Oracle® Communications ASAP

Cloud Native Deployment Guide

Release 7.4
F40784-03
April 2023

Oracle Communications ASAP Cloud Native Deployment Guide, Release 7.4

F40784-03

Contents

Preface

This guide describes how to install and administer Oracle Communications ASAP Cloud Native Deployment Option.

Audience

This document is intended for DevOps administrators and those involved in installing and maintaining Oracle Communications ASAP Cloud Native Deployment.

Documentation Accessibility

Diversity and Inclusion

1

Overview of the ASAP Cloud Native Deployment

Get an overview of Oracle Communications ASAP cloud native deployment, architecture, and the ASAP cloud native toolkit.

This chapter provides an overview of Oracle Communications ASAP deployed in a cloud native environment using container images and a Kubernetes cluster.

About the ASAP Cloud Native Deployment

You set up your own cloud native environment and can then use the ASAP cloud native toolkit to automate the deployment of ASAP instances. By leveraging the pre-configured Helm charts, you can deploy ASAP instances quickly ensuring your services are up and running in far less time than a traditional deployment. However, there are a few differences between traditional and cloud native deployments. For more information, see "Differences Between ASAP Cloud Native and ASAP Traditional Deployments".

ASAP cloud native supports the following deployment models:

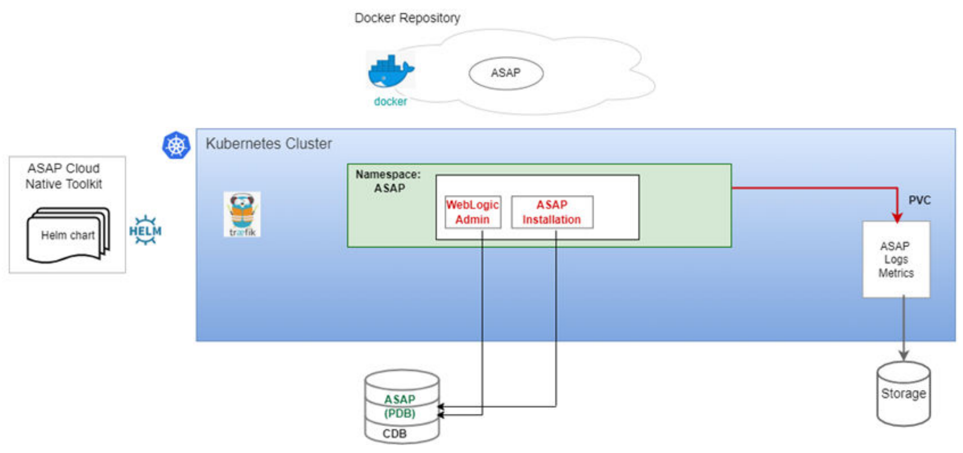
- **On Private Kubernetes Cluster**: ASAP cloud native is certified for a general deployment of Kubernetes.
- **On Oracle Cloud Infrastructure Container Engine for Kubernetes (OKE)**: ASAP cloud native is certified to run on Oracle's hosted Kubernetes OKE service.

ASAP Cloud Native Architecture

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

The following diagram illustrates the ASAP cloud native architecture.

Figure 1-1 ASAP Cloud Native Architecture



The ASAP cloud native architecture requires components such as the Kubernetes cluster. The WebLogic domain is static in the Kubernetes cluster. For any modifications, you should update the Docker image and redeploy it. The ASAP cloud native artifacts include a container image built using Docker and the ASAP cloud native toolkit.

Downloading the ASAP Cloud Native Artifacts

To deploy ASAP and Order Balancer cloud native instances in the Kubernetes cluster:

1. Download the cloud native tar file, for example **ASAP.R7_4_0_P***x***.B***version***.cn.tar** from Oracle Software Delivery Cloud or from My Oracle Support Website.

   Note:

   The cloud native tar file is present in the Linux platform only.

   Where *x* is the patch set version of the ASAP cloud native.

   version is the release version of the ASAP cloud native.
2. Copy the tar file to the machine where Docker and Kubernetes are installed.
3. Extract the tar file using the following command:

   ```
   tar -xvf ASAP.R7_4_0_Px.Bversion.cn.tar
   ```

   The artifacts in the tar file are extracted to the same directory where you ran the command.

   The following zip files are extracted:

   - asap-img-builder.zip: To build ASAP Docker image and Order Balancer Docker image.
   - asap-cntk.zip: To create ASAP instance.
   - ob-cntk.zip: To create Order Balancer instance.

About the ASAP Docker Image Toolkit

The ASAP Docker image builder toolkit (asap-img-builder.zip) creates an image with a base image as Linux 8 and installs prerequisite packages, Java, WebLogic Server, database client, and ASAP. The toolkit contains scripts to install the required packages and installers.

The image building process consists of manual steps. ASAP database is not part of the Docker image. The database should be accessible from Docker host machine and the Kubernetes cluster.

About ASAP Instance

ASAP instance in the Kubernetes cluster includes deployment, service, and Ingress route. The Docker image is created using the image builder toolkit, which deploys the ASAP instance in the Kubernetes cluster by using the ASAP cloud native toolkit.

Once the instance is up, you should not perform any configuration changes such as deploying a new cartridge, changing configuration parameters, and so on. If you want to perform such changes, update the Docker image and redeploy the instance.

About the ASAP Cloud Native Toolkit

The ASAP cloud native toolkit (asap-cntk.zip) is an archive file that includes utility scripts and samples to deploy ASAP in a cloud native environment.

Contents of the ASAP Cloud Native Toolkit

The ASAP cloud native toolkit contains the following artifacts:

- Helm charts for ASAP:
  - The Helm chart for ASAP is located in the **asap_cntk/charts/asap** directory.
- Scripts to manage the lifecycle of an ASAP instance

About Helm Overrides

The specifications of the ASAP deployment are consumed from the **values.yaml** file. The values defined in the **values.yaml** file are used by deployment, service, and ingress yaml files.

Update the values accordingly to create multiple instances.

2

Planning and Validating Your Cloud Environment

In preparation for Oracle Communications ASAP cloud native deployment, you must set up and validate prerequisite software. This chapter provides information about planning, setting up, and validating the environment for ASAP cloud native deployment.

See the following topics:

- Required Components for ASAP and Order Balancer Cloud Native
- Planning Your Cloud Native Environment
- Setting Up Persistent Storage
- Planning Your Container Engine for Kubernetes (OKE) Cloud Environment
- Validating Your Cloud Environment

If you are already familiar with traditional ASAP, for important information on the differences introduced by ASAP cloud native, see "Differences Between ASAP Cloud Native and ASAP Traditional Deployments".

Required Components for ASAP and Order Balancer Cloud Native

To run, manage, and monitor the ASAP and Order Balancer cloud native deployment, the following components and capabilities are required. These must be configured in the cloud environment:

- Kubernetes Cluster
- Container Image Management
- Helm
- Load Balancer
- Domain Name System (DNS)
- Persistent Volumes
- Secrets Management
- Kubernetes Monitoring Toolchain
- Application Logs and Metrics Toolchain

For details about the required versions of these components, see *ASAP Compatibility Matrix*.

Planning Your Cloud Native Environment

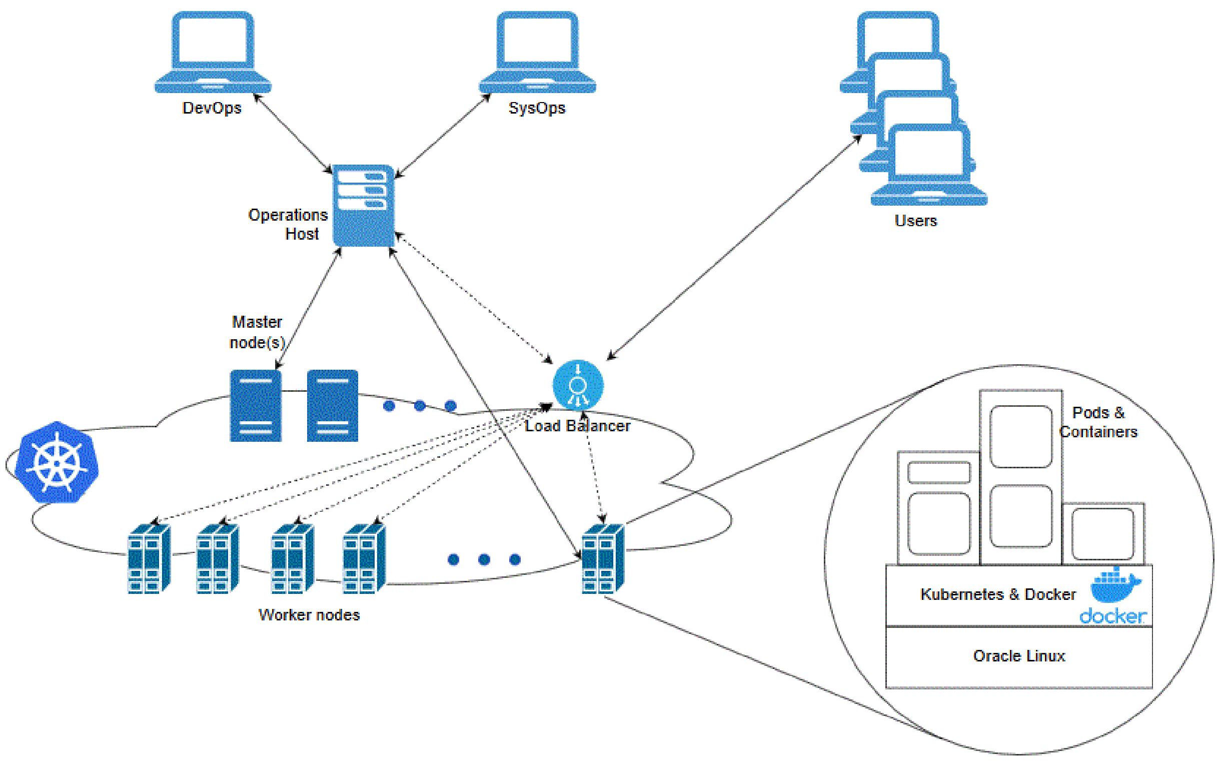
This section provides information about planning and setting up an ASAP cloud native environment. As part of preparing your environment for ASAP cloud native, you choose, install, and set up various components and services in ways that are best suited for your cloud native environment. The following sections provide information about each of those required components and services, the available options that you can choose from, and the way you must set them up for your ASAP cloud native environment.

Setting Up Your Kubernetes Cluster

For ASAP cloud native, Kubernetes worker nodes must be capable of running Linux 8.x pods with software compiled for Intel 64-bit cores. A reliable cluster must have multiple worker nodes spread over separate physical infrastructure and a very reliable cluster must have multiple master nodes spread over separate physical infrastructure.

The following diagram illustrates the Kubernetes cluster and the components that it interacts with.

Figure 2-1 Kubernetes Cluster



ASAP cloud native requires:

- Kubernetes
  To check the version, run the following command:

  ```
  kubectl version
  ```

- Flannel

  To check the version, run the following command on the master node running the kube-flannel pod:

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

- Docker
  To check the version, run the following command:

  ```
  docker version
  ```

Typically, Kubernetes nodes are not used directly to run or monitor Kubernetes workloads. You must reserve worker node resources for the execution of the Kubernetes workload. However, multiple users (manual and automated) of the cluster require a point from which to access the cluster and operate on it. This can be achieved by using `kubectl` commands (either directly on the command line and shell scripts or through Helm) or Kubernetes APIs. For this purpose, set aside a separate host or set of hosts. Operational and administrative access to the Kubernetes cluster can be restricted to these hosts and specific users can be given named accounts on these hosts to reduce cluster exposure and promote traceability of actions.

In addition, you need the appropriate tools to connect to your overall environment, including the Kubernetes cluster. For instance, for a Container Engine for Kubernetes (OKE) cluster, you must install and configure the Oracle Cloud Infrastructure Command Line Interface.

Additional integrations may need to include appropriate NFS mounts for home directories, security lists, firewall configuration for access to the overall environment, and so on.

Kubernetes worker nodes should be configured with the recommended operating system kernel parameters listed in "Configuring a UNIX ASAP Group and User" in *ASAP Installation Guide*. Use the documented values as the minimum values to set for each parameter. Ensure that Linux OS kernel parameter configuration is persistent, so as to survive a reboot.

The ASAP cloud native instance, for which specification files are provided with the toolkit for large systems, requires up to 16 GB of RAM and 2 CPUs, in terms of Kubernetes worker node capacity. For more details about database sizes, see "ASAP Server Hardware Requirements" in *ASAP Installation Guide*. A small increment is needed for Traefik. Refer to those projects for details.

Synchronizing Time Across Servers

It is important that you synchronize the date and time across all machines that are involved in testing, including client test drivers and Kubernetes worker nodes. Oracle recommends that you do this using Network Time Protocol (NTP), rather than manual synchronization, and strongly recommends it for Production environments. Synchronization is important in inter-component communications and in capturing accurate run-time statistics.

About Container Image Management

An ASAP cloud native deployment generates a container image for ASAP. Additionally, the image is downloaded for Traefik (depending on the choice of Ingress controllers).

Oracle highly recommends that you create a private container repository and ensure that all nodes have access to that repository. The image is saved in this repository and all nodes would then have access to the repository. This may require networking changes (such as routes and proxy) and include authentication for logging in to the repository. Oracle recommends that you choose a repository that provides centralized storage and management for the container image.

Failing to ensure that all nodes have access to a centralized repository will mean that image has to be synced to the hosts manually or through custom mechanisms (for example, using scripts), which are error-prone operations as worker nodes are commissioned, decommissioned, or even rebooted. When an image on a particular worker node is not available, the pods using that image are either not scheduled to that node, wasting resources, or fail on that node. If image names and tags are kept constant (such as myapp:latest), the pod may pick up a pre-existing image of the same name and tag, leading to unexpected and hard to debug behaviors.

Installing Helm

ASAP cloud native requires Helm, which delivers reliability, productivity, consistency, and ease of use.

In an ASAP cloud native environment, using Helm enables you to achieve the following:

- You can apply custom domain configuration by using a single and consistent mechanism, which leads to an increase in productivity. You no longer need to apply configuration changes through multiple interfaces such as WebLogic Console, WLST, and WebLogic Server MBeans.
- Changing the ASAP domain configuration in the traditional installations is a manual and multi-step process that may lead to errors. This can be eliminated with Helm because of the following features:
  - Helm Lint allows pre-validation of syntax issues before changes are applied
  - Multiple changes can be pushed to the running instance with a single upgrade command
  - Configuration changes may map to updates across multiple Kubernetes resources (such as domain resources, config maps, and so on). With Helm, you merely update the Helm release and its responsibility to determine which Kubernetes resources are affected.
- Including configuration in Helm charts allows the content to be managed as code, through source control, which is a fundamental principle of modern DevOps practices.

To co-exist with older Helm versions in production environments, ASAP requires Helm 3.1.3 or later saved as **helm** in PATH.

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

```
$ cd some-tmp-dir
$ wget https://get.helm.sh/helm-v3.4.1-linux-amd64.tar.gz
$ tar -zxvf helm-v3.4.1-linux-amd64.tar.gz

# Find the helm binary in the unpacked directory and move it to its
desired destination. You need root user.
$ sudo mv linux-amd64/helm /usr/local/bin/helm

# Optional: If access to the deprecated Helm repository "stable" is
required, uncomment and run
```

```
# helm repo add stable https://charts.helm.sh/stable

# verify Helm version
$ helm version
version.BuildInfo{Version:"v3.4.1",
GitCommit:"c4e74854886b2efe3321e185578e6db9be0a6e29", GitTreeState:"clean",
GoVersion:"go1.14.11"}
```

Helm leverages **kubeconfig** for users running the `helm` command to access the Kubernetes cluster. By default, this is $HOME/**.kube/config**. Helm inherits the permissions set up for this access into the cluster. You must ensure that if RBAC is configured, then sufficient cluster permissions are granted to users running Helm.

About Load Balancing and Ingress Controller

ASAP cloud native instance is running in Kubernetes. To access application endpoints, you must enable HTTP/S connectivity to the cluster through an appropriate mechanism. This mechanism must be able to route traffic to the ASAP cloud native instance in the Kubernetes cluster.

For ASAP cloud native, an ingress controller is required to expose appropriate services from the ASAP cluster and direct traffic appropriately to the cluster members. An external load balancer is an optional add-on.

Note:

ASAP does not support multiple replicas. However, if you do not want to expose Kubernetes node IP addresses to users, use a load balancer.

The ingress controller monitors the ingress objects created by the ASAP cloud native deployment, and acts on the configuration embedded in these objects to expose ASAP HTTP and HTTPS services to the external network. This is achieved using NodePort services exposed by the ingress controller.

The ingress controller must support:

- Sticky routing (based on standard session cookie)
- SSL termination and injecting headers into incoming traffic

Examples of such ingress controllers include Traefik, Voyager, and Nginx. The ASAP cloud native toolkit provides samples and documentation that use Traefik as the ingress controller.

An external load balancer serves to provide a highly reliable single-point access into the services exposed by the Kubernetes cluster. In this case, this would be the NodePort services exposed by the ingress controller on behalf of the ASAP cloud native instance. Using a load balancer removes the need to expose Kubernetes node IPs to the larger user base, and insulates the users from changes (in terms of nodes appearing or being decommissioned) to the Kubernetes cluster. It also serves to enforce access policies. The ASAP cloud native toolkit includes samples and documentation that show integration with Oracle Cloud Infrastructure LBaaS when Oracle OKE is used as the Kubernetes environment.

Using Traefik as the Ingress Controller

If you choose to use Traefik as the ingress controller, the Kubernetes environment must have the Traefik ingress controller installed and configured.

For more information about installing and configuring Traefik ingress controller, see "Installing the Traefik Container Image".

For details about the required version of Traefik, see *ASAP Compatibility Matrix*.

Using Domain Name System (DNS)

A Kubernetes cluster can have many routable entry points. Common choices are:

- External load balancer (IP and port)
- Ingress controller service (master node IPs and ingress port)
- Ingress controller service (worker node IPs and ingress port)

You must identify the proper entry point for your Kubernetes cluster.

ASAP cloud native requires hostnames to be mapped to routable entrypoints into the Kubernetes cluster. Regardless of the actual entry points (external load balancer, Kubernetes master node, or worker nodes), users who need to communicate with the ASAP cloud native instances require name resolution.

The access hostnames take the *prefix*.*domain* form. *prefix* and *domain* are determined by the specifications of the ASAP cloud native configuration for a given deployment. *prefix* is unique to the deployment, while *domain* is common for multiple deployments.

The default *domain* in ASAP cloud native toolkit is `asap.org`.

For a particular deployment, as an example, this results in the following addresses:

- `dev1.wireless.asap.org` (for HTTP access)
- `admin.dev1.wireless.asap.org` (for WebLogic Console access)

These "hostnames" must be routable to the entry point of your Ingress Controller or Load Balancer. For a basic validation, on the systems that access the deployment, edit the local hosts file to add the following entry:

Note:

The hosts file is located in **/etc/hosts** on Linux and MacOS machines and in **C:\Windows\System32\drivers\etc\hosts** on Windows machines.

```
ip_address  dev1.wireless.asap.org   admin.dev1.wireless.asap.org
t3.dev1.wireless.asap.org
```

However, the solution of editing the hosts file is not easy to scale and coordinate across multiple users and multiple access environments. A better solution is to leverage DNS services at the enterprise level.

With DNS servers, a more efficient mechanism can be adopted. The mechanism is the creation of a domain level A-record:

```
A-Record: *.asap.org IP_address
```

If the target is not a load balancer, but the Kubernetes cluster nodes themselves, a DNS service can also insulate the user from relying on any single node IP. The DNS entry can be configured to map **.asap.org* to all the current Kubernetes cluster node IP addresses. You must update this mapping as the Kubernetes cluster changes with adding a new node, removing an old node, reassigning the IP address of a node, and so on.

With these two approaches, you can set up an enterprise DNS once and modify it only infrequently.

Configuring Kubernetes Persistent Volumes

Typically, runtime artifacts in ASAP cloud native are created within the respective pod filesystems. As a result, they are lost when the pod is deleted. These artifacts include application logs and WebLogic Server logs.

While this impermanence may be acceptable for highly transient environments, it is typically desirable to have access to these artifacts outside of the lifecycle of the ASAP could native instance. It is also highly recommended to deploy a toolchain for logs to provide a centralized view with a dashboard. To allow for artifacts to survive independent of the pod, ASAP cloud native allows for them to be maintained on Kubernetes Persistent Volumes.

ASAP cloud native does not dictate the technology that supports Persistent Volumes but provides samples for NFS-based persistence. Additionally, for ASAP cloud native on an Oracle OKE cloud, you can use persistence based on File Storage Service (FSS).

Regardless of the persistence provider chosen, persistent volumes for ASAP cloud native use must be configured:

- With accessMode ReadWriteMany
- With a capacity to support the intended workload

Log size and retention policies can be configured as part of the shape specification.

About NFS-based Persistence

For use with ASAP cloud native, one or more NFS (Network File System) servers must be designated.

It is highly recommended to split the servers as follows:

- At least one for the development instance and the non-sensitive test instance (for example, for Integration testing)
- At least one for the sensitive test instance (for example, for Performance testing, Stress testing, and production staging)
- One for the production instance

In general, ensure that the sensitive instances have dedicated NFS support, so that they do not compete for disk space or network IOPS with others.

The exported filesystems must have enough capacity to support the intended workload. Given the dynamic nature of the ASAP cloud native instances, and the fact that the ASAP logging volume is highly dependent on cartridges and on the order volume, it is prudent to put in place a set of operational mechanisms to:

- Monitor disk usage and warn when the usage crosses a threshold
- Clean out the artifacts that are no longer needed

If a toolchain such as ELK Stack picks up this data, then the cleanup task can be built into this process itself. As artifacts are successfully populated into the toolchain, they can be deleted from the filesystem. You must take care to only delete log files that have rolled over.

Using Kubernetes Monitoring Toolchain

A multi-node Kubernetes cluster with multiple users and an ever-changing workload require a capable set of tools to monitor and manage the cluster. There are tools that provide data, rich visualizations, and other capabilities such as alerts. ASAP cloud native does not require any particular system to be used but recommends using such a monitoring, visualization, and alerting capability.

For ASAP cloud native, the key aspects of monitoring are:

- Worker capacity in CPU and memory. The pods take up a non-trivial amount of worker resources. For example, pods configured for production performance use 32 GB of memory.
- Worker node disk pressure
- Worker node network pressure
- The health of the core Kubernetes services
- The health of WebLogic Kubernetes Operator
- The health of Traefik (or other load balancers in the cluster)

The name spaces and pods that ASAP cloud native uses provide a cross instance view of ASAP cloud native.

About Application Logs and Metrics Toolchain

ASAP cloud native generates all logs that traditional ASAP and WebLogic Server typically generate. The logs can be sent to a shared filesystem for retention and for retrieval by a toolchain such as Elastic Stack.

In addition, ASAP cloud native generates metrics. ASAP cloud native exposes metrics for scraping by Prometheus. These can then be processed by a metrics toolchain, with visualizations like Grafana dashboards. Dashboards and alerts can be configured to enable sustainable monitoring of multiple ASAP cloud native instances throughout their lifecycles. Performance metrics include heap utilization, threads stuck, garbage collection, and so on.

Oracle highly recommends using a toolchain to effectively monitor ASAP cloud native instance. The dynamic lifecycle in ASAP cloud native, in terms of deploying, scaling and updating an instance, requires proper monitoring and management of the database resources as well. For non-sensitive environments such as development instances and some test instances, this largely implies monitoring the tablespace usage and the disk usage, and adding disk space as needed.

Setting Up Persistent Storage

ASAP and Order Balancer cloud native can be configured to use a Kubernetes Persistent Volume to store data that needs to be retained even after a pod is terminated. This data includes application logs and WebLogic Server logs. When an instance is re-created, the same persistent volume need not be available. When persistent storage is configured in the Docker image, these data files, which are written inside a pod are re-directed to the persistent volume.

Data from all ASAP and Order Balancer instances may be persisted, but each instance does not need a unique location for logging. Data is written to an *asap-instance* folder or *ob-instance* folder, so multiple instances can share the same end location without destroying data from other instances.

The final location for this data should be one that is directly visible to the users of ASAP and Order Balancer cloud native. The development instances may direct data to a shared file system for analysis and debugging by cartridge developers. Whereas formal test and production instances may need the data to be scraped by a logging toolchain such as EFK, which can then process the data and make it available in various forms. The recommendation, therefore, is to create a PV-PVC pair for each class of destination within a project. In this example, one for developers to access and one that feeds into a toolchain.

A PV-PVC pair would be created for each of these "destinations", that multiple instances can then share. A single PVC can be used by multiple ASAP and Order Balancer instances. The management of the PV (Persistent Volume) and PVC (Persistent Volume Claim) lifecycles is beyond the scope of ASAP and Order Balancer cloud native.

The ASAP and Order Balancer cloud native infrastructure administrator is responsible for creating and deleting PVs or for setting up dynamic volume provisioning.

The ASAP and Order Balancer cloud native project administrator is responsible for creating and deleting PVCs as per the standard documentation in a manner such that they consume the pre-created PVs or trigger the dynamic volume provisioning. The specific technology supporting the PV is also beyond the scope of ASAP and Order Balancer cloud native. However, samples for PV supported by NFS are provided.

Planning Your Container Engine for Kubernetes (OKE) Cloud Environment

This section provides information about planning your cloud environment if you want to use Oracle Cloud Infrastructure Container Engine for Kubernetes (OKE) for ASAP cloud native. Some of the components, services, and capabilities that are required and recommended for a cloud native environment are applicable to the Oracle OKE cloud environment as well.

- **Kubernetes** and **Container Images**: You can choose from the version options available in OKE as long as the selected version conforms to the range described in the section about planning cloud native environment.
- **Container Image Management**: ASAP cloud native recommends using Oracle Cloud Infrastructure Registry with OKE. Any other repository that you use must be able to serve images to the OKE environment in a quick and reliable manner. The ASAP cloud native image is of the order of 5 GB each.

- **Oracle Multitenant Database**: It is strongly recommended to run Oracle DB outside of OKE, but within the same Oracle Cloud Infrastructure tenancy and the region as an Oracle DB service (BareMetal, VM, or ExaData). The database version should be 19c. You can choose between a standalone DB or a multi-node RAC.
- **Helm**: Install Helm as described for the cloud native environment into the OKE cluster.
- **Persistent Volumes**: Use NFS-based persistence. ASAP cloud native recommends the use of Oracle Cloud Infrastructure File Storage service in the OKE context.
- **Monitoring Toolchains**: While the Oracle Cloud Infrastructure Console provides a view of the resources in the OKE cluster, it also enables you to use the Kubernetes Dashboard. Any additional monitoring capability must be built up.

Compute Disk Space Requirements

Given the size of the ASAP cloud native container image (approximately 5 GB), the size of the ASAP cloud native containers, and the volume of the ASAP logs generated, it is recommended that the OKE worker nodes have at least 40 GB of free space that the /**var/lib** filesystem can use. Add disk space if the worker nodes do not have the recommended free space in the /**var/lib** filesystem.

Work with your Oracle Cloud Infrastructure OKE administrator to ensure worker nodes have enough disk space. Common options are to use Compute shapes with larger boot volumes or to mount an Oracle Cloud Infrastructure Block Volume to **/var/lib/docker**.

Note:

The reference to logs in this section applies to the container logs and other infrastructure logs. The space considerations still apply even if the ASAP cloud native logs are being sent to an NFS Persistent Volume.

Connectivity Requirements

ASAP cloud native assumes the connectivity between the OKE cluster and the Oracle CDBs is LAN-equivalent in reliability, performance, and throughput. This can be achieved by creating the Oracle CDBs within the same tenancy as the OKE cluster and in the same Oracle Cloud Infrastructure region.

ASAP cloud native allows for the full range of Oracle Cloud Infrastructure "cloud-to-ground" connectivity options for integrating the OKE cluster with on-premise applications and users. Selecting, provisioning, and testing such connectivity is a critical part of adopting Oracle Cloud Infrastructure OKE.

Using Load Balancer as a Service (LBaaS)

For load balancing, you have the option of using the services available in OKE. The infrastructure for OKE is provided by Oracle's IaaS offering, Oracle Cloud Infrastructure. In OKE, the master node IP address is not exposed to the tenants. The IP addresses of the worker nodes are also not guaranteed to be static. This makes

DNS mapping difficult to achieve. Additionally, it is also required to balance the load between the worker nodes. To fulfill these requirements, you can use Load Balancer as a Service (LBaaS) of Oracle Cloud Infrastructure.

The load balancer can be created using the service descriptor in **$ASAP_CNTK/samples/oci-lb-traefik.yaml**. The subnet ID referenced in this file must be filled in from your Oracle Cloud Infrastructure environment (using the subnet configured for your LBaaS). The port values assume you have installed Traefik using the unchanged sample values.

The configuration can be applied using the following command (or for traceability, by wrapping it into a Helm chart):

```
$ kubectl apply -f oci-lb-traefik.yaml
service/oci-lb-service-traefikconfigured
```

The Load Balancer service is created for Traefik pods in the Traefik name space. Once the Load Balancer service is created successfully, an external IP address is allocated. This IP address must be used for DNS mapping.

```
$ kubectl get svc -n traefik oci-lb-service-traefik
NAME                         TYPE           CLUSTER-IP      EXTERNAL-IP
PORT(S)
oci-lb-service-traefik       LoadBalancer   10.96.103.118   100.77.24.178
80:32006/TCP,443:32307/TCP
```

For additional details, see the following:

- "Creating Load Balancers to Distribute Traffic Between Cluster Nodes" in Oracle Cloud Infrastructure documentation.
- "Load Balancer Annotations" in Oracle GitHub documentation.

About Using Oracle Cloud Infrastructure Domain Name System (DNS) Zones

While a custom DNS service can provide the addressing needs of ASAP cloud native even when ASAP is running in OKE, you can evaluate the option of Oracle Cloud Infrastructure Domain Name System (DNS) zones capability. Configuration of DNS zones (and integration with on-premise DNS systems) is not within the scope of ASAP cloud native.

Using Persistent Volumes and File Storage Service (FSS)

In the OKE cluster, ASAP cloud native can leverage the high performance, high capacity, high reliability File Storage Service (FSS) as the backing for the persistent volumes of ASAP cloud native. There are two flavors of FSS usage in this context:

- Allocating FSS by setting up NFS mount target
- Native FSS

To use FSS through an NFS mount target, see instructions for allocating FSS and setting up a Mount Target in "Creating File Systems" in the Oracle Cloud Infrastructure documentation. Note down the Mount Target IP address and the storage path and use these in the ASAP cloud native instance specification as the NFS host and path. This approach is simple to set up and leverages the NFS storage provisioner that is typically available in all Kubernetes installations. However, the data flows through the mount target, which models an NFS server.

FSS can also be used natively, without requiring the NFS protocol. This can be achieved by leveraging the FSS storage provisioner supplied by OKE. The broad outline of how to do this is available in the blog post "Using File Storage Service with Container Engine for Kubernetes" on the Oracle Cloud Infrastructure blog.

Leveraging Oracle Cloud Infrastructure Services

For your OKE environment, you can leverage existing services and capabilities that are available with Oracle Cloud Infrastructure. The following table lists the Oracle Cloud Infrastructure services that you can leverage for your OKE cloud environment.

Table 2-1 Oracle Cloud Infrastructure Services for OKE Cloud Environment

Type of Service	Service	Indicates Mandatory / Recommended / Optional
Developer Service	Container Clusters	Mandatory
Developer Service	Registry	Recommended
Core Infrastructure	Compute Instances	Mandatory
Core Infrastructure	File Storage	Recommended
Core Infrastructure	Block Volumes	Optional
Core Infrastructure	Networking	Mandatory
Core Infrastructure	Load Balancers	Recommended
Core Infrastructure	DNS Zones	Optional
Database	BareMetal, VM, and ExaData	Recommended

Validating Your Cloud Environment

Before you start using your cloud environment for deploying cloud native instances, you must validate the environment to ensure that it is set up properly and that any prevailing issues are identified and resolved. This section describes the tasks that you should perform to validate your cloud environment.

You can validate your cloud environment by:

- Performing a smoke test of the Kubernetes cluster
- Validating the common building blocks in the Kubernetes cluster

Performing a Smoke Test

You can perform a smoke test of your Kubernetes cloud environment by running nginx. This procedure validates basic routing within the Kubernetes cluster and access from outside the environment. It also allows for an initial RBAC examination as you need to have permissions to perform the smoke test. For the smoke test, you need nginx 1.14.2 container image.

Note:

The requirement of the nginx container image for the smoke test can change over time. See the content of the **deployment.yaml** file in step 3 of the following procedure to determine which image is required. Alternatively, ensure that you have logged in to Docker Hub so that the system can download the required image automatically.

To perform a smoke test:

1. Download the nginx container image from Docker Hub.

   For details on managing container images, see "About Container Image Management."

2. After obtaining the image from Docker Hub, upload it into your private container repository and ensure that the Kubernetes worker nodes can access the image in the repository.

   Oracle recommends that you download and save the container image to the private Docker repository even if the worker nodes can access Docker Hub directly. The images in the cloud native toolkit are available only through your private Docker repository.

3. Run the following commands:

```
kubectl apply -f https://k8s.io/examples/application/deployment.yaml #
the deployment specifies two replicas
kubectl get pods     # Must return two pods in the Running state
kubectl expose deployment nginx-deployment --type=NodePort --
name=external-nginx
kubectl get service external-nginx    # Make a note of the external port
for nginx
```

   These commands must run successfully and return information about the pods and the port for nginx.

4. Open the following URL in a browser:

```
http://master_IP:port/
```

   where:

   - *master_IP* is the IP address of the master node of the Kubernetes cluster or the external IP address for which routing has been set up
   - *port* is the external port for the **external-nginx** service

5. To track which pod is responding, on each pod, modify the text message in the webpage served by nginx. In the following example, this is done for the deployment of two pods:

```
$ kubectl get pods -o wide | grep nginx
nginx-deployment-5c689d88bb-g7zvh   1/1     Running   0          1d
10.244.0.149   worker1   <none>
nginx-deployment-5c689d88bb-r68g4   1/1     Running   0          1d
10.244.0.148   worker2   <none>
$ cd /tmp
$ echo "This is pod A - nginx-deployment-5c689d88bb-g7zvh - worker1" >
```

```
index.html
$ kubectl cp index.html nginx-deployment-5c689d88bb-g7zvh:/usr/
share/nginx/html/index.html
$ echo "This is pod B - nginx-deployment-5c689d88bb-r68g4 -
worker2" > index.html
$ kubectl cp index.html nginx-deployment-5c689d88bb-r68g4:/usr/
share/nginx/html/index.html
$ rm index.html
```

6. Check the index.html webpage to identify which pod is serving the page.
7. Check if you can reach all the pods by running refresh (Ctrl+R) and hard refresh (Ctrl+Shift+R) on the index.html webpage.
8. If you see the default nginx page, instead of the page with your custom message, it indicates that the pod has restarted. If a pod restarts, the custom message on the page gets deleted.

   Identify the pod that restarted and apply the custom message for that pod.
9. Increase the pod count by patching the deployment.

   For instance, if you have three worker nodes, run the following command:

   Note:

   Adjust the number as per your cluster. You may find you have to increase the pod count to more than your worker node count until you see at least one pod on each worker node. If this is not observed in your environment even with higher pod counts, consult your Kubernetes administrator. Meanwhile, try to get as much worker node coverage as reasonably possible.

   ```
   kubectl patch deployment nginx-deployment -p '{"spec":
   {"replicas":3}}' --type merge
   ```
10. For each pod that you add, repeat step 5 to step 8.

Ensuring that all the worker nodes have at least one nginx pod in the Running state ensures that all worker nodes have access to Docker Hub or to your private Docker repository.

Validating Common Building Blocks in the Kubernetes Cluster

To approach ASAP cloud native in a sustainable manner, you must validate the common building blocks that are on top of the basic Kubernetes infrastructure individually. The following sections describe how you can validate the building blocks.

Network File System (NFS)

ASAP cloud native uses Kubernetes Persistent Volumes (PV) and Persistent Volume Claims (PVC) to use a pod-remote destination filesystem for ASAP logs and performance data. By default, these artifacts are stored within a pod in Kubernetes and are not easily available for integration into a toolchain. For these to be available externally, the Kubernetes environment must implement a mechanism for fulfilling PV and PVC. The Network File System (NFS) is a common PV mechanism.

For the Kubernetes environment, identify an NFS server and create or export an NFS filesystem from it.

Ensure that this filesystem:

- Has enough space for the ASAP logs and performance data
- Is mountable on all the Kubernetes worker nodes

Create an nginx pod that mounts an NFS PV from the identified server. For details, see the documentation about "Kubernetes Persistent Volumes" on the Kubernetes website. This activity verifies the integration of NFS, PV/PVC, and the Kubernetes cluster. To clean up the environment, delete the nginx pod, the PVC, and the PV.

Ideally, data such as logs and JFR data is stored in the PV only until it can be retrieved into a monitoring toolchain such as Elastic Stack. The toolchain must delete the rolled over log files after processing them. This helps you to predict the size of the filesystem. You must also consider the factors such as the number of ASAP cloud native instances that will use this space, the size of those instances, the volume of orders they will process, and the volume of logs that your cartridges generate.

Validating the Load Balancer

For a development-grade environment, you can use an in-cluster software load balancer. ASAP cloud native toolkit provides documentation and samples that show you how to use Traefik to perform load balancing activities for your Kubernetes cluster.

It is not necessary to run through "Traefik Quick Start" as part of validating the environment. However, if the ASAP cloud native instances have connectivity issues with HTTP/HTTPS traffic, and the ASAP logs do not show any failures, it might be worthwhile to take a step back and validate Traefik separately using Traefik Quick Start.

A more intensive environment, such as a test, a production, apre-production, or performance environment can additionally require a more robust load balancing service to handle the HTTP/HTTPS traffic. For such environments, Oracle recommends using a load balancing hardware that is set up outside the Kubernetes cluster. A few examples of external load balancers are Oracle Cloud Infrastructure LBaaS for OKE, Google's Network LB Service in GKE, and F5's Big-IP for private cloud. The actual selection and configuration of an external load balancer is outside the scope of ASAP cloud native itself but is an important component to sort out in the implementation of ASAP cloud native. For more details on the requirements and options, see "Integrating ASAP".

To validate the ingress controller of your choice, you can use the same nginx deployment used in the smoke test described earlier. This is valid only when run in a Kubernetes cluster where multiple worker nodes are available to take the workload.

To perform a smoke test of your ingress setup:

1. Run the following commands:

```
kubectl apply -f https://k8s.io/examples/application/deployment.yaml
kubectl get pods -o wide    # two nginx pods in Running state; ensure
these are on different worker nodes
cat > smoke-internal-nginx-svc.yaml <<EOF
apiVersion: v1
kind: Service
metadata:
  name: smoke-internal-nginx
  namespace: default
```

```
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  sessionAffinity: None
  type: ClusterIP
EOF
kubectl apply -f ./smoke-internal-nginx-svc.yaml
kubectl get svc smoke-internal-nginx
```

2. Create your ingress targeting the **internal-nginx** service. The following text shows a sample ingress annotated to work with the Traefik ingress controller:

   ```
   apiVersion: extensions/v1beta1
   kind: Ingress
   metadata:
     annotations:
       kubernetes.io/ingress.class: traefik
     name: smoke-nginx-ingress
     namespace: default
   spec:
     rules:
     - host: smoke.nginx.asaptest.org
       http:
         paths:
         - backend:
             serviceName: smoke-internal-nginx
             servicePort: 80
   ```

   If the Traefik ingress controller is configured to monitor the default name space, then Traefik creates a reverse proxy and the load balancer for the nginx deployment. For more details, see Traefik documentation.

   If you plan to use other ingress controllers, refer to the documentation about the corresponding controllers for information on creating the appropriate ingress and make it known to the controller. The ingress definition should be largely reusable, with ingress controller vendors describing their own annotations that should be specified, instead of the Traefik annotation used in the example.

3. Create a local DNS/hosts entry in your client system mapping **smoke.nginx.asaptest.org** to the IP address of the cluster, which is typically the IP address of the Kubernetes master node, but could be configured differently.

4. Open the following URL in a browser:

   ```
   http://smoke.nginx.asaptest.org:Traefik_Port/
   ```

   where *Traefik_Port* is the external port that Traefik has been configured to expose.

5. Verify that the web address opens and displays the nginx default page.

Your ingress controller must support session stickiness for ASAP cloud native. To learn how stickiness should be configured, refer to the documentation about the ingress controller you choose. For Traefik, stickiness must be set up at the service level itself. For testing purposes, you can modify the **internal-nginx** service to enable stickiness by running the following commands:

```
kubectl delete ingress smoke-nginx-ingress
vi smoke-internal-nginx-svc.yaml
# Add an annotation section under the metadata section:
#   annotation:
#     traefik.ingress.kubernetes.io/affinity: "true"
kubectl apply -f ./smoke-internal-nginx-svc.yaml
# now apply back the ingress smoke-nginx-ingress using the above yaml
definition
```

Other ingress controllers may have different configuration requirements for session stickiness. Once you have configured your ingress controller, and `smoke-nginx-ingress` and `smoke-internal-nginx` services as required, repeat the browser-based procedure to verify and confirm if nginx is still reachable. As you refresh (Ctrl+R) the browser, you should see the page getting served by one of the pods. Repeatedly refreshing the web page should show the same pod servicing the access request.

To further test session stickiness, you can either do a hard refresh (Ctrl+Shift+R) or restart your browser (you may have to use the browser in Incognito or Private mode), or clear your browser cache for the access hostname for your Kubernetes cluster. You may observe that the same nginx pod or a different pod is servicing the request. Refreshing the page repeatedly should stick with the same pod while hard refreshes should switch to the other pod occasionally. As the deployment has two pods, the chances of a switch with a hard refresh are 50%. You can modify the deployment to increase the number of replica nginx pods (controlled by the `replicas` parameter under `spec`) to increase the odds of a switch. For example, with four nginx pods in the deployment, the odds of a switch with hard refresh rise to 75%. Before testing with the new pods, run the commands for identifying the pods to add unique identification to the new pods. See the procedure in "Performing a Smoke Test" for the commands.

To clean up the environment after the test, delete the following services and the deployment:

- `smoke-nginx-ingress`
- `smoke-internal-nginx`
- `nginx-deployment`

3

Creating an ASAP Cloud Native Image

An ASAP cloud native image is required to create and manage ASAP cloud native instances. This chapter describes creating an ASAP cloud native image.

An ASAP cloud native requires a container image and access to the database. The ASAP image is built on top of a Linux base image and the ASAP image builder script adds Java, WebLogic Server components, database client, and ASAP.

The ASAP cloud native image is created using the ASAP cloud native builder toolkit. You should run the ASAP cloud native builder toolkit on Linux and it should have access to the local Docker daemon.

See the following topics for further details:

- Downloading the ASAP Cloud Native Image Builder
- Prerequisites for Creating ASAP Image
- Creating the ASAP Cloud Native Image
- Working with Cartridges

Downloading the ASAP Cloud Native Image Builder

To build the ASAP cloud native Docker image, the **asap-img-builder.zip** file is required. For more information about downloading the ASAP cloud native Image Builder, see "Downloading the ASAP Cloud Native Artifacts".

ASAP cloud native builder kit contains:

- The scripts to install the required packages.
- The scripts to install database client, Java, WebLogic Server, and ASAP.

Prerequisites for Creating ASAP Image

The prerequisites for building ASAP cloud native image are:

- The Docker client and daemon on the build machine.
- Approximately 2 GB of swap space on the machine where the Docker daemon is running. By running the `free -m` command, you can verify the swap space.

  > **Note:**
  >
  > If the required swap space is not available, contact your administrator.

- ASAP 7.4.0.0 or later Linux Installer. Download the **.tar** file from Oracle Software Delivery Cloud:
  https://edelivery.oracle.com

- Create the **disk1** directory and copy the contents of the **.tar** file to this directory.
- Installers for WebLogic Server and JDK. Download these from Oracle Software Delivery Cloud:

  https://edelivery.oracle.com
- Oracle Database Client. Download this from Oracle Software Downloads:

  https://www.oracle.com/downloads/
- Java, installed with JAVA_HOME set in the environment.
- ASAP is installed in a silent installation mode using the **asap.properties** file. You should update the properties file with the database, WebLogic Server, ORACLE_HOME, port numbers, and all required details.
- Keep the TRAEFIK Ingress service node port details ready where it is being deployed.
- Create ASAP database users. For more information, see "Creating Oracle Database Tablespaces, Tablespace User, and Granting Permissions" in *ASAP Installation Guide*.

For details about the required and supported versions of the prerequisite software, see *ASAP Software Compatibility Matrix*.

Creating the ASAP Cloud Native Image

The ASAP cloud native image builder tool builds the ASAP cloud native Docker image, which is then pushed to the Docker repository and deployed in the Kubernetes cluster. If the Docker repository is not available, you copy the image to all the worker nodes of the cluster.

The ASAP installer is packaged with the ASAP cloud native image builder and the cloud native toolkit.

Note:

After you download the installer, locate the cloud native image builder **asap-img-builder.zip** in the cloud native tar file. The ASAP Docker images are created automatically for ASAP 7.4.0.1 or later.

To create the ASAP cloud native Docker image:

1. Copy the **asap-img-builder.zip** file to the machine where the Docker daemon is running.
2. Extract the contents of the zip file by running the following command:

   ```
   unzip asap-img-builder.zip
   ```
3. Copy the following installers to the **$asap-img-builder/installers** directory.
   - Linux Installer for ASAP 7.4.0.1 or later
   - Installers for WebLogic Server and JDK
   - Oracle Database Client
4. Copy the required cartridges to the **$asap-img-builder/cartridges** directory.

5. Set the environment variable for ASAP_IMG_BUILDER to the **$asap-img-builder** directory.
6. Update the **$asap-img-builder/scripts/tnsnames.ora** file with the database details. For example,

   ```
   orcl19c =
       (DESCRIPTION =
         (ADDRESS = (PROTOCOL = TCP)(HOST = database host)(PORT = 1521))
         (CONNECT_DATA =
           (SERVER = DEDICATED)
           (SERVICE_NAME = service name)
         )
       )
   ```

7. Update the **HTTPS_PROXY** and **HTTP_PROXY** variables in the **build_env.sh** script.

   > **Note:**
   >
   > The variables in the section Docker details, Installer filenames, and Installation locations are populated by default with the appropriate information.

   ```
   base_image=oraclelinux:8
   HTTPS_PROXY=
   HTTP_PROXY=

   # Docker details
   ASAP_IMAGE_TAG="7.4.0.0.0"
   ASAP_VOLUME=dockerhost_volume
   ASAP_CONTAINER="asap-c"
   DOCKER_HOSTNAME="asaphost"

   # Installer filenames
   JDK_FILE=jdk-8u321-linux-x64.tar.gz
   DB_CLIENT_FILE=LINUX_193000_client.zip
   FMW_FILE=fmw_14.1.1.0.0_wls_lite_Disk1_1of1.zip

   # Installation locations
   TNS_ADMIN=/scripts/
   JAVA_HOME=/usr/lib/jvm/java/jdk1.8.0_321
   ORACLE_HOME=/home/oracle/oracle/product/19.3.0/dbhome_1
   PATH=$ORACLE_HOME:$JAVA_HOME/bin:$PATH
   CV_ASSUME_DISTID=OEL8
   WLS_HOME=/home/oracle/weblogic14110/wlserver
   ```

   > **Note:**
   >
   > Ensure that the file names of **JDK_FILE, DB_CLIENT_FILE,** and **FMW_FILE** variables match with the file names in the **/asap-img-builder/installers/** folder.

8. Update the following parameters in the **$asap-img-builder/asap.properties** file with the required details for ASAP and the WebLogic Server domain. You can update the configuration parameters that must be updated after ASAP installation using **asap.properties** file. All the configuration parameters should be prefixed with **asap.properties**. You can create multiple ASAP instances by entering multiple unique environment IDs in the **asap.envid** variable separated by a comma. For example, **asap.envid=**CNE1, CNE2, CNE3 where CNE1, CNE2, and CNE3 are unique environment IDs.

   ```
   asap.tar.file=ASAP.R7_4_0.B196.linux.tar
   asap.envid=CNE1
   asap.installLocation=/scratch/oracle/asap
   asap.db.alias=
   asap.db.username=
   asap.db.password=
   asap.db.tablespace=
   asap.db.temp.ts=
   weblogic.domainName=asapDomain
   weblogic.domainLocation=/u01/oracle/user_projects/domain/
   weblogic.username=
   weblogic.password=
   weblogic.port=7601
   weblogic.channel.listenport=7602
   weblogic.channel.publicport=30301
   asap.server.adm.password=
   asap.server.ctrl.password=
   asap.server.nep.password=
   asap.server.sarm.password=
   asap.server.srp.password=
   asap.weblogic.adminPassword=
   asap.weblogic.cmwsPassword=
   asap.weblogic.monitorPassword=
   asap.weblogic.operatorPassword=
   asap.weblogic.wsPassword=
   ## ASAP.cfg properties
   asap.properties.MSGSND_RETRIES=5
   ```

Note:

- If Traefik is configured, `weblogic.channel.publicport` parameter must have the traefik ingress controller nodeport value (for http 30305 and for https 30443).
- If DNS is configured, `weblogic.channel.publicport` must have the value same as `weblogic.channel.listenport`.
- `asap.weblogic.adminPassword`, `asap.weblogic.cmwsPassword`, `asap.weblogic.monitorPassword`, `asap.weblogic.operatorPassword`, and `asap.weblogic.wsPassword` parameters must have passwords at least 8 characters long, and must contain at least 1 number or special character.
- The **asap.properties** file is available only in the host machine and not as part of the Docker image.

9. Run the **build-asap-images.sh** script using the following command:

```
./build-asap-images.sh -i asap
```

The script creates the staging Docker image by installing WebLogic Server, Java, and the database client.

The script also creates the staging ASAP Docker image for the environment IDs specified in the **asap.properties** file.

Working with Cartridges

ASAP cartridges are discrete software components developed for ASAP. An ASAP cartridge provides specific domain behavior on top of the core ASAP software. This domain behavior includes a part of, or all services on a network element (NE), element management system (EMS), or network management system (NMS). For more information about cartridges, see *ASAP Cartridge Development Guide* and for installing cartridges, see *ASAP Installation Guide*.

To provision orders to network elements, you install cartridges in the ASAP container that are available in the **$ASAP_VOLUME/cartridges/** directory. After installing the cartridges, you must exit the ASAP container by using the `exit` command.

To install cartridges:

1. Copy the required cartridges to the **$asap-img-builder/cartridges** directory.
2. Run the following command to copy installers and cartridges to the volume:

```
$asap-img-builder/upgradeASAPDockerImage.sh
```

3. Create a new container with the ASAP Docker image created by the **build-asap-images.sh** script using the following command:

```
docker run --name $ASAP_CONTAINER -dit -h $DOCKER_HOSTNAME -
p $WEBLOGIC_PORT -v $ASAP_VOLUME:/$ASAP_VOLUME ASAP-BASE-IMAGE
```

For example: docker run --name asap-c -dit -h asaphost -p 7601 -v dockerhost_volume:/dockerhost_volume asapcn:7.4.0.0

The container is created with **asap-c**.

4. Enter into the ASAP container using the following command:

   ```
   docker exec -it CONTAINER_NAME bash
   ```

   where *CONTAINER_NAME* is the $ASAP_CONTAINER. For example, **asap-c**.

   You have entered into the ASAP container.

5. Start ASAP and WebLogic Server using the `startALL.sh` script.
6. Navigate to the ASAP installation directory using `cd $ASAP_BASE`.
7. Source the environment profile using the `source Environment_Profile` script.
8. Verify the ASAP server status using the `status` command.
9. Install the cartridges present in the **/dockerhost_volume/cartridges** directory. For more information, see "Installing a Cartridge" in *ASAP Installation Guide*.
10. After you install the cartridge, create an image from the staging container using the following command:

    ```
    docker commit CONTAINER_NAME imagename:version
    ```

    Where

    - *CONTAINER_NAME* is the $ASAP_CONTAINER.
    - *version* is the version of the ASAP Docker image. This version should be higher than the previous version.

11. Exit the ASAP container by using the following command:

    ```
    exit
    ```

12. Stop and remove the containers using the following commands:

    ```
    docker stop CONTAINER_NAME
    ```

    ```
    docker rm CONTAINER_NAME
    ```

    where *CONTAINER_NAME* is the $ASAP_CONTAINER.

Securing Your ASAP Installation

ASAP security is designed to provide confidentiality, data integrity, and ensure on-demand access to services for authorized users. For information about ASAP security, see "Setting Up and Managing ASAP Security" in *ASAP System Administrator's Guide*.

4

Creating an ASAP Cloud Native Instance

This chapter describes how to create an ASAP cloud native instance in your cloud environment using the operational scripts and the base ASAP configuration provided in the ASAP cloud native toolkit. You can create an ASAP instance quickly to become familiar with the process, explore the configuration, and structure your own project. This procedure is intended to validate that you are able to create an ASAP instance in your environment.

Before you create an ASAP instance, you must do the following:

- Download the ASAP cloud native tar file and extract the **asap-cntk.zip** file. For more information about downloading the ASAP cloud native toolkit, see "Downloading the ASAP Cloud Native Artifacts".
- Install the Traefik container images

Installing the ASAP Cloud Native Artifacts and the Toolkit

Build container image for the ASAP application using the ASAP cloud native Image Builder.

You must create a private Docker repository for this image, ensuring that all nodes in the cluster have access to the repository. See "About Container Image Management" for more details.

Copy the ASAP cloud native toolkit that is the **asap-cntk.zip** file to one of the nodes in the Kubernetes cluster:

- **On Oracle Linux**: Where Kubernetes is hosted on Oracle Linux, download and extract the tar archive to each host that has connectivity to the Kubernetes cluster.
- **On OKE**: For an environment where Kubernetes is running in OKE, extract the contents of the tar archive on each OKE client host. The OKE client host is the bastion host/s that is set up to communicate with the OKE cluster.

Set the variable for the installation directory by running the following command, where *asap_cntk_path* is the installation directory of the ASAP cloud native toolkit:

```
$ export ASAP_CNTK=asap_cntk_path
```

Installing the Traefik Container Image

Note:

If you are installing Order Balancer in the ASAP namespace, ignore this section.

To leverage the ASAP cloud native samples that integrate with Traefik, the Kubernetes environment must have the Traefik ingress controller installed and configured.

If you are working in an environment where the Kubernetes cluster is shared, confirm whether Traefik has already been installed and configured for ASAP cloud native. If Traefik is already installed and configured, set your `TRAEFIK_NS` environment variable to the appropriate name space.

The instance of Traefik that you installed to validate your cloud environment must be removed as it does not leverage the ASAP cloud native samples. Ensure that you have removed this installation in addition to purging the Helm release. Check that any roles and rolebindings created by Traefik are removed. There could be a **clusterrole** and **clusterrolebinding** called "traefik-operator". There could also be a **role** and **rolebinding** called "traefik-operator" in the $TRAEFIK_NS name space. Delete all of these before you set up Traefik.

To download and install the Traefik container image:

1. Ensure that Docker in your Kubernetes cluster can pull images from Docker Hub. See *ASAP Compatibility Matrix* for the required and supported versions of the Traefik image.
2. Run the following command to create a name space ensuring that it does not already exist:

   Note:

   You might want to add the traefik name space to the environment setup such as `.bashrc`.

   ```
   kubectl get namespaces
   export TRAEFIK_NS=traefik
   kubectl create namespace $TRAEFIK_NS
   ```

3. Run the following commands to install Traefik using the **$ASAP_CNTK/samples/charts/traefik/values.yaml** file in the samples:

   Note:

   Set **kubernetes.namespaces** and the chart version specifically using command-line.

   ```
   helm repo add traefik repo_link
   helm install traefik-operator traefik/traefik \
    --namespace $TRAEFIK_NS \
    --values $ASAP_CNTK/samples/charts/traefik/values.yaml \
     --set "kubernetes.namespaces={$TRAEFIK_NS}"
   ```

   where *repo_link* is https://helm.traefik.io/traefik or https://traefik.github.io/charts based on the helm chart version. For more details, see: https://github.com/traefik/traefik-helm-chart

After the installation, Traefik monitors the name spaces listed in its `kubernetes.namespaces` field for Ingress objects. The scripts in the toolkit manage this name space list as part of creating and tearing down ASAP cloud native projects.

When the **values.yaml** Traefik sample in the ASAP cloud native toolkit is used as is, Traefik is exposed to the network outside of the Kubernetes cluster through port 30305. To use a different port, edit the YAML file before installing Traefik. Traefik metrics are also available for Prometheus to scrape from the standard annotations.

Traefik function can be viewed using the Traefik dashboard. Create the Traefik dashboard by running the instructions provided in the **$ASAP_CNTK/samples/charts/traefik/traefik-dashboard.yaml** file. To access this dashboard, the URL is: `http://traefik.asap.org`. This is if you use the **values.yaml** file provided with the ASAP cloud native toolkit; it is possible to change the hostname as well as the port to your desired values.

Creating an ASAP Instance

This section describes how to create an ASAP instance.

Setting Environment Variables

Order Balancer cloud native relies on access to certain environment variables to run seamlessly. Ensure the following variables are set in your environment:

- Path to your private specification repository
- Traefik name space

To set the environment variables:

- Set the `TRAEFIK_NS` variable for Traefik name space as follows:

  ```
  $ export TRAEFIK_NS=Treafik Namespace
  ```

Creating Secrets

You must store sensitive data and credential information in the form of Kubernetes Secrets that the scripts and Helm charts in the toolkit consume. Managing secrets is out of the scope of the toolkit and must be implemented while adhering to your organization's corporate policies. Additionally, Order Balancer cloud native does not establish password policies.

For an ASAP could native instance, the following secrets are required:

- **imagepull-secret**: If the private registry or repository is password protected, create this secret.
- **tls-secret**: If the traefik ingress is ssl-enabled, create this secret. For more information about creating **tls-secret**, see "Setting Up ASAP Cloud Native for Incoming Access."

To create imagepull-secret:

1. Run the following command:

   ```
   docker login
   ```

2. Enter the credentials or access token.

The login process creates or updates the `config.json` file that contains an authorization token.

3. Run the following command to run the secret:

```
kubectl create secret generic asap-imagepull -n namespace
--from-file=.dockerconfigjson=$HOME/.docker/config.json --
type=kubernetes.io/dockerconfigjson
```

Registering the Namespace

After you set the environment variables, register the name space. To register the name space, run the following command:

```
$ASAP_CNTK/scripts/register-namespace.sh -p sr -t targets
# For example, $ASAP_CNTK/scripts/register-namespace.sh -p sr -t
traefik
```

Where `-p` is the namespace where ASAP is being deployed.

Note:

`traefik` is the name of the targets for registration of the namespace `sr`. The script uses **TRAEFIK_NS** to find these targets. Do not provide the `traefik` target if you are not using Traefik.

Configuring Failed ASAP Instances to Restart Automatically

You configure the readiness and liveness probes in Kubernetes to restart the failed ASAP instances automatically and to restore the services. Kubernetes uses the liveness and readiness probes to find the failed ASAP instances and restarts those instances to restore the services automatically.

You update the liveness and readiness probes in the **$ASAP_CNTK/charts/asap/values.yaml** file with the following values:

```
readiness:
  enabled: true
  initialDelaySeconds: 240
  periodSeconds: 60
liveness:
  enabled: true
  periodSeconds: 120
  initialDelaySeconds: 120
  failureThreshold: 3
```

For detailed description of the readiness and liveness parameters, see step 1 in "Creating an ASAP Instance".

Creating an ASAP Instance

This procedure describes how to create an ASAP instance in your environment using the scripts that are provided with the toolkit.

To create an ASAP instance:

1. Update the **$ASAP_CNTK/charts/asap/values.yaml** file with the following values:

```
# Default values for asap.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

replicaCount: 1
image:
  repository: asapcn
  pullPolicy: IfNotPresent
  tag: "7.4.0.0.0"
imagePullSecrets:
  - name: asap-imagepull
asapEnv:
  envid: cne1
  port: 7601
  host: asaphost
persistence:
  enabled: false
readiness:
  enabled: true
  initialDelaySeconds: 240
  periodSeconds: 60
liveness:
  enabled: true
  periodSeconds: 120
  initialDelaySeconds: 120
  failureThreshold: 3
nameOverride: ""
fullnameOverride: ""

serviceAccount:
  # Specifies whether a service account should be created
  create: true
  # Annotations to add to the service account
  annotations: {}
  # The name of the service account to use.
  # If not set and create is true, a name is generated using the fullname
template
  name: ""

podAnnotations: {}
podSecurityContext: {}
  # fsGroup: 2000
securityContext: {}
  # capabilities:
  #   drop:
```

```
  #   - ALL
  # readOnlyRootFilesystem: true
  # runAsNonRoot: true
  # runAsUser: 1000
servicechannel:
  name: channelport
  type: ClusterIP
  port: 7601
service:
  name: adminport
  type: ClusterIP
  port: 7602

ingress:
  type: TRAEFIK
  enabled: true
  sslIncoming: false
  adminsslhostname: adminhost.asap.org
  adminhostname: adminhostnonssl.asap.org
  secretName: project-instance-asapcn-tls-cert
  hosts:
    - host: adminhost.asap.org
      paths: []
  tls: []
  #  - secretName: chart-example-tls
  #    hosts:
  #      - chart-example.local

resources: {}
autoscaling:
  enabled: false
nodeSelector: {}
tolerations: []

affinity: {}
```

Where:

- *repository* is the repository name of the configured container registry
- *tag* is the version name that is used when you create a final Docker image from the container
- *name* is the name of the imagepull-secret if it is configured. For more information about imagepull-secret, see "Creating Secrets".
- *envid* is the unique environment ID of the instance. This ID should include only the lower-case alphanumeric characters. For example, asapinstance1.
- *port* is the port number of the WebLogic Server where ASAP is deployed.
- *host* is the hostname of the docker container.

> **Note:**
>
> The hostname should match with the hostname when you create the ASAP Docker image. If the hostname mismatches, the ASAP servers may not start.

- `persistence.`*enabled* is set to **true** if PVC is enabled.
- `readiness.`*enabled* is set to **true** to enable the readiness probe in Kubernetes. Kubernetes uses the readiness probe to know when a container is ready to start accepting traffic.
- *initialDelaySeconds* is the number of seconds after the ASAP instance has started before the readiness probes are initiated.
- *periodSeconds* specifies how often (in seconds) to perform the readiness probe.
- `liveness.`*enabled* is set to **true** to enable the liveness probe. Kubernetes uses the liveness probes to know when to restart a container.
- *periodSeconds* specifies how often (in seconds) to perform the liveness probe.
- *initialDelaySeconds* is the number of seconds after the ASAP instance has started before the liveness probes are initiated.
- *failureThreshold* is the number of times that Kubernetes retry the liveness probes before restarting the ASAP instance.
- `servicechannel.`*port* is the channel port when you create a channel in the WebLogic domain.
- `service.`*port* is the admin port of the WebLogic Server.
- *type* is the ingress controller type. This type can be TRAEFIK or GENERIC or OTHER.
- *enabled* is the status of the ingress controller whether it is enabled or not. The value is *true* or *false*. By default, this is set to *true*.
- *sslIncoming* is the status of the SSL/TLS configuration on incoming connections whether it is enabled or not. The configuration value is *true* or *false*. By default, this is set to *false*. If you want to set the value to **true**, create keys, certificate, and secret by following the instructions in the "Setting Up ASAP Cloud Native for Incoming Access" section.
- *adminsslhostname* is the hostname of the https access.
- *adminhostname* is the hostname of the http access.
- *secretName* is the secret name of the certificate created for SSL/TLS. For more information about creating keys and secret name, see "Setting Up ASAP Cloud Native for Incoming Access."

> **Note:**
>
> `adminsslhostname` and `secretName` are applicable only if `sslIncoming` is set to **true**.

2. Create PV and PVC if the persistence.enabled is set to **true** in the **$ASAP_CNTK/charts/asap/values.yaml** file for the ASAP instance. The PV path is used to store the

logs of the ASAP instance. The sample files are available in the **asap_cntk.zip** at **$ASAP_CNTK/samples/nfs/** file.

```
# This is pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: <project>-<asapEnv.envid>-nfs-pv
  labels:
    type: local
spec:
  storageClassName: asaplogs
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
#  hostPath:
#   path: "/mnt/asap/logs/
  nfs:
    server: <server>
    path: <path>
# This is pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: <obEnv.envid>-pvc
  namespace: sr
spec:
  storageClassName: asaplogs
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Where:

- <project> is the namespace. In the above example, value is *sr*.
- <*asapEnv.envid*> is the environment ID provided in the **$ASAP_CNTK/samples/charts/traefik/values.yaml** file. In the above example, value is *cn96*.

This path will be mounted on the Pod as `/scratch/oracle/asap/DATA/logs/`

3. Run the following command to create the ASAP instance:

```
$ASAP_CNTK/scripts/create-instance.sh -p sr -i quick
```

The **create-instance.sh** script uses the Helm chart located in the **charts/asap** directory to deploy the ASAP docker image, service, and ingress controller for your instance. If the script fails, see "Troubleshooting Issues with the Scripts" before you make additional attempts.

4. Validate the important input details such as Image name and tag, specification files used (Values Applied), hostname, and port for ingress routing:

   ```
   $ASAP_CNTK/scripts/create-instance.sh -p sr -i quick

   1 chart(s) linted, 0 chart(s) failed
   Project Namespace : sr
   Instance Fullname : sr-quick

   NAME: sr-quick
   LAST DEPLOYED: Sun Feb 27 17:49:25 2022
   NAMESPACE: sr
   STATUS: deployed
   REVISION: 1
   TEST SUITE: None
   ```

5. If you query the status of the ASAP pod, the **READY** state of the ASAP pod displays **0/1** for several minutes when the ASAP application is starting.
   When the **READY** state shows **1/1**, your ASAP instance is up and running. You can then validate the instance by submitting work orders.

The base hostname required to access this instance using HTTP is `quick.sr.asap.org`. See "Planning and Validating Your Cloud Environment" for details about hostname resolution.

The **create-instance** script prints out the following valuable information that you can use when you work with your ASAP domain:

- The T3 URL: `http://t3.quick.sr.asap.org` This is required for external client applications such as JMS and WLST.
- The URL for accessing the WebLogic UI, which is provided through the ingress controller at host: `http://admin.quick.sr.asap.org:30305/console`.

Validating the ASAP Instance

After creating an instance, you can validate it by accessing the WebLogic Server console.

Run the following command to display the pod details of the ASAP instance that you have created:

```
$ kubectl get all -n sr
NAME                                       READY   STATUS    RESTARTS   AGE
pod/asapinstance1-deployment-9845fbcb6-8qq2h    1/1     Running   0
5d15h
NAME                   TYPE        CLUSTER-IP      EXTERNAL-IP
PORT(S)             AGE
service/asapinstance1-service   ClusterIP   10.99.231.206   <none>
7602/TCP,7601/TCP   5d21h
NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/asapinstance1-deployment   1/1     1             1
5d21h
NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/asapinstance1-deployment-d5bd787f8    0          0
0       5d15h
```

To get the ASAP server status enter into the pod by using the following command:

```
kubectl exec -it asapinstance1-deployment-9845fbcb6-8qq2h bash -n sr
```

You are now entered into the ASAP pod. Navigate to the ASAP installation directory by using the following command:

```
cd $ASAP_BASE
source Environment_Profile
status
```

The status of ASAP servers are displayed. For example:

```
[root@asaphost asap]# status
               **** ASAP Application Status ****
    #  CPU       PID      Program
Application Location
    -- --------- --------- ----------------------------------------
----------- --------
    1  00:00:01  6712     $ASAP_BASE/programs/ctrl_svr
CTRLCN89    LOCAL
    2  00:00:00  6824     $ASAP_BASE/programs/fork_agent
CTRLCN89    LOCAL
    3  00:00:18  6966     java
JNEP_CN89   LOCAL
    4  00:00:01  7018     $ASAP_BASE/programs/asc_nep
NEP_CN89    LOCAL
    5  00:00:01  7073     $ASAP_BASE/programs/admn_svr
ADM_CN89    LOCAL
    6  00:00:07  7131     $ASAP_BASE/programs/sarm
SARMCN89    LOCAL
    7  00:00:06  7234     java
DAEMCN89    LOCAL
    8  00:00:02  7335     $ASAP_BASE/programs/srp_emul
SRP_CN89    LOCAL
               **** End of Application Status ****
```

`CN89` is the ENV_ID.

Note:

After an ASAP instance is created, it may take a few minutes to start ASAP servers and WebLogic Server.

To access WebLogic Administration Console outside the cluster, enter the following URL in the browser:

```
http://adminhostnonssl.asap.org:30305/console
```

The system prompts for the user name and password. Enter the WebLogic domain user name and password.

Update the hosts file with the **hostname** and **master_ip** address on the machine where the URL is getting accessed with the following information:

```
ip_address adminhostnonssl.asap.org
```

Note:

The hosts file is located in **/etc/hosts** on Linux and MacOS machines and in **C:\Windows\System32\drivers\etc\hosts** on Windows machines.

Submitting Orders

ASAP is installed with the default POTS cartridge. You can submit ASAP orders over JMS and Web Services.

To submit ASAP orders over JMS, use an external runJMSclient. The endpoint must be as follows:

```
System.$
{ENV_ID}.ApplicationType.ServiceActivation.Application.1-0;7-4;ASAP.Comp.Mess
ageQueue
```

The connection factory's provider URL must be as follows:

For non-SSL:

```
http://adminhostnonssl.asap.org:30305/
```

For SSL:

```
https://adminhostssl.asap.org:30443/
```

To submit ASAP orders over Web Services, type the following URL in the web browser and access the ASAP Web Services WSDL:

```
http://adminhostnonssl.asap.org:30305/env_id/Oracle/CGBU/Mslv/Asap/Ws?WSDL
```

HTTP protocol is used for a handshake with the application server to authenticate and request a web service client stub, which is used as the launch pad to talk to Web Services. Then the client can communicate with the ASAP Web Services using HTTP or HTTPS protocols.

Deleting and Recreating Your ASAP Instance

Deleting Your ASAP Instance

To delete your ASAP instance, run the following command:

```
$ASAP_CNTK/scripts/delete-instance.sh -p sr -i quick
```

Recreating Your ASAP Instance

When you delete an ASAP instance, the database state for that instance still remains unaffected. You can re-create an ASAP instance by using the same Docker image.

To re-create an ASAP instance, run the following command:

```
$ASAP_CNTK/scripts/create-instance.sh -p sr -i quick
```

Note:

After recreating an instance, client applications such as SoapUI may need to be restarted to avoid using expired cache information.
If another ASAP instance is created in the same database using the same Environment ID, the ASAP installer deletes the previous ASAP database users and recreates new users.

You should not create multiple ASAP instances with the same Docker image.

Cleaning Up the Environment

To clean up the environment:

1. Delete the instance:

   ```
   $ASAP_CNTK/scripts/delete-instance.sh -p sr -i quick
   ```

2. Delete the name space, which in turn deletes the Kubernetes name space and the secrets:

   ```
   $ASAP_CNTK/scripts/unregister-namespace.sh -p sr -d -t target
   ```

   Note:

   `traefik` is the name of the target for registration of the name space. The script uses TRAEFIK_NS to find this target. Do not provide the "traefik" target if you are not using Traefik.

3. Delete the ASAP database users.

Troubleshooting Issues with the Scripts

This section provides information about troubleshooting some issues that you may encounter when running the scripts.

If you experience issues when running the scripts, do the following:

- Check the "Status" section of the domain to see if there is useful information:

  ```
  kubectl describe pod name -n sr
  ```

Cleanup Failed Instance

When a create-instance script fails, you must clean up the instance before making another attempt at instance creation.

> **Note:**
>
> Do not retry running the **create-instance** script or the **upgrade-instance** script immediately to fix any errors, as they would return errors. The **upgrade-instance** script may work but re-running it does not complete the operation.

To clean up the failed instance:

1. Delete the instance:

   ```
   $ASAP_CNTK/scripts/delete-instance.sh -p sr -i quick
   ```

Recreating an Instance

If you encounter issues when creating an instance, do not try to re-run the **create-instance.sh** script as this will fail. Instead, perform the cleanup activities and then run the following command:

```
$ASAP_CNTK/scripts/create-instance.sh -p sr -i quick
```

Accessing the OCA Client

Similar to the ASAP traditional deployment, ASAP cloud native also connects through the Order Control Application (OCA) client using thick and thin clients.

To configure the OCA client:

1. Add an entry to the **hosts** file to the machine (Unix or Windows) from which the user is planning to launch the OCA client to resolve DNS.
   This entry contains the IP address of the master node and hostname of the ASAP instance running in a cloud native. You can obtain the hostname from the **values.yaml** file.

   The hosts configuration file is located at:

   - On Windows: **C:\Windows\System32\drivers\etc\hosts**
   - On Linux: **/etc/hosts**

   For example:

   ```
   Kubernetes_Cluster_Master_IP <hostname provided in the values.yaml file>
   ```

2. Update the **tbl_server_info** table in the CTRL database.
   The hostname column contains the hostname mentioned in the **values.yaml** file and INFO column contains ‘HTTPS:<ingressport>’

For example:

```
update tbl_server_info set HOST_NAME='adminhost.asap.org';
update tbl_server_info set INFO='HTTP:30305';
update tbl_server_info set INFO='HTTPS:30443';
update tbl_server_info set INFO='HTTPS:80';
```

> **Note:**
>
> Based on the Ingressroute configuration use `HTTP` or `HTTPS`. The port number is the traefik nodeport.

3. Launch the thin client once the database is updated with the following URL:
   `https://<HOST>:<ingressport>/<ENVID>/OCA`
   For the thick client, the **OCA.cfg** file contains **HTTPS_PORT** inside **SESSION** specifying the ingressport.

Next Steps

The ASAP instance is ready to add the Order Balancer cloud native instance. The URL for adding the Order Balancer instance is:

```
t3://<service-name>.<namespace>.svc.cluster.local:<portnumber>
```

Where:

- `<service-name>` is the name of the service.
- `<namespace>` is the namespace of the ASAP instance.
- `<portnumber>` is the port number of the WebLogic Server Domain in the ASAP instance.

Here is an example for adding the ASAP instance to Order Balancer:

- ./addASAPServer -asapSrvName ASAP1 -asapSrvURL t3://cn96-service.sr.svc.cluster.local:7601 -asapSrvUser weblogic -asapSrvRequestQueue "System.CN96.ApplicationType.ServiceActivation.Application.1-0;7-4;ASAP.Comp.MessageQueue"

For more information about managing ASAP instances, see "Setting Up ASAP for High Availability" in *ASAP System Administrator's Guide*.

5

Creating an Order Balancer Cloud Native Image

Order Balancer cloud native image is required to create and manage Order Balancer cloud native instances. This chapter describes creating an Order Balancer cloud native image.

Order Balancer cloud native instance requires a container image and access to the database. The Order Balancer image is built on top of a Linux base image and the Order Balancer image builder script adds Java, WebLogic Server components, and Order Balancer.

The Order Balancer cloud native image is created using the Order Balancer cloud native builder toolkit. You should run the Order Balancer cloud native builder toolkit on Linux and it should have access to the local Docker daemon.

See the following topics for further details:

- Downloading the Order Balancer Cloud Native Image Builder
- Prerequisites for Creating an Order Balancer Image
- Creating the Order Balancer Cloud Native Image

Downloading the Order Balancer Cloud Native Image Builder

To build the Order Balancer cloud native Docker image, the **asap-img-builder.zip** file is required. This zip file is packaged as part of the ASAP cloud native tar file. For more information about downloading the Order Balancer cloud native Image Builder, see "Downloading the ASAP Cloud Native Artifacts".

See *ASAP Software Compatibility Matrix* for details about the latest recommended installer versions of Order Balancer.

The Order Balancer cloud native builder kit contains:

- The scripts to install the required packages.
- The scripts to install database client, install Java, WebLogic Server, and ASAP.

Prerequisites for Creating an Order Balancer Image

The prerequisites for building an Order Balancer cloud native image are:

- The Docker client and daemon on the build machine.
- Approximately 2 GB of swap space on the machine where the Docker daemon is running. By running the `free -m` command, you can verify the swap space.

> **Note:**
>
> If the required swap space is not available, contact your administrator.

- Order Balancer Installer file. For example, **ASAP.R7_4_0_P*x*.B*yy*.ob.tar** file. Download this file from the Oracle Software Delivery Cloud website:https://edelivery.oracle.com.
- Installers for WebLogic Server and JDK. Download these from the Oracle Software Delivery Cloud website: https://edelivery.oracle.com.
- Java, installed with **JAVA_HOME** set in the environment.
- TRAEFIK Ingress service node port should be ready where it is being deployed.
- Order Balancer database users should be created. For more information, see "Planning Your Installation" in *ASAP System Administrator's Guide*.

See *ASAP Software Compatibility Matrix* for details about the required and supported versions of the prerequisite software.

Creating the Order Balancer Cloud Native Image

The Order Balancer cloud native image builder tool builds the Order Balancer cloud native Docker image, which is then pushed to the Docker repository and deployed in the Kubernetes cluster. If the Docker repository is not available, you copy the image to all the worker nodes of the cluster.

The ASAP installer is packaged with the Order Balancer cloud native image builder and the cloud native toolkit.

Note:

After you download the installer, locate the Order Balancer cloud native image builder **asap-img-builder.zip** in the ASAP cloud native tar file. The Order Balancer Docker images are created automatically for ASAP 7.4.0.1 or later.

To create the Order Balancer cloud native image:

1. Copy the **asap-img-builder.zip** file to the machine where the Docker daemon is running.
2. Extract the contents of the zip file by running the following command:

   ```
   unzip asap-img-builder.zip
   ```
3. Copy the following installers to the **$asap-img-builder/installers** directory.
   - Order Balancer Installer
   - Installers for WebLogic Server and JDK
4. Update the following parameters in the **$asap-img-builder/ob.properties** file:

   ```
   ob.tar.file=ASAP.R7_4_0.B196.ob.tar
   ob.weblogic.username=weblogic
   ob.weblogic.password=
   ob.weblogic.port=7501
   ob.weblogic.domainName=ob
   ob.weblogic.channel.listenport=7502
   ob.weblogic.channel.publicport=30301
   ```

```
ob.ssl.incoming=0
ob.cacheexpiry=60
#Time in seconds
ob.all.servers.down.wait.interval=3600
ob.all.servers.down.retry.interval=120
ob.server.down.retry.interval=2
ob.server.poll.interval=60
ob.webservice.res.timeout=0
ob.asap.conn.timeout=10
## Values allowed: SEVERE, WARNING, INFO , FINE , FINEST ,ALL
ob.logger.info=INFO
ob.db.host=
ob.db.port=1521
ob.db.service.name=
ob.db.user=
ob.db.password=
ob.jms.user=
ob.jms.password=
```

where

- `ob.weblogic.username` is the user name to log in to WebLogic Server.
- `ob.weblogic.password` is the password to log in to WebLogic Server.
- `ob.weblogic.port` is the port of the WebLogic Server.
- `ob.weblogic.domainName` is the WebLogic Server domain.
- `ob.weblogic.channel.listenport` is the channel listen port of the WebLogic Server.
- `ob.weblogic.channel.publicport` is the public port of the WebLogic Server.
- `ob.ssl.incoming` is set to enable SSL on Order Balancer WebLogic Server. The default value is 0 which specifies non-SSL.
- `ob.cacheexpiry` specifies the duration in seconds that Order Balancer JPA shared query refreshes the cache. Cache is refreshed upon next request after the duration expires. The default value is 60.
- `ob.all.servers.down.wait.interval` specifies the duration in seconds that Order Balancer waits before routing the request back to queue when all the ASAP instances are down. The default value is 3600.
- `ob.all.servers.down.retry.interval` specifies the duration in seconds that Order Balancer waits before retrying to connect to fetch for an active ASAP member instance while waiting when all servers are down. The default value is 120.
- `ob.server.down.retry.interval` specifies the duration in seconds that Order Balancer waits before reattempting to route the order to the same instance. If the re-attempt fails, the instance is marked as down. The default value is 2.
- `ob.server.poll.interval` specifies the duration in seconds that Order Balancer waits before it retries to check the ASAP instance status. The default value is 60.
- `ob.webservice.res.timeout` specifies the duration in seconds that Order Balancer waits for response before the read times-out. Order Balancer Web Service waits for a response from ASAP member instance after invoking the operation. A value of zero means Order Balancer will wait indefinitely until it receives a response from ASAP. The default value is 0 seconds (no read time-out).

- `ob.asap.conn.timeout` specifies the duration in seconds that Order Balancer reattempts the connection to the ASAP instance. The default value is 10.
- `ob.logger.info` specifies the log level for initializing the Order Balancer application root logger. The valid values are SEVERE, WARNING, INFO, FINE, FINEST, and ALL.
- `ob.db.host` is the database host name or IP address.
- `ob.db.port` is the database port.
- `ob.db.service.name` is the database service name.
- `ob.db.user` is the database user name.
- `ob.db.password` is the database password.
- `ob.jms.user` is the JMS user.
- `ob.jms.password` is the JMS password.

> **Note:**
>
> Do not add an ASAP instance when you are building the Order Balancer Docker image. The wallet store is mounted dynamically in the Kubernetes cluster. The wallet files created in the Docker image are not accessible in the Kubernetes Pod.

In the cloud native deployment, the WebLogic domain is non-SSL and the ingress controller is configured as SSL.

5. Update the **HTTPS_PROXY** and **HTTP_PROXY** variables in the **build_ob_env.sh** script:

```
base_image=oraclelinux:8
HTTPS_PROXY=
HTTP_PROXY=

# Docker details
OB_IMAGE_TAG="obcn:7.4.0.0.0"
OB_VOLUME=obhost_volume
OB_CONTAINER="ob-c"
DOCKER_HOSTNAME="obhost"

# Installer filenames
WEBLOGIC_DOMAIN=/u01/oracle/user_projects/domains/
JDK_FILE=jdk-8u321-linux-x64.tar.gz
FMW_FILE=fmw_14.1.1.0.0_wls_lite_Disk1_1of1.zip

# Installation locations
JAVA_HOME=/usr/lib/jvm/java/jdk1.8.0_321
PATH=$JAVA_HOME/bin:$PATH
WEBLOGIC_HOME=/home/oracle/weblogic141100
```

Note:

The file names of JDK_FILE and FMW_FILE variables must match with the file names in the **/asap-img-builder/installers/** folder.

6. Run the **build-asap-images.sh** script to build the Order Balancer docker images:

   ```
   ./build-asap-images.sh -i ob
   ```

   The script creates the Order Builder Docker images by running the docker container and commits the Order Builder image.

6

Creating an Order Balancer Cloud Native Instance

This chapter describes how to create an Order Balancer cloud native instance in your cloud environment using the operational scripts and the base Order Balancer configuration provided in the Order Balancer cloud native toolkit. You can create an Order Balancer instance quickly to become familiar with the process, explore the configuration, and structure your own project. You can create multiple Order Balancer instances by using the same Order Balancer Docker image. This procedure is intended to validate that you are able to create an Order Balancer instance in your environment.

Before you create an Order Balancer instance, you must do the following:

- Download the ASAP cloud native tar file and extract the **ob-cntk.zip** file. For more information about downloading the Order Balancer cloud native toolkit, see "Downloading the ASAP Cloud Native Artifacts".
- Install the Traefik container images

Installing the Order Balancer Artifacts and the Toolkit

Build container image for the Order Balancer application using the Order Balancer cloud native Image Builder.

You must create a private Docker repository for this image, ensuring that all nodes in the cluster have access to the repository. See "About Container Image Management" for more details.

Copy the Order Balancer cloud native toolkit that is the **ob-cntk.zip** file to one of the nodes in the Kubernetes cluster and do the following:

- **On Oracle Linux**: Where Kubernetes is hosted on Oracle Linux, download and extract the tar archive to each host that has connectivity to the Kubernetes cluster.
- **On OKE**: For an environment where Kubernetes is running in OKE, extract the contents of the tar archive on each OKE client host. The OKE client host is the bastion host/s that is set up to communicate with the OKE cluster.

Set the variable for the installation directory by running the following command:

```
$ export OB_CNTK=ob_cntk_path
```

Where *ob_cntk_path* is the installation directory of the Order Balancer cloud native toolkit.

Installing the Traefik Container Image

To leverage the Order Balancer cloud native samples that integrate with Traefik, the Kubernetes environment must have the Traefik ingress controller installed and configured.

If you are working in an environment where the Kubernetes cluster is shared, confirm whether Traefik has already been installed and configured for Order Balancer cloud native. If Traefik is already installed and configured, set your `TRAEFIK_NS` environment variable to the appropriate name space.

The instance of Traefik that you installed to validate your cloud environment must be removed as it does not leverage the Order Balancer cloud native samples. Ensure that you have removed this installation in addition to purging the Helm release. Check that any roles and rolebindings created by Traefik are removed. There could be a **clusterrole** and **clusterrolebinding** called "traefik-operator". There could also be a **role** and **rolebinding** called "traefik-operator" in the $TRAEFIK_NS name space. Delete all of these before you set up Traefik.

To download and install the Traefik container image:

1. Ensure that Docker in your Kubernetes cluster can pull images from Docker Hub. See *ASAP Compatibility Matrix* for the required and supported versions of the Traefik image.
2. Run the following command to create a name space ensuring that it does not already exist:

   Note:

   You might want to add the traefik name space to the environment setup such as `.bashrc`.

   ```
   kubectl get namespaces
   export TRAEFIK_NS=traefik
   kubectl create namespace $TRAEFIK_NS
   ```

3. Run the following commands to install Traefik using the **$OB_CNTK/samples/charts/traefik/values.yaml** file in the samples:

   Note:

   Set **kubernetes.namespaces** and the chart version specifically using command-line.

   ```
   helm repo add traefik repo_link
   helm install traefik-operator traefik/traefik \
    --namespace $TRAEFIK_NS \
    --values $OB_CNTK/samples/charts/traefik/values.yaml \
     --set "kubernetes.namespaces={$TRAEFIK_NS}"
   ```

   where *repo_link* is https://helm.traefik.io/traefik or https://traefik.github.io/charts based on the helm chart version. For more details, see: https://github.com/traefik/traefik-helm-chart

After the installation, Traefik monitors the name spaces listed in its `kubernetes.namespaces` field for Ingress objects. The scripts in the toolkit manage this name space list as part of creating and tearing down Order Balancer cloud native projects.

When the **values.yaml** Traefik sample in the Order Balancer cloud native toolkit is used as is, Traefik is exposed to the network outside of the Kubernetes cluster through port 30305. To use a different port, edit the YAML file before installing Traefik. Traefik metrics are also available for Prometheus to scrape from the standard annotations.

Traefik function can be viewed using the Traefik dashboard. Create the Traefik dashboard by running the instructions provided in the **$OB_CNTK/samples/charts/traefik/traefik-dashboard.yaml** file. To access this dashboard, the URL is: `http://traefik.asap.org`. This is if you use the **values.yaml** file provided with the Order Balancer cloud native toolkit; it is possible to change the hostname as well as the port to your desired values.

Creating an Order Balancer Instance

This section describes how to create an Order Balancer instance.

Setting Environment Variables

ASAP cloud native relies on access to certain environment variables to run seamlessly. Ensure the following variables are set in your environment:

- Path to your private specification repository
- Traefik name space

To set the environment variables:

- Set the `TRAEFIK_NS` variable for Traefik name space as follows:

  ```
  $ export TRAEFIK_NS=Treafik Namespace
  ```

Creating Secrets

You must store sensitive data and credential information in the form of Kubernetes Secrets that the scripts and Helm charts in the toolkit consume. Managing secrets is out of the scope of the toolkit and must be implemented while adhering to your organization's corporate policies. Additionally, ASAP cloud native does not establish password policies.

For an Order Balancer could native instance, the following secrets are required:

- **imagepull-secret**: If the private registry or repository is password protected, create this secret.
- **tls-secret**: If the traefik ingress is ssl-enabled, create this secret. For more information about creating **tls-secret**, see "Setting Up ASAP Cloud Native for Incoming Access."

To create imagepull-secret:

1. Run the following command:

   ```
   docker login
   ```

2. Enter the credentials or access token.
   The login process creates or updates the `config.json` file that contains an authorization token.

3. Run the following command to run the secret:

   ```
   kubectl create secret generic asap-imagepull -n namespace
   --from-file=.dockerconfigjson=$HOME/.docker/config.json --
   type=kubernetes.io/dockerconfigjson
   ```

Registering the Namespace

After you set the environment variables, register the name space. To register the name space, run the following command:

```
$OB_CNTK/scripts/register-namespace.sh -p sr -t targets
# For example, $OB_CNTK/scripts/register-namespace.sh -p sr -t traefik
```

Where `-p` is the namespace where Order Balancer is being deployed.

Note:

`traefik` is the name of the targets for registration of the namespace sr. The script uses **TRAEFIK_NS** to find these targets. Do not provide the `traefik` target if you are not using Traefik.

Creating an Order Balancer Instance

This procedure describes how to create an Order Balancer instance in your environment using the scripts that are provided with the toolkit.

To create an Order Balancer instance:

1. Update the `$OB_CNTK/charts/asap/values.yaml` file with the following values:

   ```
   replicaCount: 1
   image:
     repository: obcn
     pullPolicy: IfNotPresent
     tag: "7.4.0.0"
   imagePullSecrets:
     - name: asap-imagepull
   obEnv:
     envid: ob96
     port: 7501
     host: obhost
     domain: ob
   persistence:
     enabled: true

   nameOverride: ""
   fullnameOverride: ""

   serviceAccount:
     # Specifies whether a service account should be created
   ```

```
  create: true
  # Annotations to add to the service account
  annotations: {}
  # The name of the service account to use.
  # If not set and create is true, a name is generated using the fullname
template
  name: ""

podAnnotations: {}


podSecurityContext: {}
  # fsGroup: 2000

securityContext: {}
  # capabilities:
  #   drop:
  #   - ALL
  # readOnlyRootFilesystem: true
  # runAsNonRoot: true
  # runAsUser: 1000
servicechannel:
  name: channelport
  type: ClusterIP
  port: 7502

service:
  name: adminport
  type: ClusterIP
  port: 7501

ingress:
  type: TRAEFIK
  enabled: true
  sslIncoming: false
  adminsslhostname: adminobhostssl.asap.org
  adminhostname: adminobhost.asap.org

  secretName: project-instance-obcn-tls-cert

  hosts:
    - host: adminobhost.asap.org
      paths: []
  tls: []
  #  - secretName: chart-example-tls
  #    hosts:
  #      - chart-example.local

resources: {}

autoscaling:
  enabled: false

nodeSelector: {}
```

```
tolerations: []

# To avoid co-locating two OB replicas/pods on the same node remove
{} and
# uncomment below lines. Replace <OB-ENV-ID> with OB Env ID
affinity: {}
#  podAntiAffinity:
#    requiredDuringSchedulingIgnoredDuringExecution:
#      - labelSelector:
#          matchExpressions:
#            - key: "app"
#              operator: In
#              values:
#              - obEnv.envid
#        topologyKey: "kubernetes.io/hostname"
```

Where:

- *replicaCount* is the number of Order Balancer instances to be created.
- *repository* is the repository name of the configured container registry.
- *tag* is the version name that is used when you create a final Docker image from the container.
- *name* is the name of the imagepull-secret if it is configured. For more information about imagepull-secret, see "Creating Secrets".
- *envid* is the unique environment ID of the instance. This ID should include only the lower-case alphanumeric characters. For example, asapinstance1.
- *port* is the port number of the WebLogic Server where Order Balancer is deployed.
- *host* is the hostname of the docker container.
- *domain* is the Order Balancer WebLogic domain name as configured in **ob.properties** while creating image.
- `servicechannel`.*port* is the channel port when you create a channel in the WebLogic domain.
- `service`.*port* is the admin port of the WebLogic Server.
- *type* is the ingress controller type. This type can be TRAEFIK or GENERIC or OTHER.
- *enabled* is the status of the ingress controller whether it is enabled or not. The value is *true* or *false*. By default, this is set to *true*.
- *sslIncoming* is the status of the SSL/TLS configuration on incoming connections whether it is enabled or not. The configuration value is *true* or *false*. By default, this is set to *false*. If you want to set the value to **true**, create keys, certificate, and secret by following the instructions in the "Setting Up ASAP Cloud Native for Incoming Access" section.
- *adminsslhostname* is the hostname of the https access.
- *adminhostname* is the hostname of the http access.

- *secretName* is the secret name of the certificate created for SSL/TLS. For more information about creating keys and secret name, see "Setting Up ASAP Cloud Native for Incoming Access."
- *podAntiAffinity* can be uncommented to avoid co-locating two Order Balancer pods on the same node.
- *requiredDuringSchedulingIgnoredDuringExecution* tells the Kubernetes Scheduler that it should never co-locate two Pods which have *app* label as *envid* in the domain defined by the *topologyKey.*
- *topologyKey kubernetes.io/hostname* indicates that the domain is an individual node.

> **Note:**
>
> `adminsslhostname` and `secretName` are applicable only if `sslIncoming` is set to **true**.

2. Create PV and PVC for the Order Balancer instance. It is a mandatory step for Order Balancer instance. The pv path is used to store the wallet and logs of the ASAP instance added to the Order Balancer. The sample files are available in the **ob_cntk.zip at $OB_CNTK/samples/nfs/** file.

```
# This is pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: <project>-<obEnv.envid>-nfs-pv
  labels:
    type: local
spec:
  storageClassName: wallet
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteMany
# Valid values are Retain, Delete or Recycle
persistentVolumeReclaimPolicy: Retain
# hostPath:
  nfs:
    server: <server>
    path: <path>
# This is pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: <obEnv.envid>-pvc
  namespace: <project>
spec:
  storageClassName: wallet
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
```

Where:

- <project> is the namespace. In the above example, value is *sr*.
- <*obEnv.envid*> is the environment ID provided in the **$OB_CNTK/samples/charts/traefik/values.yaml** file. In the above example, value is ob96.

The below paths from both Order Balancer pods are mounted on to the PV path as:

For wallet files: `/u01/oracle/user_projects/domains/<domainName>/oracle_communications/asap` to `PV_mount_path/asap`

For log files of each Order Balancer pod: `/u01/oracle/user_projects/domains/<domainName>/oracle_communications/logs` to `PV_mount_path/OB_pod_name/logs`

For example, `/mnt/OB99-1/logs and /mnt/OB99-0/logs`

3. Run the following command to create pv and pvc:

```
Kubectl apply -f pv.yaml
Kubectl apply -f pvc.yaml -n sr
```

4. Verify that whether pv and pvc are created successfully or not by running the following command:

```
kubectl get pv
kubectl get pvc -n sr
```

5. Run the following command to create the Order Balancer instance:

```
$OB_CNTK/scripts/create-instance.sh -p sr -i quick
```

The **create-instance.sh** script uses the Helm chart located in the **charts/ob** directory to deploy the Order Balancer docker image, service, and ingress controller for your instance. If the script fails, see "Troubleshooting Issues with the Scripts" before you make additional attempts.

6. Validate the important input details such as Image name and tag, specification files used (Values Applied), hostname, and port for ingress routing:

```
$OB_CNTK/scripts/create-instance.sh -p sr -i quick

Calling helm lint
==> Linting ../charts/ob
[INFO] Chart.yaml: icon is recommended

1 chart(s) linted, 0 chart(s) failed
Project Namespace : sr
Instance Fullname : sr-quick
NAME              : sr-quick
LB_PORT           : 30305
LAST DEPLOYED     : Timestamp
```

```
NAMESPACE          : sr
STATUS             : deployed
REVISION           : 1
TEST SUITE         : n/a
```

If PV/PVC is not configured before, the Pod will be in the pending state as shown below:

```
Kubectl describe pod <pod-name> -n sr
Events:
  Type     Reason            Age    From                 Message
  ----     ------            ----   ----                 -------
  Warning  FailedScheduling  2m31s  default-scheduler
persistentvolumeclaim "sr-pvc" not found
  Warning  FailedScheduling  2m31s  default-scheduler
persistentvolumeclaim "sr-pvc" not found
```

7. When the Pod state shows **READY 1/1**, your Order Balancer instance is up and running.

The base hostname required to access this instance using HTTP is `adminobhostnonssl.asap.org`. See "Planning and Validating Your Cloud Environment" for details about hostname resolution.

The **create-instance** script prints out the following valuable information that you can use when you work with your Order Balancer domain:

- The URL for accessing the WebLogic UI, which is provided through the ingress controller at host: `http://adminobhostnonssl.asap.org:30305/console`.

Validating the Order Balancer Instance

After creating an instance, you can validate it by accessing the WebLogic Server console.

Run the following command to display the Pod details of the Order Balancer instances that you have created:

```
$ kubectl get all -n OB_namespace
NAME                       READY        STATUS     RESTARTS    AGE
pod/ob96-0                  1/1        Running         0       24h
pod/ob96-1                  1/1        Running         0       24h
NAME                         TYPE        CLUSTER-IP    EXTERNAL-IP
PORT(S)            AGE
service/ob96-hlservice     ClusterIP    None            <none>     6723/
TCP,6724/TCP    3d18h
service/ob96-service       ClusterIP    10.106.240.147 <none>     6723/
TCP,6724/TCP    3d18h
NAME                        READY    AGE
statefulset.apps/ob96      2/2      3d18h
```

Note:

After the Order Balancer instances are created, it may take a few minutes to start Order Balancer servers.

To access WebLogic Administration Console outside the cluster, enter the following URL in the browser:

For non-SSL:

```
http://adminhostnonssl.asap.org:30305/console
```

For SSL:

```
https://adminhostssl.asap.org:30443/console
```

The system prompts for the user name and password. Enter the WebLogic domain user name and password.

Update the hosts file with the **hostname** and **master_ip** address on the machine where the URL is getting accessed.

Note:

The hosts file is located in **/etc/hosts** on Linux and MacOS machines and in **C:\Windows\System32\drivers\etc\hosts** on Windows machines.

```
ip_address adminhostnonssl.asap.org
```

Scaling the Order Balancer Instance

You can create multiple instances of Order Balancer by setting the *replicaCount* accordingly in the **values.yaml** file. The Order Balancer helm chart templates create multiple instances as StatefulSet pods. The unique pod names are automatically configured by suffixing an ordinal number with the StatefulSet Name. For example, if the StatefulSet Name is OBapp, the pod names are OBapp-0, OBapp-1 and so on. The StatefulSet policy ensures that only one pod with the given name is run at a given point of time.

The number of Order Balancer instances can be scaled up or down by modifying the replicas count in StatefulSet:

```
kubectl scale statefulsets stateful_set_name --replicas=replica_count
```

Where

stateful_set_name is the name of the StatefulSet. In this example, it is OBapp.

replica_count is the number of the Order Balancer instances to be created.

Deleting and Recreating Your Order Balancer Instance

Deleting Your Order Balancer Instance

To delete your Order Balancer instance, run the following command:

```
$OB_CNTK/scripts/delete-instance.sh -p sr -i quick
```

Re-creating Your Order Balancer Instance

When you delete an Order Balancer instance, the database state for that instance still remains unaffected. You can re-create multiple Order Balancer instance by using the same Docker image.

To re-create an Order Balancer instance, run the following command:

```
$OB_CNTK/scripts/create-instance.sh -p sr -i quick
```

Cleaning Up the Environment

To clean up the environment:

1. Delete the instance:

   ```
   $OB_CNTK/scripts/delete-instance.sh -p sr -i quick
   ```

2. Delete the name space, which in turn deletes the Kubernetes name space and the secrets:

   ```
   $OB_CNTK/scripts/unregister-namespace.sh -p sr -d -t target
   ```

   Note:

   `traefik` is the name of the target for registration of the name space. The script uses TRAEFIK_NS to find this target. Do not provide the "traefik" target if you are not using Traefik.

3. Delete the Order Balancer database users.
4. Delete pv and pvc using the following commands:

   ```
   Kubectl delete pv <pv-name>
   Kubectl delete pvc <pvc-name> -n sr
   ```

Troubleshooting Issues with the Scripts

This section provides information about troubleshooting some issues that you may come across when running the scripts.

If you experience issues when running the scripts, do the following:

- Check the "Status" section of the domain to see if there is useful information:

  ```
  kubectl describe pod name -n sr
  ```

Cleanup Failed Instance

When a create-instance script fails, you must clean up the instance before making another attempt at instance creation.

Note:

Do not retry running the **create-instance** script or the **upgrade-instance** script immediately to fix any errors, as they would return errors. The **upgrade-instance** script may work but re-running it does not complete the operation.

To clean up the failed instance:

1. Delete the instance:

   ```
   $OB_CNTK/scripts/delete-instance.sh -p sr -i quick
   ```

Recreating an Instance

If you encounter issues when creating an instance, do not try to re-run the **create-instance.sh** script as this will fail. Instead, perform the cleanup activities and then run the following command:

```
$OB_CNTK/scripts/create-instance.sh -p sr -i quick
```

Next Steps

The Order Balancer instance is ready to add the ASAP cloud native instance.

For more information about managing ASAP instances, see "Setting Up ASAP for High Availability" in *ASAP System Administrator's Guide*.

7

Planning Infrastructure

In Creating an ASAP Cloud Native Instance, you learned how to create an ASAP instance in your cloud native environment. This chapter provides details about setting up infrastructure and structuring ASAP instances for your organization.

See the following topics:

- Sizing Considerations
- Securing Operations in Kubernetes Cluster

Sizing Considerations

The hardware utilization for an ASAP cloud native deployment is approximately the same as that of the ASAP traditional deployment.

Consider the following when sizing for your cloud native deployment:

- For ASAP cloud native, ensure that the database is sized to account for work orders residing in the database. For details, see "ASAP Oracle Database Tablespace Sizing Requirements" in *ASAP Installation Guide*.
- Oracle recommends sizing using a configuration as a building block by adjusting the **ASAP.cfg** file to meet target order volumes.

  > **Note:**
  >
  > Update the ASAP.cfg file when you build the Docker image.

  For more details, see "Installing a Pre-tuned Configuration", in *ASAP System Administrator's Guide*.
- In addition to planning hardware for a production instance, Oracle recommends planning for a Disaster Recovery size and key non-production instances to support functional, integration, and performance tests. The Disaster Recovery instance can be created against an Active Data Guard Standby database when needed and terminated when no longer needed to improve hardware utilization.
- Non-production instances can likewise be created when needed, either against new or existing database instances.

Contact Oracle Support for further assistance with sizing.

Securing Operations in Kubernetes Cluster

This section describes how to secure the operations of ASAP and Order Balancer cloud native users in a Kubernetes cluster. A well-organized deployment of ASAP and Order Balancer cloud native ensures that individual users have specific privileges that are limited to the requirements for their approved actions. The Kubernetes objects concerned are service accounts and RBAC objects.

All ASAP and Order Balancer cloud native users fall into the following three categories:

- Infrastructure Administrator
- Project Administrator
- ASAP User

Infrastructure Administrator

Infrastructure Administrators perform the following operations:

- Create a project for ASAP and Order Balancer cloud native and configure the projects
- After creating a new project, run the **register-namespace.sh** script provided with the ASAP cloud native toolkit
- Before deleting the ASAP and Order Balancer cloud native projects, run the **unregister-namespace.sh** script
- Delete the ASAP and Order Balancer cloud native projects

Project Administrator

Project Administrators can perform all the tasks related to an instance level ASAP and and Order Balancer cloud native deployments within a given project. This includes creating and updating ASAP and Order Balancer cloud native instances. A project administrator can work on one specific project. However, a given human user may be assigned Project Administrator privileges on more than one project.

RBAC Requirements

The RBAC requirements for Traefik is documented in its user guide. The Infrastructure Administrator must be able to create and delete name spaces and Traefik name space (if Traefik is used as the ingress controller). Depending on the specifics of your Kubernetes cluster and RBAC environment, this may require cluster-admin privileges.

The Project Administrator has limited RBAC privileges. For a start, it would be limited to only that project's name space. Further, it would be limited to the set of actions and objects that the instance-related scripts manipulate when run by the Project Administrator. This set of actions and objects is documented in the ASAP and Order Balancer cloud native toolkit sample located in the **samples/rbac** directory.

Structuring Permissions Using the RBAC Sample Files

There are many ways to structure permissions within a Kubernetes cluster. There are clustering applications and platforms that add their own management and control of these permissions. Given this, the ASAP and Order Balancer cloud native toolkit provides a set of RBAC files as a sample. You will have to translate this sample into a configuration that is appropriate for your environment. These samples are in the **samples/rbac** directory within the toolkit.

The key files are **project-admin-role.yaml** and **project-admin-rolebinding.yaml**. These files govern the basic RBAC for a Project Administrator.

Do the following with these files:

1. Make a copy of both these files for each particular project, renaming them with the *project*/*namespace* name in place of "project". For example, for a project called "biz", these files would be **biz-admin-role.yaml** and **biz-admin-rolebinding.yaml**.
2. Edit both the files, replacing all occurrences of *project* with the actual *project*/*namespace* name.

   For the *project***-admin-rolebinding.yaml** file, replace the contents of the "subjects" section with the list of users who will act as Project Administrators for this particular project.

   Alternatively, replace the contents with reference to a group that contains all users who will act as Project Administrators for this project.
3. Once both files are ready, they can be activated in the Kubernetes cluster by the cluster administrator using **kubectl apply -f** *filename*.

   It is strongly recommended that these files be version controlled as they form part of the overall ASAP cloud native configuration.

In addition to the main Project Administrator role and its binding, the samples contain two additional and optional role-rolebinding sets:

- **project-admin-addon-role.yaml** and **project-admin-addon-rolebinding.yaml**: This role is per project and is an optional adjunct to the main Project Administrator role. It contains authorization for resources and actions in the project name space that is not required by the toolkit, but might be of some use to the Project Administrator for debugging purposes.

8

Exploring Alternate Configuration Options

The ASAP cloud native toolkit provides samples and documentation for setting up your ASAP cloud native environment using standard configuration options. However, you can choose to explore alternate configuration options for setting up your environment, based on your requirements. This chapter describes alternate configurations you can explore, allowing you to decide how best to configure your ASAP cloud native environment to suit your needs.

You can choose alternate configuration options for the following:

- Choosing Worker Nodes for Running ASAP Cloud Native
- Working with Ingress, Ingress Controller, and External Load Balancer
- Using an Alternate Ingress Controller
- Managing Logs
- Managing ASAP Cloud Native Metrics

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

Choosing Worker Nodes for Running ASAP Cloud Native

By default, ASAP cloud native has its pods scheduled on all worker nodes in the Kubernetes cluster in which it is installed. However, in some situations, you may want to choose a subset of nodes where pods are scheduled.

For example, these situations include:

- Non-license restrictions: Limitation on the deployment of ASAP on specific worker nodes per each team for reasons such as capacity management, chargeback, budgetary reasons, and so on.

To choose a subset of nodes where pods are scheduled, you can use the configuration in the **asap-cntk/charts/asap/values.yaml** file.

```
# If ASAP cloud native instances must be targeted to a subset of worker
nodes in the
# Kubernetes cluster, tag those nodes with a label name and value, and choose
# that label+value here.
#   key    : any node label key
#   values : list of values to choose the node.
#            If any of the values is found for the above label key, then that
#            node is included in the pod scheduling algorithm.
#
# This can be overriden in instance specification if required.
nodeSelector: {} # This empty declaration should be removed if adding items
here.
#asapcnTargetNodes:
#  nodeLabel:
##   oracle.com/licensed-for-coherence is just an indicative example, any
label and its values can be used for choosing nodes.
```

```
#    key: oracle.com/licensed-for-coherence
#    values:
#      - true
```

Consider the following when you update the configuration:

- There is no restriction on the node label key. Any valid node label can be used.
- There can be multiple valid values for a key.

Working with Ingress, Ingress Controller, and External Load Balancer

A Kubernetes ingress is responsible for establishing access to back-end services. However, creating an ingress is not sufficient. An Ingress controller connects the back-end services with the front-end services based on Ingress rules. In ASAP cloud native, you can configure an ingress controller in the **asap-cntk/charts/asap/values.yaml** file. For example,

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

The Traefik ingress controller works by creating an operator in its own "traefik" name space and exposing a NodePort service. However, all ingress controllers do not behave the same way. To accommodate all types of ingress controllers, by default, the **values.yaml** file provides the `loadBalancerPort` parameter.

If an external load balancer is used, it needs to be connected to the NodePort service of the Ingress controller. Hence, `externalLoadBalancerIP` also needs to be present in the **values.yaml** file.

For the Traefik ingress controller, do the following:

- If an external load balancer is not configured, fetch `loadBalancerPort` by running the following command:

  ```
  $kubectl -n $TRAEFIK_NS get service traefik-operator --
  output=jsonpath="{..spec.ports[?(@.name=='http')].nodePort}"
  ```

- If an external load balancer is used, fetch `loadBalancerPort` by running the following command:

  ```
  kubectl -n $TRAEFIK_NS get service traefik-operator --
  output=jsonpath="{..spec.ports[?(@.name=='http')].port}"
  ```

Populate the values in the **values.yaml** file before invoking **create-instance.sh** command to create an instance:

```
# If external hardware or software load balancer is used, set this
value to that frontend host IP.
# If OCI load balancer is used, then set externalLoadBalancerIP from
OCI LBaaS
#externalLoadBalancerIP: ""
```

```
# For Traefik Ingress Controller:
# If external load balancer is used, then this would be 80, else traefik
pod's Nodeport (30305)
loadBalancerPort: 80
```

Note:

If you choose Traefik or any other ingress controller such as GENERIC or OTHER you can update the **ingress:** section in the **asap_cntk/charts/asap/values.yaml** file.

Using an Alternate Ingress Controller

By default, ASAP cloud native supports Traefik and provides sample files for integration. However, you can use any Ingress controller that supports host-based routing and session stickiness with cookies. ASAP cloud native uses the term "generic" ingress for scenarios where you want to leverage the Ingress capabilities that the Kubernetes platform may provide.

To use a generic ingress controller, you must create the ingress object and configure your ASAP instance to use it. The toolkit uses an ingress Helm chart (**$ASAP_CNTK/samples/charts/asap/templates/traefik-ingress.yaml**) and scripts for creating the ingress objects. If you want to use a generic ingress controller, these samples can be used as a reference and customized as necessary.

If your ASAP cloud native instance needs to secure incoming communications, then look at the **$ASAP_CNTK/samples/charts/asap/templates/traefik-ingress.yaml** file. This file demonstrates the configuration for a TLS-enabled Traefik ingress that can be used as a sample.

The host-based rules and the corresponding back-end Kubernetes service mapping are provided using the following definition:

- asapUID: Combination of *project-instance*. For example, **sr-quick**.

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

Table 8-1 Service Name and Service Ports for Ingress Rules

Rule	Service Name	Service Port	Purpose
instance.project.loadBalancerDomainName	*domainUID*-cluster-*clusterName*	8001	For access to ASAP through UI, XMLAPI, Web Services, and so on.
t3.*instance.project.loadBalancerDomainName*	t3.*instance.project.loadBalancerDomainName*	30303	ASAP T3 Channel access for WLST, JMS, and SAF clients.
admin.*instance.project.loadBalancerDomainName*	*domainUID*-admin	7001	For access to ASAP WebLogic Admin Console UI.

You must update the value of the `ingressController` parameter in the **$ASAP_CNTK/charts/asap/values.yaml** file. For example,

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

If any of the supported Ingress controllers or even a generic ingress does not meet your requirements, you can choose "OTHER".

By choosing this option, ASAP cloud native does not create or manage any ingress required for accessing the ASAP cloud native services. However, you may choose to create your own ingress objects based on the service and port details mentioned in the above table.

Note:

Regardless of the choice of Ingress controller, it is mandatory to provide the value of `loadBalancerPort` in one of the specification files. This is used for establishing a front-end cluster.

Managing Logs

ASAP cloud native generates traditional textual logs. By default, these log files are generated in the managed server pod but can be re-directed to a Persistent Volume Claim (PVC) supported by the underlying technology that you choose. See "Setting Up Persistent Storage" for details.

When you update the staging container, update the LOGDIR attribute in the **$ASAP_BASE/Environment_Profile** file:

```
# LOGDIR=/asaplogs
```

- The ASAP application logs can be found at: *pv-directory/asapinstance/*logs

Update the WebLogic logs path to the PVC mounted path.

Managing ASAP Cloud Native Metrics

ASAP WebLogic Server exposes the work order metrics deployed in ASAP cloud native. The following is the work order metrics path:

```
asapcn.metricspath: /ENV_ID/OrderMetrics
```

ASAP cloud native metrics expose the Order Balancer metrics along with the work order metrics. The following is the Order Balancer metrics path:

```
metrics_path: /ASAPOB/metrics
```

Configuring Prometheus for ASAP Cloud Native Metrics

ASAP provides the following metrics for monitoring based on the work order status:

- Completed
- Completed in last interval
- Loading Work Orders
- Failed
- Canceled
- In progress

Work order metrics can be queried with different parameter values. To query the work order metrics, use the following URL in the browser:

`http://host:port/env_id/OrderMetrics?query=parameter`

The supported parameter values are:

- **total**: Provides total work order count. This is the the default parameter used.
- **today**: Provides todays total work order count.
- **last_interval**: Provides total work order count in the last interval.
- **all**: Provides all the three work order counts (total + today + last_interval).

The Order Balancer metrics provided are:

- **ASAP Count**: Provides the number of ASAP instances that are registered with Order Balancer.
- **ASAP instance status**: Provides the status of each ASAP instance that is registered in Order Balancer.
- **Order Distribution**: Provides the order distribution between the ASAP instances.

Configure the scrape job in Prometheus by updating the **prometheus.yml** file as follows:

```
- job_name: 'asapmetrics'
    scrape_interval: 120s
    scrape_timeout: 60s
    metrics_path: /ENV_ID/OrderMetrics
    scheme: http/https
    basic_auth:
      username: WebLogic user name
      password: WebLogic password
    static_configs:
      - targets: ['hostname:port number']
    params:
         query: [all]

  - job_name: 'obmetrics'
    scrape_interval: 120s
    scrape_timeout: 60s
    metrics_path: /ASAPOB/metrics
    scheme: http/https
    basic_auth:
      username: WebLogic user name
      password: WebLogic password
    static_configs:
      - targets: ['hostname:port number']
```

Where

- *WebLogic user name* is the user name of WebLogic Server.
- *WebLogic password* is the password of WebLogic Server.
- *hostname* is the configured host name in the **values.yaml** file.
  - ASAP: **$asap_cntk/charts/asap/values.yaml**
  - Order Balancer: **$ob-cntk/charts/ob/values.yaml**
- *port number* is the traefik node port number.

Note:

The filter options are: **all**, **today**, and **total**.

If you use a filter, update `query: [filter]` in the **prometheus.yml** file.

If you do not use a filter, comment out `params: query: [filter]` in the **prometheus.yml** file.

If multiple ASAP instances are added, add the respective jobs in **prometheus.yml** file

Viewing ASAP Cloud Native Metrics Without Using Prometheus

You can view the ASAP cloud native metrics, such as the work order metrics and the Order Balancer metrics using the following URLs:

Work order metrics:

```
http://hostname:traefik_Port/ENV_ID/OrderMetrics
```

Order Balancer metrics:

```
http://hostname:traefik_Port/ASAPOB/metrics
```

Where

- *hostname* is the configured host name in the **values.yaml** file
  - ASAP: **$asap_cntk/charts/asap/values.yaml**
  - Order Balancer: **$ob-cntk/charts/ob/values.yaml**
- *traefik_Port* is the traefik node port number.

These only provide metrics of the WebLogic Server that is serving the request. They does not provide consolidated metrics for the entire cluster. Prometheus Query and Grafana dashboards provide consolidated metrics.

Viewing ASAP Cloud Native Metrics in Grafana

ASAP cloud native metrics and Order Balancer metrics scraped by Prometheus can be made available for further processing and visualization. The ASAP cloud native toolkit comes with sample Grafana dashboards to get you started with visualizations.

Import the dashboard JSON files from **$ASAP_CNTK/samples/grafana** into your Grafana environment.

The sample dashboard displays the following:

- ASAP Count
- ASAP Instance Status
- Order Distribution
- Work order count by order state
- Completed work order count in a configured interval

Exposed ASAP Order Metrics

The following ASAP metrics are exposed via ASAP Servlet APIs.

Order Metrics

The following table lists the order metrics exposed.

Table 8-2 Order Metrics Exposed via ASAP Servlet APIs

Name	Notes
asap_wo_complete_total	The total work orders in the completed state.
asap_wo_loading_total	The total work orders in the loading state.
asap_wo_failed_total	The total work orders in the failed state.
asap_wo_cancelled_total	The total work orders in the canceled state.
asap_wo_inprogress_total	The total work orders in the in progress state.
asap_wo_complete_last_interval	The total work orders that are in the completed state in the last interval.
asap_wo_complete_today	The total work orders that are in the completed state as of the current date.
asap_wo_loading_today	The total work orders that are in the loading state as of the current date.
asap_wo_failed_today	The total work orders that are in the failed state as of the current date.
asap_wo_cancelled_today	The total work orders that are in the canceled state as of the current date.
asap_wo_inprogress_today	The total work orders that are in the in-progress state as of the current date.

9

Integrating ASAP

Typical usage of ASAP involves the ASAP application receiving work orders from upstream. Upstream interacts with ASAP using t3/t3s or http/https. This chapter examines the considerations involved in integrating ASAP cloud native instances into a larger solution ecosystem.

This section describes the following topics and tasks:

- Integrating with ASAP cloud native instances
- Applying the WebLogic patch for external systems
- Configuring SAF on External Systems
- Setting up Secure Communication with SSL/TLS

Integrating With ASAP Cloud Native Instances

Functionally, the interaction requirements of ASAP do not change when ASAP is run in a cloud native environment. All of the categories of interaction that are applicable for connectivity with traditional ASAP instances are applicable and must be supported for ASAP cloud native.

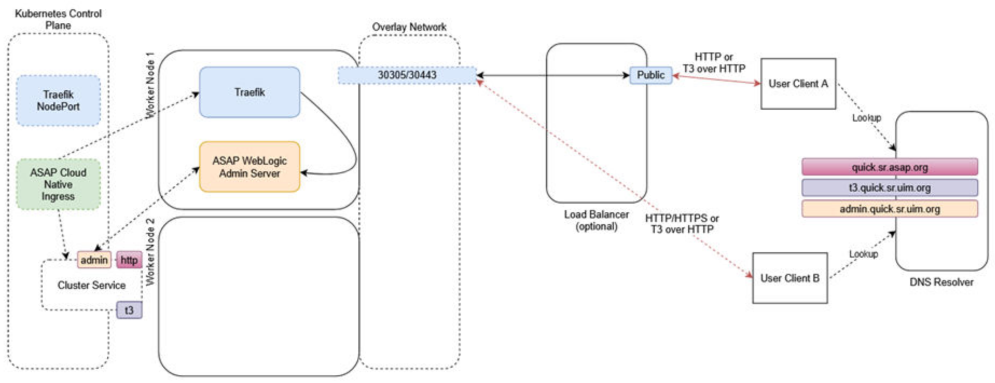
Note:

Connectivity with the OCA client and SRT are not supported in ASAP cloud native environment.

Connectivity Between the Building Blocks

The following diagram illustrates the connectivity between the building blocks in an ASAP cloud native environment using an example:

Figure 9-1 Connectivity Between Building Blocks in ASAP Cloud Native Environment



Invoking the ASAP cloud native Helm chart creates a new ASAP instance. In the above illustration, the name of the instance is "quick" and the name of the project is "sr". The instance consists of an ASAP pod and a Kubernetes service.

The Cluster Service contains endpoints for both HTTP and T3 traffic. The instance creation script creates the ASAP cloud native Ingress object. The Ingress object has metadata to trigger the Traefik ingress controller as a sample. Traefik responds by creating new front-ends with the configured "hostnames" for the cluster (**quick.sr.asap.org** and **t3.quick.sr.uim.org** in the illustration). The IngressRoute connects the hostname to the service exposed on the pod. The service is created on the ASAP WebLogic admin server port.

The prior installation of Traefik has already exposed Traefik itself via a selected port number (**30305** in the example) on each worker node.

Inbound HTTP Requests

An ASAP instance is exposed outside of the Kubernetes cluster for HTTP access via an Ingress Controller and potentially a Load Balancer.

Because the Traefik port (30305) is common to all ASAP cloud native instances in the cluster, Traefik must be able to distinguish between the incoming messages headed for different instances. It does this by differentiating on the basis of the "hostname" mentioned in the HTTP messages. This means that a client (User Client B in the illustration) must believe it is talking to the "host" **dev2.mobilecom.asap.org** when it sends HTTP messages to port 30305 on the access IP. This might be the Master node IP, or IP address of one of the worker nodes, depending on your cluster setup. The "DNS Resolver" provides this mapping.

In this mode of communication, there are concerns around resiliency and load distribution. For example, If the DNS Resolver always points to the IP address of Worker Node 1 when asked to resolve **dev2.mobilecom.asap.org**, then that Worker node ends up taking all the inbound traffic for the instance. If the DNS Resolver is configured to respond to any ***.mobilecom.asap.org** requests with that IP, then that worker node ends up taking all the inbound traffic for all the instances. Since this latter configuration in the DNS Resolver is desired, to minimize per-instance touches, the setup creates a bottleneck on Worker node 1. If Worker node 1 were to fail, the DNS Resolver would have to be updated to point ***.mobilecom.asap.org** to Worker node 2.

This leads to an interruption of access and requires intervention. The recommended pattern to avoid these concerns is for the DNS Resolver to be populated with all the applicable IP addresses as resolution targets (in our example, it would be populated with the IPs of both Worker node 1 and node 2), and have the Resolver return a random selection from that list.

An alternate mode of communication is to introduce a load balancer configured to balance incoming traffic to the Traefik ports on all the worker nodes. The DNS Resolver is still required, and the entry for ***.mobilecom.asap.org** points to the load balancer. Your load balancer documentation describes how to achieve resiliency and load management. With this setup, a user (User Client A in our example) sends a message to **dev2.mobilecom.asap.org**, which actually resolves to the load balancer - for instance, **http://dev2.mobilecom.asap.org:8080/OrderManagement/Login.jsp**. Here, 8080 is the public port of the load balancer. The load balancer sends this to Traefik, which routes the message, based on the "hostname" targeted by the message to the HTTP channel of the ASAP cloud native instance.

By adding the hostname resolution such that **admin.dev2.mobilecom.asap.org** also resolves to the Kubernetes cluster access IP (or Load Balancer IP), User Client B can access the WebLogic console via **http://admin.dev2.mobilecom.asap.org/console** and the credentials specified while setting up the "wlsadmin" secret for this instance.

Note:

Access to the WebLogic Admin console is provided for review and debugging use only. Do not use the console to change the system state or configuration. As a result, any such manual changes (whether using the console or using WLST or other such mechanisms) are not retained in pod reschedule or reboot scenarios. The only way to change the state or configuration of the WebLogic domain or the ASAP installation is inside the Docker image.

Inbound JMS Requests

JMS messages use the T3 protocol. Since Ingress Controllers and Load Balancers do not understand T3 for routing purposes, ASAP cloud native requires all incoming JMS traffic to be "T3 over HTTP". Hence, the messages are still HTTP but contain a T3 message as a payload. ASAP cloud native requires the clients to target the "t3 hostname" of the instance - **t3.dev2.mobilecom.asap.org**, in the example. This "t3 hostname" should behave identically as the regular "hostname" in terms of the DNS Resolver and the Load Balancer. Traefik however not only identifies the instance this message is meant for (dev2.mobilecom) but also that it targets the T3 channel of instance.

The "T3 over HTTP" requirement applies for all inbound JMS messages - whether generated by direct or foreign JMS API calls or generated by SAF. The procedure in SAF QuickStart explains the setup required by the message producer or SAF agent to achieve this encapsulation. If SAF is used, the fact that T3 is riding over HTTP does not affect the semantics of JMS. All the features such as reliable delivery, priority, and TTL, continue to be respected by the system. See "Applying the WebLogic Patch for External Systems".

An ASAP instance can be configured for secure access, which includes exposing the T3 endpoint outside the Kubernetes cluster for HTTPS access. See "Configuring Secure Incoming Access with SSL" for details on enabling SSL.

Applying the WebLogic Patch for External Systems

When an external system is configured with a SAF sender towards ASAP cloud native, using HTTP tunneling, a patch is required to ensure the SAF sender can connect to the ASAP cloud native instance. This is regardless of whether the connection resolves to an ingress controller or to a load balancer. Each such external system that communicates with ASAP through SAF must have the WebLogic patch 30656708 installed and configured, by adding `–Dweblogic.rjvm.allowUnknownHost=true` to the WebLogic startup parameters.

For environments where it is not possible to apply and configure this patch, a workaround is available. On each host running a Managed Server of the external system, add the following entries to the **/etc/hosts** file:

```
0.0.0.0 project-instance-ms1
0.0.0.0 project-instance-ms2
0.0.0.0 project-instance-ms3
0.0.0.0 project-instance-ms4
0.0.0.0 project-instance-ms5
0.0.0.0 project-instance-ms6
0.0.0.0 project-instance-ms7
0.0.0.0 project-instance-ms8
0.0.0.0 project-instance-ms9
0.0.0.0 project-instance-ms10
0.0.0.0 project-instance-ms11
0.0.0.0 project-instance-ms12
0.0.0.0 project-instance-ms13
0.0.0.0 project-instance-ms14
0.0.0.0 project-instance-ms15
0.0.0.0 project-instance-ms16
0.0.0.0 project-instance-ms17
0.0.0.0 project-instance-ms18
```

You should add these entries for all the ASAP cloud native instances that the external system interacts with. Set the IP address to 0.0.0.0. The server in the ASAP cloud native instance must be listed.

Configuring SAF On External Systems

To create SAF and JMS configuration on your external systems to communicate with the ASAP cloud native instance, use the configuration samples provided as part of the SAF sample as your guide.

It is important to retain the "Per-JVM" and "Exactly-Once" flags as provided in the sample.

All connection factories must have the "Per-JVM" flag, as must SAF foreign destinations.

Each external queue that is configured to use SAF must have its QoS set to "Exactly-Once".

Enabling Domain Trust

To enable domain trust, in your domain configuration, under **Advanced**, edit the **Credential** and **ConfirmCredential** fields with the same password you used to create the global trust secret in ASAP cloud native.

Setting Up Secure Communication with SSL/TLS

When ASAP cloud native is involved in secure communication with other systems, you should additionally configure SSL/TLS. The configuration may involve the WebLogic domain, the ingress controller, or the URL of remote endpoints, but it always involves participating in an SSL handshake with the other system. The procedures for setting up SSL use self-signed certificates for demonstration purposes. However, replace the steps as necessary to use signed certificates.

If an external client is communicating with ASAP cloud native instance by using SSL/TLS, you should configure secure incoming access with SSL.

Configuring Secure Incoming Access with SSL

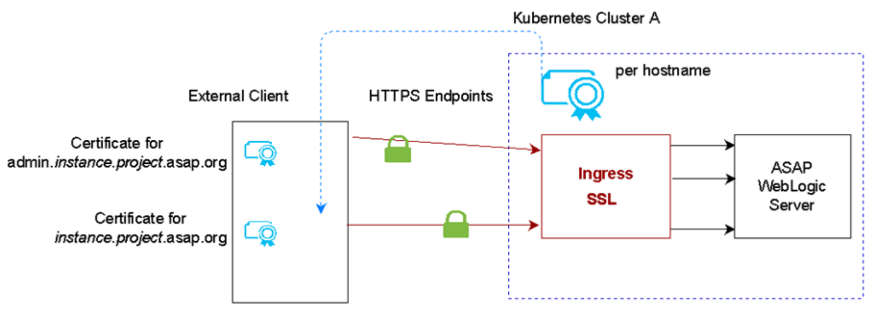
This section demonstrates how to secure incoming access to ASAP cloud native. In this scenario, SSL termination happens at the ingress. The traffic coming in from external clients must use one of the HTTPS endpoints. When SSL terminates at the ingress, it also means that communication within the cluster from Traefik ASAP cloud native instances is not secured.

The ASAP cloud native toolkit provides the sample configuration for Traefik ingress. If you use Voyager or other ingress, you can look at the **$ASAP-CNTK/charts/asap/templates/traefik-ingress.yaml** file to see what configuration is applied.

Generating SSL Certificates for Incoming Access

The following illustration shows when certificates are generated.

Figure 9-2   Generating SSL Certificates



When ASAP cloud native dictates secure communication, then it is responsible for generating the SSL certificates. These certificates must be provided to the appropriate client.

Setting Up ASAP Cloud Native for Incoming Access

The ingress controller routes unique hostnames to different backend services. You can see this if you look at the ingress controller YAML file (obtained by running **kubectl get ingress -n** *project ingress_name* **-o yaml**):

Note:

Traefik 2.x moved to use IngressRoute (a CustomResourceDefinition) instead of the Ingress object. If you are using Traefik, change all references of `ingress` to `ingressroute` in the following command :

```
rules:
- host: admin.instance.project.asap.org
  http:
    paths:
    - backend:
        serviceName: ENV_ID-service
        servicePort: 7601
```

To set up ASAP cloud native for incoming access:

1. Generate key pairs for each hostname corresponding to an endpoint that ASAP cloud native exposes to the outside world:

   ```
   # Create a directory to save your keys and certificates. This is
   for sample only. Proper management policies should be used to store
   private keys.

   mkdir $ASAP_CNTK/charts/asap/ssl

   # Generate key and certificates
   openssl req -x509 -nodes -days 365 -newkey rsa:2048 -
   keyout $ASAP_CNTK/charts/asap/ssl/admin.key -out $ASAP_CNTK/charts/
   asap/ssl/admin.crt -subj "/CN=admin.instance.project.asap.org"

   # Create secrets to hold each of the certificates. The secret name
   must be in the format below. Do not change the secret names
   kubectl create secret -n project tls project-instance-admin-tls-
   cert --key $ASAP_CNTK/charts/asap/ssl/admin.key --cert $ASAP_CNTK/
   charts/asap/ssl/admin.crt
   ```

2. Edit the **values.yaml** file and set `incoming` to **true**:

   ```
   ingress:
     sslIncoming: true
   ```

3. After creating the instance by running the **create-instance.sh** script, you can validate the configuration by describing the ingress controller for your instance.

You should see each of the certificates you generated, terminating one of the hostnames:

```
kubectl get ingress -n project
```

Once you have the name of your ingress, run the following command:

```
kubectl describe ingress -n project ingress

TLS:
  project-instance-admin-tls-cert terminates
admin.instance.project.asap.org
```

Now the ASAP instance is created with the secure connection to the ingress controller.

Configuring Incoming HTTP and JMS Connectivity for External Clients

This section describes how to configure incoming HTTP and JMS connectivity for external clients.

Note:

Remember to have your DNS resolution set up on any remote hosts that will connect to the ASAP cloud native instance.

Incoming HTTPS Connectivity

External Web clients that are connecting to ASAP cloud native must be configured to accept the certificates from ASAP cloud native. They will then connect using the HTTPS endpoint and port **30443**.

Incoming JMS Connectivity

For external servers that are connected to ASAP cloud native through JMS queues, the certificate for the t3 endpoint needs to be copied to the host where the external client is running.

If your external WebLogic configuration uses "CustomIdentityAndJavaSTandardTrust", follow these instructions to upload the certificate to the Java Standard Trust. If, however, you are using a CustomTrust, then you must upload the certificate into the custom trust keystore.

The keytool is found in the **bin** directory of your JDK installation. The alias should uniquely describe the environment where this certificate is from.

```
./keytool -importcert -v -trustcacerts -alias alias -file /path-to-copied-t3-
certificate/t3.crt -keystore /path-to-jdk/jdk1.8.0_202/jre/lib/security/
cacerts -storepass default_password


# For example
./keytool -importcert -v -trustcacerts -alias asapcn -file /scratch/t3.crt -
keystore /jdk1.8.0_202/jre/lib/security/cacerts -storepass default_password
```

Debugging SSL

To debug SSL, do the following:

- Verify Hostname
- Enable SSL logging

Verifying Hostname

When the keystore is generated for the on-premise server, if FQDN is not specified, then you may have to disable hostname verification. This is not secure and should only be done in development environments.

To do so, when you build the Docker image, update the **build_env.sh** script and add the following Java options:

```
    #JAVA_OPTIONS for all managed servers at project level
    java_options: "-
Dweblogic.security.SSL.ignoreHostnameVerification=true"
```

Enabling SSL Logging

When trying to establish the handshake between servers, it is important to enabling the SSL-specific logging.

To do so, when you build the Docker image, update the **build_env.sh** script and append the following Java options:

```
  project:
    #JAVA_OPTIONS for all managed servers at project level
    java_options: "-Dweblogic.StdoutDebugEnabled=true -Dssl.debug=true
-Dweblogic.security.SSL.verbose=true -
Dweblogic.debug.DebugSecuritySSL=true -Djavax.net.debug=ssl"
```

10 Upgrading the ASAP Cloud Native Environment

This chapter describes the tasks you perform in order to apply a change or upgrade to a component in the cloud native environment.

ASAP supports only one replica per instance. If the same Docker image is used in two instances, the behavior is undefined. Due to these constraints, ASAP supports only offline upgrades.

ASAP Cloud Native Upgrade Procedures

ASAP cloud native owns the component and therefore the upgrade procedure applies for the component. ASAP cloud native provides the mechanism to perform the upgrade using the scripts that are bundled with the Docker image and cloud native toolkit. The upgrade procedure includes upgrading the ASAP cloud native Docker image and deploying the Docker image in the instance.

To upgrade the ASAP installer, Java, WebLogic Server, Database client, and Cartridge install or uninstall require an upgrade in the ASAP Docker image.

To upgrade the ASAP Docker Image:

1. Delete the running instance using the `delete-instance.sh` script.
2. Copy the required installers to the **$asap-img-builder/installers** directory.
3. Copy the required cartridges to the **$asap-img-builder/cartridges** directory.
4. Run the following commands to copy installers and cartridges to the volume:

   ```
   $asap-img-builder/upgradeASAPDockerImage.sh
   ```

5. Create a new container using the previous version of the Docker image using the following command:

   ```
   docker run --name $ASAP_CONTAINER -dit -h $DOCKER_HOSTNAME -
   p $WEBLOGIC_PORT -v $ASAP_VOLUME:/$ASAP_VOLUME ASAP-BASE-IMAGE
   ```

   For example: docker run --name asap-c -dit -h asaphost -p 7601 -v dockerhost_volume:/dockerhost_volume asapcn:7.4.0.0

   The container will be created with **asap-c**.

6. Enter into the ASAP container using the following command:

   ```
   docker exec -it asap-c bash
   ```

   You have entered into the ASAP container. Now, you have to upgrade the ASAP installation in the console mode.

7. Upgrade the ASAP installer using the console mode, manually.
8. Run the `./startWeblogic.sh` script to start WebLogic Server in the background.
9. Navigate to the installation directory of ASAP as shown below:

   ```
   cd /dockerhost_volume/installers/new installer
   ```

10. Run the following command to install ASAP:

    ```
    /asap74ServerLinux -console
    ```

11. Enter the details for the prompted options.
12. Enter the hostname of WebLogic Server as: 127.0.0.1
13. Enter the port number provided in the domain.xml file.
14. Enter the credentials of the Oracle WebLogic Server Administrator provided when you created the domain.
15. Select the server as AdminServer.
    ASAP installation in the console mode is completed.

Upgrading cartridges
To upgrade cartridges, uninstall the previous cartridges and install the new cartridges.

To uninstall and install cartridges:

1. Repeat steps 1 to 6 to create the staging container.
   Cartridges are present in the **/dockerhost_volume/cartridges** container.
2. Start ASAP and WebLogic Server using the `startALL.sh` script.
3. Navigate to the ASAP installation directory using `cd $ASAP_BASE`.
4. Source the environment profile using the `source Environment_Profile` script.
5. Verify the ASAP server status using the `status` command.
6. Uninstall the cartridges. For more information, see "Uninstalling a Cartridge" in *ASAP Installation Guide*.
7. Install the cartridges present in the **/dockerhost_volume/cartridges** directory. For more information, see "Installing a Cartridge" in *ASAP Installation Guide*.

To upgrade Java, WebLogic Server, and database client, see "About Upgrading ASAP" in *ASAP Installation Guide*.

Creating an Image from the Staging Container
The staging container is deployed with all the required updates to provision network elements. Save this container as a Docker image to deploy in the Kubernetes cluster.

To create an image from the staging container:

1. Run the following command to create an image from the staging container:

   ```
   docker commit asap-c imagename:version
   ```

   Where *version* is the version of the ASAP Docker image. This version should be higher than the previous version.

2. To deploy a new Docker image in the Kubernetes cluster, the image should be available in the configured Docker registry or on all worker nodes. To push the Docker image to the Kubernetes docker registry, run the following commands:

   ```
   docker images | grep image name
   docker tag imageid <tag-imageid>
   -bash-4.2$ docker push  tag-imageid
   ```

3. Stop and remove the containers using the following commands:

   ```
   docker stop asap-c
   docker rm asap-c
   ```

4. Update the Docker image in the **$ASAP_CNTK/charts/asap/values.yaml** file.
5. Create the ASAP instance using the following command:

   ```
   $ASAP_CNTK/scripts/create-instance.sh -p sr -i quick
   ```

Now the ASAP instance is upgraded successfully.

Order Balancer Cloud Native Upgrade Procedures

To upgrade the Order Balancer Docker Image:

1. Delete the running instance using the `delete-instance.sh` script.
2. Copy the required installers to the **$asap-img-builder/installers** directory.
3. Run the following command to copy installers to the volume:

   ```
   $asap-img-builder/upgradeOBDockerImage.sh
   ```

4. Create a new container using the previous version of the Docker image using the following command:

   ```
   docker run --name $OB_CONTAINER   -dit -h $OB_HOSTNAME    -
   p $WEBLOGIC_PORT -v $OB_VOLUME:/$OB_VOLUME <OB-BASE-IMAGE>
   ```

   For example: docker run --name ob-c -dit -h obhost -p 7601 -v obhost_volume:/ obhost_volume obcn:7.4.0.0

   The container will be created with **ob-c**.
5. Enter into the container using the following command:

   ```
   docker exec -it ob-c bash
   ```

   You have entered into the Order Balancer container. For upgrading Order Balancer, see "Updating and Redeploying Order Balancer" in *ASAP System Administrator's Guide*.

Creating an Image from the Staging Container

The staging container is deployed with all the required updates to route work orders to ASAP instances. Save this container as a Docker image to deploy in the Kubernetes cluster.

To create an image from the staging container:

1. Run the following command to create an image from the staging container:

   ```
   docker commit ob-c imagename:version
   ```

   Where *version* is the version of the Order Balancer Docker image. This version should be higher than the previous version.
2. To deploy the new Docker image in the Kubernetes cluster, the image should be available in the configured Docker registry or on all worker nodes. To push the Docker image to the Kubernetes docker registry, run the following commands:

   ```
   docker images | grep image name
   docker tag imageid <tag-imageid>
   -bash-4.2$ docker push  tag-imageid
   ```

3. Stop and remove the containers using the following commands:

   ```
   docker stop ob-c
   docker rm ob-c
   ```

4. Update the Docker image in the **$OB_CNTK/charts/ob/values.yaml** file.
5. Create the Order Balancer instance using the following command:

   ```
   $OB_CNTK/scripts/create-instance.sh -p sr -i quick
   ```

Now the Order Balancer instance is upgraded successfully.

Upgrades to Infrastructure

From the point of view of ASAP instances, upgrades to the cloud infrastructure fall into two categories:

- Rolling upgrades
- One-time upgrades

> **Note:**
>
> All infrastructure upgrades must continue to meet the supported types and versions listed in the ASAP documentation's certification statement.

Rolling upgrades are where, with proper high-availability planning (like anti-affinity rules), the instance as a whole remains available as parts of it undergo temporary outages. Examples of this are Kubernetes worker node OS upgrades, Kubernetes version upgrades and Docker version upgrades.

One-time upgrades affect a given instance all at once. The instance as a whole suffers either an operational outage or a control outage. Examples of this is Ingress controller upgrade.

Kubernetes and Docker Infrastructure Upgrades

Follow standard Kubernetes and Docker practices to upgrade these components. The impact at any point should be limited to one node - master (Kubernetes and OS) or worker (Kubernetes, OS, and Docker). If a worker node is going to be upgraded, drain and cordon the node first. This will result in all pods moving away to other worker nodes. This is assuming your cluster has the capacity for this - you may have to temporarily add a worker node or two. For ASAP instances, any pods on the cordoned worker will suffer an outage until they come up on other workers. However, their messages and orders are redistributed to surviving pods and processing continues at a reduced capacity until the affected pods relocate and initialize. As each worker undergoes this process in turn, pods continue to terminate and start up elsewhere, but as long as the instance has pods in both affected and unaffected nodes, it will continue to process orders.

Ingress Controller Upgrade

Follow the documentation of your chosen Ingress Controller to perform an upgrade. Depending on the Ingress Controller used and its deployment in your Kubernetes environment, the ASAP instances it serves may see a wide set of impacts, ranging from no impact at all (if the Ingress Controller supports a clustered approach and can be upgraded that way) to a complete outage.

The new Traefik can be installed into a new name space, and one-by-one, projects can be unregistered from the old Traefik and registered with the new Traefik.

```
export TRAEFIK_NS=old-namespace $ASAP_CNTK/scripts/unregister-namespace -p
project -t traefik
export TRAEFIK_NS=new-namespace $ASAP_CNTK/scripts/register-namespace -p
project -t traefik
```

During this transition, there will be an outage in terms of the outside world interacting with ASAP. Any data that flows through the ingress will be blocked until the new Traefik takes over. This includes GUI traffic, order injection, API queries, and SAF responses from external systems. This outage will affect all the instances in the project being transitioned.

Miscellaneous Upgrade Procedures

This section describes miscellaneous upgrade scenarios.

Network File System (NFS)

If an instance is created successfully, but a change to the NFS configuration is required, then the change cannot be made to a running ASAP instance. In this case, the procedure is as follows:

1. Delete the ASAP instance.
2. Update the `nfs` details in the pv.yaml and pvc.yaml files.
3. Start the instance.

11

Moving to ASAP Cloud Native from a Traditional Deployment

You can move to an ASAP cloud native deployment from your existing ASAP traditional deployment. This chapter describes tasks that are necessary for moving from a traditional ASAP deployment to an ASAP cloud native deployment.

Supported Releases

You can move to ASAP cloud native from all supported traditional ASAP releases. In addition, you can move to ASAP cloud native within the same release, starting with the ASAP release 7.3.0.6.0.

About the Move Process

The move to ASAP cloud native involves offline preparation as well as maintenance outage. This section outlines the general process as well as the details of the steps involved in the move to ASAP cloud native. However, there are various places where choices have to be made. It is recommended that a specific procedure be put together after taking into account these choices in your deployment context.

The ASAP cloud native application layer runs on different hardware locations (within a Kubernetes cluster) than the ASAP traditional application layer.

The process of moving to ASAP cloud native involves the following sets of activities:

- Pre-move development activities, which includes the following tasks:
  - Building ASAP cloud native images (cloud native task)
  - Creating project specification ASAP cloud native (cloud native and solution task)
  - Creating an ASAP cloud native instance for testing (cloud native task)
  - Validating your solution cartridges (solution task)
  - Deleting the test ASAP cloud native instance (cloud native task)
- Data synchronization activities, which include the following tasks:
  - Preparing a new database server (database task)
  - Synchronizing the current database server (database task)
- Tasks for moving to ASAP cloud native, which include the following:
  - Quiescing the ASAP traditional instance (solution task)
  - Backing up the database (database task)
  - Creating an ASAP cloud native instance (cloud native task)
  - Performing a smoke test (solution task)
  - Importing the database (database task)

– Switching all upstream systems (solution task)
  For more information about creating backup and rolling back the ASAP database, see "Creating a Backup of the ASAP Schemas" and "Rolling Back the ASAP Database" in *ASAP Installation Guide*.

Pre-move Development Activities

In preparation to move your traditional ASAP instance into an ASAP cloud native environment, you must do the following activities:

1. Build the ASAP cloud native image with the same ENV_ID, database users credentials, default users credentials, same port numbers, and cartridges deployed in the traditional deployment. This task includes creating the ASAP Docker image and using the ASAP cloud native download packages. See "Creating an ASAP Cloud Native Image" for details.

   Note:

   The values of ENV_ID and port numbers are present in the **asap73ServerLinux.response** file of the ASAP installation directory.

2. Create an ASAP cloud native test instance and test your instance.
3. Validate the solution.
4. Shut down your test instance and remove the associated secrets and ingress.

Moving to an ASAP Cloud Native Deployment

Moving to an ASAP cloud native deployment from an ASAP traditional deployment requires performing the following tasks:

1. Quiesce the ASAP traditional instance. See "Quiescing the Traditional Instance of ASAP".
2. Create the ASAP cloud native image. See "Creating the ASAP Cloud Native Image".
3. Import the restored data from the traditional instance to the cloud native instance.
4. Create the ASAP cloud native instance. See "Creating an ASAP Cloud Native Instance".
5. Perform a smoke test. See "Performing a Smoke Test". Once the ASAP cloud native instance passes the smoke test and is optionally resized to the desired target value, shut down the ASAP traditional instance fully.
6. Switch all upstream systems to the ASAP cloud native instance. See "Switching Integration with Upstream Systems".

Quiescing the Traditional Instance of ASAP

At the start of the maintenance window, the ASAP traditional instance must be quiesced. This involves stopping database jobs, stopping all upstream and peer systems from sending messages (for example, http/s, JMS, and SAF) to ASAP, and ensuring all human users are logged out. It also involves pausing the JMS queues so that no messages get queued or dequeued. The result is that ASAP is up and running, but completely idle.

Restoring the Database

Before creating the final Docker image, restore the database. For more information, see "Rolling Back the ASAP Database" in *ASAP Installation Guide* and follow these steps:

1. Update hostname in `tbl_listeners` of the CTRL database by running the following command:

   ```
   update tbl_listeners set HOST_NAME='asaphost';
   ```

   where '*asaphost*' is the name of the host in which ASAP is deployed.
2. Update the hostname in the `TBL_ASAP_SRP` table of the SARM by running the following command:

   ```
   update TBL_ASAP_SRP set HOST_NAME='asaphost';
   ```

3. Update the **SRP_HOST_NAME** field in the **TBL_ASAP_SRP** table by running the following command:

   ```
   update TBL_ASAP_SRP set SRP_HOST_NAME='asaphost';
   ```

Switching Integration with Upstream Systems

After you shut down the ASAP traditional instance fully, do the following:

- Ensure that the ASAP cloud native instance has its JMS.
- Configure the upstream to resume sending messages. See "Integrating ASAP" for more details.

Reverting to Your ASAP Traditional Deployment

During the move to ASAP cloud native, if there is a need to revert to your ASAP traditional deployment, the exact sequence of steps that you need to perform depends on the options you have chosen while moving to ASAP cloud native.

In general, the ASAP traditional deployment application layer should be undisturbed through the upgrade process. The ASAP traditional instance can simply be started up again, still pointing to its database.

Cleaning Up

Once the ASAP cloud native instance is deemed operational, you can release the resources used for the ASAP traditional application layer.

You can delete the database used for ASAP traditional instance and release its resources as well.

12

Debugging and Troubleshooting

This chapter provides information about debugging and troubleshooting issues that you may face while setting up an ASAP cloud native environment and creating ASAP cloud native instances.

This chapter describes information about the following:

- Troubleshooting Issues with Traefik and WebLogic Administration Console
- Common Error Scenarios
- Known Issues

Troubleshooting Issues with Traefik and WebLogic Administration Console

This section describes how to troubleshoot issues with access to WLST, and WebLogic Administration Console.

It is assumed that Traefik is used as the default Ingress controller and the domain name suffix is `asap.org`. You can modify the instructions to suit any other domain name suffix that you may have chosen.

The following table lists the URLs for accessing the WebLogic Administration Console when the Oracle Cloud Infrastructure load balancer is used and not used:

Table 12-1 URLs for Accessing ASAP Clients

Client	If Not Using Oracle Cloud Infrastructure Load Balancer	If Using Oracle Cloud Infrastructure Load Balancer
WebLogic Admin Console	http://admin.instance.project.asap.org:30305/console	http://admin.instance.project.asap.org:80/console

Error: Http 404 Page not found

This is the most common problem that you may encounter.

To resolve this issue:

1. Verify the Domain Name System (DNS) configuration.

> **Note:**
>
> These steps apply for local DNS resolution via the **hosts** file. For any other DNS resolution, such as corporate DNS, follow the corresponding steps.

The hosts configuration file is located at:

- On Windows: **C:\Windows\System32\drivers\etc\hosts**
- On Linux: **/etc/hosts**

Verify if the following entry exists in the hosts configuration file of the client machine from where you are trying to connect to ASAP:

- Local installation of Kubernetes without Oracle Cloud Infrastructure load balancer:

  ```
  Kubernetes_Cluster_Master_IP <hostname provided in the
  values.yaml file>
  ```

- If Oracle Cloud Infrastructure load balancer is used:

  ```
  Load_balancer_IP  instance.project.asap. <hostname given in the
  values.yaml file>
  ```

Resolve the DNS configuration.

2. Verify the browser settings and ensure that `*.asap.org` is added to the **No proxy** list, if your proxy cannot route to it.
3. Verify if the Traefik pod is running and installing or updating the Traefik Helm chart:

   ```
   kubectl -n traefik get pod
   NAME                                 READY    STATUS     RESTARTS   AGE
   traefik-operator-657b5b6d59-njxwg    1/1      Running    0
   128m
   ```

4. Verify if the Traefik service is running:

   ```
   kubectl -n traefik get svc
   NAME                           TYPE           CLUSTER-IP
   EXTERNAL-IP      PORT(S)                        AGE
   oci-lb-service-traefik         LoadBalancer   10.96.136.31
   100.77.18.141    80:31115/TCP                   20d     <---- Is
   expected in OCI environment only --
   traefik-operator               NodePort       10.98.176.16
   <none>           443:30443/TCP,80:30305/TCP     141m
   traefik-operator-dashboard     ClusterIP      10.103.29.101
   <none>           80/TCP                         141m
   ```

   Note:

   If the Traefik service is not running, install or update the Traefik Helm chart.

5. Verify if the Traefik back-end systems are registered, by using one of the following options:

- Run the following commands to check if your project name space is being monitored by Traefik. The absence of your project name space means that your managed server back-end systems are not registered with Traefik.

  ```
  $ cd $ASAP_CNTK
  $ source scripts/common-utils.sh
  $ find_namespace_list 'namespaces' traefik traefik-operator
  "traefik","project_1", "project_2"
  ```

- Verify the Traefik Dashboard and add the following DNS entry in your hosts configuration file:

  ```
  Kubernetes_Access_IP traefik.asap.org
  ```

  Add the same entry regardless of whether you are using Oracle Cloud Infrastructure load balancer or not. Navigate to: `http://traefik.asap.org:30305/dashboard/` and check the back-end systems that are registered. If you cannot find your project name space, install or upgrade the Traefik Helm chart. See "Installing the Traefik Container Image" for more information.

Reloading Instance Backend Systems

If your instance's ingress is present, yet Traefik does not recognize the URLs of your instance, try to unregister and register your project name space again. You can do this by using the **unregister-namespace.sh** and **register-namespace.sh** scripts in the toolkit.

> **Note:**
>
> Unregistering a project name space will stop access to any existing instances in that name space that was working prior to the unregistration.

Debugging Traefik Access Logs

To increase the log level and debug Traefik access logs:

1. Run the following command:

   ```
   $ helm upgrade traefik-operator traefik/traefik   --version 9.11.0  --
   namespace traefik   --reuse-values   --set logs.access.enabled=true
   ```

   A new instance of the Traefik pod is created automatically.

2. Look for the pod that is created most recently:

   ```
   $ kubectl get po -n traefik
   NAME                                    READY    STATUS     RESTARTS    AGE
   traefik-operator-pod_name   1/1       Running    0          0           5s

   $ kubectl -n traefik logs -f traefik-operator-pod_name
   ```

3. Enabling access logs generates large amounts of information in the logs. After debugging is complete, disable access logging by running the following command:

   ```
   $ helm upgrade traefik-operator traefik/traefik   --version 9.11.0
   --namespace traefik   --reuse-values   --set
   logs.access.enabled=false
   ```

Cleaning Up Traefik

> **Note:**
>
> Clean up is not usually required. It should be performed as a desperate measure only. Before cleaning up, make a note of the monitoring project name spaces. Once Traefik is re-installed, run **$ASAP_CNTK/scripts/register-namespace.sh** for each of the previously monitored project name spaces.
>
> **Warning**: Uninstalling Traefik in this manner will interrupt access to all ASAP instances in the monitored project name spaces.

To clean up the Traefik Helm chart, run the following command:

```
helm uninstall traefik-operator -n traefik
```

Cleaning up of Traefik does not impact actively running ASAP instances. However, they cannot be accessed during that time. Once the Traefik chart is re-installed with all the monitored name spaces and registered as Traefik back-end systems successfully, ASAP instances can be accessed again.

Setting up Logs

As described earlier in this guide, ASAP and WebLogic logs can be stored in the individual pods or in a location provided via a Kubernetes Persistent Volume. The PV approach is strongly recommended, both to allow for proper preservation of logs (as pods are ephemeral) and to avoid straining the in-pod storage in Kubernetes.

Within the pod, logs are available at: **/u01/oracle/user_projects/domains/domain/servers/AdminServer/logs**.

ASAP logs: **/scratch/oracle/asap/DATA/logs/**

When a PV is configured, logs are available at the following path starting from the root of the PV storage:

*project-instance***/logs**.

Common Problems and Solutions

This section describes some common problems that you may experience because you have run a script or a command erroneously or you have not properly followed the recommended procedures and guidelines regarding setting up your cloud environment, components, tools, and services in your environment. This section provides possible solutions for such problems.

Pod Status

While the introspection is running, you can check the status of the introspection pod by running the following command:

```
kubectl get pods -n namespace
## healthy status looks like this
NAME                                           READY   STATUS    RESTARTS   AGE
project-instance-introspect-domain-job-hzh9t   1/1     Running   0
3s
```

The **READY** field is showing 1/1, which indicates that the pod status is healthy.

If there is an issue accessing the image specified in the instance specification, then it shows the following:

```
NAME                                               READY   STATUS
RESTARTS   AGE
project-instance-introspect-domain-job-r2d6j   0/1     ErrImagePull
0          5s
### OR
NAME                                               READY   STATUS
RESTARTS   AGE
project-instance-introspect-domain-job-r2d6j   0/1     ImagePullBackOff
0          45s
```

This shows that the introspection pod status is not healthy. If the image can be pulled, it is possible that it took a long time to pull the image.

To resolve this issue, verify the image name and the tag and that it is accessible from the repository by the pod.

You can also try the following:

- Pull the container image manually on all Kubernetes nodes where the ASAP cloud native pods can be started up.

Known Issues

This section describes known issues that you may come across, their causes, and the resolutions.

Email Plugin
The ASAP Email plugin is currently not supported. Users who require this capability can create their own plugin for this purpose.

A

Differences Between ASAP Cloud Native and ASAP Traditional Deployments

If you are moving from a traditional deployment of ASAP to a cloud native deployment, this section describes the differences between ASAP cloud native and ASAP traditional.

- **ASAP Installer**

  Distributed installations are not supported in the ASAP cloud native environment. All ASAP components, including WebLogic Server, must be installed in the same container.

  Also, SRT, custom SRPs, and custom NEPs are not supported in the ASAP cloud native environment.

- **WebLogic Domain Configuration**

  In a traditional deployment of ASAP, the WebLogic domain configuration is done using WLST or the WebLogic Admin Console. In ASAP cloud native, domain configuration is done by using WLST. ASAP cloud native does not support the deployment of ASAP in a Managed Server.

- **Incoming JMS and SAF**

  For incoming JMS and SAF messages, the originator must use T3 over HTTPS tunneling.

- **ASAP OCA**

  The Order Control Application (OCA) is available in both ASAP traditional and ASAP cloud native deployments. In a cloud native environment, you can access OCA using the hostname configured in the ASAP **values.yaml** file and the port number in the Traefik **values.yaml** file. For example, to access the OCA, use:

  ```
  https://adminhostssl.asap.org:30443/<ENV_ID>/OCA
  ```

- **Web Services API**

  The Web Services API is supported in the ASAP cloud native environment. The external transport protocols are HTTP, HTTPS, and JMS and the data service formats are SOAP v1.1 and 1.2.

  For details about the ASAP Web Services API supported, see ASAP *Developer's Guide*.