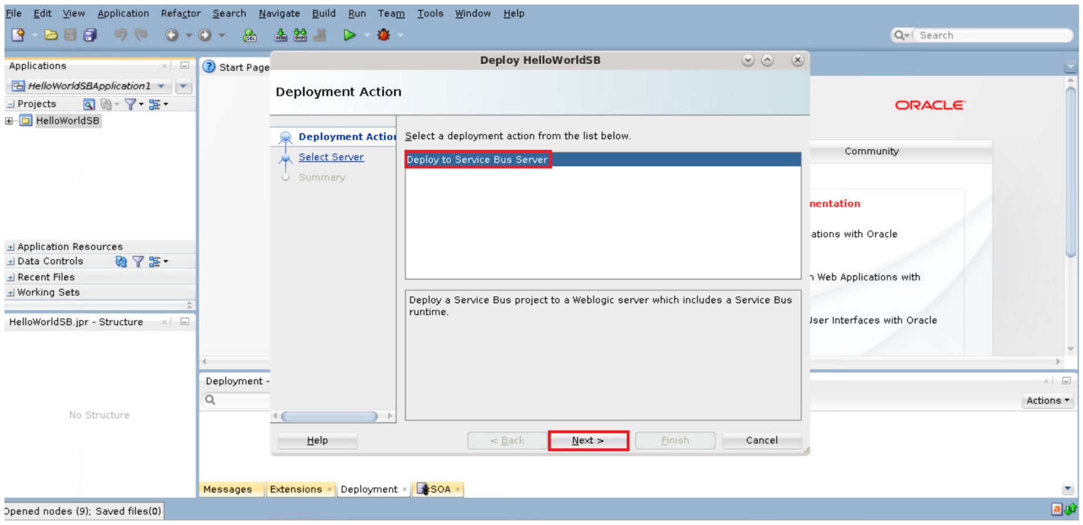
SOA Composer
BPM Worklist

Deploy Oracle Service Bus Composite Applications Using JDeveloper

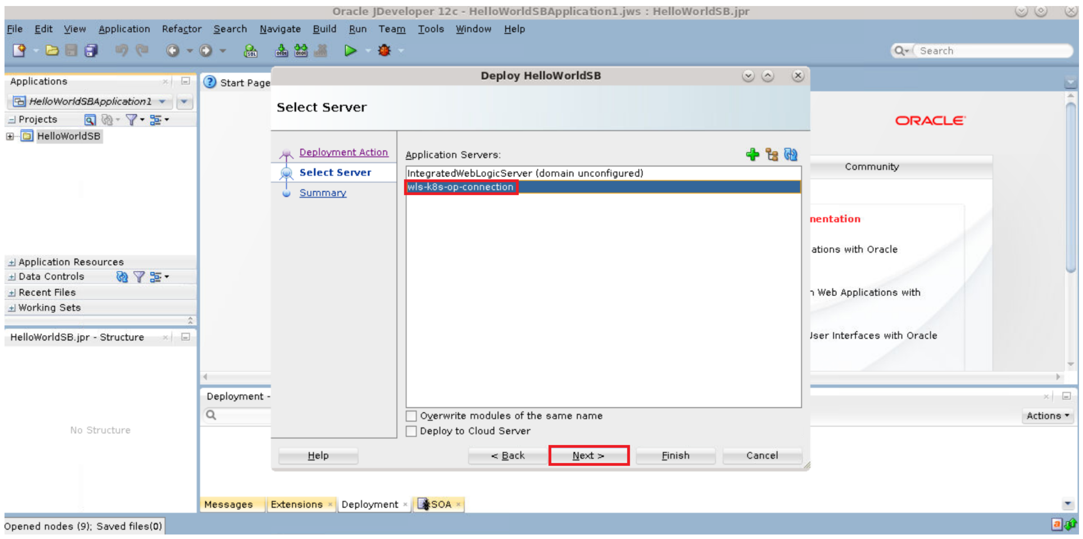
1. In JDeveloper, right-click the Oracle Service Bus project you want to deploy and select **Deploy** to display the deployment wizard.


File Edit View Application Refactor Search Navigate Build Run Team Tools Window Help
Search
Applications
Start Page
HelloWorldS
Projects
HelloWorl
New
Edit Project Source Paths...
Delete Project
Find Project Files
Show Classpath
Show Overview
Find in Files... Ctrl+Shift-F
Replace in Files...
Deploy
HelloWorldSB...
New Deployment Profile...
Find Usages... Ctrl+Alt-U
Make HelloWorldSB.jpr Ctrl-F9
Rebuild HelloWorldSB.jpr Alt-F9
Run
Debug
Run Unit Tests in HelloWorldSB.jpr
Debug Unit Tests in HelloWorldSB.jpr
Refactor
Compare With
Replace With
Restore from Local History
SAP Adapter Migration Tool
Project Properties...
Application Re
Data Controls
Recent Files
Working Sets
HelloWorldSB.jp
EVELOPER
ORACLE
arn & Explore
Get Started
Community
Featured Tutorials
d with the JDeveloper IDE
Developing Rich Web Applications with Oracle ADF
Building and Using Web Services
Featured Documentation
Developing Applications with Oracle JDeveloper
Developing Fusion Web Applications with Oracle ADF
Developing Web User Interfaces with Oracle ADF Faces
Demos
Actions
SOA
Opened nodes (9); Saved files(0)

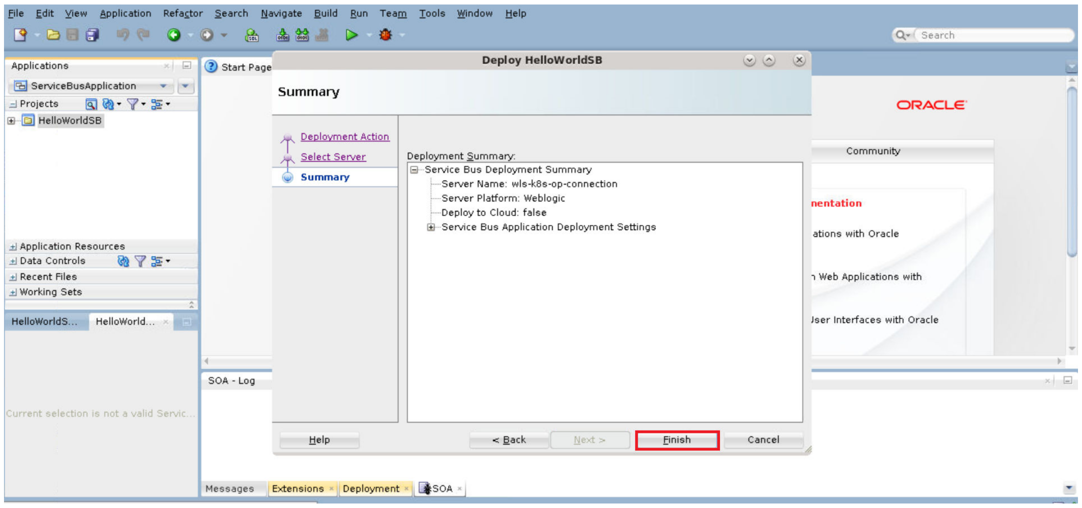
2. In the Deployment Action page, select **Deploy to Application Server** and click **Next**.


Deploy HelloWorldSB
Deployment Action
Deployment Action
Select Server
Summary
Select a deployment action from the list below.
Deploy to Service Bus Server
Deploy a Service Bus project to a Weblogic server which includes a Service Bus runtime.
Help
< Back
Next >
Finish
Cancel

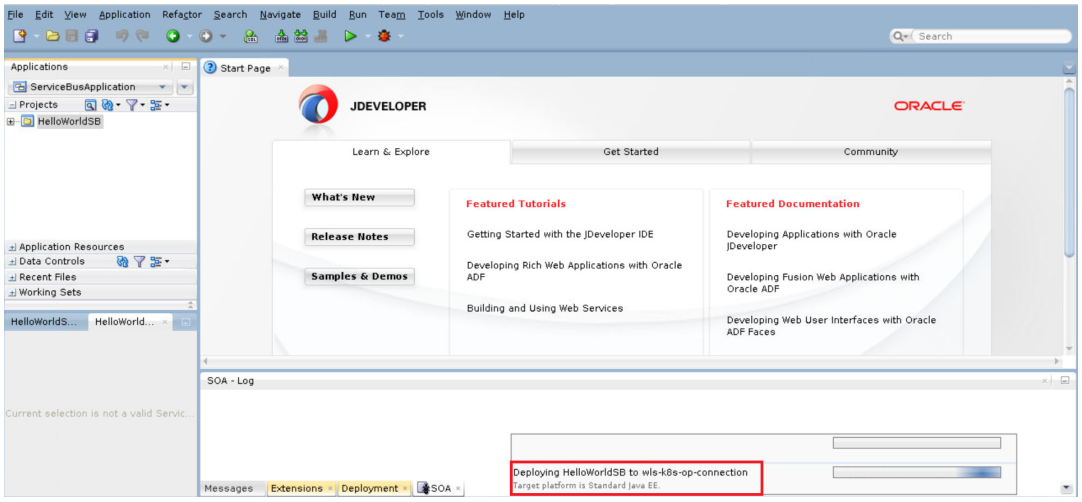
3. In the Select Server page, select the application server connection (`wls-k8s-op-connection`) that was created earlier and click **Next**.



4. On the Summary page, click **Finish** to start deploying the composites to the Oracle Service Bus cluster.

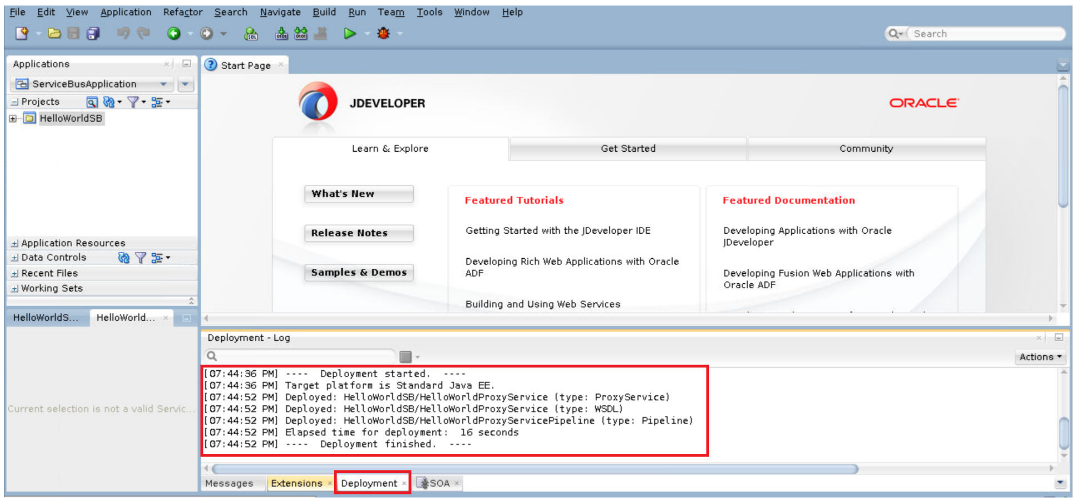


Deployment Summary:
```
⊟ Service Bus Deployment Summary
    Server Name: wls-k8s-op-connection
    Server Platform: Weblogic
    Deploy to Cloud: false
  ⊞ Service Bus Application Deployment Settings
```


JDEVELOPER
ORACLE
Learn & Explore
Get Started
Community
What's New
Release Notes
Samples & Demos
Featured Tutorials
Getting Started with the JDeveloper IDE
Developing Rich Web Applications with Oracle ADF
Building and Using Web Services
Featured Documentation
Developing Applications with Oracle JDeveloper
Developing Fusion Web Applications with Oracle ADF
Developing Web User Interfaces with Oracle ADF Faces
SOA - Log
Messages
Extensions
Deployment
SOA

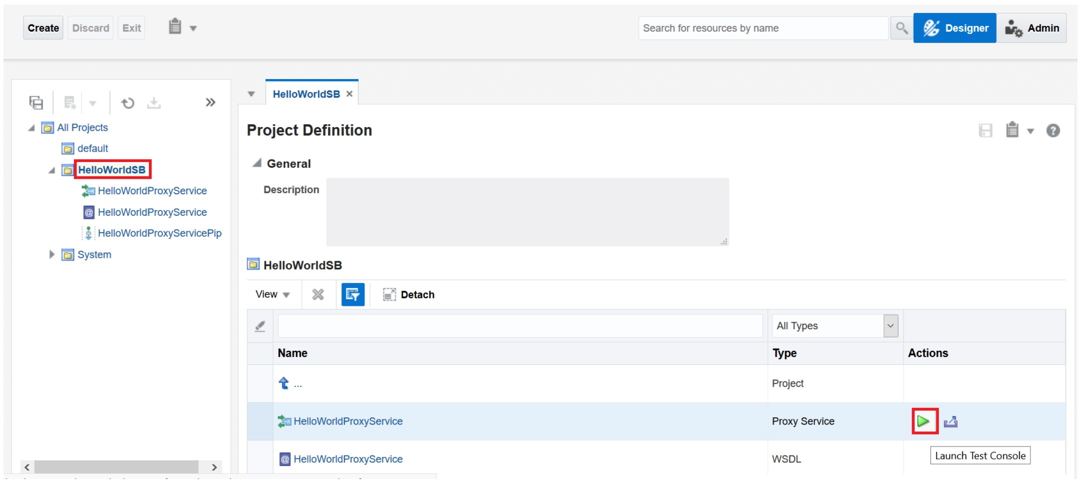
```
Deploying HelloWorldSB to wls-k8s-op-connection
Target platform is Standard Java EE.
```

5. In JDeveloper, verify logs to confirm successful deployment.



```
[07:44:36 PM] ----  Deployment started.  ----
[07:44:36 PM] Target platform is Standard Java EE.
[07:44:52 PM] Deployed: HelloWorldSB/HelloWorldProxyService (type: ProxyService)
[07:44:52 PM] Deployed: HelloWorldSB/HelloWorldProxyService (type: WSDL)
[07:44:52 PM] Deployed: HelloWorldSB/HelloWorldProxyServicePipeline (type: Pipeline)
[07:44:52 PM] Elapsed time for deployment:  16 seconds
[07:44:52 PM] ----  Deployment finished.  ----
```

6. In the Oracle Service Bus Console, click **Launch Test Console** to verify that the Oracle Service Bus composite application is deployed successfully.



Deploy Oracle SOA Suite and Oracle Service Bus applications using JDeveloper to Oracle SOA Suite on Kubernetes

This section provides details on how to deploy Oracle SOA Suite and Oracle Service Bus composite applications from Oracle JDeveloper to Oracle SOA Suite on Kubernetes environment.

perform the following instructions to deply using JDeveloper to Oracle SOA Suite on Kubernetes environment:

1. Create an Oracle SOA Quickstart container image.

   a. Download the Oracle SOA Quickstart binaries `fmw_14.1.2.0.0_soa_quickstart_generic.jar` and `fmw_14.1.2.0.0_soa_quickstart_generic2.jar` and copy both binaries into the folder `$WORKDIR/create-soa-domain/jdev-support/containerfiles`.
   By default, the scripts use docker to build the Oracle SOA Quickstart image. To use `podman`, set the below parameters before proceeding to the next step:

   ```
   $ export BUILD_CLI=podman$ export BUILD_OPTS="--format docker"
   ```

   b. Next you need an Oracle JDK17 image `container-registry.oracle.com/java/jdk:17` that can be pulled from the Oracle Container Registry. Navigate to Oracle Container Registry, sign in using your Oracle Account, and accept the Oracle Standard Terms. If you do not already have an Oracle Account, create a new account. After you log in, on your build host, use the docker or podman login command:

   ```
   $ podman login container-registry.oracle.com
   Username: <Oracle Account Username>
   Password: <auth token>
   Login successful.
   ```

c. Once you have successfully logged in, pull the JDK image by running the following command:

```
$ podman pull container-registry.oracle.com/java/jdk:17
```

d. Create the Oracle SOA Quickstart image for JDeveloper by running the following command:

```
$ sh build.sh -t 14.1.2.0.0
```

This step creates the Oracle SOA Quickstart image `oracle/soajeveloper:14.1.2.0.0` along with the required packages for VNC server access.

2. Push the Oracle SOA Quickstart image `oracle/soajeveloper:14.1.2.0.0` to a container registry so that your Kubernetes Cluster hosting the domain can pull.

```
$ podman tag oracle/soajeveloper:14.1.2.0.0 your-registry.com/oracle/
soajeveloper:14.1.2.0.0
$ podman push your-registry/oracle/soajeveloper:14.1.2.0.0
```

3. Place the SOA/Oracle Service Bus composite project at a share location (for example at /share/apps). Make sure to provide oracle user (uid: 1000 and gid: 0) permission to directory /share/apps, so that it is accessible and writable inside the container.

```
$ sudo chown -R 1000:0 /share/apps
```

4. Create a PersistentVolume and PersistentVolumeClaim (`soadeploy-pv.yaml` and `soadeploy-pvc.yaml`) with your SOA/Oracle Service Bus composite project placed at /share/apps.

a. Create a PersistentVolume with the sample provided (soadeploy-pv.yaml), which uses NFS (you can use hostPath or any other supported PV type):

```
apiVersion: v1
kind: PersistentVolume
metadata:
name: soadeploy-pv
spec:
storageClassName: soadeploy-storage-class
capacity:
   storage: 10Gi
accessModes:
   - ReadWriteMany
# Valid values are Retain, Delete or Recycle
persistentVolumeReclaimPolicy: Retain
# hostPath:
nfs:
   server: X.X.X.X
   path: "/share/apps"
```

b. Create a PersistentVolumeClaim `soadeploy-pvc.yaml`:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: soadeploy-pvc
namespace: soans
spec:
storageClassName: soadeploy-storage-class
accessModes:
   - ReadWriteMany
resources:
   requests:
      storage: 10Gi


$ kubectl apply -f soadeploy-pvc.yaml
```

5. Create an image pull secret if a credential is required to pull the image your-registry.com/oracle/soajeveloper:14.1.2.0.0 by the Kubernetes pod.

```
$ kubectl create secret docker-registry image-secret -n <domain-namespace>
--docker-server=your-registry.com --docker-username=xxxxxx --docker-
password=xxxxxxx  --docker-email=my@company.com
```

6. You need to create the JDeveloper environment in the same namespace as the domain namespace using the script `jdev_helper.sh`.

```
$ cd jdev_support/scripts
$ ./jdev_helper.sh -h

   This is a helper script for SOA jdeveloper access setup in container
environment.

   Please see README.md for more details.

   Usage:

      jdev_helper.sh [-c persistentVolumeClaimName] [-m mountPath]  [-n
namespace] [-i image] [-u imagePullPolicy] [-t serviceType] [-d
vncpassword] [-k killvnc] [-h]"

      [-c | --claimName]                : Persistent volume claim name.

      [-m | --mountPath]                : Mount path of the persistent
volume in jdevhelper deployment.

      [-n | --namespace]                : Domain namespace. Default is
'default'.

      [-i | --image]                    : Container image for the
jdevhelper deployment (optional). Default is 'ghcr.io/oracle/
oraclelinux:8'.

      [-u | --imagePullPolicy]          : Image pull policy for the helper
deployment (optional). Default is 'IfNotPresent'.

      [-p | --imagePullSecret]          : Image pull secret for the helper
deployment (optional). Default is 'None'.

      [-t | --serviceType]              : Kubernetes service type for VNC
```

```
port. Default is 'NodePort'. Supported values are NodePort and
LoadBalancer.

      [-d | --vncpassword]             : Password for VNC access. Default
is 'vncpassword'.

      [-k | --killvnc                  : Removes the Kubernetes resources
created in the namespace created for SOA JDeveloper access through VNC.

      [-h | --help]                    : This help.
```

Sample command with the PersistentVolumeClaim, image pull secret and Oracle SOA Quickstart image created above is:

```
./jdev_helper.sh -n soans -i your-registry.com/oracle/
soajeveloper:14.1.2.0.0  -t NodePort -d welcome -p image-secret -c
soadeploy-pvc
```

Output:

```
[2024-07-12T10:05:17.250564395Z][INFO] Creating deployment 'jdevhelper'
using image 'your-registry.com/oracle/soajeveloper:14.1.2.0.0', persistent
volume claim 'soadeploy-pvc' and mount path '/shared'.
deployment.apps "jdevhelper" deleted
deployment.apps/jdevhelper created
service/jdevhelper unchanged
[2024-07-12T10:05:30.917075513Z][INFO]
=========================================== VNC environment details
====================================================
[2024-07-12T10:05:30.920164769Z][INFO] VNCSERVER started on DISPLAY= <NODE
PORT>
[2024-07-12T10:05:30.921882024Z][INFO] To start using Oracle JDeveloper
==> connect via VNC viewer with <NODE NAME>:<NODE PORT>
[2024-07-12T10:05:30.924094226Z][INFO]
[2024-07-12T10:05:30.928925615Z][INFO] Your projects/applications hosted
at persistentvolumeClaim soadeploy-pvc, are available for JDeveloper
access at /shared
[2024-07-12T10:05:30.930312223Z][INFO]
===========================================================================
=============================================
[2024-07-12T10:05:30.931901926Z][INFO] Navigate to the following location
from VNCViewer on terminal
[2024-07-12T10:05:30.933337276Z][INFO] $ cd /u01/oracle/jdeveloper/jdev/bin
[2024-07-12T10:05:30.934801172Z][INFO]
[2024-07-12T10:05:30.936312907Z][INFO] For example, to connect to secure
Oracle SOA Domain with DemoTrust, run the following command:
[2024-07-12T10:05:30.939333282Z][INFO] $ ./jdev -J-
Dweblogic.security.SSL.ignoreHostnameVerify=true -J-
Dweblogic.security.TrustKeyStore=DemoTrust
[2024-07-12T10:05:30.941008547Z][INFO]
[2024-07-12T10:05:30.949161692Z][INFO] While creating Application Server
Connection, use Administration server pod name and internal configured
ports.
[2024-07-12T10:05:30.9514636857Z][INFO] For example 'WebLogic Hostname'
value is 'soainfra-adminserver'
```

```
[2024-07-12T10:05:30.956109261Z][INFO] 'SSL Port' is '9002' for secure
domain or 'Port' is '7001' for non-secure domain
[2024-07-12T10:05:30.957739730Z][INFO]
============================================================================
=============================================
[2024-07-12T10:05:30.959372215Z][INFO]
[2024-07-12T10:05:30.960767620Z][INFO]
[2024-07-12T10:05:30.962397769Z][INFO]
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<<<<<<
[2024-07-12T10:05:30.963984357Z][INFO] >>>>>> To cleanup the Kubernetes
resources created for Oracle JDeveloper access through VNC
[2024-07-12T10:05:30.965782926Z][INFO] >>>>>> Run: $ ./jdev_helper.sh -k -
n soans
[2024-07-12T10:05:30.967358203Z][INFO]
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<<<<<<
$
```

Sample command to create a JDeveloper environment with VNC access using password as `welcome` without a presistenceVolumeClaimName:

```
$ ./jdev_helper.sh -n soans -i your-registry.com/oracle/
soajdeveloper:14.1.2.0.0 -t NodePort -d welcome -p image-secret
```

Sample output:

```
 [2024-07-08T17:59:38.277941559Z][INFO] CAUTION!!!! The
presistenceVolumeClaimName is not provided. Projects/applications stored
in this setup is ephemeral and not persistent
 [2024-07-08T17:59:38.279462462Z][INFO] CAUTION!!!! In case you want the
projects/applications created to be persistent, recommended to pass a
persistentVolumeClaimName with option -c
 [2024-07-08T17:59:38.281126958Z][INFO] Creating deployment 'jdevhelper'
using image 'your-registry.com/oracle/soajdeveloper:14.1.2.0.0',
persistent volume claim '' and mount path '/shared'.
 deployment.apps "jdevhelper" deleted
 deployment.apps/jdevhelper created
 service/jdevhelper unchanged
 [2024-07-08T17:59:51.926915480Z][INFO]
=========================================== VNC environment details
====================================================
 [2024-07-08T17:59:51.928832925Z][INFO] VNCSERVER started on DISPLA=
<NODE PORT>
 [2024-07-08T17:59:51.930804513Z][INFO] To start using Oracle JDeveloper
==> connect via VNC viewer with <NODE NAME>:<NODE PORT>
 [2024-07-08T17:59:51.932756905Z][INFO]
 [2024-07-08T17:59:51.935113101Z][INFO] Your projects/applications created
are ephemeral and not persistent as no persistentvolumeClaim is used
 [2024-07-08T17:59:51.937168808Z][INFO]
============================================================================
=============================================
 [2024-07-08T17:59:51.940554959Z][INFO] Navigate to the following location
from VNCViewer on terminal
 [2024-07-08T17:59:51.942509904Z][INFO] $ cd /u01/oracle/jdeveloper/
```

```
jdev/bin
 [2024-07-08T17:59:51.944227450Z][INFO]
 [2024-07-08T17:59:51.945968141Z][INFO] For example, to connect to secure
Oracle SOA Domain with DemoTrust, run the following command:
 [2024-07-08T17:59:51.948112907Z][INFO] $ ./jdev -J-
Dweblogic.security.SSL.ignoreHostnameVerify=true -J-
Dweblogic.security.TrustKeyStore=DemoTrust
 [2024-07-08T17:59:51.950122988Z][INFO]
 [2024-07-08T17:59:51.952409522Z][INFO] While creating Application Server
Connection, use Administration server pod name and internal configured
ports.
 [2024-07-08T17:59:51.954187243Z][INFO] For example 'WebLogic Hostname'
value is 'soainfra-adminserver'
 [2024-07-08T17:59:51.955624588Z][INFO] 'SSL Port' is '9002' for secure
domain. 'Port' is '7001' for non-secure domain
 [2024-07-08T17:59:51.956996289Z][INFO]
===========================================================================
=============================================
 [2024-07-08T17:59:51.958355926Z][INFO]
 [2024-07-08T17:59:51.959729069Z][INFO]
 [2024-07-08T17:59:51.961092794Z][INFO]
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<
<<<<<<<<<<<<<<<<<<<<<<<<<<<<
 [2024-07-08T17:59:51.962525360Z][INFO] >>>>>> To cleanup the Kubernetes
resources created for Oracle JDeveloper access through VNC
 [2024-07-08T17:59:51.963854601Z][INFO] >>>>>> Run: $ ./jdev_helper.sh -k -
n soans
 [2024-07-08T17:59:51.965774791Z][INFO]
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<
<<<<<<<<<<<<<<<<<<<<<<<<<<<<
 $
```

7. To start using Oracle JDeveloper, connect via VNC viewer with `<NODE NAME>:<NODE PORT>` obtained from the `jdev_helper.sh` output.
8. Navigate to the following location from VNCViewer:

   ```
   $ cd /u01/oracle/jdeveloper/jdev/bin
   ```

9. Connect to secure Oracle SOA Domain for example with DemoTrust by running the following command:

   ```
   $ ./jdev -J-Dweblogic.security.SSL.ignoreHostnameVerify=true -J-
   Dweblogic.security.TrustKeyStore=DemoTrust
   ```

10. While creating Application Server Connection, use Administration server pod name and internal configured ports. For example `WebLogic Hostname` value is `soainfra-adminserver`. For a secure domain SSL Port is `9002` and for a non-secure domain Port is `7001`.
11. To cleanup the Kubernetes resources created for Oracle JDeveloper access through VNC, run the following command:

    ```
    $ ./jdev_helper.sh -k -n <domain namespace>
    ```

Deploy Using Maven and ANT

Learn how to deploy Oracle SOA Suite and Oracle Service Bus composite applications using the Maven and Ant based approach in an Oracle SOA Suite in WebLogic Kubernetes Operator environment.

Before deploying composite applications, we need to create a Kubernetes pod in the same cluster where the Oracle SOA Suite domain is running, so that composite applications can be deployed using the internal Kubernetes Service for the Administration Server URL.

Place the SOA/Oracle Service Bus composite project at a share location (for example at `/share/soa-deploy`) mounted at `/composites` inside container. Make sure to provide `oracle` user *(uid: 1000 and gid: 0)* permission to directory `/share/soa-deploy`, so that it is accessible and writable inside the container.

```
sudo chown -R 1000:0 /share/soa-deploy
```

Follow the steps in this section to create a container and then use it to deploy Oracle SOA Suite and Oracle Service Bus composite applications using Maven or Ant.

- Create a composite deployment container
- Maven based build and deploy
- Ant based build and deploy

Create a Composite Deployment Container

Before creating a Kubernetes pod, make sure that the Oracle SOA Suite Docker image is available on a node, or you can create an image pull secret so that the pod can pull the Docker image on the host where it gets created.

1. Create an image pull secret to pull image soasuite:release-version by the Kubernetes pod:

   ```
   kubectl create secret docker-registry image-secret -n soans --docker-
   server=your-registry.com --docker-username=xxxxxx --docker-
   password=xxxxxxx  --docker-email=my@company.com
   ```

2. Create a PersistentVolume and PersistentVolumeClaim (`soadeploy-pv.yaml` and `soadeploy-pvc.yaml`) with sample composites for build and deploy placed at `/share/soa-deploy`.

   a. Create a PersistentVolume with the sample provided (`soadeploy-pv.yaml`), which uses NFS (you can use `hostPath` or any other supported PV type):

      ```
      apiVersion: v1
      kind: PersistentVolume
      metadata:
        name: soadeploy-pv
      spec:
        storageClassName: soadeploy-storage-class
        capacity:
         storage: 10Gi
        accessModes:
          - ReadWriteMany
      ```

```
  # Valid values are Retain, Delete or Recycle
  persistentVolumeReclaimPolicy: Retain
  # hostPath:
  nfs:
    server: X.X.X.X
    path: "/share/soa-deploy"
```

b. Apply the YAML:

```
kubectl apply -f soadeploy-pv.yaml
```

c. Create a PersistentVolumeClaim (soadeploy-pvc.yaml):

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: soadeploy-pvc
  namespace: soans
spec:
  storageClassName: soadeploy-storage-class
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
```

d. Apply the YAML:

```
kubectl apply -f soadeploy-pvc.yaml
```

3. Create a composite deploy pod using `soadeploy.yaml` to mount the composites inside pod at `/composites`:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: soadeploy
  name: soadeploy
  namespace: soans
spec:
  imagePullSecrets:
  - name: image-secret
  containers:
  - image: soasuite:release-version
    name: soadeploy
    env:
    - name: M2_HOME
      value: /u01/oracle/oracle_common/modules/thirdparty/apache-
maven_bundle/3.9.4.0.0/apache-maven-3.9.4
    command: ["/bin/bash", "-c", "echo 'export PATH=$PATH:$M2_HOME/bin'
>> $HOME/.bashrc; sleep infinity"]
    imagePullPolicy: IfNotPresent
    volumeMounts:
    - name: mycomposite
```

```
        mountPath: /composites
    volumes:
    - name: mycomposite
      persistentVolumeClaim:
         claimName: soadeploy-pvc
```

4. Create the pod:

   ```
   kubectl apply -f soadeploy.yaml
   ```

5. Once the Kubernetes pod is deployed, exec into the pod to perform Maven/Ant based build and deploy:

   ```
   kubectl exec -it -n soans soadeploy -- bash
   ```

Maven Based Build and Deploy

Set up proxy details for Maven to pull dependencies from the internet.

Note:

Make sure to execute these commands inside the `soadeploy` pod.

If your environment is not running behind a proxy, then skip this step. Otherwise, replace `REPLACE-WITH-PROXY-HOST`, `REPLACE-WITH-PROXY-PORT` and the value for `nonProxyHosts` attribute per your environment and create the `settings.xml`:

```
mkdir $HOME/.m2
cat <<EOF > $HOME/.m2/settings.xml
<settings>
<proxies>
<proxy>
<active>true</active>
<protocol>http</protocol>
<host>REPLACE-WITH-PROXY-HOST</host>
<port>REPLACE-WITH-PROXY-PORT</port>
<nonProxyHosts>soainfra-cluster-soa-cluster|soainfra-adminserver</
nonProxyHosts>
</proxy>
</proxies>
</settings>
EOF
```

1. For Oracle SOA Suite composite applications
   a. Set up the environment for Maven:

      ```
      #Perform Maven Sync
      cd /u01/oracle/oracle_common/plugins/maven/com/oracle/maven/oracle-
      maven-sync/14.1.2/
      mvn install:install-file \
          -DpomFile=oracle-maven-sync-14.1.2.pom \
      ```

```
    -Dfile=oracle-maven-sync-14.1.2.jar

#install Maven plugin
mvn help:describe \
    -Dplugin=com.oracle.maven:oracle-maven-sync \
    -Ddetail

#push libraries into internal repository
mvn com.oracle.maven:oracle-maven-sync:push \
    -DoracleHome=/u01/oracle \
    -DtestingOnly=false

mvn archetype:crawl \
    -Dcatalog=$HOME/.m2/archetype-catalog.xml \
    -DarchetypeArtifactId=oracle-soa-application \
    -DarchetypeVersion=14.1.2.0-0
```

   b. Build the SOA Archive (SAR) for your sample deployment available at `/composites/mavenproject/my-soa-app`:

```
cd /composites/mavenproject/my-soa-app
mvn package
```

   The SAR will be generated at `/composites/mavenproject/my-soa-app/my-project/target/sca_my-project.jar`.

   c. Deploy into the Oracle SOA Suite instance. For example, if the instance URL is http://soainfra-cluster-soa-cluster:7003 with credentials username: weblogic and password: REPLACE-WITH-WEBLOGIC-PASSWORD, enter the following commands:

```
cd /composites/mavenproject/my-soa-app
mvn pre-integration-test \
    -DoracleServerUrl=http://soainfra-cluster-soa-cluster:7003 \
    -DsarLocation=/composites/mavenproject/my-soa-app/my-project/target/
sca_my-project.jar \
    -Doverwrite=true \
    -DforceDefault=true \
    -Dcomposite.partition=default \
    -Duser=weblogic  -Dpassword=Welcome1
```

2. For Oracle Service Bus composite applications

   a. Set up the environment for Maven:

```
#Perform Maven Sync
cd /u01/oracle/oracle_common/plugins/maven/com/oracle/maven/oracle-
maven-sync/14.1.2/
mvn install:install-file \
    -DpomFile=oracle-maven-sync-14.1.2.pom \
    -Dfile=oracle-maven-sync-14.1.2.jar

#push libraries into internal repository
mvn com.oracle.maven:oracle-maven-sync:push \
    -DoracleHome=$ORACLE_HOME
mvn archetype:crawl \
    -Dcatalog=$HOME/.m2/archetype-catalog.xml
```

```
#Verify the mvn setup
mvn help:describe \
    -DgroupId=com.oracle.servicebus.plugin \
    -DartifactId=oracle-servicebus-plugin \
    -Dversion=14.1.2.0-0
```

b. Build the Oracle Service Bus Archive (`sbconfig.sbar`).

   Build `sbconfig.sbar` for your sample deployment, available at `/composites/mavenproject/HelloWorldSB`:

```
cd /composites/mavenproject/HelloWorldSB
mvn com.oracle.servicebus.plugin:oracle-servicebus-plugin:package
```

   The Oracle Service Bus Archive (SBAR) will be generated at `/composites/mavenproject/HelloWorldSB/.data/maven/sbconfig.sbar`.

c. Deploy the generated `sbconfig.sbar` into the Oracle Service Bus instance. For example, if the Administration URL is `http://soainfra-adminserver:7001` with credentials username: `weblogic` and password: `Welcome1`, enter the following commands:

```
cd /composites/mavenproject/HelloWorldSB
mvn pre-integration-test   \
    -DoracleServerUrl=t3://soainfra-adminserver:7001 \
    -DoracleUsername=weblogic -DoraclePassword=Welcome1
```

Ant based build and deploy

Make sure to execute these commands inside the `soadeploy` pod.

1. For Oracle SOA Suite composite applications:

   a. Build an Oracle SOA Suite composite application using Ant. For example, if the composite application to be deployed is available at `/composites/antproject/Project`, enter the following commands:

```
cd /u01/oracle/soa/bin
ant -f ant-sca-package.xml \
      -DcompositeDir=/composites/antproject/Project \
      -DcompositeName=Project \
      -Drevision=0.1
```

   The SOA Archive is generated at `/composites/antproject/Project/deploy/sca_Project_rev0.1.jar`, which will be used for deploying.

   b. Deploy into the Oracle SOA Suite instance using Ant:

```
cd /u01/oracle/soa/bin
ant -f ant-sca-deploy.xml \
      -DserverURL=http://soainfra-cluster-soa-cluster:7003  \
      -DsarLocation=/composites/antproject/Project/deploy/
sca_Project_rev0.1.jar \
      -Doverwrite=true \
      -Duser=weblogic -Dpassword=REPLACE-WITH-WEBLOGIC-PASSWORD
```

2. For Oracle Service Bus composite applications:

   See Developing Services Using Oracle Service Bus to deploy Oracle Service Bus composite applications using Ant.

Deploy Using Composites in a Persistent Volume or Image

Learn how to deploy Oracle SOA Suite and Oracle Service Bus composite applications artifacts in a Kubernetes persistent volume or in an image to an Oracle SOA Suite environment deployed using a WebLogic Kubernetes Operator.

The deployment methods described in Deploy using JDeveloper and Deploy using Maven and Ant are manual processes. If you have the deployment artifacts (archives) already built, then you can package them either into a Kubernetes persistent volume or in an image and use this automated process to deploy the artifacts to an Oracle SOA Suite domain.

Prepare to use the deploy artifacts script

The sample scripts for deploying artifacts are available at `${WORKDIR}/create-soa-domain/domain-home-on-pv/` You must edit `deploy-artifacts-inputs.yaml` (or a copy of it) to provide the details of your domain and artifacts. Refer to the configuration parameters below to understand the information that you must provide in this file.

Configuration parameters

The following parameters can be provided in the inputs file.

Parameter	Definition	Default value
adminPort	Port number of the Administration Server inside the Kubernetes cluster.	7001
adminServerName	Name of the Administration Server.	AdminServer
domainUID	Unique ID that is used to identify the domain. This ID cannot contain any characters that are not valid in a Kubernetes service name.	soainfra
domainType	Type of the domain. Mandatory input for Oracle SOA Suite domains. You must provide one of the supported domain type values: soa (deploys artifacts into an Oracle SOA Suite domain), osb (deploys artifacts into an Oracle Service Bus domain), or soaosb (deploys artifacts into both Oracle SOA Suite and Oracle Service Bus domains).	soa

Parameter	Definition	Default value
soaClusterName	Name of the SOA WebLogic Server cluster instance in the domain. By default, the cluster name is soa_cluster. This configuration parameter is applicable only for soa and soaosb domain types.	soa_cluster
image	SOA Suite Docker image. The artifacts deployment process requires Oracle SOA Suite 14.1.2.0. Refer to Obtain the Oracle SOA Suite Docker image for details on how to obtain or create the image.	soasuite:release-version
imagePullPolicy	Oracle SOA Suite Docker image pull policy. Valid values are IfNotPresent, Always, Never.	IfNotPresent
imagePullSecretName	Name of the Kubernetes secret to access the Docker Store to pull the Oracle SOA Suite Docker image. The presence of the secret will be validated when this parameter is specified.	
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.	soainfra-domain-credentials
namespace	Kubernetes namespace in which the domain was created.	soans
artifactsSourceType	The deploy artifacts source type. Set to PersistentVolume for deploy artifacts available in a persistent volume and Image for deploy artifacts available as an image.	Image
persistentVolumeClaimName	Name of the persistent volume claim created that hosts the deployment artifacts. If not specified, the value is derived from the domainUID as <domainUID>-deploy-artifacts-pvc.	soainfra-deploy-artifacts-pvc
artifactsImage	Deploy artifacts image. Required if artifactsSourceType is Image.	artifacts:release-version
artifactsImagePullPolicy	Deploy artifacts image pull policy. Valid values are IfNotPresent, Always, Never.	IfNotPresent
artifactsImagePullSecretName	Name of the Kubernetes secret to access the deploy artifacts image. The presence of the secret will be validated when this parameter is specified.	

Parameter	Definition	Default value
deployScriptFilesDir	Directory on the host machine to locate the required files to deploy artifacts to the Oracle SOA Suite domain, including the script that is specified in the deployScriptName parameter. By default, this directory is set to the relative path deploy.	deploy
deployScriptsMountPath	Mount path where the deploy artifacts scripts are located inside a pod. The deploy-artifacts.sh script creates a Kubernetes job to run the script (specified by the deployScriptName parameter) in a Kubernetes pod to deploy the artifacts. Files in the deployScriptFilesDir directory are mounted to this location in the pod, so that the Kubernetes pod can use the scripts and supporting files to deploy artifacts.	/u01/weblogic
deployScriptName	Script that the deploy artifacts script uses to deploy artifacts to the Oracle SOA Suite domain. For Oracle SOA Suite, the script placed in the soa directory is used. For Oracle Service Bus, the script placed in the osb directory is used. The deploy-artifacts.sh script creates a Kubernetes job to run this script to deploy artifacts. The script is located in the in-pod directory that is specified by the deployScriptsMountPath parameter.	deploy.sh
soaArtifactsArchivePath	Directory inside container where Oracle SOA Suite archives are placed.	/u01/sarchives
osbArtifactsArchivePath	Directory inside container where Oracle Service Bus archives are placed.	/u01/sbarchives

The sample demonstrates how to deploy Oracle SOA Suite composites or Oracle Service Bus applications to an Oracle SOA Suite domain home.

Run the deploy artifacts script

Run the deploy artifacts script, specifying your inputs file and an output directory to store the generated artifacts:

```
./deploy-artifacts.sh \
  -i deploy-artifacts-inputs.yaml \
  -o <path to output-directory>
```

The script performs the following steps:

- Creates a directory for the generated Kubernetes YAML files for the artifacts deployment process if it does not already exist. The path name is `<path to output-directory>/weblogic-domains/<domainUID>/<YYYYMMDD-hhmmss>`. If the directory already exists, its contents must be removed before running this script.
- Creates a Kubernetes job that starts a utility Oracle SOA Suite container and run scripts to deploy artifacts provided either in an image or in a persistent volume.

Deploy artifacts from an image

1. Create an image with artifacts.
   a. A sample Dockerfile to create the artifacts in an image is available at $WORKDIR/create-soa-domain/domain-home-on-pv/deploy-docker-file. This expects the Oracle SOA Suite related archives to be available in the soa directory and Oracle Service Bus archives to be available in the osb directory.
   b. Create the `soa` directory and copy the Oracle SOA Suite archives to be deployed to the directory:

      ```
      cd $WORKDIR/create-soa-domain/domain-home-on-pv/deploy-docker-file
      mkdir soa
      cp /path/sca_sampleBPEL.jar soa
      ```

   c. Create the `osb` directory and copy the Oracle Service Bus archives to be deployed to the directory:

      ```
      cd $WORKDIR/create-soa-domain/domain-home-on-pv/deploy-docker-file
      mkdir osb
      cp /path/simple_sbconfig.jar osb
      ```

   d. Create the image using build.sh. This script creates the image with default tag release-version (artifacts:release-version):

      ```
      cd $WORKDIR/create-soa-domain/domain-home-on-pv/deploy-docker-file
      ./build.sh  -h
        Usage: build.sh -t [tag]
        Builds a Docker Image with Oracle SOA/OSB artifacts
        Parameters:
           -h: view usage
           -t: tag for image, default is release-version
      ```

      Sample output of script with tag 14.1.2.0-v1:

      ```
        ```
        $ ./build.sh -t 14.1.2.0-v1
          Sending build context to Docker daemon 36.35kB
          Step 1/13 : FROM busybox
           ---> 16ea53ea7c65
          Step 2/13 : ARG SOA_ARTIFACTS_ARCHIVE_PATH=/u01/sarchives
           ---> Using cache
           ---> 411edf07f267
          Step 3/13 : ARG OSB_ARTIFACTS_ARCHIVE_PATH=/u01/sbarchives
           ---> Using cache
           ---> c4214b9cf0ae
          Step 4/13 : ARG USER=oracle
           ---> Using cache
      ```
      ```

```
     ---> c8ebcd5ee546
    Step 5/13 : ARG USERID=1000
     ---> Using cache
     ---> 5780beb0c3cf
    Step 6/13 : ARG GROUP=root
     ---> Using cache
     ---> 048e67c71f92
    Step 7/13 : ENV SOA_ARTIFACTS_ARCHIVE_PATH=$
{SOA_ARTIFACTS_ARCHIVE_PATH}
     ---> Using cache
     ---> 31ae33cfd9bb
    Step 8/13 : ENV OSB_ARTIFACTS_ARCHIVE_PATH=$
{OSB_ARTIFACTS_ARCHIVE_PATH}
     ---> Using cache
     ---> 79602bf64dc0
    Step 9/13 : RUN adduser -D -u ${USERID} -G $GROUP $USER
     ---> Using cache
     ---> 07c12cea52f9
    Step 10/13 : COPY soa/ ${SOA_ARTIFACTS_ARCHIVE_PATH}/
     ---> bfeb138516d8
    Step 11/13 : COPY osb/ ${OSB_ARTIFACTS_ARCHIVE_PATH}/
     ---> 0359a11f8f76
    Step 12/13 : RUN chown -R $USER:$GROUP $
{SOA_ARTIFACTS_ARCHIVE_PATH}/ ${OSB_ARTIFACTS_ARCHIVE_PATH}/
     ---> Running in 285fb2bd8434
    Removing intermediate container 285fb2bd8434
     ---> 2e8d8c337de0
    Step 13/13 : USER $USER
     ---> Running in c9db494e46ab
    Removing intermediate container c9db494e46ab
     ---> 40295aa15317
    Successfully built 40295aa15317
    Successfully tagged artifacts:14.1.2.0-v1
    INFO: Artifacts image for Oracle SOA suite
          is ready to be extended.
          --> artifacts:14.1.2.0-v1
    INFO: Build completed in 4 seconds.
  ```
```

2. Update the image details in deploy-artifacts-inputs.yaml for parameter artifactsImage and invoke deploy-artifacts.sh to perform deployment of artifacts.
   Sample output of deployment for domainType of soaosb:

```
./deploy-artifacts.sh -i deploy-artifacts-inputs.yaml -o out-deploy
 Input parameters being used
 export version="deploy-artifacts-inputs-v1"
 export adminPort="7001"
 export adminServerName="AdminServer"
 export domainUID="soainfra"
 export domainType="soaosb"
 export soaClusterName="soa_cluster"
 export soaManagedServerPort="7003"
 export image="soasuite:14.1.2.0"
 export imagePullPolicy="IfNotPresent"
 export weblogicCredentialsSecretName="soainfra-domain-credentials"
 export namespace="soans"
```
```

```
 export artifactsSourceType="Image"
 export artifactsImage="artifacts:14.1.2.0"
 export artifactsImagePullPolicy="IfNotPresent"
 export deployScriptsMountPath="/u01/weblogic"
 export deployScriptName="deploy.sh"
 export deployScriptFilesDir="deploy"
 export soaArtifactsArchivePath="/u01/sarchives"
 export osbArtifactsArchivePath="/u01/sbarchives"

 Generating out-deploy/deploy-artifacts/soainfra/20211022-152335/deploy-
artifacts-job.yaml
 Checking to see if the secret soainfra-domain-credentials exists in
namespace soans
 configmap/soainfra-deploy-scripts-soa-job-cm created
 Checking the configmap soainfra-deploy-scripts-soa-job-cm was created
 configmap/soainfra-deploy-scripts-soa-job-cm labeled
 configmap/soainfra-deploy-scripts-osb-job-cm created
 Checking the configmap soainfra-deploy-scripts-osb-job-cm was created
 configmap/soainfra-deploy-scripts-osb-job-cm labeled
 Checking if object type job with name soainfra-deploy-artifacts-
job-20211022-152335 exists
 Deploying artifacts by creating the job out-deploy/deploy-artifacts/
soainfra/20211022-152335/deploy-artifacts-job.yaml
 job.batch/soainfra-deploy-artifacts-job-20211022-152335 created
 Waiting for the job to complete...
 status on iteration 1 of 20 for soainfra
 pod soainfra-deploy-artifacts-job-20211022-152335-r7ffj status is NotReady
 status on iteration 2 of 20 for soainfra
 pod soainfra-deploy-artifacts-job-20211022-152335-r7ffj status is
Completed
 configmap "soainfra-deploy-scripts-soa-job-cm" deleted
 configmap "soainfra-deploy-scripts-osb-job-cm" deleted
 The following files were generated:
   out-deploy/deploy-artifacts/soainfra/20211022-152335/deploy-artifacts-
inputs.yaml
   out-deploy/deploy-artifacts/soainfra/20211022-152335/deploy-artifacts-
job.yaml


 Completed

kubectl get all -n soans|grep deploy
pod/soainfra-deploy-artifacts-job-20211022-152335-r7ffj   0/2
Completed   0          15m
job.batch/soainfra-deploy-artifacts-job-20211022-152335   1/1
43s        15m
```

Note:

When you are running the script for domainType soaosb, a deployment pod is created with two containers, one for Oracle SOA Suite artifacts deployments and another for Oracle Service Bus artifacts deployments. When the deployment completes for one container while other container is still running, the pod status will move from Ready to NotReady. Once both the deployments complete successfully, the status of the pod moves to Completed.

Deploy artifacts from a persistent volume

1. Copy the artifacts for Oracle SOA Suite to the soa directory and Oracle Service Bus to the osb directory at the share location. For example, with location /share, artifacts for Oracle SOA Suite are in /share/soa and Oracle Service Bus are in /share/osb.

```
ls /share/soa
sca_sampleBPEL.jar
ls /share/osb/
simple_sbconfig.jar
```

2. Create a PersistentVolume with the sample provided (artifacts-pv.yaml):

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: soainfra-deploy-artifacts-pv
spec:
  storageClassName: deploy-storage-class
  capacity:
    storage: 10Gi
  accessModes:
    - ReadOnlyMany
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: "/share"


kubectl apply -f artifacts-pv.yaml
```

3. Create a PersistentVolumeClaim with the sample provided (artifacts-pvc.yaml):

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: soainfra-deploy-artifacts-pvc
  namespace: soans
spec:
  storageClassName: deploy-storage-class
  accessModes:
    - ReadOnlyMany
  resources:
    requests:
      storage: 10Gi


kubectl apply -f artifacts-pvc.yaml
```

4. Update the artifactsSourceType to PersistentVolume and provide the name for persistentVolumeClaimName in deploy-artifacts-inputs.yaml.
5. Invoke deploy-artifacts.sh to deploy artifacts for artifacts present in persistentVolumeClaimName.
   Ssample output of deployment for domainType of soaosb:

```
./deploy-artifacts.sh -i deploy-artifacts-inputs.yaml -o out-deploy
 Input parameters being used
 export version="deploy-artifacts-inputs-v1"
 export adminPort="7001"
 export adminServerName="AdminServer"
 export domainUID="soainfra"
 export domainType="soaosb"
 export soaClusterName="soa_cluster"
 export soaManagedServerPort="7003"
 export image="soasuite:14.1.2.0"
 export imagePullPolicy="IfNotPresent"
 export weblogicCredentialsSecretName="soainfra-domain-credentials"
 export namespace="soans"
 export artifactsSourceType="PersistentVolume"
 export persistentVolumeClaimName="soainfra-deploy-artifacts-pvc"
 export deployScriptsMountPath="/u01/weblogic"
 export deployScriptName="deploy.sh"
 export deployScriptFilesDir="deploy"
 export soaArtifactsArchivePath="/u01/sarchives"
 export osbArtifactsArchivePath="/u01/sbarchives"

 Generating out-deploy/deploy-artifacts/soainfra/20211022-164735/deploy-
artifacts-job.yaml
 Checking to see if the secret soainfra-domain-credentials exists in
namespace soans
 configmap/soainfra-deploy-scripts-soa-job-cm created
 Checking the configmap soainfra-deploy-scripts-soa-job-cm was created
 configmap/soainfra-deploy-scripts-soa-job-cm labeled
 configmap/soainfra-deploy-scripts-osb-job-cm created
 Checking the configmap soainfra-deploy-scripts-osb-job-cm was created
 configmap/soainfra-deploy-scripts-osb-job-cm labeled
 Checking if object type job with name soainfra-deploy-artifacts-
job-20211022-164735 exists
 Deploying artifacts by creating the job out-deploy/deploy-artifacts/
soainfra/20211022-164735/deploy-artifacts-job.yaml
 job.batch/soainfra-deploy-artifacts-job-20211022-164735 created
 Waiting for the job to complete...
 status on iteration 1 of 20 for soainfra
 pod soainfra-deploy-artifacts-job-20211022-164735-66fvn status is NotReady
 status on iteration 2 of 20 for soainfra
 pod soainfra-deploy-artifacts-job-20211022-164735-66fvn status is
Completed
 configmap "soainfra-deploy-scripts-soa-job-cm" deleted
 configmap "soainfra-deploy-scripts-osb-job-cm" deleted
 The following files were generated:
   out-deploy/deploy-artifacts/soainfra/20211022-164735/deploy-artifacts-
inputs.yaml
   out-deploy/deploy-artifacts/soainfra/20211022-164735/deploy-artifacts-
job.yaml


 Completed

kubectl get all -n soans |grep deploy
pod/soainfra-deploy-artifacts-job-20211022-164735-66fvn   0/2
Completed   0          3m1s
job.batch/soainfra-deploy-artifacts-job-20211022-164735   1/1
37s        3m1s
```

Note:

When you are running the script for domainType of soaosb, a deployment pod is created with two containers, one for Oracle SOA Suite artifacts deployments and one for Oracle Service Bus artifacts deployments. When the deployment completes for one container while other container is still running, the pod status moves from Ready to NotReady. Once both the deployments successfully complete, the status of the pod moves to Completed.

Verify the deployment logs

To confirm the deployment of artifacts was successful, verify the output using the `kubectl` logs command:

Note:

Replace `<YYYYMMDD-hhmmss>`, `<domainUID>` and `<namespace>` with values for your environment.

For Oracle SOA Suite artifacts:

```
kubectl logs job.batch/<domainUID>-deploy-artifacts-job-<YYYYMMDD-hhmmss> -n
<namespace>  soa-deploy-artifacts-job
```

For Oracle Service Bus artifacts:

```
kubectl logs job.batch/<domainUID>-deploy-artifacts-job-<YYYYMMDD-hhmmss> -n
<namespace>  osb-deploy-artifacts-job
```

Persist Adapter Customizations

The lifetime for any customization done in a file on a server pod is up to the lifetime of that pod. The changes are not persisted once the pod goes down or is restarted.

For example, the following configuration updates `DbAdapter.rar` to create a new connection instance and creates data source CoffeeShop on the WebLogic Remote Console for the same with `jdbc/CoffeeShopDS`.

File location: `/u01/oracle/soa/soa/connectors/DbAdapter.rar`

```
<connection-instance>
  <jndi-name>eis/DB/CoffeeShop</jndi-name>
  <connection-properties>
    <properties>
      <property>
        <name>XADataSourceName</name>
        <value>jdbc/CoffeeShopDS</value>
      </property>
      <property>
        <name>DataSourceName</name>
        <value></value>
      </property>
      <property>
        <name>PlatformClassName</name>
        <value>org.eclipse.persistence.platform.database.Oracle10Platform</
value>
      </property>
    </properties>
   </connection-properties>
</connection-instance>
```

If you need to persist the customizations for any of the adapter files under the SOA Oracle Home in the server pod, use one of the following methods.

Method 1: Customize the Adapter file using the WebLogic Remote Console

1. Log in to the WebLogic Remote Console, and update the "Outbound Connection Pools" and `Plan.xml` referring to https://oracle.github.io/weblogic-remote-console/userguide/providers/administration-server/#configure-deployment.
2. The location by default for Plan.xml will be in `${ORACLE_HOME}/soa/soa` which is not under Persistent Volume (PV). Therefore, provide the domain's PV location such as `{DOMAIN_HOME}/soainfra/servers`.
   Now the `Plan.xml` will persist under this location for each Managed Server.

Method 2: Customize the Adapter file on the Worker Node

1. Copy `ABC.rar` from the server pod to a PV path:

   ```
   kubectl cp <namespace>/<SOA Managed Server pod name>:<full path of .rar
   file>  <destination path inside PV>
   ```

   For example:

   ```
   kubectl cp soans/soainfra-soa-server1:/u01/oracle/soa/soa/connectors/
   ABC.rar ${DockerVolume}/domains/soainfra/servers/ABC.rar
   ```

   or do a normal file copy between these locations after entering (using `kubectl exec`) in to the Managed Server pod.

2. Unrar `ABC.rar`.
3. Update the new connection details in the `weblogic-ra.xml` file under `META_INF`.
4. In the WebLogic Remote Console, under **Deployments**, select **ABC.rar**, then **Update**. Select the `ABC.rar` path as the new location, which is `${DOMAIN_HOME}/user_projects/domains/soainfra/servers/ABC.rar` and click **Update**.
5. Verify that the `plan.xml` or updated `.rar` should be persisted in the PV.

Perform WLST Operations

You can use the WebLogic Scripting Tool (WLST) to manage a domain running in a Kubernetes cluster. Some of the many ways to do this are provided here.

If the Administration Server was configured to expose a T3 channel using exposeAdminT3Channel when creating the domain, refer to Use WLST.

If you do not want to expose additional ports and perform WLST administration operations using the existing Kubernetes services created by the WebLogic Server Kubernetes operator, then follow this documentation. Here we will be creating and using a helper pod in the same Kubernetes cluster as the Oracle SOA Suite domain to perform WLST operations.

Note:

To avoid any misconfigurations, Oracle recommends that you do not use the Administration Server pod directly for WLST operations.

1. Create a Kubernetes helper pod
2. Perform WLST operations
3. Sample WLST operations

Create a Kubernetes helper pod

Before creating a Kubernetes helper pod, make sure that the Oracle SOA Suite Docker image is available on the node, or you can create an image pull secret so that the pod can pull the Docker image on the host where it gets created.

1. Create an image pull secret to pull image `soasuite:14.1.2.0` by the helper pod.

   Skip this step if you are not using an image pull secret.

   ```
   kubectl create secret docker-registry <secret-name> --namespace soans \
     --docker-server=<docker-registry-name> \
     --docker-username=<docker-user> \
     --docker-password=<docker-user> \
     --docker-email=<email-id>
   ```

   For example:

   ```
   kubectl create secret docker-registry image-secret --namespace soans \
      --docker-server=your-registry.com \
      --docker-username=xxxxxx \
   ```

```
    --docker-password=xxxxxxx  \
    --docker-email=my@company.com
```

2. Create a helper pod.

   ```
   kubectl run helper \
     --image <image_name> \
     --namespace <domain_namespace> \
     --overrides='{ "apiVersion": "v1", "spec": { "imagePullSecrets":
   [{"name": "<secret-name>"}] } }' \
     -- sleep infinity
   ```

   For example:

   ```
   kubectl run helper \
     --image soasuite:14.1.2.0 \
     --namespace soans \
     --overrides='{ "apiVersion": "v1", "spec": { "imagePullSecrets":
   [{"name": "image-secret"}] } }' \
     -- sleep infinity
   ```

   Note:

   If you are not using the image pull secret, remove `--overrides='{ "apiVersion": "v1", "spec": { "imagePullSecrets": [{"name": "<secret-name>"}] } }'`.

Perform WLST operations

Once the Kubernetes helper pod is deployed, you can exec into the pod, connect to servers using `t3` or `t3s` and perform WLST operations. By default, `t3s` is not enabled for the Administration Server or Managed Servers. If you enabled SSL with `sslEnabled` when creating the domain, then you can use `t3s` to perform WLST operations.

Interactive mode

1. Start a bash shell in the helper pod:

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

   For example:

   ```
   kubectl exec -it helper -n soans -- /bin/bash
   ```

   This opens a bash shell in the running helper pod:

   ```
   [oracle@helper oracle]$
   ```

2. Invoke WLST:

   ```
   [oracle@helper oracle]$ cd $ORACLE_HOME/oracle_common/common/bin
   [oracle@helper bin]$ ./wlst.sh
   ```

   The output will look similar to the following:

   ```
   [oracle@helper bin]$ ./wlst.sh
   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>
   ```

3. Connect using `t3`:

   a. To connect to the Administration Server or Managed Servers using `t3`, you can use the Kubernetes services created by the WebLogic Server Kubernetes operator:

      ```
      wls:/offline> connect('weblogic','<password>','t3://<domainUID>-
      <WebLogic Server Name>:<Server Port>')
      ```

      For example, if the domainUID is soainfra, Administration Server name is AdminServer, and Administration Server port is 7001, then you can connect to the Administration Server using t3:

      ```
      wls:/offline> connect('weblogic','<password>','t3://soainfra-
      adminserver:7001')
      ```

      The output will look similar to the following:

      ```
      wls:/offline> connect('weblogic','<password>','t3://soainfra-
      adminserver:7001')
      Connecting to t3://soainfra-adminserver:7001 with userid weblogic ...
      Successfully connected to Admin Server "AdminServer" that belongs to
      domain "soainfra".

      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:/soainfra/serverConfig/>
      ```

   b. To connect a WebLogic Server cluster (SOA or Oracle Service Bus) using `t3`, you can use the Kubernetes services created by the WebLogic Server Kubernetes operator:

      ```
      wls:/offline> connect('weblogic','<password>','t3://<domainUID>-cluster-
      <Cluster name>:<Managed Server Port>')
      ```

For example, if the domainUID is soainfra, SOA cluster name is soa-cluster, and SOA Managed Server port is 7003, then you can connect to SOA Cluster using t3:

```
wls:/offline> connect('weblogic','<password>','t3://soainfra-cluster-
soa-cluster:7003')
```

The output will look similar to the following:

```
wls:/offline> connect('weblogic','<password>','t3://soainfra-cluster-
soa-cluster:7003')
Connecting to t3://soainfra-cluster-soa-cluster:7003 with userid
weblogic ...
Successfully connected to Managed Server "soa_server1" that belongs to
domain "soainfra".

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:/soainfra/serverConfig/>
```

4. If you enabled SSL with sslEnabled when creating the domain, then you can use t3s to perform WLST operations:
   a. Set the required environment values, sample provided with DemoTrust::

      ```
      [oracle@helper oracle] $export WLST_PROPERTIES='-
      Dweblogic.security.TrustKeyStore=DemoTrust -
      Dweblogic.security.SSL.ignoreHostnameVerification=true'
            [oracle@helper oracle]
      ```

   b. Connect to WLST and set the required environment variable before connecting using `t3s`:

      ```
      [oracle@helper oracle]$ cd
       $ORACLE_HOME/oracle_common/common/bin
      [oracle@helper bin]$ ./wlst.sh
      ```

   c. Access `t3s` for the Administration Server in secure domain.
      For example, if the domainUID is soainfra, SOA cluster name is soa-cluster, and SOA Managed Server SSL port is 7004, connect to the SOA cluster as follows:

      ```
      wls:/offline> connect('weblogic','<password>','t3s://soainfra-
      adminserver:7004')
      ```

   d. Access t3s for the SOA cluster in non-secure domain.
      For example, if the domainUID is soainfra, SOA cluster name is soa-cluster, and Administration port of SOA server is 9004, connect to the SOA cluster as follows:

      ```
      wls:/offline> connect('weblogic','<password>','t3s://soainfra-
      adminserver:9004')
      ```

Script mode

In script mode, scripts contain WLST commands in a text file with a .py file extension (for example, mywlst.py). Before invoking WLST using the script file, you must copy the .py file into the helper pod.

To copy the .py file into the helper pod using WLST operations in script mode:

1. Create a `.py` file containing all the WLST commands.
2. Copy the `.py` file into the helper pod:

   ```
   kubectl cp <filename>.py <domain namespace>/helper:<directory>
   ```

   For example:

   ```
   kubectl cp mywlst.py soans/helper:/u01/oracle
   ```

3. Run `wlst.sh` on the `.py` file by exec into the helper pod:

   ```
   kubectl exec -it helper -n <domain_namespace> -- /bin/bash
   [oracle@helper oracle]$ cd $ORACLE_HOME/oracle_common/common/bin
   [oracle@helper oracle]$ ./wlst.sh <directory>/<filename>.py
   ```

Note:

Refer to Interactive mode for details on how to connect using `t3` or `t3s`.

Sample WLST operations

For a full list of WLST operations, refer to WebLogic Server WLST Online and Offline Command Reference.

Display servers

```
kubectl exec -it helper -n soans -- /bin/bash
[oracle@helper oracle]$ cd $ORACLE_HOME/oracle_common/common/bin
[oracle@helper bin]$ ./wlst.sh

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> connect('weblogic','Welcome1','t3://soainfra-adminserver:7001')
Connecting to t3://soainfra-adminserver:7001 with userid weblogic ...
Successfully connected to Admin Server "AdminServer" that belongs to domain
"soainfra".

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:/soainfra/serverConfig/>  cd('/Servers')
wls:/soainfra/serverConfig/Servers> ls()
dr--   AdminServer
dr--   osb_server1
dr--   osb_server2
dr--   osb_server3
dr--   osb_server4
dr--   osb_server5
dr--   soa_server1
dr--   soa_server2
dr--   soa_server3
dr--   soa_server4
dr--   soa_server5

wls:/soainfra/serverConfig/Servers>
```

7 Patch and Upgrade

Patch an existing Oracle SOA Suite image or upgrade the infrastructure, such as upgrading the underlying Kubernetes cluster to a new release and upgrading the WebLogic Kubernetes Operator release.

- Patch an image

  Create a patched Oracle SOA Suite image using the WebLogic Image Tool.
- Upgrade an Operator Release

  Upgrade the WebLogic Kubernetes Operator release to a newer version.
- Upgrade a Kubernetes Cluster

  Upgrade the underlying Kubernetes cluster version in a running SOA Kubernetes environment.

Patch an Image

Oracle releases Oracle SOA Suite images regularly with the latest bundle and recommended interim patches in My Oracle Support (MOS). However, if you need to create images with new bundle and interim patches, you can build these images using the WebLogic Image Tool.

If you have access to the Oracle SOA Suite patches, you can patch an existing Oracle SOA Suite image with a bundle patch and interim patches. Oracle recommends that you use the WebLogic Image Tool to patch the Oracle SOA Suite image.

Recommendations:

- Use the WebLogic Image Tool create feature for patching the Oracle SOA Suite Docker image with a bundle patch and multiple interim patches. This is the recommended approach because it optimizes the size of the image.
- Use the WebLogic Image Tool update feature for patching the Oracle SOA Suite Docker image with a single interim patch. Note that the patched image size may increase considerably due to additional image layers introduced by the patch application tool.

Apply the patched Oracle SOA Suite image

To update an Oracle SOA Suite domain with a patched image, first make sure the patched image is pulled or created and available on the nodes in your Kubernetes cluster. Once the patched image is available, you can follow these steps to update the Oracle SOA Suite domain with a patched image:

- Stop all servers
- Update user permissions of the domain PV storage
- Address post-installation requirements
- Apply the patched image

Stop all servers

> **Note:**
>
> The following steps are applicable only for non-Zero Downtime Patching. For Zero Downtime Patching, go to Address post-installation requirements.

Before applying the patch, stop all servers in the domain:

1. In the domain.yaml configuration file, update the spec.serverStartPolicy field value to Never.
2. Shut down the domain (stop all servers) by applying the updated domain.yaml file:

   ```
   kubectl apply -f domain.yaml
   ```

Address post-installation requirements

If the patches in the patched Oracle SOA Suite image have any post-installation steps, follow these steps:

- Create a Kubernetes pod with domain home access
- Perform post-installation steps

Create a Kubernetes pod with domain home access

1. Get domain home persistence volume claim details for the Oracle SOA Suite domain. For example, to list the persistent volume claim details in the namespace soans:

   ```
   kubectl get pvc -n soans
   ```

   Sample output showing the persistent volume claim is soainfra-domain-pvc:

   ```
   NAME                  STATUS   VOLUME                CAPACITY   ACCESS
   MODES   STORAGECLASS                         AGE
   soainfra-domain-pvc   Bound    soainfra-domain-pv    10Gi
   RWX            soainfra-domain-storage-class   xxd
   ```

2. Create a YAML soapostinstall.yaml using the domain home persistence volume claim. For example, using soainfra-domain-pvc per the sample output:

   > **Note:**
   >
   > Replace `soasuite:14.1.2.0-XXXXXX` with the patched image in the following sample YAML:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     labels:
        run: soapostinstall
   ```

```
  name: soapostinstall
  namespace: soans
spec:
 containers:
 - image: soasuite:14.1.2.0-XXXXXX
   name: soapostinstall
   command: ["/bin/bash", "-c", "sleep infinity"]
   imagePullPolicy: IfNotPresent
   volumeMounts:
   - name: soainfra-domain-storage-volume
     mountPath: /u01/oracle/user_projects
 volumes:
 - name: soainfra-domain-storage-volume
   persistentVolumeClaim:
    claimName: soainfra-domain-pvc
```

3. Apply the YAML to create the Kubernetes pod:

   ```
   kubectl apply -f soapostinstall.yaml
   ```

Perform post-installation steps

If you need to perform any post-installation steps on the domain home:

1. Start a bash shell in the soapostinstall pod:

   ```
   kubectl exec -it -n soans soapostinstall -- bash
   ```

   This opens a bash shell in the running soapostinstall pod:

   ```
   [oracle@soapostinstall oracle]$
   ```

2. Use the bash shell of the soapostinstall pod and perform the required steps on the domain home.
3. After successful completion of the post-installation steps, you can delete the soapostinstall pod:

   ```
   kubectl delete -f  soapostinstall.yaml
   ```

Apply the patched image

After completing the required post-installation steps, start up the domain:

1. In the domain.yaml configuration file, update the image field value with the patched image: For example:

   ```
     image: soasuite:14.1.2.0-XXXXXX
   ```

2. In case of non-Zero Downtime Patching, update the spec.serverStartPolicy field value to IfNeeded in domain.yaml.
3. Apply the updated `domain.yaml` configuration file to start up the domain.

   ```
   kubectl apply -f domain.yaml
   ```

> **Note:**
>
> In case of non-Zero downtime patching, update the spec.serverStartPolicy field value to IfNeeded in domain.yaml.

4. Verify the domain is updated with the patched image:

   ```
   kubectl describe domain <domainUID> -n <domain-namespace>|grep "Image:"
   ```

   Sample output:

   ```
   kubectl describe domain soainfra -n soans |grep "Image:"
   Image:                          soasuite:14.1.2.0-XXXXXXX
   ```

Upgrade an Operator Release

Upgrade a running WebLogic Kubernetes Operator by using the `helm upgrade` command. The following instructions are applicable for upgrading operator to version 4.2.9.

```
helm repo add weblogic-operator https://oracle.github.io/weblogic-kubernetes-
operator/charts
      --force-update
helm upgrade \
  --reuse-values \
  --set version=4.2.9 \
  --namespace opns \
  --wait \
  weblogic-kubernetes-operator \
  charts/weblogic-operator
```

For more details, see here.

Upgrade Kubernetes Cluster

These instructions describe how to upgrade a Kubernetes cluster created using kubeadm on which an Oracle SOA Suite domain is deployed. A rolling upgrade approach is used to upgrade nodes (master and worker) of the Kubernetes cluster.

> **Note:**
>
> It is expected that there will be a down time during the upgrade of the Kubernetes cluster as the nodes need to be drained as part of the upgrade process.

Prerequisites

- Review Prerequisites and ensure that your Kubernetes cluster is ready for upgrade.
- Make sure your environment meets all prerequisites. Make sure the database used for the SOA domain deployment is up and running during the upgrade process.

Upgrade the Kubernetes version

An upgrade of Kubernetes is supported from one MINOR version to the next MINOR version, or between PATCH versions of the same MINOR. For example, you can upgrade from 1.x to 1.x+1, but not from 1.x to 1.x+2. To upgrade a Kubernetes version, first all the master nodes of the Kubernetes cluster must be upgraded sequentially, followed by the sequential upgrade of each worker node.

- See here) for Kubernetes official documentation to upgrade from 1.29 to 1.30
- See here for Kubernetes official documentation to upgrade from 1.28 to 1.29
- See here for Kubernetes official documentation to upgrade from 1.27 to 1.28
- See here for Kubernetes official documentation to upgrade from 1.26 to 1.27
- See here for Kubernetes official documentation to upgrade from 1.25 to 1.26

Release Upgrade

Learn hot to upgrade to Oracle Fusion Middleware SOA Suite and Service Bus 14c (14.1.2.0.0). The section explains how to upgrade a production version of Oracle SOA Suite and Service Bus, including the Oracle Fusion Middleware component configurations in that domain.

Planning an Upgrade to 14c (14.1.2.0.0)

Before you begin an upgrade, understand the limitations of the upgrade and how the upgrade impacts your production environment.

See here for details.

Upgrading and Preparing Your Oracle Databases for 14c

It is important to understand the database requirements for upgrade. If required, prior to the upgrade you may need to upgrade your database to a supported version.

See here for details on some of the tasks associated with preparing your Oracle database for an upgrade to 14c (14.1.2.0.0).

Understand how your pre-upgrade environment will be affected by the upgrade.

When upgrading your existing SOA Suite 12c environment to SOA Suite 14c (14.1.2.0.0), you should understand how your pre-upgrade environment will be affected by the upgrade. For example, schemas and domain directory upgrades are performed "in place" which updates the existing files during the upgrade. The 14c (14.1.2.0.0) Oracle home binaries are upgraded using the new 14c container image.

The upgrade to 14c (14.1.2.0.0) includes the midtier and the schemas. You cannot perform a midtier-only or schema-only upgrade.

The following list describes how the upgrade is performed for the following Infrastructure and SOA Suite components:

- Oracle WebLogic Server, JRF and SOA Oracle Home Binaries

  You will be using new 14c SOA Suite container image which has Oracle Infrastructure (WebLogic Server and JRF) 14c (14.1.2.0.0) and the Oracle SOA Suite and Serivce Bus 14c (14.1.2.0.0) distribution binaries.
- Schemas - Upgraded In Place

The schemas are upgraded "in place" which means that the Upgrade Assistant updates and overwrites the existing 12c schemas during the upgrade process. The servers must be down during this process.

- Instances - Migrated during the schema upgrade

  The upgrade of active and closed instances happens automatically as part of the schema upgrade. You can manage the upgrade using administration scripts.

- Domain Directory Reconfiguration - Upgraded In Place

  The existing SOA domain is upgraded "in place". During the upgrade you will provide the location of the existing 12c SOA domain and this domain will be reconfigured to new SOA 14c (14.1.2.0.0) home directory.

Understanding the Starting Points for a SOA Suite 14c (14.1.2.0.0) Upgrade

Verify that the pre-upgrade environment is at a supported version before an upgrade.

You can upgrade to Oracle SOA Suite and Service Bus 14c (14.1.2.0.0) from the following production starting point:

- SOA Suite and Service Bus 12c (12.2.1.4.0)

SOA Domain Compatibility and Upgrade Restrictions

Read and understand how all of the components within your pre-upgrade domain interact with the upgraded 14c (14.1.2.0.0) components. See Understanding the Interoperability and Compatibility Restrictions Before You Upgrade for details.

Review the domain upgrade restrictions before starting the upgrade. See Understanding SOA Domain Upgrade Restrictions for details.

Oracle SOA Domain Pre-Upgrade Tasks

Before you start the upgrade process be sure to complete the required pre-upgrade tasks for your components and environment. See Oracle Fusion Middleware Pre-Upgrade Tasks for details.

In addition to the Oracle Fusion Middleware pre-upgrade requirements, you may also be required to complete additional SOA-specific upgrade tasks depending on your pre-upgrade environment.

Review the SOA-specific and Servic Bus specific pre-upgrade tasks that apply to your environment.

Upgrading Oracle SOA Suite and Oracle Service Bus

This section provides the end-to-end procedure for upgrading a 12.2.1.4 Oracle SOA Suite and Oracle Service bus production environment in Kubernetes to 14c (14.1.2.0.0).

Oracle strongly recommends to create a complete backup (backup of database and domain home in persistence volume) of your existing domain before you begin the upgrade.

Refer Upgrading SOA Suite and Upgrading Oracle Service Bus for details.

Using GUI Mode

To perform Oracle SOA Suite domain upgrade in a container environment using GUI mode, we need display terminal.Perform below steps to start a VNC session using the Oracle SOA Suite 14.1.2.0.0 container image and the existing 12.2.1.4.0 domain home in persistent volume. Then using this VNC session you can perform upgrade using GUI mode.

Using silent mode

Alternatively, to perform Oracle SOA Suite domain upgrade in a container environment in silent mode, you can run the helper upgrade script `domain-upgrade.sh` available at `${WORKDIR}/create-soa-domain/domain-upgrade` to perform Oracle SOA Suite domain upgrade from 12.2.1.4.0 to 14.1.2.0.0.

Refer Oracle SOA Suite Domain Release Upgrade.

Oracle SOA Suite Domain Release Upgrade

This section provides details on how to upgrade Oracle SOA Suite domains from release 12.2.1.4 to 14.1.2.0.

Prerequisites

Ensure the following before starting the automated domain upgrade process:

- Administration and all collocated managed servers will be shutdown before upgrade process. Make sure that the servers in the domain can be brought down.
- All affected data is backed up.
- Domain home is backed up.
- Database version is certified by Oracle for Fusion Middleware upgrade.
- Certification and system requirements have been met.
- Review here for impacted scenarios after upgrade.

Prepare to use the domain upgrade scripts

The sample scripts for automating the domain upgrade (schema upgrade and domain home upgrade) for Oracle OracleSOASuite domain are available at ${WORKDIR}/create-soa-domain/domain-upgrade.

You must edit domain-upgrade-inputs.yaml (or a copy of it) to provide the details for domain upgrade.

Refer to the configuration parameters below to understand the information that you must provide in this file

Configuration parameters

The following parameters can be provided in the inputs file.

Parameter	Definition	Default
domainHome	Home directory of the OracleSOASuite domain.	/u01/oracle/user_projects/domains/soainfra
domainPVMountPath	Mount path of the domain persistent volume.	/u01/oracle/user_projects
domainUID	WebLogic Server domain name.	soainfra
image	OracleSOASuite 14.1.2 container image.	soasuite:release-version
imagePullPolicy	OracleSOASuite container image pull policy. Valid values are IfNotPresent, Always, and Never.	IfNotPresent

Parameter	Definition	Default
imagePullSecretName	Name of the Kubernetes secret to access the container registry to pull the OracleSOASuite container image. The presence of the secret will be validated when this parameter is specified.	
namespace	Kubernetes namespace of the domain.	soans
persistentVolumeClaimName	Name of the persistent volume claim used for the domain home.	soainfra-domain-pvc
rcuSchemaPrefix	The schema prefix.	SOA1
rcuDatabaseURL	The database URL.	oracle-db.default.svc.cluster.local:1521/devpdb.k8s
rcuCredentialsSecret	The Kubernetes secret containing the database credentials.	soainfra-rcu-credentials
secureEnabled	Boolean indicating if secure to be enabled for the domain.	false

Run the domain upgrade script

Run the domain upgrade script, specifying your inputs file and an output directory to store the generated artifacts:

```
$ cd domain-upgrade
$ ./domain-upgrade.sh \
  -i domain-upgrade-inputs.yaml \
  -o <path to output-directory>
```

The script will perform the following steps:

- Stops the domain
- Creates pod using the 14.1.2 image and the persistent volume claim used for the domain home for performing the domain upgrade.
- Performs UA schema upgrade.
- Performs domain home upgrade.
- Sets the database values for new WLS_RUNTIME schema.
- Enables secure domain if flag secureEnabled is set to true.
- Updates the domain spec with the 14.1.2 image.
- Starts and waits for the domain to be up and running with 14.1.2 image.

Upgrade ingress

This step is required in case you have enabled secureEnabled to true during domain upgrade. For secure domain only sslType=E2ESSL is supported. In case the ingress controller is not installed with support for E2ESSL, for example, for NGINX, recreate the ingress controller accordingly. Then run the helm upgrade to update the ingress-per-domain with --set wlsDomain.secureEnabled=true. Sample command as below:

```
$ cd $WORKDIR
$ helm install REPLACE-WITH-INGRESS-PER-DOMAIN-RELEASE-NAME charts/ingress-
per-domain \
  --reuse-values --set wlsDomain.secureEnabled=true
```

Setting up VNC session in a container environment

To perform any GUI based operations using display terminal in container environment, deploy additional packages.

This section provides details on how to update an existing Oracle SOA Suite image with additional packages using WebLogic Image Tool and then use this image to create Kubernetes deployment for setting up VNC session in a container environment.

Before you begin

Make sure that you have set up the WebLogic Image Tool. See setup for details. WebLogic Image Tool by default uses docker CLI. To override the default with say podman, set the WLSIMG_BUILDER with full path to the executable. For example, `WLSIMG_BUILDER="/usr/bin/podman"`

Update the container image

1. Review and update the additionalBuildCmds.txt. The sample additionalBuildCmds.txt is provided for Oracle Linux 8 with the packages required for installing and setting up a VNC session.
2. Update additionalBuildCmds.txt if you want to add additional packages.
3. Run the imagetool.sh.
4. To update the existing Oracle SOA Suite container image `soasuite:14.1.2.0` by applying the additionalBuildCmds.txt and tag it as `soasuite:14.1.2.0-vnc` and run the below command:

   ```
   $ cd $WORKDIR/create-soa-domain/domain-upgrade/vncsetup
   $ imagetool update --fromImage soasuite:14.1.2.0 --tag soasuite:14.1.2.0-
   vnc --chown oracle:root --additionalBuildCommands ./
   additionalBuildCmds.txt
   ```

5. Push the image to remote or local registry
   - Remote registry - Push the updated image soasuite:14.1.2.0-vnc to a container registry so that your Kubernetes Cluster hosting can you pull.

     ```
     $ podman tag soasuite:14.1.2.0-vnc  your-registry.com/oracle/
     soasuite:14.1.2.0-vnc$ podman push your-registry.com/oracle/
     soasuite:14.1.2.0-vnc
     ```

   - Local registry - This requires access to worker nodes.
     First archive the image:

     ```
     $ podman save -o soasuite-14.1.2.0-vnc.tar.gz soasuite:14.1.2.0-vnc
     ```

Copy the archive `soasuite-14.1.2.0-vnc.tar.gz` into worker nodes and load the image to local container storage:

```
$ podman load < soasuite-14.1.2.0-vnc.tar.gz
```

Setup the VNC access

1. To create the VNC session in a container environment you can use the script start_vnc.sh:

   ```
   $ cd $WORKDIR/create-soa-domain/domain-upgrade/vncsetup/scripts
   $ ./start_vnc.sh -h

   This is a helper script for starting VNC session in a container
   environment.

   Please see README.md for more details.

   Usage:

       start_vnc.sh [-c persistentVolumeClaimName] [-m mountPath]  [-n
   namespace] [-i image] [-u imagePullPolicy] [-t serviceType] [-d
   vncpassword] [-k killvnc] [-h]"

       [-c | --claimName]                : Persistent volume claim name.

       [-m | --mountPath]                : Mount path of the persistent
   volume in vnchelper deployment.

       [-n | --namespace]                : Namespace. Default is 'default'.

       [-i | --image]                    : Container image for the vnchelper
   deployment (optional). Default is 'ghcr.io/oracle/oraclelinux:8'.

       [-u | --imagePullPolicy]          : Image pull policy for the
   vnchelper deployment (optional). Default is 'IfNotPresent'.

       [-p | --imagePullSecret]          : Image pull secret for the
   vnchelper deployment (optional). Default is 'None'.

       [-t | --serviceType]              : Kubernetes service type for VNC
   port. Default is 'NodePort'. Supported values are NodePort and
   LoadBalancer.

       [-d | --vncpassword]              : Password for VNC access. Default
   is 'vncpassword'.

       [-k | --killvnc                   : Removes the Kubernetes resources
   created in the namespace created for VNC session.

       [-h | --help]                     : This help.
   ```

2. In case you want to invoke GUI based command like Upgrade Assistant (ua) or reconfig.sh to perform release upgrade, use the start_vnc.sh script with same domain home Persistent volume claim name and mount path used by the Domain. Here is an example for Oracle SOASuite domain deployed in soans namespace, using soainfra-domain-pvc Persistent

volume claim mounted inside container at mount path /u01/oracle/user_projects, you can start the script with below command:

```
$ ./start_vnc.sh -n soans -i soasuite:14.1.2.0-vnc -c soainfra-domain-pvc -
t NodePort -d welcome -m /u01/oracle/user_projects
```

Sample output:

```
[2024-11-07T17:45:55.893964348Z][INFO] Creating deployment 'vnchelper'
using image 'soasuite:14.1.2.0-vnc', persistent volume claim 'soainfra-
domain-pvc' and mount path '/u01/oracle/user_projects'.
configmap/vnchelper-scripts-cm configured
Checking the configmap vnchelper-scripts-cm was created
secret/vnchelper-scripts-secret configured
Checking the secret vnchelper-scripts-secret was created
deployment.apps "vnchelper" deleted
deployment.apps/vnchelper created
service/vnchelper created
[2024-11-07T17:46:06.838237729Z][INFO]
=========================================== VNC environment details
====================================================
[2024-11-07T17:46:06.840832397Z][INFO] VNCSERVER started on DISPLAY= <NODE
PORT>
[2024-11-07T17:46:06.842955900Z][INFO] To start using VNC Session ==>
connect via VNC viewer with <NODE NAME>:<NODE PORT>
[2024-11-07T17:46:06.845361778Z][INFO]
[2024-11-07T17:46:06.847710989Z][INFO] Your data hosted at
persistentvolumeClaim soainfra-domain-pvc, are available for access
at /u01/oracle/user_projects
[2024-11-07T17:46:06.850974036Z][INFO]
===========================================================================
=============================================
[2024-11-07T17:46:06.853056442Z][INFO]
[2024-11-07T17:46:06.855075115Z][INFO]
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<
<<<<<<<<<<<<<<<<<<<<<<<<<<<<
[2024-11-07T17:46:06.857105300Z][INFO] >>>>>> To cleanup the Kubernetes
resources created for VNC session
[2024-11-07T17:46:06.859015919Z][INFO] >>>>>> Run: $ ./start_vnc.sh -k -n
soans
[2024-11-07T17:46:06.861045303Z][INFO]
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<
<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

3. To start using VNC session, connect via VNC viewer with <NODE NAME>:<NODE PORT> obtained from the start_vnc.sh output.
4. Finally to remove the VNC access and cleanup the Kubernetes resources created, run the following command:

```
$ ./start_vnc.sh -k -n <namespace>
```

8
Create or Update an Image

If you have access to the My Oracle Support (MOS), and there is a need to build a new image with a patch (bundle or interim), it is recommended to use the WebLogic Image Tool to build an Oracle SOA Suite image for production deployments.

- Create or update an Oracle SOA Suite Docker image using the WebLogic Image Tool
  - Set up the WebLogic Image Tool
  - Create an image
  - Update an image

Create or update an Oracle SOA Suite Docker image using the WebLogic Image Tool

Using the WebLogic Image Tool, you can create a new Oracle SOA Suite Docker image (can include patches as well) or update an existing image with one or more patches (bundle patch and interim patches).

Note:

- Use create for creating a new Oracle SOA Suite Docker image either:
  - without any patches
  - or, containing the Oracle SOA Suite binaries, bundle patch and interim patches. This is the recommended approach if you have access to the Oracle SOA Suite patches because it optimizes the size of the image.
- Use update for patching an existing Oracle SOA Suite Docker image with a single interim patch. Note that the patched image size may increase considerably due to additional image layers introduced by the patch application tool.

Set up the WebLogic Image Tool

- Prerequisites
- Set up the WebLogic Image Tool
- Validate setup
- WebLogic Image Tool build directory
- WebLogic Image Tool cache
- Set up additional build scripts

Prerequisites

Verify that your environment meets the following prerequisites:

- Docker client and daemon on the build machine, with minimum Docker version 18.03.1.ce.
- Bash version 4.0 or later, to enable the command complete feature.

- JAVA_HOME environment variable set to the appropriate JDK location.

Set up the WebLogic Image Tool

To set up the WebLogic Image Tool:

1. Create a working directory and change to it. In these steps, this directory is `imagetool-setup`.

    ```
    mkdir imagetool-setup
    cd imagetool-setup
    ```

2. Download the latest version of the WebLogic Image Tool from the releases page.
3. Unzip the release ZIP file to the imagetool-setup directory.
4. Execute the following commands to set up the WebLogic Image Tool on a Linux environment:

    ```
    cd imagetool-setup/imagetool/bin
    source setup.sh
    ```

Validate setup

To validate the setup of the WebLogic Image Tool:

1. Enter the following command to retrieve the version of the WebLogic Image Tool:

    ```
    imagetool --version
    ```

2. Enter `imagetool` then press the Tab key to display the available `imagetool` commands:

    ```
    imagetool <TAB>
    cache   create  help    rebase  update
    ```

WebLogic Image Tool build directory

The WebLogic Image Tool creates a temporary Docker context directory, prefixed by `wlsimgbuilder_temp`, every time the tool runs. Under normal circumstances, this context directory will be deleted. However, if the process is aborted or the tool is unable to remove the directory, it is safe for you to delete it manually. By default, the WebLogic Image Tool creates the Docker context directory under the user's home directory. If you prefer to use a different directory for the temporary context, set the environment variable `WLSIMG_BLDDIR`:

```
export WLSIMG_BLDDIR="/path/to/buid/dir"
```

WebLogic Image Tool cache

The WebLogic Image Tool maintains a local file cache store. This store is used to look up where the Java, WebLogic Server installers, and WebLogic Server patches reside in the local file system. By default, the cache store is located in the user's `$HOME/cache` directory. Under this directory, the lookup information is stored in the `.metadata` file. All automatically downloaded patches also reside in this directory. You can change the default cache store location by setting the environment variable `WLSIMG_CACHEDIR`:

```
export WLSIMG_CACHEDIR="/path/to/cachedir"
```

Set up additional build scripts

Creating an Oracle SOA Suite Docker image using the WebLogic Image Tool requires additional container scripts for Oracle SOA Suite domains.

1. Clone the docker-images repository to set up those scripts. In these steps, this directory is DOCKER_REPO:

```
cd imagetool-setup
git clone https://github.com/oracle/docker-images.git
```

2. Copy the additional WebLogic Image Tool build files from the operator source repository to the imagetool-setup location:

```
mkdir -p imagetool-setup/docker-images/OracleSOASuite/imagetool/14.1.2.0.0
cd imagetool-setup/docker-images/OracleSOASuite/imagetool/14.1.2.0.0
cp -rf ${WORKDIR}/imagetool-scripts/* .
```

Note:

If you want to create the image continue with the following steps, otherwise to update the image see update an image.

Create an image

After setting up the WebLogic Image Tool and required build scripts, follow these steps to use the WebLogic Image Tool to create a new Oracle SOA Suite Docker image.

Download the Oracle SOA Suite installation binaries

You must download the required Oracle SOA Suite installation binaries as listed below from the Oracle Software Delivery Cloud and save them in a directory of your choice. In these steps, this directory is download location.

The installation binaries required for release 14.1.2.0.0 are:

- JDK
  - jdk-21.0.4_linux-x64.tar.gz or jdk-17.0.12_linux-x64.tar.gz
- Fusion Middleware Infrastructure installer:
  - fmw_14.1.2.0.0_infrastructure.jar
- Oracle SOA Suite installers:
  - fmw_14.1.2.0.0_soa.jar
  - fmw_14.1.2.0.0_osb.jar
  - fmw_14.1.2.0.0_b2bhealthcare.jar

  Note:

  In this release, Oracle B2B is not supported to be configured, but the installer is required for completeness.

Update required build files

The following files in the code repository location `<imagetool-setup-location>/docker-images/OracleSOASuite/imagetool/14.1.2.0.0` are used for creating the image:

- additionalBuildCmds.txt
- buildArgs

1. In the buildArgs file, update all occurrences of %DOCKER_REPO% with the docker-images repository location, which is the complete path of <imagetool-setup-location>/docker-images.
   For example, update:

   ```
   %DOCKER_REPO%/OracleSOASuite/imagetool/14.1.2.0.0/
   ```

   to:

   ```
   <imagetool-setup-location>/docker-images/OracleSOASuite/imagetool/
   14.1.2.0.0/
   ```

2. Similarly, update the placeholders %JDK_VERSION% and %BUILDTAG% with appropriate values.
3. Update the response file <imagetool-setup-location>/docker-images/OracleFMWInfrastructure/dockerfiles/14.1.2.0.0/install.file to add the parameter INSTALL_TYPE="Fusion Middleware Infrastructure" in the [GENERIC] section.

Create the image

1. Add a JDK package to the WebLogic Image Tool cache:

   ```
   imagetool cache addInstaller --type jdk --version 21u03 --path <download
   location>/jdk-21.0.4_linux-x64.tar.gz
   ```

2. Add the downloaded installation binaries to the WebLogic Image Tool cache:

   ```
   imagetool cache addInstaller --type fmw --version 14.1.2.0.0 --path
   <download location>/fmw_14.1.2.0.0_infrastructure.jar

   imagetool cache addInstaller --type soa --version 14.1.2.0.0 --path
   <download location>/fmw_14.1.2.0.0_soa.jar

   imagetool cache addInstaller --type osb --version 14.1.2.0.0 --path
   <download location>/fmw_14.1.2.0.0_osb.jar

   imagetool cache addInstaller --type b2b --version 14.1.2.0.0 --path
   <download location>/fmw_14.1.2.0.0_b2bhealthcare.jar
   ```

3. Example buildArgs file

   ```
   create
   --jdkVersion 21u03
   --type soa_osb_b2b
   --version 14.1.2.0.0
   --tag oracle/soasuite:14.1.2.0.0
   --pull
   --fromImage ghcr.io/oracle/oraclelinux:8-slim
   ```

```
--chown oracle:root
--additionalBuildCommands <imagetool-setup-location>/docker-images/
OracleSOASuite/imagetool/14.1.2.0.0/additionalBuildCmds.txt
--additionalBuildFiles <imagetool-setup-location>/docker-images/
OracleSOASuite/dockerfiles/14.1.2.0/container-scripts
--installerResponseFile <imagetool-setup-location>/docker-images/
OracleFMWInfrastructure/dockerfiles/14.1.2.0/install.file,<imagetool-setup-
location>/docker-images/OracleSOASuite/dockerfiles/14.1.2.0/install/
soasuite.response,<imagetool-setup-location>/docker-images/OracleSOASuite/
dockerfiles/14.1.2.0/install/osb.response,<imagetool-setup-location>/
docker-images/OracleSOASuite/dockerfiles/14.1.2.0/install/b2b.response
```

Note:

In the buildArgs file:

- --jdkVersion value must match the --version value used in the imagetool cache addInstaller command for --type jdk.
- --version value must match the --version value used in the imagetool cache addInstaller command for --type soa.
- --pull always pulls the latest base Linux image

Refer to this page for the complete list of options available with the WebLogic Image Tool create command.

4. Create the Oracle SOA Suite image:

```
imagetool @<absolute path to buildargs file>
```

Note:

Make sure that the absolute path to the buildargs file is prepended with a @ character, as shown in the example above. For example:

```
imagetool @<imagetool-setup-location>/docker-images/OracleSOASuite/
imagetool/14.1.2.0.0/buildArgs
```

Sample Dockerfile generated with the `imagetool` command:

```
########## BEGIN DOCKERFILE ##########
# Copyright (c) 2019, 2021, Oracle and/or its affiliates.
# Licensed under the Universal Permissive License v 1.0 as shown at
https://oss.oracle.com/licenses/upl.
#

FROM ghcr.io/oracle/oraclelinux:8-slim as os_update
LABEL com.oracle.weblogic.imagetool.buildid="b4554a25-22dd-4793-
b121-9989bd4be40a"
USER root
```

```
# Use package manager to make sure that unzip, tar, and other required
packages are installed
#
# Copyright (c) 2021, Oracle and/or its affiliates.
#
# Licensed under the Universal Permissive License v 1.0 as shown at
https://oss.oracle.com/licenses/upl.
#
# Ensure necessary OS packages are installed
    RUN microdnf update \
    && microdnf install gzip tar unzip libaio libnsl jq findutils
diffutils hostname perl freetype fontconfig  \
    && microdnf clean all
    && rm -rf /tmp/imagetool


# Create the Oracle user that will be the owner of the installed software
#
# Copyright (c) 2021, Oracle and/or its affiliates.
#
# Licensed under the Universal Permissive License v 1.0 as shown at
https://oss.oracle.com/licenses/upl.
#
# Create user and group
RUN if [ -z "$(getent group root)" ]; then hash groupadd &> /dev/null &&
groupadd root || exit -1 ; fi \
&& if [ -z "$(getent passwd oracle)" ]; then hash useradd &> /dev/null &&
useradd -g root oracle || exit -1; fi \
&& mkdir -p /u01 \
&& chown oracle:root /u01 \
&& chmod 775 /u01

# If Java is not already in the base image, install it
    # Copyright (c) 2021, Oracle and/or its affiliates.
# Licensed under the Universal Permissive License v 1.0 as shown at
https://oss.oracle.com/licenses/upl.
#
# Installing Java

FROM os_update as jdk_build
LABEL com.oracle.weblogic.imagetool.buildid="b4554a25-22dd-4793-
b121-9989bd4be40a"

ENV JAVA_HOME=/u01/jdk

COPY --chown=oracle:root jdk-17.0.12-8-linux-x64.tar.gz /tmp/imagetool/

USER oracle


RUN tar xzf /tmp/imagetool/jdk-17.0.12-8-linux-x64.tar.gz -C /u01 \
&& $(test -d /u01/jdk* && mv /u01/jdk* /u01/jdk || mv /u01/graal* /u01/
jdk) \
&& rm -rf /tmp/imagetool \
&& rm -f /u01/jdk/javafx-src.zip /u01/jdk/src.zip
```

```
# If an Oracle Home is not already in the base image, install the
middleware components
    # Copyright (c) 2021, Oracle and/or its affiliates.
# Licensed under the Universal Permissive License v 1.0 as shown at
https://oss.oracle.com/licenses/upl.
#
# Installing Middleware

FROM os_update as wls_build
LABEL com.oracle.weblogic.imagetool.buildid="b4554a25-22dd-4793-
b121-9989bd4be40a"

ENV JAVA_HOME=/u01/jdk \
ORACLE_HOME=/u01/oracle \
OPATCH_NO_FUSER=true

RUN mkdir -p /u01/oracle \
&& mkdir -p /u01/oracle/oraInventory \
&& chown oracle:root /u01/oracle/oraInventory \
&& chown oracle:root /u01/oracle

COPY --from=jdk_build --chown=oracle:root /u01/jdk /u01/jdk/

COPY --chown=oracle:root fmw_14.1.2.0.0_infrastructure.jar
install.file /tmp/imagetool/
COPY --chown=oracle:root fmw_14.1.2.0.0_soa.jar soasuite.response /tmp/
imagetool/
COPY --chown=oracle:root fmw_14.1.2.0.0_osb.jar osb.response /tmp/
imagetool/
COPY --chown=oracle:root fmw_14.1.2.0.0_b2bhealthcare.jar
b2b.response /tmp/imagetool/
COPY --chown=oracle:root oraInst.loc /u01/oracle/

USER oracle


RUN echo "INSTALLING MIDDLEWARE" \
    && echo "INSTALLING fmw" \
    &&  \
    /u01/jdk/bin/java -Xmx1024m -jar /tmp/imagetool/
fmw_14.1.2.0.0_infrastructure.jar -silent ORACLE_HOME=/u01/oracle \
    -responseFile /tmp/imagetool/install.file -invPtrLoc /u01/oracle/
oraInst.loc -ignoreSysPrereqs -force -novalidation \
    && echo "INSTALLING soa" \
    &&  \
    /u01/jdk/bin/java -Xmx1024m -jar /tmp/imagetool/fmw_14.1.2.0.0_soa.jar
-silent ORACLE_HOME=/u01/oracle \
    -responseFile /tmp/imagetool/soasuite.response -invPtrLoc /u01/oracle/
oraInst.loc -ignoreSysPrereqs -force -novalidation \
    && echo "INSTALLING osb" \
    &&  \
    /u01/jdk/bin/java -Xmx1024m -jar /tmp/imagetool/fmw_14.1.2.0.0_osb.jar
-silent ORACLE_HOME=/u01/oracle \
    -responseFile /tmp/imagetool/osb.response -invPtrLoc /u01/oracle/
```

```
oraInst.loc -ignoreSysPrereqs -force -novalidation \
    && echo "INSTALLING b2b" \
    &&  \
    /u01/jdk/bin/java -Xmx1024m -jar /tmp/imagetool/
fmw_14.1.2.0.0_b2bhealthcare.jar -silent ORACLE_HOME=/u01/oracle \
    -responseFile /tmp/imagetool/b2b.response -invPtrLoc /u01/oracle/
oraInst.loc -ignoreSysPrereqs -force -novalidation \
&& test $? -eq 0 \
&& chmod -R g+r /u01/oracle \
|| (grep -vh "NOTIFICATION" /tmp/OraInstall*/install*.log && exit 1)

#
# Copyright (c) 2021, Oracle and/or its affiliates.
#
# Licensed under the Universal Permissive License v 1.0 as shown at
https://oss.oracle.com/licenses/upl.
#
# Update OPatch and apply WebLogic patches
RUN if [ -f "${ORACLE_HOME}/soa/soa/thirdparty/edifecs/
XEngine_8_4_1_23.tar.gz" ]; then \
        cd $ORACLE_HOME/soa/soa/thirdparty/edifecs && \
        tar -zxvf  XEngine_8_4_1_23.tar.gz; \
    else \
        echo -e "\nXEngine_8_4_1_23.tar.gz not present in $
{ORACLE_HOME}/soa/soa/thirdparty/edifecs directory. Skipping untar."; \
    fi
# zip as few log files grow larger when patches are installed.
RUN if ls /u01/oracle/cfgtoollogs/opatch/*.log; then \
        gzip /u01/oracle/cfgtoollogs/opatch/*.log; \
    fi

FROM os_update as final_build

ENV ORACLE_HOME=/u01/oracle \
    JAVA_HOME=/u01/jdk \
    PATH=${PATH}:/u01/jdk/bin:/u01/oracle/oracle_common/common/bin:/u01/
oracle/wlserver/common/bin:/u01/oracle

LABEL com.oracle.weblogic.imagetool.buildid="b4554a25-22dd-4793-
b121-9989bd4be40a"

    COPY --from=jdk_build --chown=oracle:root /u01/jdk /u01/jdk/

    COPY --from=wls_build --chown=oracle:root /u01/oracle /u01/oracle/



USER oracle
WORKDIR /u01/oracle

#ENTRYPOINT /bin/bash


    ENV ORACLE_HOME=/u01/oracle \
        VOLUME_DIR=/u01/oracle/user_projects \
        SCRIPT_FILE=/u01/oracle/container-scripts/* \
        HEALTH_SCRIPT_FILE=/u01/oracle/container-scripts/
get_healthcheck_url.sh \
        JAVA_OPTIONS="-Doracle.jdbc.fanEnabled=false -
Dweblogic.StdoutDebugEnabled=false" \
        PATH=$PATH:/u01/oracle/container-scripts:/u01/oracle/oracle_common/
modules/thirdparty/org.apache.ant/1.10.5.0.0/apache-ant-1.10.5/bin

    USER root
    RUN mkdir -p $VOLUME_DIR && chown oracle:root /u01 $VOLUME_DIR && \
        mkdir -p /u01/oracle/container-scripts

    COPY --chown=oracle:root files/container-scripts/ /u01/oracle/
container-scripts/
    RUN chmod +xr $SCRIPT_FILE

    USER oracle

    HEALTHCHECK --start-period=5m --interval=1m CMD curl -k -s --fail
`$HEALTH_SCRIPT_FILE` || exit 1
    WORKDIR ${ORACLE_HOME}
    CMD ["/u01/oracle/container-scripts/createDomainAndStart.sh"]

########## END DOCKERFILE ##########
```

5. Check the created image using the docker images command:

```
  $ docker images | grep soasuite
```

Update an image

After setting up the WebLogic Image Tool and required build scripts, use the WebLogic Image Tool to update an existing Oracle SOA Suite Docker image:

1. Enter the following command to add the OPatch patch to the WebLogic Image Tool cache:

```
imagetool cache addEntry --key 28186730_13.9.4.2.17 --value <download
location>/p28186730_1394217_Generic.zip
```

2. Execute the `imagetool cache addEntry` command for each patch to add the required patch(es) to the WebLogic Image Tool cache. For example, to add patch `pXXXXXXXX_141200_Generic.zip`:

```
imagetool cache addEntry --key=XXXXXXXX_14.1.2.0.0 --value <downloaded-
patches-location>/pXXXXXXXX_141200_Generic.zip
```

3. Provide the following arguments to the WebLogic Image Tool update command:
   - –-fromImage - Identify the image that needs to be updated. In the example below, the image to be updated is soasuite:14.1.2.0.
   - –-patches - Multiple patches can be specified as a comma-separated list.
   - --tag - Specify the new tag to be applied for the image being built.

   Refer here for the complete list of options available with the WebLogic Image Tool update command.

Note:

The WebLogic Image Tool cache should have the latest OPatch zip. The WebLogic Image Tool will update the OPatch if it is not already updated in the image.

Examples:

Example for update command:

```
imagetool update --fromImage soasuite:14.1.2.0 --chown oracle:root --
tag=soasuite:14.1.2.0-XXXXXXXX --patches=XXXXXXXX_14.1.2.0.0 --
opatchBugNumber=28186730_13.9.4.2.17

  [INFO   ] Image Tool build ID: bd21dc73-b775-4186-ae03-8219bf02113e
  [INFO   ] Temporary directory used for docker build context: <work-
directory>/wlstmp/wlsimgbuilder_temp1117031733123594064
  [INFO   ] Using patch 28186730_13.9.4.2.17 from cache: <downloaded-
patches-location>/p28186730_1394217_Generic.zip
  [WARNING] skipping patch conflict check, no support credentials provided
  [WARNING] No credentials provided, skipping validation of patches
  [INFO   ] Using patch XXXXXXXX_14.1.2.0.0 from cache: <downloaded-
patches-location>/pXXXXXXXX_141200_Generic.zip
  [INFO   ] docker cmd = docker build --force-rm=true --no-cache --tag
soasuite:14.1.2.0-XXXXXXXX --build-arg http_proxy=http://<YOUR-COMPANY-
PROXY> --build-arg https_proxy=http://<YOUR-COMPANY-PROXY> --build-arg
no_proxy=<IP addresses and Domain address for no_proxy>,/var/run/
docker.sock <work-directory>/wlstmp/wlsimgbuilder_temp1117031733123594064
  Sending build context to Docker daemon  53.47MB

  Step 1/7 : FROM soasuite:14.1.2.0 as FINAL_BUILD
  ---> 445b649a3459
  Step 2/7 : USER root
  ---> Running in 27f45e6958c3
  Removing intermediate container 27f45e6958c3
  ---> 150ae0161d46
  Step 3/7 : ENV OPATCH_NO_FUSER=true
  ---> Running in daddfbb8fd9e
  Removing intermediate container daddfbb8fd9e
  ---> a5fc6b74be39
  Step 4/7 : LABEL com.oracle.weblogic.imagetool.buildid="bd21dc73-
b775-4186-ae03-8219bf02113e"
  ---> Running in cdfec79c3fd4
  Removing intermediate container cdfec79c3fd4
  ---> 4c773aeb956f
  Step 5/7 : USER oracle
  ---> Running in ed3432e43e89
  Removing intermediate container ed3432e43e89
  ---> 54fe6b07c447
  Step 6/7 : COPY --chown=oracle:oracle patches/* /tmp/imagetool/patches/
  ---> d6d12f02a9be
  Step 7/7 : RUN /u01/oracle/OPatch/opatch napply -silent -oh /u01/oracle -
phBaseDir /tmp/imagetool/patches     && /u01/oracle/OPatch/opatch util
cleanup -silent -oh /u01/oracle     && rm -rf /tmp/imagetool
  ---> Running in a79addca4d2f
  Oracle Interim Patch Installer version 13.9.4.2.17
  Copyright (c) 2020, Oracle Corporation.  All rights reserved.


  Oracle Home       : /u01/oracle
  Central Inventory : /u01/oracle/oraInventory
    from           : /u01/oracle/oraInst.loc
  OPatch version    : 13.9.4.2.17
  OUI version       : 13.9.4.0.0
  Log file location : /u01/oracle/cfgtoollogs/opatch/
opatch2024-06-01_10-56-13AM_1.log


  OPatch detects the Middleware Home as "/u01/oracle"

  Verifying environment and performing prerequisite checks...
  OPatch continues with these patches:   XXXXXXXX

  Do you want to proceed? [y|n]
  Y (auto-answered by -silent)
  User Responded with: Y
  All checks passed.

  Please shutdown Oracle instances running out of this ORACLE_HOME on the
local system.
  (Oracle Home = '/u01/oracle')


  Is the local system ready for patching? [y|n]
  Y (auto-answered by -silent)
  User Responded with: Y
  Backing up files...
  Applying interim patch 'XXXXXXXX' to OH '/u01/oracle'
  ApplySession: Optional component(s)
[ oracle.org.bouncycastle.bcprov.ext.jdk15on, 1.55.0.0.0 ] ,
[ oracle.org.bouncycastle.bcprov.ext.jdk15on, 1.55.0.0.0 ] ,
[ oracle.org.bouncycastle.bcprov.ext.jdk15on, 1.5.0.0.0 ] ,
[ oracle.org.bouncycastle.bcprov.ext.jdk15on, 1.5.0.0.0 ] ,
[ oracle.org.bouncycastle.bcprov.jdk15on, 1.55.0.0.0 ] ,
[ oracle.org.bouncycastle.bcprov.jdk15on, 1.55.0.0.0 ] ,
[ oracle.org.bouncycastle.bcprov.jdk15on, 1.52.0.0.0 ] ,
[ oracle.org.bouncycastle.bcprov.jdk15on, 1.52.0.0.0 ] ,
[ oracle.org.bouncycastle.bcprov.ext.jdk15on, 1.48.0.0.0 ] ,
[ oracle.org.bouncycastle.bcprov.ext.jdk15on, 1.48.0.0.0 ] ,
[ oracle.org.bouncycastle.bcpkix.jdk15on, 1.49.0.0.0 ] ,
[ oracle.org.bouncycastle.bcpkix.jdk15on, 1.49.0.0.0 ] ,
[ oracle.org.bouncycastle.bcprov.jdk15on, 1.51.0.0.0 ] ,
[ oracle.org.bouncycastle.bcprov.jdk15on, 1.51.0.0.0 ] ,
[ oracle.org.bouncycastle.bcprov.jdk15on, 1.54.0.0.0 ] ,
[ oracle.org.bouncycastle.bcprov.jdk15on, 1.54.0.0.0 ] ,
[ oracle.org.bouncycastle.bcprov.ext.jdk15on, 1.54.0.0.0 ] ,
[ oracle.org.bouncycastle.bcprov.ext.jdk15on, 1.54.0.0.0 ] ,
[ oracle.org.bouncycastle.bcpkix.jdk15on, 1.5.0.0.0 ] ,
[ oracle.org.bouncycastle.bcpkix.jdk15on, 1.5.0.0.0 ] ,
[ oracle.org.bouncycastle.bcpkix.jdk15on, 1.54.0.0.0 ] ,
[ oracle.org.bouncycastle.bcpkix.jdk15on, 1.54.0.0.0 ] ,
[ oracle.org.bouncycastle.bcpkix.jdk15on, 1.55.0.0.0 ] ,
[ oracle.org.bouncycastle.bcpkix.jdk15on, 1.55.0.0.0 ] ,
[ oracle.org.bouncycastle.bcprov.jdk15on, 1.49.0.0.0 ] ,
[ oracle.org.bouncycastle.bcprov.jdk15on, 1.49.0.0.0 ] ,
[ oracle.org.bouncycastle.bcprov.jdk15on, 1.5.0.0.0 ] ,
[ oracle.org.bouncycastle.bcprov.jdk15on, 1.5.0.0.0 ]  not present in the
Oracle Home or a higher version is found.

  Patching component oracle.org.bouncycastle.bcprov.jdk15on, 1.60.0.0.0...

  Patching component oracle.org.bouncycastle.bcprov.jdk15on, 1.60.0.0.0...

  Patching component oracle.org.bouncycastle.bcprov.ext.jdk15on,
1.60.0.0.0...

  Patching component oracle.org.bouncycastle.bcprov.ext.jdk15on,
1.60.0.0.0...

  Patching component oracle.org.bouncycastle.bcpkix.jdk15on, 1.60.0.0.0...

  Patching component oracle.org.bouncycastle.bcpkix.jdk15on, 1.60.0.0.0...
  Patch XXXXXXXX successfully applied.
  Log file location: /u01/oracle/cfgtoollogs/opatch/
opatch2024-06-01_10-56-13AM_1.log

  OPatch succeeded.
  Oracle Interim Patch Installer version 13.9.4.2.17
  Copyright (c) 2020, Oracle Corporation.  All rights reserved.


  Oracle Home       : /u01/oracle
  Central Inventory : /u01/oracle/oraInventory
    from           : /u01/oracle/oraInst.loc
  OPatch version    : 13.9.4.2.17
  OUI version       : 13.9.4.0.0
  Log file location : /u01/oracle/cfgtoollogs/opatch/
opatch2024-06-01_10-57-19AM_1.log


  OPatch detects the Middleware Home as "/u01/oracle"

  Invoking utility "cleanup"
  OPatch will clean up 'restore.sh,make.txt' files and 'scratch,backup'
directories.
  You will be still able to rollback patches after this cleanup.
  Do you want to proceed? [y|n]
  Y (auto-answered by -silent)
  User Responded with: Y

  Backup area for restore has been cleaned up. For a complete list of
files/directories
  deleted, Please refer log file.

  OPatch succeeded.
  Removing intermediate container a79addca4d2f
  ---> 2ef2a67a685b
  Successfully built 2ef2a67a685b
  Successfully tagged soasuite:14.1.2.0-XXXXXXXX
  [INFO   ] Build successful. Build time=112s. Image tag=soasuite:14.1.2.0-
XXXXXXXX
```

Example Dockerfile generated by the WebLogic Image Tool with the '--dryRun' option:

```
imagetool update --fromImage soasuite:14.1.2.0 --chown oracle:root --
tag=soasuite:14.1.2.0-XXXXXXXX --patches=XXXXXXXX_14.1.2.0.0 --
opatchBugNumber=28186730_13.9.4.2.17 --dryRun

[INFO ] Image Tool build ID: f9feea35-c52c-4974-b155-eb7f34d95892
[INFO ] Temporary directory used for docker build context: <work-
directory>/wlstmp/wlsimgbuilder_temp1799120592903014749
[INFO ] Using patch 28186730_13.9.4.2.17 from cache: <downloaded-patches-
location>/p28186730_1394217_Generic.zip
[WARNING] skipping patch conflict check, no support credentials provided
[WARNING] No credentials provided, skipping validation of patches
[INFO ] Using patch XXXXXXXX_14.1.2.0.0 from cache: <downloaded-patches-
location>/pXXXXXXXX_141200_Generic.zip
[INFO ] docker cmd = docker build --force-rm=true --no-cache --tag
soasuite:14.1.2.0-XXXXXXXX --build-arg http_proxy=http://
www.yourcompany.proxy.com:80 --build-arg https_proxy=http://
www.yourcompany.proxy.com:80 --build-arg
no_proxy=localhost,127.0.0.1,/var/run/docker.sock <work-directory>/wlstmp/
wlsimgbuilder_temp1799120592903014749
########## BEGIN DOCKERFILE ##########
#
# Copyright (c) 2019, 2020, Oracle and/or its affiliates.
#
# Licensed under the Universal Permissive License v 1.0 as shown at
https://oss.oracle.com/licenses/upl.
#
#

FROM soasuite:14.1.2.0 as FINAL_BUILD
USER root

ENV OPATCH_NO_FUSER=true


LABEL com.oracle.weblogic.imagetool.buildid="f9feea35-c52c-4974-b155-
eb7f34d95892"

USER oracle


COPY --chown=oracle:oracle patches/* /tmp/imagetool/patches/

RUN /u01/oracle/OPatch/opatch napply -silent -oh /u01/oracle -
phBaseDir /tmp/imagetool/patches \
&& /u01/oracle/OPatch/opatch util cleanup -silent -oh /u01/oracle \
&& rm -rf /tmp/imagetool
```

```
########## END DOCKERFILE ##########
```

4. Check the built image using the `docker images` command.

```
$ docker images | grep soasuite
  soasuite   14.1.2.0-XXXXXXXX
  2ef2a67a685b        About a minute ago   4.84GB
  $
```

9 Uninstall

Learn how to clean up the Oracle SOA Suite domain setup.

Remove the domain

1. Remove the domain's ingress (for example, Traefik ingress) using Helm:

   ```
   helm uninstall soa-domain-ingress -n sample-domain1-ns
   ```

   For example:

   ```
   helm uninstall soainfra-traefik -n soans
   ```

2. Remove the domain resources by using the sample delete-weblogic-domain-resources.sh script present at ${WORKDIR}/delete-domain:

   ```
   cd ${WORKDIR}/delete-domain
   ./delete-weblogic-domain-resources.sh -d sample-domain1
   ```

   For example:

   ```
   cd ${WORKDIR}/delete-domain
   ./delete-weblogic-domain-resources.sh -d soainfra
   ```

3. Use `kubectl` to confirm that the server pods and domain resource are deleted:

```
kubectl get pods -n sample-domain1-ns
kubectl get domains -n sample-domain1-ns
kubectl get clusters -n sample-domain1-ns
```

For example:

```
kubectl get pods -n soans
kubectl get domains -n soans
kubectl get clusters -n soans
```

Drop the RCU schemas

Follow these steps to drop the RCU schemas created for Oracle SOA Suite domains.

Remove the domain namespace

1. Configure the installed ingress load balancer (for example, Traefik) to stop managing the ingresses in the domain namespace:

   ```
   helm upgrade traefik traefik/traefik \
       --namespace traefik \
   ```

```
    --reuse-values \
    --set "kubernetes.namespaces={traefik}" \
    --wait
```

2. Delete the domain namespace:

   ```
   $ kubectl delete namespace sample-domain1-ns
   ```

   For example:

   ```
   kubectl delete namespace soans
   ```

Remove the operator

1. Remove the operator:

   ```
   helm uninstall sample-weblogic-operator -n sample-weblogic-operator-ns
   ```

   For example:

   ```
   helm uninstall weblogic-kubernetes-operator -n opns
   ```

2. Remove the operator’s namespace:

   ```
   kubectl delete namespace sample-weblogic-operator-ns
   ```

   For example:

   ```
   kubectl delete namespace opns
   ```

Remove the load balancer

1. Remove the installed ingress based load balancer (for example, Traefik):

   ```
   helm uninstall traefik -n traefik
   ```

2. Remove the Traefik namespace:

   ```
   kubectl delete namespace traefik
   ```

Delete the domain home

To remove the domain home that is generated using the `create-domain.sh` script, with appropriate privileges manually delete the contents of the storage attached to the domain home persistent volume (PV). For example, for the domain’s persistent volume of type host_path:

```
rm -rf /scratch/k8s_dir/SOA/*
```

10

Frequently Asked Questions

Overriding tuning parameters is not supported using configuration overrides

The WebLogic Kubernetes Operator enables you to override some of the domain configuration using configuration overrides (also called situational configuration). See supported overrides. Overriding the tuning parameters such as **MaxMessageSize** and **PAYLOAD**, for Oracle SOA Suite domains is not supported using the configuration overrides feature. However, you can override them using the following steps:

1. Specify the new value using the environment variable `K8S_REFCONF_OVERRIDES` in `serverPod.env` section in `domain.yaml` configuration file (example path: <domain-creation-output-directory>/weblogic-domains/soainfra/domain.yaml) based on the servers to which the changes are to be applied. For example, to override the value at the Administration Server pod level:

```
  spec:
    adminServer:
      serverPod:
        env:
        - name: K8S_REFCONF_OVERRIDES
          value: "-Dweblogic.MaxMessageSize=78787878"
        - name: USER_MEM_ARGS
          value: '-Djava.security.egd=file:/dev/./urandom -Xms512m -
Xmx1024m '
      serverStartState: RUNNING
```

   For example, to override the value at a specific cluster level (soa_cluster or osb_cluster):

```
  apiVersion: "weblogic.oracle/v1"
  kind: Cluster
  metadata:
    name: soainfra-soa-cluster
    # Update this with the namespace your domain will run in:
    namespace: soans
    labels:
      # Update this with the `domainUID` of your domain:
      weblogic.domainUID: soainfra
  spec:
    clusterName: soa_cluster
    serverService:
      precreateService: true
    serverPod:
      env:
      - name: K8S_REFCONF_OVERRIDES
        value: "-Dsoa.payload.threshold.kb=102410"
```

Note:

When multiple system properties are specified for

```
serverPod.env.value
```

, make sure each system property is separated by a space.

2. Apply the updated

   ```
   domain.yaml
   ```

   file:

   ```
   kubectl apply -f domain.yaml
   ```

   Note:

   Note: The server pod(s) will be automatically restarted (rolling restart).

Enterprise Manager Console may display ADF_FACES-30200 error

In an Oracle SOA Suite environment deployed using the operator, the Enterprise Manager Console may intermittently display the following error when the domain servers are restarted:

```
ADF_FACES-30200: For more information, please see the server's error log for
an entry beginning with: The UIViewRoot is null. Fatal exception during
PhaseId: RESTORE_VIEW 1.
```

You can refresh the Enterprise Manager Console URL to successfully log in to the Console.

Configure the external URL access for Oracle SOA Suite composite applications

For Oracle SOA Suite composite applications to access the external URLs over the internet (if your cluster is behind a http proxy server), you must configure the following proxy parameters for Administration Server and Managed Server pods.

```
-Dhttp.proxyHost=www-your-proxy.com
-Dhttp.proxyPort=proxy-port
-Dhttps.proxyHost=www-your-proxy.com
-Dhttps.proxyPort=proxy-port
-Dhttp.nonProxyHosts="localhost|soainfra-adminserver|soainfra-soa-server1|
soainfra-osb-server1|...soainfra-soa-serverN|*.svc.cluster.local|
*.your.domain.com|/var/run/docker.sock"
```

To do this, edit the `domain.yaml` configuration file and append the proxy parameters to the `spec.serverPod.env.JAVA_OPTIONS` environment variable value.

For example:

```
  serverPod:
    env:
    - name: JAVA_OPTIONS
      value: -Dweblogic.StdoutDebugEnabled=false -Dweblogic.ssl.Enabled=true -
Dweblogic.security.SSL.ignoreHostnameVerification=true -Dhttp.proxyHost=www-
your-proxy.com -Dhttp.proxyPort=proxy-port -Dhttps.proxyHost=www-your-
proxy.com -Dhttps.proxyPort=proxy-port -Dhttp.nonProxyHosts="localhost|
soainfra-adminserver|soainfra-soa-server1|soainfra-osb-server1|...soainfra-
soa-serverN|*.svc.cluster.local|*.your.domain.com|/var/run/docker.sock"
    - name: USER_MEM_ARGS
      value: '-Djava.security.egd=file:/dev/./urandom -Xms256m -Xmx1024m '
    volumeMounts:
```

Note:

The `-Dhttp.nonProxyHosts` parameter must have the pod names of the Administration Server and each Managed Server. For example: `soainfra-adminserver`, `soainfra-soa-server1`, `soainfra-osb-server1`, and so on.

Apply the updated `domain.yaml` file:

```
 $ kubectl apply -f domain.yaml
```

Note:

The server pod(s) will be automatically restarted (rolling restart).

Configure the external access for the Oracle Enterprise Scheduler WebServices WSDL URLs

In an Oracle SOA Suite domain deployed including the Oracle Enterprise Scheduler (ESS) component, the following ESS WebServices WSDL URLs shown in the table format in the `ess/essWebServicesWsdl.jsp` page are not reachable outside the Kubernetes cluster.

```
ESSWebService
EssAsyncCallbackService
EssWsJobAsyncCallbackService
```

Follow these steps to configure the external access for the Oracle Enterprise Scheduler WebServices WSDL URLs:

1. Log in to the WebLogic Remote Console URL and connect to the Administration Server.
2. In the Home Page, click **Clusters**. Then click the **soa_cluster**.
3. Click the **HTTP** tab and then click **Lock & Edit** in the Change Center panel.
4. Update the following values:
   - Frontend Host: host name of the load balancer. For example, domain1.example.com.

- Frontend HTTP Port: load balancer port. For example, 30305.
- Frontend HTTPS Port: load balancer https port. For example, 30443.

5. Click **Save**.
6. Click **Activate Changes** in the Change Center panel.
7. Restart the servers using `domain-lifecycle` [scripts] in the SOA cluster. For more information, refer Restart the servers .

Note:

Do not restart servers from the Remote Console.

WebLogic Kubernetes Operator FAQs

See the general frequently asked questions for using the WebLogic Kubernetes Operator.

11 Appendix

This section provides information on miscellaneous tasks related to Oracle SOA Suite domains deployment on Kubernetes.

- Domain Resource Sizing

  Describes the resourse sizing information for Oracle SOA Suite domains setup on Kubernetes cluster.

- Quick Start Deployment On-Premise

  Describes how to quickly get an Oracle SOA Suite domain instance running (using the defaults, nothing special) for development and test purposes.

- Security Hardening

  Review resources for the Docker and Kubernetes cluster hardening.

Domain Resource Sizing

Oracle SOA cluster sizing minimum requirements

Oracle SOA	Normal Usage	Moderate Usage	High Usage
Administration Server	No of CPU core(s) : 1, Memory : 4GB	No of CPU core(s) : 1, Memory : 4GB	No of CPU core(s) : 1, Memory : 4GB
Number of Managed Servers	2	2	4
Configurations per Managed Server	No of CPU core(s) : 2, Memory : 16 GB	No of CPU core(s) : 4, Memory : 16 GB	No of CPU core(s) : 6, Memory : 16 - 32 GB
PV Storage	Minimum 250 GB	Minimum 250 GB	Minimum 500 GB

Quick Start Deployment On-Premise

Use this Quick Start to create an Oracle SOA Suite domain deployment in a Kubernetes cluster (on-premise environments) with the WebLogic Kubernetes Operator.

Note that this walkthrough is for demonstration purposes only, not for use in production. These instructions assume that you are already familiar with Kubernetes. If you need more detailed instructions, refer to the Install Guide.

Hardware requirements

The Linux kernel supported for deploying and running Oracle SOA Suite domains with the operator is Oracle Linux 8 and Red Hat Enterprise Linux 8. Refer to the prerequisites for more details. For this exercise, the minimum hardware requirements to create a single-node Kubernetes cluster and then deploy the `soaosb` (Oracle SOA Suite, Oracle Service Bus, and Enterprise Scheduler (ESS)) domain type with one Managed Server for Oracle SOA Suite and one for the Oracle Service Bus cluster, along with Oracle Database running as a container are:

Hardware	Size
RAM	32 GB
Disk Space	250 GB+
CPU core(s)	6

See here for resource sizing information for Oracle SOA Suite domains set up on a Kubernetes cluster.

Set up Oracle SOA Suite in an on-premise environment

Use the steps in this topic to create a single-instance on-premise Kubernetes cluster and then create an Oracle SOA Suite soaosb domain type, which deploys a domain with Oracle SOA Suite, Oracle Service Bus, and Oracle Enterprise Scheduler (ESS).

1. Step 1 - Prepare a Virtual Machine for the Kubernetes Cluster
2. Step 2 - Set Up a Single Instance Kubernetes Cluster
3. Step 3 - Get Scripts and Images
4. Step 4 - Install the WebLogic Kubernetes Operator
5. Step 5 - Install the Traefik (ingress-based) Load Balancer
6. Step 6 - Create and Configure an Oracle SOA Suite Domain

Prepare a Virtual Machine for the Kubernetes Cluster

For illustration purposes, these instructions are for Oracle Linux 8. If you are using a different flavor of Linux, you will need to adjust the steps accordingly.

Note:

These steps must be run with the `root` user, unless specified otherwise. Any time you see `YOUR_USERID` in a command, you should replace it with your actual `userid`.

1. Prerequisites
   a. Choose the directories where your Kubernetes files will be stored. The Kubernetes directory is used for the `/var/lib/kubelet` file system and persistent volume storage.

      ```
      export kubelet_dir=/u01/kubelet
      mkdir -p $kubelet_dir
      ln -s $kubelet_dir /var/lib/kubelet
      ```

   b. Verify that IPv4 forwarding is enabled on your host.

Note:

Replace eth0 with the ethernet interface name of your compute resource if it is different.

```
/sbin/sysctl -a 2>&1|grep -s 'net.ipv4.conf.eth0.forwarding'
/sbin/sysctl -a 2>&1|grep -s 'net.ipv4.conf.lo.forwarding'
/sbin/sysctl -a 2>&1|grep -s 'net.ipv4.ip_nonlocal_bind'
```

For example: Verify that all are set to 1:

```
net.ipv4.conf.eth0.forwarding = 1
net.ipv4.conf.lo.forwarding = 1
net.ipv4.ip_nonlocal_bind = 1
```

Solution: Set all values to 1 immediately:

```
/sbin/sysctl net.ipv4.conf.eth0.forwarding=1
/sbin/sysctl net.ipv4.conf.lo.forwarding=1
/sbin/sysctl net.ipv4.ip_nonlocal_bind=1
```

c. To preserve the settings permanently: Update the above values to 1 in files in /usr/lib/sysctl.d/, /run/sysctl.d/, and /etc/sysctl.d/.

d. Verify the iptables rule for forwarding. Kubernetes uses iptables to handle many networking and port forwarding rules. A standard container installation may create a firewall rule that prevents forwarding. Verify if the iptables rule to accept forwarding traffic is set:

```
/sbin/iptables -L -n | awk '/Chain FORWARD / {print $4}' | tr -d ")"
```

If the output is “DROP”, then run the following command:

```
/sbin/iptables -P FORWARD ACCEPT
```

Verify if the iptables rule is properly set to “ACCEPT”:

```
/sbin/iptables -L -n | awk '/Chain FORWARD / {print $4}' | tr -d ")"
```

e. Disable and stop firewalld:

```
systemctl disable firewalld
systemctl stop firewalld
```

2. Install CRI-O and Podman

> **Note:**
>
> If you have already configured CRI-O and Podman, continue to Install and configure Kubernetes.

a. Make sure that you have the right operating system version:

```
uname -a
more /etc/oracle-release
```

Example output:

```
Linux xxxxxx 5.15.0-100.96.32.el8uek.x86_64 #2 SMP Tue Feb 27 18:08:15
PDT 2024 x86_64 x86_64 x86_64 GNU/Linux
Oracle Linux Server release 8.6
```

b. Installing CRI-O:

Add OLCNE(Oracle Cloud Native Environment) Repository to dnf config-manager. This allows dnf to install the additional packages required for CRI-O installation.

On Oracle Linux 8:

```
dnf config-manager --add-repo https://yum.oracle.com/repo/
OracleLinux/OL8/olcne18/x86_64
```

On Oracle Linux 9:

```
dnf config-manager --add-repo https://yum.oracle.com/repo/
OracleLinux/OL9/olcne18/x86_64
```

Install the cri-o:

```
dnf install -y cri-o
```

> **Note:**
>
> Note: To install a different version of CRI-O or on a different operating system, see CRI-O Installation Instructions.

c. Start the CRI-O service:

Set up Kernel Modules and Proxies

```
### Enable kernel modules overlay and br_netfilter which are required
for Kubernetes Container Network Interface (CNI) plugins
modprobe overlay
modprobe br_netfilter

### To automatically load these modules at system start up create
config as below
```

```
cat <<EOF > /etc/modules-load.d/crio.conf
overlay
br_netfilter
EOF
sysctl --system

### Set the environmental variable CONTAINER_RUNTIME_ENDPOINT to
crio.sock to use crio as the container runtime
export CONTAINER_RUNTIME_ENDPOINT=unix:///var/run/crio/crio.sock

### Setup Proxy for CRIO service
cat <<EOF > /etc/sysconfig/crio
http_proxy=http://REPLACE-WITH-YOUR-COMPANY-PROXY-HOST:PORT
https_proxy=http://REPLACE-WITH-YOUR-COMPANY-PROXY-HOST:PORT
HTTPS_PROXY=http://REPLACE-WITH-YOUR-COMPANY-PROXY-HOST:PORT
HTTP_PROXY=http://REPLACE-WITH-YOUR-COMPANY-PROXY-HOST:PORT
no_proxy=localhost,127.0.0.0/8,ADD-YOUR-INTERNAL-NO-PROXY-LIST,/var/run/
crio/crio.sock
NO_PROXY=localhost,127.0.0.0/8,ADD-YOUR-INTERNAL-NO-PROXY-LIST,/var/run/
crio/crio.sock
EOF
```

Set the runtime for CRI-O

```
### Setting the runtime for crio
## Update crio.conf
vi /etc/crio/crio.conf
## Append following under [crio.runtime]
conmon_cgroup = "kubepods.slice"
cgroup_manager = "systemd"
## Uncomment following under [crio.network]
network_dir="/etc/cni/net.d"
plugin_dirs=[
    "/opt/cni/bin",
    "/usr/libexec/cni",
]
```

Start the CRI-O Service

```
## Restart crio service
systemctl restart crio.service
systemctl enable --now crio
```

d. Installing Podman:

On Oracle Linux 8, if podman is not available, then install Podman and related tools with following command syntax:

```
shell$ sudo dnf module install container-tools:ol8
```

On Oracle Linux 9, if podman is not available, then install Podman and related tools with following command syntax:

```
$ sudo dnf install container-tools
```

Since the setup uses docker CLI commands, on Oracle Linux 8/9, install the podman-docker package if not available, that effectively aliases the docker command to podman,with following command syntax:

```
$ sudo dnf install podman-docker
```

e. Configure Podman rootless:

For using podman with your User ID (Rootless environment), Podman requires the user running it to have a range of UIDs listed in the files /etc/subuid and /etc/subgid. Rather than updating the files directly, the usermod program can be used to assign UIDs and GIDs to a user with the following commands:

```
$ sudo /sbin/usermod --add-subuids 100000-165535 --add-subgids
100000-165535
              <REPLACE_USER_ID>
            $ podman system migrate
```

Note:

The above podman system migrate needs to be executed with your User ID and not root.

Verify the user-id addition

```
cat /etc/subuid
cat /etc/subgid
```

Expected similar output

```
opc:100000:65536
<user-id>:100000:65536
```

3. Install and Configure Kubernetes

a. Add the external Kubernetes repository:

```
cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.28/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.28/rpm/repodata/
repomd.xml.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
EOF
```

b. Set SELinux in permissive mode (effectively disabling it):

export PATH=/sbin:$PATH setenforce 0 sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

c. Export proxy and enable kubelet:

```
### Get the nslookup IP address of the master node to use with
apiserver-advertise-address during setting up Kubernetes master
### as the host may have different internal ip (hostname -i) and
nslookup $HOSTNAME
ip_addr=`nslookup $(hostname -f) | grep -m2 Address | tail -n1| awk -F:
'{print $2}'| tr -d " "`
echo $ip_addr

### Set the proxies
export NO_PROXY=localhost,127.0.0.0/8,ADD-YOUR-INTERNAL-NO-PROXY-
LIST,/var/run/crio/crio.sock,$ip_addr,.svc
export no_proxy=localhost,127.0.0.0/8,ADD-YOUR-INTERNAL-NO-PROXY-
LIST,/var/run/crio/crio.sock,$ip_addr,.svc
export http_proxy=http://REPLACE-WITH-YOUR-COMPANY-PROXY-HOST:PORT
export https_proxy=http://REPLACE-WITH-YOUR-COMPANY-PROXY-HOST:PORT
export HTTPS_PROXY=http://REPLACE-WITH-YOUR-COMPANY-PROXY-HOST:PORT
export HTTP_PROXY=http://REPLACE-WITH-YOUR-COMPANY-PROXY-HOST:PORT

### Install the kubernetes components and enable the kubelet service so
that it automatically restarts on reboot
dnf install -y kubeadm kubelet kubectl
systemctl enable --now kubelet
```

d. Ensure net.bridge.bridge-nf-call-iptables is set to 1 in your sysctl to avoid traffic routing issues:

```
cat <<EOF >  /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
EOF
sysctl --system
```

e. Disable swap check:

```
sed -i 's/KUBELET_EXTRA_ARGS=/KUBELET_EXTRA_ARGS="--fail-swap-
on=false"/' /etc/sysconfig/kubelet
cat /etc/sysconfig/kubelet
### Reload and restart kubelet
systemctl daemon-reload
systemctl restart kubelet
```

f. Pull the images using crio:

```
kubeadm config images pull --cri-socket unix:///var/run/crio/crio.sock
```

4. Set up Helm

a. Install Helm v3.10.2+.

1. a. Download Helm from https://github.com/helm/helm/releases.

For example, to download Helm v3.10.2:

```
wget https://get.helm.sh/helm-v3.10.2-linux-amd64.tar.gz
```

2. Unpack tar.gz:

```
tar -zxvf helm-v3.10.2-linux-amd64.tar.gz
```

Find the Helm binary in the unpacked directory, and move it to its desired destination:

```
mv linux-amd64/helm /usr/bin/helm
```

b. Run `helm version` to verify its installation:

```
helm version
  version.BuildInfo{Version:"v3.10.2",
GitCommit:"50f003e5ee8704ec937a756c646870227d7c8b58",
GitTreeState:"clean", GoVersion:"go1.18.8"}
```

Set Up a Single Instance Kubernetes Cluster

Note:

- These steps must be run with the root user, unless specified otherwise!
- If you choose to use a different CIDR block (that is, other than 10.244.0.0/16 for the --pod-network-cidr= in the kubeadm init command), then also update NO_PROXY and no_proxy with the appropriate value.
  – Also make sure to update kube-flannel.yaml with the new value before deploying.
- Replace the following with appropriate values:
  – ADD-YOUR-INTERNAL-NO-PROXY-LIST
  – REPLACE-WITH-YOUR-COMPANY-PROXY-HOST:PORT

Set up the master node

1. Create a shell script that sets up the necessary environment variables. You can append this to the user's .bashrc so that it will run at login. You must also configure your proxy settings here if you are behind an HTTP proxy:

```
## grab my IP address to pass into  kubeadm init, and to add to no_proxy
vars
ip_addr=`nslookup $(hostname -f) | grep -m2 Address | tail -n1| awk -F:
'{print $2}'| tr -d " "`
export pod_network_cidr="10.244.0.0/16"
export service_cidr="10.96.0.0/12"
export PATH=$PATH:/sbin:/usr/sbin
```

```
### Set the proxies
export NO_PROXY=localhost,.svc,127.0.0.0/8,ADD-YOUR-INTERNAL-NO-PROXY-
LIST,/var/run/crio/crio.sock,$ip_addr,$pod_network_cidr,$service_cidr
export no_proxy=localhost,.svc,127.0.0.0/8,ADD-YOUR-INTERNAL-NO-PROXY-
LIST,/var/run/crio/crio.sock,$ip_addr,$pod_network_cidr,$service_cidr
export http_proxy=http://REPLACE-WITH-YOUR-COMPANY-PROXY-HOST:PORT
export https_proxy=http://REPLACE-WITH-YOUR-COMPANY-PROXY-HOST:PORT
export HTTPS_PROXY=http://REPLACE-WITH-YOUR-COMPANY-PROXY-HOST:PORT
export HTTP_PROXY=http://REPLACE-WITH-YOUR-COMPANY-PROXY-HOST:PORT
```

2. Source the script to set up your environment variables:

   ```
   . ~/.bashrc
   ```

3. To implement command completion, add the following to the script:

   ```
   [ -f /usr/share/bash-completion/bash_completion ] && . /usr/share/bash-
   completion/bash_completion
   source <(kubectl completion bash)
   ```

4. Run kubeadm init to create the master node:

   ```
   kubeadm init \
     --pod-network-cidr=$pod_network_cidr \
     --apiserver-advertise-address=$ip_addr \
     --ignore-preflight-errors=Swap  > /tmp/kubeadm-init.out 2>&1
   ```

5. Log in to the terminal with YOUR_USERID:YOUR_GROUP. Then set up the ~/.bashrc similar to steps 1 to 3 with YOUR_USERID:YOUR_GROUP.

   > **Note:**
   >
   > From now on we will be using YOUR_USERID:YOUR_GROUP to execute any kubectl commands and not root.

6. Set up YOUR_USERID:YOUR_GROUP to access the Kubernetes cluster:

   ```
   mkdir -p $HOME/.kube
   sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
   sudo chown $(id -u):$(id -g) $HOME/.kube/config
   ```

7. Verify that YOUR_USERID:YOUR_GROUP is set up to access the Kubernetes cluster using the kubectl command:

   ```
   kubectl get nodes
   ```

   > **Note:**
   >
   > At this step, the node is not in ready state as we have not yet installed the pod network add-on. After the next step, the node will show status as Ready.

8. Install a pod network add-on (flannel) so that your pods can communicate with each other.

Note:

If you are using a different CIDR block than 10.244.0.0/16, then download and update kube-flannel.yml with the correct CIDR address before deploying into the cluster:

```
wget https://github.com/flannel-io/flannel/releases/download/v0.25.1/kube-
flannel.yml
### Update the CIDR address if you are using a CIDR block other than the
default 10.244.0.0/16
kubectl apply -f kube-flannel.yml
```

9. Verify that the master node is in Ready status:

```
kubectl get nodes
```

Sample output:

```
NAME                 STATUS      ROLES          AGE    VERSION
mymasternode         Ready     control-plane    12h    v1.27.2
```

or:

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

Sample output:

```
NAME                                        READY       STATUS
RESTARTS    AGE
pod/coredns-86c58d9df4-58p9f                    1/1         Running
0       3m59s
pod/coredns-86c58d9df4-mzrr5                    1/1         Running
0       3m59s
pod/etcd-mymasternode                           1/1         Running
0       3m4s
pod/kube-apiserver-node                         1/1         Running
0       3m21s
pod/kube-controller-manager-mymasternode    1/1         Running
0       3m25s
pod/kube-flannel-ds-6npx4                       1/1         Running
0       49s
pod/kube-proxy-4vsgm                            1/1         Running
0       3m59s
pod/kube-scheduler-mymasternode                 1/1         Running
0       2m58s
```

10. To schedule pods on the master node, taint the node:

```
kubectl taint nodes --all node-role.kubernetes.io/control-plane-
```

Congratulations! Your Kubernetes cluster environment is ready to deploy your Oracle SOA Suite domain.

Refer to the official documentation to set up a Kubernetes cluster.

Get Scripts and Images

1. Set up the source code repository to deploy Oracle SOA Suite domains
   a. Create a working directory to set up the source code:

      ```
      mkdir $HOME/soa_14.1.2
      cd $HOME/soa_14.1.2
      ```

   b. Download the WebLogic Kubernetes Operator source code and Oracle SOA Suite Kubernetes deployment scripts from the SOA repository. Required artifacts are available at OracleSOASuite/kubernetes.

      ```
      git clone https://github.com/oracle/fmw-kubernetes.git
      export WORKDIR=$HOME/soa_14.1.2/fmw-kubernetes/OracleSOASuite/kubernetes
      ```

2. Get required Docker images and add them to your local registry
   a. Pull the WebLogic Kubernetes Operator image:

      ```
      podman pull ghcr.io/oracle/weblogic-kubernetes-operator:4.2.9
      ```

   b. Obtain the Oracle Database image and Oracle SOA Suite Docker image from the Oracle Container Registry:

      https://container-registry.oracle.com/

      For first time users, to pull an image from the Oracle Container Registry, navigate to https://container-registry.oracle.com and log in using the Oracle Single Sign-On (SSO) authentication service. If you do not already have SSO credentials, you can create an Oracle Account using: https://profile.oracle.com/myprofile/account/create-account.jspx.

      Use the web interface to accept the Oracle Standard Terms and Restrictions for the Oracle software images that you intend to deploy. Your acceptance of these terms are stored in a database that links the software images to your Oracle Single Sign-On login credentials.

      To obtain the image, log in to the Oracle Container Registry:

      ```
      podman login container-registry.oracle.com
      ```

      Find and then pull the Oracle Database image for 19.3.0.0:

      ```
      podman pull container-registry.oracle.com/database/enterprise:19.3.0.0
      ```

      Find and then pull the prebuilt Oracle SOA Suite image 14.1.2.0.0 install image:

      ```
      podman pull container-registry.oracle.com/middleware/soasuite:14.1.2.0-
      <17 or 21>-<ol8 or ol9>-<tag>
      ```

Note:

This image is only the base GA product and does not contain any additional Oracle SOA Suite product patches.

Install the WebLogic Kubernetes Operator

1. Prepare for the WebLogic Kubernetes Operator.
   a. Create a namespace opns for the operator:

      ```
      kubectl create namespace opns
      ```

   b. Create a service account op-sa for the operator in the operator's namespace:

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

2. Install the WebLogic Kubernetes Operator.

   Use Helm to install and start the operator from the directory you just cloned:

   ```
   helm repo add weblogic-operator https://oracle.github.io/weblogic-
   kubernetes-operator/charts --force-update
   helm install weblogic-kubernetes-operator weblogic-operator/weblogic-
   operator --version 4.2.9 --namespace opns --set serviceAccount=op-sa --set
   "javaLoggingLevel=FINE" --wait
   ```

3. Verify the WebLogic Kubernetes Operator.
   a. Verify that the operator's pod is running by listing the pods in the operator's namespace. You should see one for the operator:

      ```
      kubectl get pods -n opns
      ```

   b. Verify that the operator is up and running by viewing the operator pod's logs:

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

      The WebLogic Kubernetes Operator v4.2.9 has been installed. Continue with the load balancer and Oracle SOA Suite domain setup.

Install the Traefik (ingress-based) Load Balancer

The WebLogic Kubernetes Operator supports these load balancers: Traefik, NGINX, and Apache. Samples are provided in the documentation.

This Quick Start demonstrates how to install the Traefik ingress controller to provide load balancing for an Oracle SOA Suite domain.

1. Create a namespace for Traefik:

   ```
   kubectl create namespace traefik
   ```

2. Set up Helm for 3rd party services:

```
helm repo add traefik https://helm.traefik.io/traefik --force-update
```

3. Install the Traefik operator in the traefik namespace with the provided sample values:

```
cd ${WORKDIR}
helm install traefik traefik/traefik \
 --namespace traefik \
 --values charts/traefik/values.yaml \
 --set "kubernetes.namespaces={traefik}" \
 --set "service.type=NodePort" \
 --wait
```

Create and Configure an Oracle SOA Suite Domain

1. Prepare for an Oracle SOA Suite domain.
   a. Create a namespace that can host Oracle SOA Suite domains. Label the namespace with `weblogic-operator=enabled` to manage the domain.

   ```
   kubectl create namespace soans
   kubectl label namespace soans weblogic-operator=enabled
   ```

   b. Create the Kubernetes persistence volume and persistence volume claim.
      i. a. Create the Oracle SOA Suite domain home directory. Determine if a user already exists on your host system with uid:gid of 1000:0:

      ```
      sudo getent passwd 1000
      ```

      If this command returns a username (which is the first field), you can skip the following `useradd` command. If not, create the oracle user with `useradd`:

      ```
      sudo useradd -u 1000 -g 0 oracle
      ```

      Create the directory that will be used for the Oracle SOA Suite domain home:

      ```
      sudo mkdir /scratch/k8s_dir
      sudo chown -R 1000:0 /scratch/k8s_dir
      ```

      ii. The `create-pv-pvc-inputs.yaml` has the following values by default:
      - baseName: domain
      - domainUID: soainfra
      - namespace: soans
      - weblogicDomainStoragePath: /scratch/k8s_dir

      Review and update if any changes required.

      ```
      cp create-pv-pvc-inputs.yaml create-pv-pvc-inputs-domain.yaml
            vi create-pv-pvc-inputs-domain.yaml
      ```

iii. Run the `create-pv-pvc.sh` script to create the PV and PVC configuration files using updated `create-pv-pvc-inputs-domain.yaml`:

```
cd ${WORKDIR}/create-weblogic-domain-pv-pvc
./create-pv-pvc.sh -i create-pv-pvc-inputs-domain.yaml -o output-
domain
```

iv. Create the PV and PVC using the configuration files created in the previous step:

```
kubectl create -f  output-domain/pv-pvcs/soainfra-domain-pv.yaml
kubectl create -f  output-domain/pv-pvcs/soainfra-domain-pvc.yaml
```

c. Install the database for the Oracle SOA Suite domain.

This step is required only when a standalone database is not already set up and you want to use the database in a container. You can refer to [here to create an Oracle Database service in a Kubernetes. In this guide we will be using Single Instance Database on Kubernetes using helm chart to provision the Database without using an Oracle Database Operator.

i. Clone the Oracle Database GitHub repository:

```
cd ${WORKDIR}
git clone https://github.com/oracle/docker-images.git
cd docker-images/OracleDatabase/SingleInstance/helm-charts
```

ii. Oracle database data files and configurations are stored at the /opt/oracle/oradata path of the container. Storage Class for Persistent Volume Claims has to be configured to keep the data persistence across deployments. If persistence is not desired, then you can skip the below steps and override the `persistence` to `null` using `-set` option. Follow the below steps to create the required Storage Class for Persistent Volume Claims:

i. Create a directory used for database persistence.

```
sudo mkdir /scratch/k8s_dir_db
sudo chown -R 54321 /scratch/k8s_dir_db
```

ii. Update the create-pv-pvc-inputs.yaml for creating the Persistent Volume with below values:

- baseName: `db`
- namespace: `default`
- weblogicDomainStorageSize: `100Gi`
- weblogicDomainStoragePath: `/scratch/k8s_dir_db`

```
cd ${WORKDIR}/create-weblogic-domain-pv-pvc
cp create-pv-pvc-inputs.yaml create-pv-pvc-inputs-db.yaml
sed -i -e "s/baseName: domain/baseName: db/g" create-pv-pvc-
inputs-db.yaml
sed -i -e "s/namespace: soans/namespace: default/g" create-pv-
pvc-inputs-db.yaml
sed -i -e "s/k8s_dir/k8s_dir_db/g" create-pv-pvc-inputs-db.yaml
sed -i -e "s/10Gi/100Gi/g" create-pv-pvc-inputs-db.yaml
```

iii. Run the create-pv-pvc.sh script to create the PV configuration file using updated create-pv-pvc-inputs-db.yaml:

```
cd ${WORKDIR}/create-weblogic-domain-pv-pvc
./create-pv-pvc.sh -i create-pv-pvc-inputs-db.yaml -o output-db
```

> **Note:**
>
> Do not create the PVC because helm charts use the StorageClass name and create the PVC.

iv. The above script creates for both PV and PVC, but we will be **only using PV configuration file** created in previous step:

```
kubectl create -f  output-db/pv-pvcs/soainfra-db-pv.yaml
```

i. Create a database in a container using the helm-charts::

```
cd ${WORKDIR}/docker-images/OracleDatabase/SingleInstance/
helm-charts
helm  install db19c \
        --set persistence.storageClass=soainfra-db-storage-
class \
        --set imagePullPolicy=IfNotPresent \
        oracle-db
```

ii. Verify the logs for database deployment until shows the message “DATABASE IS READY TO USE”:

```
kubectl  logs deployment.apps/db19c-oracle-db -f
```

iii. Get the password of the SYS user:

```
kubectl get secret db19c-oracle-db -o
jsonpath={.data.oracle_pwd} | base64 --decode; echo
```

The helm-chart creates the required service to access the database using connection string db19c-oracle-db.default.svc.cluster.local:1521/ ORCLPDB1 which will be used as an rcuDatabaseURL parameter in the create-domain-inputs.yaml file. In case you have used different Helm release name or configuration parameters for creating the database update the connection string accordingly.

iv. Create Oracle SOA Suite schemas for the domain type (for example, soaosb).

i. Create a secret that contains the database’s SYSDBA username and password.

```
kubectl -n default create secret generic oracle-rcu-
secret \
   --from-literal='sys_username=sys' \
   --from-literal='sys_password=REPLACE-WITH-SYS-
```

```
PASSWORD' \
   --from-literal='password=REPLACE-WITH-SCHEMA-PASSWORD'
```

To install the Oracle SOA Suite schemas, run the create-rcu-schema.sh script with the following inputs:

- -s <RCU PREFIX>
- -t <SOA domain type>
- -d <Oracle Database URL>
- -i <SOASuite image>
- -n <Namespace>
- -c <Name of credentials secret containing SYSDBA username and password and RCU schema owner password>
- -r <Comma-separated variables>
- -l <Timeout limit in seconds. (optional). (default: 300)>

For example:

```
ccd ${WORKDIR}/create-rcu-schema
./create-rcu-schema.sh \
-s SOA1 \
-t soaosb \
-d db19c-oracle-db.default.svc.cluster.local:1521/
ORCLPDB1 \
-i container-registry.oracle.com/middleware/
soasuite:14.1.2.0-<17 or 21>-<ol8 or ol9>-<tag> \
-n default \
-b EBR \
-c oracle-rcu-secret \
-r SOA_PROFILE_TYPE=SMALL,HEALTHCARE_INTEGRATION=NO
```

v. Create Kubernetes secrets required for domain creation.

   i. Create a Kubernetes secret for the domain in the same Kubernetes namespace as the domain. In this example, the username is weblogic, the password is Welcome1, and the namespace is soans:

```
cd ${WORKDIR}/create-weblogic-domain-credentials
./create-weblogic-credentials.sh \
     -u weblogic \
     -p Welcome1 \
     -n soans    \
     -d soainfra \
     -s soainfra-domain-credentials
```

   ii. Create a Kubernetes secret for the RCU in the same Kubernetes namespace as the domain:

   - Schema user : SOA1
   - Schema password : REPLACE-WITH-SCHEMA-PASSWORD
   - DB sys user password : REPLACE-WITH-SYS-PASSWORD
   - Domain name : soainfra

- Domain Namespace : soans
- Secret name : soainfra-rcu-credentials

```
cd ${WORKDIR}/create-rcu-credentials
./create-rcu-credentials.sh \
      -u SOA1 \
      -p REPLACE-WITH-SCHEMA-PASSWORD \
      -a sys \
      -q REPLACE-WITH-SYS-PASSWORD \
      -d soainfra \
      -n soans \
      -s soainfra-rcu-credentials
```

Now the environment is ready to start the Oracle SOA Suite domain creation.

2. Create an Oracle SOA Suite domain.
   a. The sample scripts for Oracle SOA Suite domain deployment are available at `${WORKDIR}/create-soa-domain/domain-home-on-pv`. You must edit `create-domain-inputs.yaml` (or a copy of it) to provide the details for your domain.

      Update `create-domain-inputs.yaml` with the following values for domain creation:

      - `domainType: soaosb`
      - `initialManagedServerReplicas:1`

```
cd ${WORKDIR}/create-soa-domain/domain-home-on-pv/

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

sed -i -e "s:domainType\: soa:domainType\: soaosb:g" create-domain-
inputs.yaml
sed -i -e "s:initialManagedServerReplicas\:
2:initialManagedServerReplicas\: 1:g" create-domain-inputs.yaml
sed -i -e "s:image\: soasuite\:release-version:image\: container-
registry.oracle.com/middleware/soasuite\:14.1.2.0-<17 or 21>-<ol8 or
ol9>-<tag>:g" create-domain-inputs.yaml
```

   b. Run the `create-domain.sh` script to create a domain:

```
cd ${WORKDIR}/create-soa-domain/domain-home-on-pv/
./create-domain.sh -i create-domain-inputs.yaml -o output
```

   c. Create a Kubernetes domain object:

      Once the `create-domain.sh` is successful, it generates `output/weblogic-domains/soainfra/domain.yaml`, which you can use to create the Kubernetes resource domain to start the domain and servers:

```
cd ${WORKDIR}/create-soa-domain/domain-home-on-pv
kubectl create -f output/weblogic-domains/soainfra/domain.yaml
```

d. Verify that the Kubernetes domain object named `soainfra` is created:

```
kubectl get domain -n soans
NAME       AGE
soainfra   3m18s
```

e. Once you create the domain, the *introspect* pod is created. This inspects the domain home and then starts the `soainfra-adminserver` pod. Once the `soainfra-adminserver` pod starts successfully, the Managed Server pods are started in parallel. Watch the `soans` namespace for the status of domain creation:

```
kubectl get pods -n soans -w
```

f. Verify that the Oracle SOA Suite domain server pods and services are created and in Ready state:

```
kubectl get all -n soans
```

3. Configure Traefik to access Oracle SOA Suite domain services.

   a. Configure Traefik to manage ingresses created in the Oracle SOA Suite domain namespace (`soans`):

```
helm upgrade traefik traefik/traefik \
  --reuse-values \
  --namespace traefik \
  --set "kubernetes.namespaces={traefik,soans}" \
  --wait
```

   b. Create an ingress for the domain in the domain namespace by using the sample Helm chart:

```
cd ${WORKDIR}
export LOADBALANCER_HOSTNAME=$(hostname -f)
helm install soa-traefik-ingress charts/ingress-per-domain \
--namespace soans \
--values charts/ingress-per-domain/values.yaml \
--set "traefik.hostname=${LOADBALANCER_HOSTNAME}" \
--set domainType=soaosb
```

   c. Verify the created ingress per domain details:

```
kubectl describe ingress soainfra-traefik -n soans
```

4. Verify that you can access the Oracle SOA Suite domain URL.

   a. Get the `LOADBALANCER_HOSTNAME` for your environment:

```
export LOADBALANCER_HOSTNAME=$(hostname -f)
```

   b. Verify the following URLs are available for Oracle SOA Suite domains of domain type `soaosb`:

   Credentials

   username: weblogic

password: Welcome1

```
http://${LOADBALANCER_HOSTNAME}:30305/em
http://${LOADBALANCER_HOSTNAME}:30305/servicebus
http://${LOADBALANCER_HOSTNAME}:30305/soa-infra
http://${LOADBALANCER_HOSTNAME}:30305/soa/composer
http://${LOADBALANCER_HOSTNAME}:30305/integration/worklistapp
http://${LOADBALANCER_HOSTNAME}:30305/ess
http://${LOADBALANCER_HOSTNAME}:30305/EssHealthCheck
```

Security Hardening

This section provides references on how to securely configure Docker and Kubernetes.

Securing a Kubernetes cluster involves hardening on multiple fronts - securing the API servers, etcd, nodes, container images, container run-time, and the cluster network. Apply principles of defense in depth, principle of least privilege, and minimize the attack surface. Use security tools such as Kube-Bench to verify the cluster's security posture. Since Kubernetes is evolving rapidly refer to Kubernetes Security Overview for the latest information on securing a Kubernetes cluster. Also ensure the deployed Docker containers follow the Docker Security guidance.

References

1. Docker hardening
   - https://docs.docker.com/engine/security/security/
   - https://blog.aquasec.com/docker-security-best-practices
2. Kubernetes hardening
   - https://kubernetes.io/docs/concepts/security/overview/
   - https://kubernetes.io/docs/concepts/security/pod-security-standards/
   - https://blogs.oracle.com/developers/5-best-practices-for-kubernetes-security
3. Security best practices for Oracle WebLogic Server Running in Docker and Kubernetes
   - https://blogs.oracle.com/weblogicserver/security-best-practices-for-weblogic-server-running-in-docker-and-kubernetes

12 Troubleshooting

This document describes common issues that may occur during the deployment of Oracle SOA Suite on Kubernetes and the steps to troubleshoot them. Also refer to the FAQs page for frequent issues and steps to resolve them.

- WebLogic Kubernetes Operator installation failure
- RCU schema creation failure
- Domain creation failure
- Common domain creation issues
- Server pods not started after applying domain configuration file
- Ingress controller not serving the domain URLs
- Security warnings reported in WebLogic Remote console
- Disable Remote Anonymous RMI T3 and IIOP Requests

WebLogic Kubernetes Operator installation failure

If the WebLogic Kubernetes Operator installation failed with timing out:

- Check the status of the operator Helm release using the command `helm ls -n <operator-namespace>`.
- Check if the operator pod is successfully created in the operator namespace.
- Describe the operator pod using `kubectl describe pod <operator-pod-name> -n <operator-namespace>` to identify any obvious errors.

RCU schema creation failure

When creating the RCU schema using `create-rcu-schema.sh`, the possible causes for RCU schema creation failure are:

- Database is not up and running
- Incorrect database connection URL used
- Invalid database credentials used
- Schema prefix already exists

Make sure that all the above causes are reviewed and corrected as needed.

Also drop the existing schema with the same prefix before rerunning the `create-rcu-schema.sh` with correct values.

Domain creation failure

If the Oracle SOA Suite domain creation fails when running `create-domain.sh`, perform the following steps to diagnose the issue:

1. Run the following command to diagnose the create domain job:

   ```
   $ kubectl logs jobs/<domain_job> -n <domain_namespace>
   ```

For example:

```
$ kubectl logs jobs/soainfra-create-soa-infra-domain-job -n soans
```

Also run:

```
$ kubectl describe pod <domain_job> -n <domain_namespace>
```

Use the output to diagnose the problem and resolve the issue.

2. Clean up the failed domain creation:
   a. Delete the failed domain creation job in the domain namespace using the command:

      ```
      kubectl delete job <domain-creation-job-name> -n
          <domain-namespace>
      ```

      .
   b. Delete the contents of the domain home directory.
   c. Drop the existing RCU schema.
3. Recreate the domain:
   a. Recreate the RCU schema
   b. Make sure the Persistent Volume and Persistent Volume Claim used for the domain are created with correct permissions and bound together.
   c. Rerun the create domain script

Common domain creation issues

A common domain creation issue is error

```
Failed to build JDBC Connection object
```

in the create domain job logs.

Stack Trace

```
Configuring the Service Table DataSource...
fmwDatabase  jdbc:oracle:thin:@orclcdb.soainfra-domain-ns-293-10202010:1521/
orclpdb1
Getting Database Defaults...
Error: getDatabaseDefaults() failed. Do dumpStack() to see details.
Error: runCmd() failed. Do dumpStack() to see details.
Problem invoking WLST - Traceback (innermost last):
File "/u01/weblogic/..2021_10_20_20_29_37.256759996/createSOADomain.py", line
943, in ?
File "/u01/weblogic/..2021_10_20_20_29_37.256759996/createSOADomain.py", line
75, in createSOADomain
File "/u01/weblogic/..2021_10_20_20_29_37.256759996/createSOADomain.py", line
695, in extendSoaB2BDomain
File "/u01/weblogic/..2021_10_20_20_29_37.256759996/createSOADomain.py", line
588, in configureJDBCTemplates
File "/tmp/WLSTOfflineIni956349269221112379.py", line 267, in
getDatabaseDefaults
File "/tmp/WLSTOfflineIni956349269221112379.py", line 19, in command
Failed to build JDBC Connection object:
   at
```

```
com.oracle.cie.domain.script.jython.CommandExceptionHandler.handleException(Co
mmandExceptionHandler.java:69)
   at
com.oracle.cie.domain.script.jython.WLScriptContext.handleException(WLScriptCo
ntext.java:3085)
   at
com.oracle.cie.domain.script.jython.WLScriptContext.runCmd(WLScriptContext.jav
a:738)
   at sun.reflect.GeneratedMethodAccessor152.invoke(Unknown Source)
   at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.j
ava:43)
   at java.lang.reflect.Method.invoke(Method.java:498)
com.oracle.cie.domain.script.jython.WLSTException:
com.oracle.cie.domain.script.jython.WLSTException: Got exception when auto
configuring the schema component(s) with data obtained from shadow table:
Failed to build JDBC Connection object:
ERROR: /u01/weblogic/create-domain-script.sh failed.
```

This error is reported when there is an issue with database schema access during domain creation. The possible causes are:

- Incorrect schema name specified in

  ```
  create-domain-inputs.yaml
  ```

  .
- RCU schema credentials specified in the secret

  ```
  soainfra-rcu-credentials
  ```

  are different from the credentials specified while creating the RCU schema using

  ```
  create-rcu-schema.sh
  ```

  .

To resolve these possible causes, check that the schema name and credentials used during the domain creation are the same as when the RCU schema was created.

Server pods not started after applying domain configuration file

When a domain configuration file (YAML) is deployed and no introspector or server pods are initiated, as well as there is no mention of the domain in the operator log, ensure that the domain's namespace has been configured to be managed by WebLogic Kubernetes Operator.

See Namespace management :: WebLogic Kubernetes Operator for details.

The default value is `LabelSelector`. The operator will manage namespaces with Kubernetes labels that match the label selector defined by your Helm chart configuration `domainNamespaceLabelSelector` attribute, which defaults to `weblogic-operator=enabled`.

Verify if the label `weblogic-operator=enabled` is specified for domain namespace that is to be managed by the operator, by running the following command:

```
$ kubectl get ns --selector="weblogic-operator=enabled"
```

For example, if your domain namespace is `soans` and the preceding command did not list the `soans` namespace, then execute the following command for operator to manage the domain namespace:

```
$ kubectl label namespace soans weblogic-operator=enabled
```

Ingress controller not serving the domain URLs

To diagnose this issue:

1. Verify that the Ingress controller is installed successfully. For example, to verify the Traefik Ingress controller status, run the following command:

   ```
   $ helm list -n traefik
   NAME                    NAMESPACE       REVISION
   UPDATED                                 STATUS
   CHART                APP VERSION
   traefik                  traefik         2                2022-11-30
   11:31:18.599876918 +0000 UTC deployed        traefik-20.5.3       v2.9.5
   $
   ```

2. Verify that the Ingress controller is setup to monitor the domain namespace. For example, to verify the Traefik Ingress controller manages the soans domain namespace, run the following command and check the values under namespaces section.

   ```
   $ helm get values traefik-operator -n traefik
   USER-SUPPLIED VALUES:
   kubernetes:
      namespaces:
      - traefik
      - soans
   $
   ```

3. Verify that the Ingress chart is installed correctly in domain namespace. For example, run the following command:

   ```
   $ helm list -n soans
   NAME                    NAMESPACE       REVISION
   UPDATED                                 STATUS
   CHART                          APP VERSION
   soainfra-traefik        soans           1               2021-10-27
   11:24:31.7572791 +0000 UTC   deployed        ingress-per-domain-0.1.0
   1.0
   $
   ```

4. Verify that the Ingress URL paths and hostnames are configured correctly by running the following commands:

See the following to see the sample commands and output:

```
$ kubectl get ingress soainfra-traefik -n soans
NAME               CLASS
HOSTS                                                      ADDRESS
PORTS   AGE
soainfra-traefik   <none>   <Hostname>             80      20h
$
$ kubectl describe ingress soainfra-traefik -n soans
Name:             soainfra-traefik
Namespace:        soans
Address:
Default backend:  default-http-backend:80 (<error: endpoints "default-http-
backend" not found>)
Rules:
Host                                                      Path  Backends
----                                                      ----  --------
<Hostname>
                                                          /
console                   soainfra-adminserver:7001 (10.244.0.123:7001)
                                                          /em
         soainfra-adminserver:7001 (10.244.0.123:7001)
                                                          /weblogic/
ready            soainfra-adminserver:7001 (10.244.0.123:7001)
                                                          /soa-
infra                 soainfra-cluster-soa-cluster:8001
(10.244.0.126:8001,10.244.0.127:8001)
                                                          /soa/
composer              soainfra-cluster-soa-cluster:8001
(10.244.0.126:8001,10.244.0.127:8001)
                                                          /integration/
worklistapp   soainfra-cluster-soa-cluster:8001
(10.244.0.126:8001,10.244.0.127:8001)
                                                          /
EssHealthCheck            soainfra-cluster-soa-cluster:8001
(10.244.0.126:8001,10.244.0.127:8001)
Annotations:                                                kubernetes.io/
ingress.class: traefik
                                                          meta.helm.sh/
release-name: soainfra-traefik
                                                          meta.helm.sh/
release-namespace: soans
Events:                                                     <none>
$
```

Security warnings reported in WebLogic Remote console

WebLogic Server regularly validates your domain configuration settings against a set of security configuration guidelines to determine whether the domain meets key security guidelines recommended by Oracle. If your domain does not meet a recommendation for a security configuration setting, a warning is logged in the Security Warnings Report in the WebLogic Remote Console.

See Review Potential Security Issues in Securing a Production Environment for Oracle WebLogic Server for more information.

Disable Remote Anonymous RMI T3 and IIOP Requests

If you see security warning message that `Remote Anonymous RMI T3 or IIOP requests are enabled`, resolve this warning by setting `RemoteAnonymousRMIT3Enabled` and `RemoteAnonymousRMIIIOPEnabled` attributes to false in `domain.yaml` with `JAVA_OPTIONS` before starting the domain as shown below:

```
serverPod:
    # an (optional) list of environment variable to be set on the servers
    env:
    - name: JAVA_OPTIONS
      value: "-Dweblogic.StdoutDebugEnabled=false -Dweblogic.ssl.Enabled=true
-Dweblogic.security.SSL.ignoreHostnameVerification=true -
Dweblogic.security.remoteAnonymousRMIT3Enabled=false -
Dweblogic.security.remoteAnonymousRMIIIOPEnabled=false"
```

See link for more details.