ORACLE

Containerized Oracle Application Deployment on Oracle Roving Edge

Version 1.0

Public

ORACLE

Purpose statement

The purpose of this solution paper is to present a validated approach for deploying Oracle enterprise applications in a containerized architecture on Oracle Roving Edge Devices (REDs). Designed for cloud architects, DevOps engineers, and edge computing teams, this guide enables secure, high-performance execution of Oracle applications, such as middleware, databases, and custom workloads at the edge.

Disclaimer

ORACLE

Table of contents

Purpose statement	**2**
Introduction	**4**
Architecture Overview	**5**
Solution Deployment	**6**
Accessing Oracle Container Registry	**10**
Deploying Oracle Containerized Applications	**12**
Deploying NGINX Server from Docker Hub Repository	**19**

Introduction

Edge computing is revolutionizing the way enterprises run applications and services in real time, especially where cloud access is constrained or unavailable. Oracle Roving Edge Devices (REDs) deliver cloud-grade compute, storage, and networking capabilities in a portable, ruggedized form factor ideal for field deployments and tactical use cases.

This solution paper outlines the process of containerizing and deploying Oracle Applications on REDs using K3s, a lightweight Kubernetes distribution optimized for resource-constrained environments. By leveraging Oracle Linux and RED's secure edge infrastructure, organizations can run critical applications, including Oracle middleware, databases, and analytics platforms closer to where data is generated.

Note: This content is provided for informational purposes and self-supported guidance only. Consultancy or other assistance related to the content is not covered under the Oracle Support contract or associated service requests. If you have questions or additional needs, then please reach out to your Oracle Sales contact directly.

Architecture Overview

This solution leverages a lightweight, modular architecture designed specifically for edge computing environments by deploying Oracle enterprise applications, such as Oracle SOA, or custom microservices on a K3s Kubernetes cluster running locally on Oracle Roving Edge Devices (REDs). The goal is to enable secure, resilient, and cloud-independent operation of mission-critical applications directly at the edge.

The deployment architecture consists of the following core components:

Oracle Roving Edge Device (RED)

Serves as the physical host (ruggedized or not), delivering secure, high-performance compute, storage, and networking capabilities. RED runs Oracle Linux and is optimized to host containerized workloads and Kubernetes clusters, even in remote or disconnected environments.

K3s Lightweight Kubernetes

A minimal, production-grade Kubernetes distribution installed directly on RED. K3s orchestrates the lifecycle of containerized Oracle applications with low resource overhead, enabling rapid deployment, scaling, and monitoring of edge workloads.

Oracle Applications (Containerized)

Deployed as Docker containers orchestrated by K3s. Depending on the use case, the stack may include:

- **Oracle Database:** For local data persistence, caching, or synchronization with central systems.
- **Oracle WebLogic or Java EE apps**: For enterprise middleware services.
- **Custom Microservices:** For telemetry, analytics, APIs, or integrations.
- **Optional Services:** like NGINX, Grafana, or Prometheus for management, logging, or observability.

All services run locally on the RED, ensuring zero dependency on external cloud services or third-party infrastructure.

Key Benefits of This Architecture

- **Cloud-independent operation:** Applications run autonomously in low or no-connectivity environments.
- **Secure and portable:** Built on Oracle Linux with hardened configurations for field deployment.
- **Optimized for resource-constrained environments:** Minimal overhead with K3s and container-native design.
- **Flexible application stack**: Supports Oracle and third-party containers with full Kubernetes orchestration.
- **Use cases:** Ideal for tactical, industrial, and remote edge scenarios.

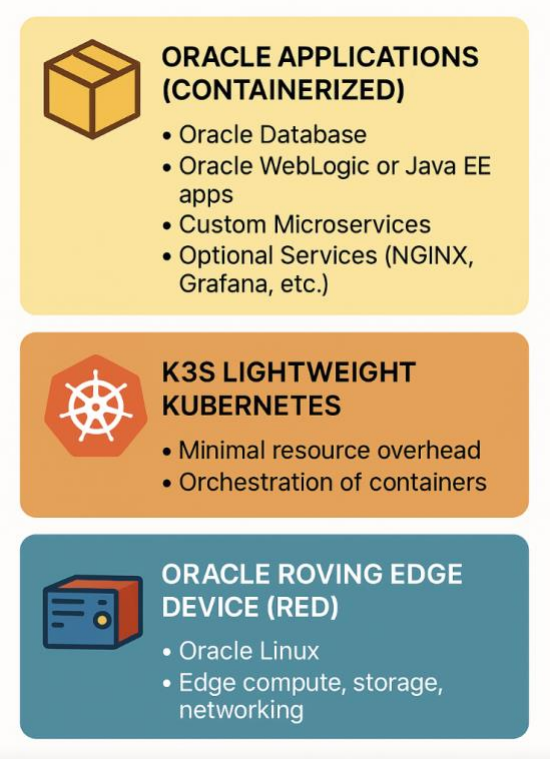


Figure 1. Architecture diagram of Oracle containerized applications running on Oracle Roving Edge

Solution Deployment

This section outlines the detailed steps to deploy Oracle Applications in containers using K3s on an Oracle Roving Edge Device running Oracle Linux 8.10. The deployment utilizes K3s as a lightweight Kubernetes orchestrator and connects securely to the Oracle Container Registry (OCR) to pull containerized Oracle applications such as Oracle Database, WebLogic Server, and supporting services. The process begins by preparing the RED environment, installing K3s, authenticating with the Oracle Container Registry, and deploying the desired Oracle applications as containers. This approach enables rapid provisioning, localized application execution, and full-stack orchestration at the edge, which is ideal for disconnected or low-latency environments.

Step 1. As root user, enable the Oracle Linux repositories, update the instance to the latest package releases, and check if the basic packages are installed.

```
sudo dnf update -y
sudo dnf install -y oraclelinux-release-el8
sudo dnf install -y epel-release
```

Step2. Install K3s (Lightweight Kubernetes), verify the installation, and configure K3s to start on boot:

- Run the following command to install K3s: **`curl -sfL https://get.k3s.io | sh -`**

Listed below is the output of the command when successfully executed:

```
[INFO]  Finding release for channel stable
[INFO]  Using v1.32.4+k3s1 as release
[INFO]  Downloading hash https://github.com/k3s-
io/k3s/releases/download/v1.32.4+k3s1/sha256sum-amd64.txt
[INFO]  Downloading binary https://github.com/k3s-io/k3s/releases/download/v1.32.4+k3s1/k3s
[INFO]  Verifying binary download
[INFO]  Installing k3s to /usr/local/bin/k3s
[INFO]  Finding available k3s-selinux versions
Rancher K3s Common (stable)

7.4 kB/s | 2.6 kB     00:00
Last metadata expiration check: 0:00:01 ago on Wed 14 May 2025 04:31:04 PM GMT.
Dependencies resolved.
================================================================================
================================================================================
================================================================================
====
 Package                                                 Architecture
Version
Repository                                                      Size
================================================================================
================================================================================
================================================================================
============
Installing:
 k3s-selinux                                             noarch
1.6-1.el8
rancher-k3s-common-stable                                       20 k
Installing dependencies:
container-selinux                                       noarch
2:2.229.0-2.module+el8.10.0+90541+332b2aa7
ol8_appstream                                                   69 k
Enabling module streams:
 container-tools
ol8

Transaction Summary
================================================================================
================================================================================
================================================================================
====
Install  2 Packages
Total download size: 90 k
Installed size: 161 k
Downloading Packages:
(1/2): k3s-selinux-1.6-1.el8.noarch.rpm
264 kB/s |  20 kB     00:00
(2/2): container-selinux-2.229.0-2.module+el8.10.0+90541+332b2aa7.noarch.rpm
203 kB/s |  69 kB     00:00
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----
```

```
Total
258 kB/s |  90 kB     00:00
Rancher K3s Common (stable)
22 kB/s | 2.4 kB     00:00
Importing GPG key 0xE257814A:
 Userid     : "Rancher (CI) <ci@rancher.com>"
 Fingerprint: C8CF F216 4551 26E9 B9C9 18BE 925E A29A E257 814A
 From       : https://rpm.rancher.io/public.key
Key imported successfully
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing    :

1/1
  Running scriptlet: container-selinux-2:2.229.0-2.module+el8.10.0+90541+332b2aa7.noarch
1/2
  Installing       : container-selinux-2:2.229.0-2.module+el8.10.0+90541+332b2aa7.noarch
1/2
  Running scriptlet: container-selinux-2:2.229.0-2.module+el8.10.0+90541+332b2aa7.noarch
1/2
  Running scriptlet: k3s-selinux-1.6-1.el8.noarch
2/2
  Installing       : k3s-selinux-1.6-1.el8.noarch
2/2
  Running scriptlet: k3s-selinux-1.6-1.el8.noarch
2/2
  Running scriptlet: container-selinux-2:2.229.0-2.module+el8.10.0+90541+332b2aa7.noarch
2/2
  Running scriptlet: k3s-selinux-1.6-1.el8.noarch
2/2
  Verifying        : container-selinux-2:2.229.0-2.module+el8.10.0+90541+332b2aa7.noarch
1/2
  Verifying        : k3s-selinux-1.6-1.el8.noarch
2/2

Installed:
  container-selinux-2:2.229.0-2.module+el8.10.0+90541+332b2aa7.noarch
k3s-selinux-1.6-1.el8.noarch

Complete!
[INFO]  Creating /usr/local/bin/kubectl symlink to k3s
[INFO]  Creating /usr/local/bin/crictl symlink to k3s
[INFO]  Creating /usr/local/bin/ctr symlink to k3s
[INFO]  Creating killall script /usr/local/bin/k3s-killall.sh
[INFO]  Creating uninstall script /usr/local/bin/k3s-uninstall.sh
[INFO]  env: Creating environment file /etc/systemd/system/k3s.service.env
[INFO]  systemd: Creating service file /etc/systemd/system/k3s.service
[INFO]  systemd: Enabling k3s unit
Created symlink /etc/systemd/system/multi-user.target.wants/k3s.service →
/etc/systemd/system/k3s.service.
[INFO]  systemd: Starting k3s
```

- To ensure that the k3s command is accessible, update the root user's bash profile. As root user, add k3s

```
echo 'export PATH=$PATH:/usr/local/bin' >> ~/.bashrc
source /root/.bashrc
```

- As root user, run the following command to verify the K3s installation and the kubernetes nodes currently available, up, and running.

systemctl status k3s

```
● k3s.service - Lightweight Kubernetes
   Loaded: loaded (/etc/systemd/system/k3s.service; enabled; vendor preset: disabled)
   Active: active (running) since Wed 2025-05-14 16:37:33 GMT; 3min 30s ago
     Docs: https://k3s.io
  Process: 16245 ExecStartPre=/sbin/modprobe overlay (code=exited, status=0/SUCCESS)
  Process: 16239 ExecStartPre=/sbin/modprobe br_netfilter (code=exited, status=0/SUCCESS)
  Process: 16236 ExecStartPre=/bin/sh -xc ! /usr/bin/systemctl is-enabled --quiet nm-cloud-
setup.service 2>/dev/null (code=exited, status=0/SUCCESS)
 Main PID: 16253 (k3s-server)
    Tasks: 146
   Memory: 1.5G
   CGroup: /system.slice/k3s.service
           ├─16253 /usr/local/bin/k3s server
           ├─16335 containerd
           ├─17405
/var/lib/rancher/k3s/data/e1730ceee3d97d63f58b7ccd96fe08638e972abfd3b1ebdf497b52572f85b316/bi
n/containerd-shim-runc-v2 -namespace k8s.io -id
cd203b62b2bec616c1a50679f667b886e48cb0a51b3fcec7de9012710ba19949 -address
/run/k3s/containerd/containerd.sock
           ├─17415
/var/lib/rancher/k3s/data/e1730ceee3d97d63f58b7ccd96fe08638e972abfd3b1ebdf497b52572f85b316/bi
n/containerd-shim-runc-v2 -namespace k8s.io -id
0682fc320cdfd0517b8b577a409d8579afc9bad306543e30306e3747943affb1 -address
/run/k3s/containerd/containerd.sock
           ├─17448
/var/lib/rancher/k3s/data/e1730ceee3d97d63f58b7ccd96fe08638e972abfd3b1ebdf497b52572f85b316/bi
n/containerd-shim-runc-v2 -namespace k8s.io -id
717501964f7b08129af01424573cc463e63f27b221d46286e68a2b1f9d2fa891 -address
/run/k3s/containerd/containerd.sock
           ├─18827
/var/lib/rancher/k3s/data/e1730ceee3d97d63f58b7ccd96fe08638e972abfd3b1ebdf497b52572f85b316/bi
n/containerd-shim-runc-v2 -namespace k8s.io -id
c30594ec747d082e79005b5cb83766a98717a92e8918b62d29ed9305a743ca14 -address
/run/k3s/containerd/containerd.sock
           └─18877
/var/lib/rancher/k3s/data/e1730ceee3d97d63f58b7ccd96fe08638e972abfd3b1ebdf497b52572f85b316/bi
n/containerd-shim-runc-v2 -namespace k8s.io -id
38dff38d1eb03ef7d90e67c97e643d4331407bd563be93fe321946f649778f4b -address
/run/k3s/containerd/containerd.sock
```

```
May 14 16:38:09 k3s k3s[16253]: I0514 16:38:09.402944   16253 resource_quota_monitor.go:227]
"QuotaMonitor created object count evaluator" resource="apibundles.hub.traefik.io"
May 14 16:38:09 k3s k3s[16253]: I0514 16:38:09.402967   16253 resource_quota_monitor.go:227]
"QuotaMonitor created object count evaluator" resource="ingressrouteudps.traefik.io"
May 14 16:38:09 k3s k3s[16253]: I0514 16:38:09.403309   16253 shared_informer.go:313] Waiting
for caches to sync for resource quota
May 14 16:38:09 k3s k3s[16253]: I0514 16:38:09.804536   16253 shared_informer.go:320] Caches
are synced for resource quota
May 14 16:38:09 k3s k3s[16253]: I0514 16:38:09.824913   16253 shared_informer.go:313] Waiting
for caches to sync for garbage collector
May 14 16:38:09 k3s k3s[16253]: I0514 16:38:09.925218   16253 shared_informer.go:320] Caches
are synced for garbage collector
May 14 16:38:18 k3s k3s[16253]: I0514 16:38:18.650025   16253 replica_set.go:679] "Finished
syncing" kind="ReplicaSet" key="kube-system/traefik-c98fdf6fb" duration="9.888711ms"
May 14 16:38:18 k3s k3s[16253]: I0514 16:38:18.650500   16253 replica_set.go:679] "Finished
syncing" kind="ReplicaSet" key="kube-system/traefik-c98fdf6fb" duration="69.594µs"
May 14 16:38:18 k3s k3s[16253]: I0514 16:38:18.659122   16253 event.go:389] "Event occurred"
object="kube-system/traefik" fieldPath="" kind="Service" apiVersion="v1" type="Normal"
reason="UpdatedLoadBalancer" message="Updated LoadBalancer with new IPs: [10.0.0.4] ->
[10.0.0.4]"
May 14 16:38:34 k3s k3s[16253]: I0514 16:38:34.530175   16253 range_allocator.go:247]
"Successfully synced" key="k3s"
```

- Run the following command to enable the K3s during the boot of the instance:

```
systemctl enable k3s
```

- Run the following command to check K3s cluster information:

```
k3s kubectl get nodes
```

```
NAME   STATUS   ROLES                  AGE     VERSION
k3s    Ready    control-plane,master   6m14s   v1.32.4+k3s1
```

- The output above means that the K3s cluster is up and running correctly. You have a single node named "k3s" that acts as the **control plane** (master node), and it has been running without issues for about 6 minutes. The Kubernetes version deployed by K3s is **v1.32.4+k3s1**.

Accessing Oracle Container Registry

- Go to Oracle Container Registry.
- Log in with your Oracle SSO account.

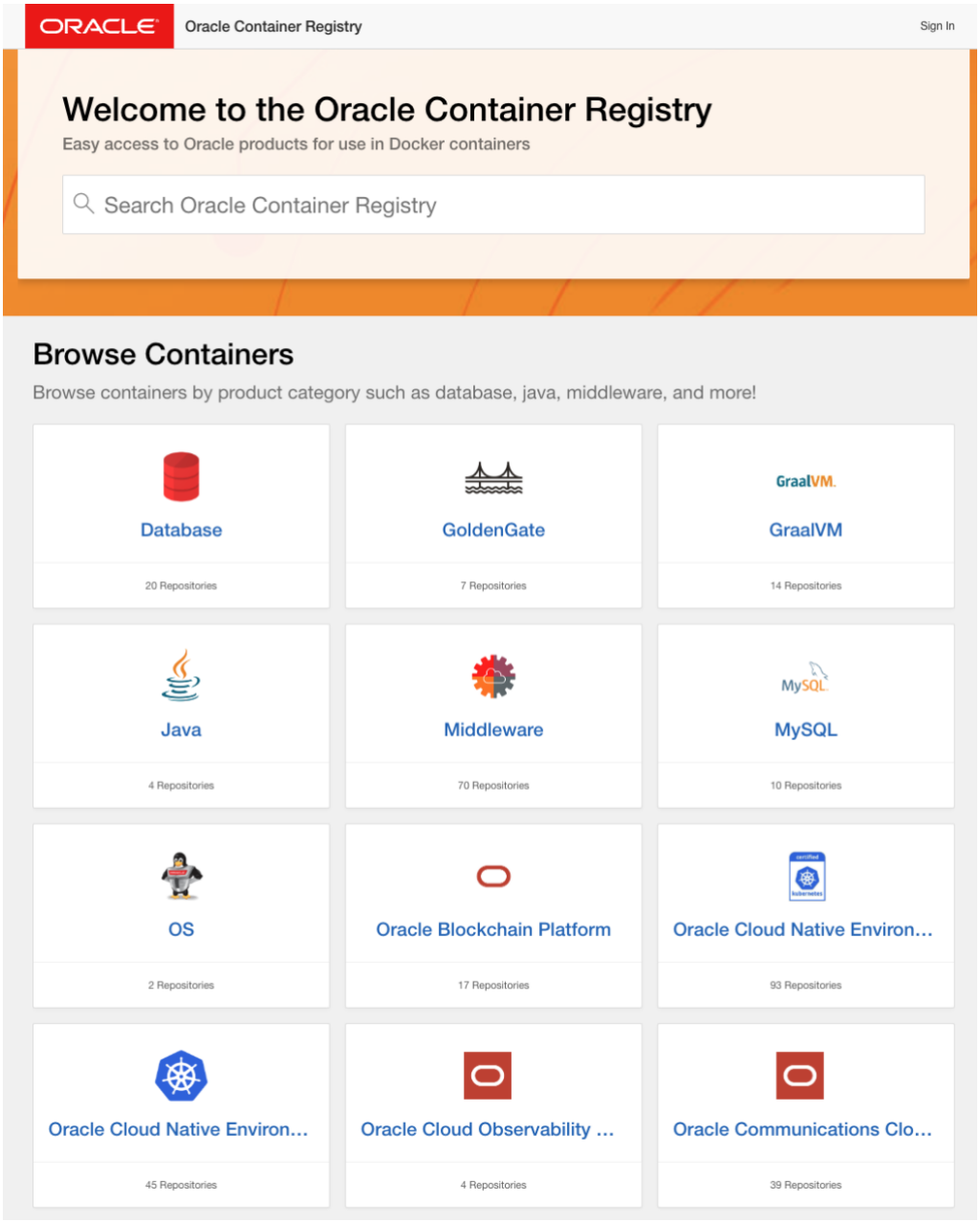
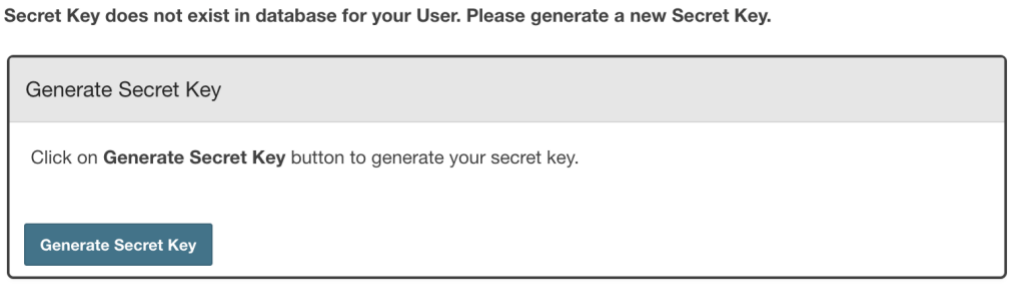
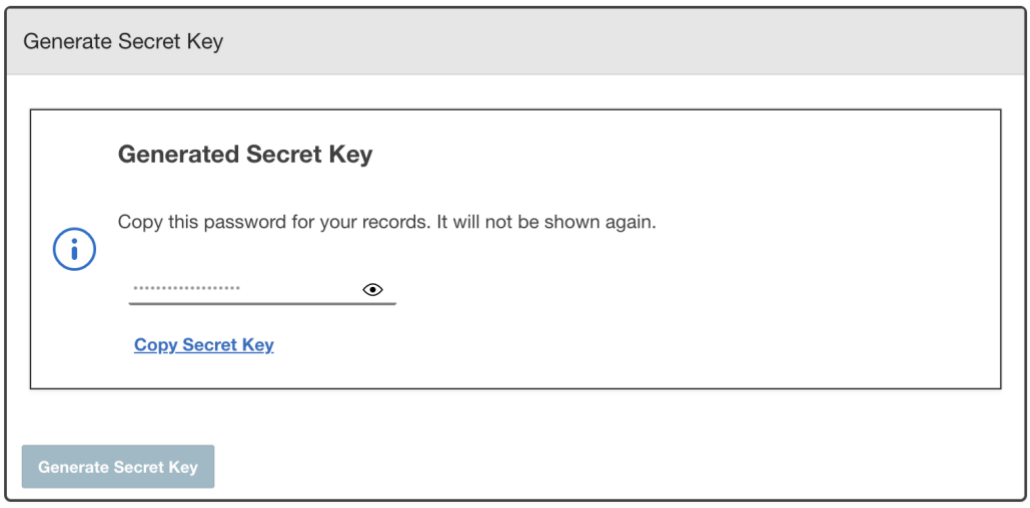


Figure 2. Architecture diagram of Oracle containerized applications running on Oracle Roving Edge

- Click on your username on the top lief toc the screen, then select Auth Token.

- On the Generate Secret Key screen, click Generate Secret Key, then copy it. This key will be utilized along with your username to login on the Oracle Container Registry via podman in your Oracle Linux instance.





- In your Oracle Linux instance, login to the Oracle Container Registry using Podman:

  ```
  docker login container-registry.oracle.com
  ```

- You will be prompted to enter your Oracle SSO username and password.
- Verify Login: After logging in, check your credentials using:

  ```
  docker login container-registry.oracle.com
  Username: your-email
  Password:
  Login Succeeded!
  ```

  This command confirms that you are logged in to the registry.

Deploying Oracle Containerized Applications

As enterprises modernize their infrastructure, containerized application delivery has become essential for agility, portability, and operational efficiency. Oracle provides a robust portfolio of containerized applications, including Oracle Database, WebLogic Server, and enterprise middleware all available via the Oracle Container Registry. This section introduces the foundational steps to deploy these applications on K3s running on Roving Edge. As example, we will be deploying Oracle SOA from Oracle Container Register and NGINX server from Docker hub repository.

NOTE: The Oracle Container Registry provides comprehensive documentation for deploying Oracle containerized applications.

Running Oracle SOA Suite in containers

Sample configurations to facilitate installation, configuration, and environment setup for Oracle SOA Suite 12.2.1.4 or 14.1.2.

At the end of this configuration there will be at least two running containers:

- (Optional) Oracle Database container (only when RCU schema is created in a database running in a container)
- Oracle WebLogic Server Administration Server container
- Two Oracle WebLogic Server Managed Server containers (Oracle SOA Server or Oracle Service Bus Server)

To create the Podman network and run containers, follow these steps:

Create a network

In this configuration, the creation of a user-defined network will enable the communication between the containers just using container names. For this setup we will use a user-defined network using bridge driver.

Create a user-defined network using the bridge driver:

$ podman network create -d bridge <network name>

For example:

$ podman network create -d bridge SOANet

Mount a host directory as a data volume

Data volumes are designed to persist data, independent of the container's lifecycle. Podman automatically creates volumes when you specify a volume name with the -v option, without the need to predefine directories on the host. In this project, the volumes will be used to store Database data files and WebLogic Server domain files. These volumes will be automatically created and managed by Podman. The names of the volumes are specified in the podman run commands.

$ podman -d --name soadb -v soadb_vol:/opt/oracle/oradata

The default storage location for Podman volumes is determined by Podman's storage configuration. To identify the location of a volume, run:

$ podman volume inspect <volume_name>

The Mountpoint entry should point to the location of the volume in the host.

Podman creates volumes with default permissions. Ensure that the container's oracle user has the necessary read/write/execute permissions on the auto-created volume. This may require setting proper permissions or ownership using a post-creation script, depending on your environment.

$ sudo chmod -R 777 $HOME/.local/shared/containers/storage/volumes/soadb_vol

To determine if a user already exists on your node system with uid:gid of 1000, run:

$ getent passwd 1000

If this command returns a username (which is the first field), you can skip the following useradd command. If not, create the oracle user manually:

$ useradd -u 1000 -g 0 oracle

Create the database

You need to have a running database container or a database running on any machine. The database connection details are required for creating SOA-specific RCU schemas while configuring the SOA domain. While using a 19.3.0.0 CDB/PDB DB, ensure a PDB is used to load the schemas. RCU loading on a CDB is not supported.

The Oracle database server container requires custom configuration parameters for starting up the container. These custom configuration parameters correspond to the data source parameters in the SOA image to connect to the database running in the container.

To run the database container to host the RCU schemas:

- Add the following parameters to a db.env.txt file:
- ORACLE_SID=soadb
- ORACLE_PDB=soapdb
- ORACLE_PWD=Oradoc_db1
    - ENABLE_ARCHIVELOG=true
- Enter the following command:
    - $ podman run -d --name soadb --network=SOANet -p 1521:1521 -p 5500:5500 -v soadb_vol:/opt/oracle/oradata --env-file ./db.env.txt container-registry.oracle.com/database/enterprise:19.3.0.0
- Verify that the database is running and healthy. The STATUS field should show healthy in the output of podman ps.

Obtain the Oracle SOA Suite container image

Pull the required version of Oracle SOA Suite image available in the Oracle Container Registry and update all occurrences of <REPLACE-WITH-RELEASE-VERSION> in the document with the exact tag value.

Create a container for the Administration Server

Start a container to launch the Administration Server from the image created using the steps above. The environment variables used to configure the domain are defined in adminserver.env.list. Replace in adminserver.env.list the values for the Database and WebLogic Server passwords.

Create an environment file adminserver.env.list:

CONNECTION_STRING=<Database container name>:<port#>/<ORACLE_PDB>
RCUPREFIX=<RCU_Prefix>
DB_PASSWORD=<database_sys_password>
DB_SCHEMA_PASSWORD=<soa-infra schema password>
ADMIN_PASSWORD=<admin_password>
DOMAIN_NAME=soainfra

DOMAIN_TYPE=<soa/osb/soaosb>
ADMIN_HOST=<Administration Server hostname>
ADMIN_PORT=<Node port number mapping Administration Server container port `7001`>
PERSISTENCE_STORE=<jdbc | file>

IMPORTANT: DOMAIN_TYPE must be carefully chosen and specified depending on the use case. It can't be changed once you proceed. For Oracle SOA Suite domains, the supported domain types are soa, osb and soaosb.

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

For example:

CONNECTION_STRING=soadb:1521/soapdb
RCUPREFIX=SOA1
DB_PASSWORD=Your-password
DB_SCHEMA_PASSWORD= Your-password
ADMIN_PASSWORD= Your-password
DOMAIN_NAME=soainfra
DOMAIN_TYPE=soa
ADMIN_HOST=<Administration Server hostname>
ADMIN_PORT=7001
PERSISTENCE_STORE=jdbc

If PERSISTENCE_STORE is not specified, the default value is jdbc. When PERSISTENCE_STORE=jdbc, a JDBC persistence store will be configured for all servers for TLOG + SOAJMS/UMSJMS servers. If PERSISTENCE_STORE=file, file-based persistence stores will be used instead.

To start a Podman container with a SOA domain and the WebLogic Server Administration Server, use the podman run command and pass the adminserver.env.list file.

For example:

$ podman run -it --name soaas --network=SOANet -p 7001:7001 -v soadomain_vol:/u01/oracle/user_projects --env-file ./adminserver.env.list container-registry.oracle.com/middleware/soasuite:<REPLACE-WITH-RELEASE-VERSION>

The options -it in the above command runs the container in interactive mode and you will be able to see the commands running in the container. This includes the command for RCU creation, domain creation, and configuration, followed by starting the Administration Server.

IMPORTANT: You need to wait until all the above commands are run before you can access the Administration Server Web Console. The following lines highlight when the Administration Server is ready to be used:
INFO: Admin server is running

INFO: Admin server running, ready to start Managed Server

These lines indicate that the Administration Server started successfully with the name soaas. Mapping container port 7001 to node port 7001 enables access to the WebLogic Server node outside of the local node. Connecting to the SOANet network enables access to the DB container by its name (soadb).

To view the Administration Server logs, enter the following command:

$ podman logs -f \<Administration Server container name\>

Create SOA Managed Server containers

Note: These steps are required only for the soa and soaosb domain type. You can start containers to launch the SOA Managed Servers from the image created.

Create an environment variables file specific to each Managed Server in the cluster in the SOA domain. For example, soaserver1.env.list and soaserver2.env.list for a SOA cluster:

MANAGED_SERVER=<Managed Server name, either soa_server1 or soa_server2>
DOMAIN_NAME=soainfra
ADMIN_HOST=<Administration Server hostname>
ADMIN_PORT=<Node port number mapping Administration Server container port `7001`>
ADMIN_PASSWORD=<admin_password>
MANAGED_SERVER_CONTAINER=true
MANAGEDSERVER_PORT=<Container port number where Managed Server is running. Refer below note for details.>

IMPORTANT: In the Managed Servers environment variables file

- MANAGED_SERVER value must be soa_server1 or soa_server2 for the soa and soaosb domain type.
- ADMIN_PORT must match the **node** port mapping the Administration Server container port 7001.
- MANAGEDSERVER_PORT:
    - For 12.2.1.4 must be 8001 for soa_server1 or 8002 for soa_server2
    - For 14.1.2 must be 7003 for soa_server1 or 7005 for soa_server2.

Example, soaserver1.env.list for 12.2.1.4 will be:

MANAGED_SERVER=soa_server1
DOMAIN_NAME=soainfra
ADMIN_HOST=<Administration Server hostname>
ADMIN_PORT=7001
ADMIN_PASSWORD= Your-password
MANAGED_SERVER_CONTAINER=true
MANAGEDSERVER_PORT=8001
Example, soaserver1.env.list for 14.1.2 will be:
MANAGED_SERVER=soa_server1
DOMAIN_NAME=soainfra
ADMIN_HOST=<Administration Server hostname>
ADMIN_PORT=7001
ADMIN_PASSWORD= Your-password
MANAGED_SERVER_CONTAINER=true
MANAGEDSERVER_PORT=7003

To start a Podman container for the SOA server (for soa_server1), you can use the podman run command passing soaserver1.env.list with correct port values.

For example, the command for 12.2.1.4 will be:

$ podman run -it --name soams1 --network=SOANet -p 8001:8001 -v soadomain_vol:/u01/oracle/user_projects --env-file ./soaserver1.env.list container-registry.oracle.com/middleware/soasuite:<REPLACE-WITH-RELEASE-VERSION> "/u01/oracle/container-scripts/startMS.sh"

Similarly, for 14.1.2 command will be:

$ podman run -it --name soams1 --network=SOANet -p 7003:7003 -v soadomain_vol:/u01/oracle/user_projects --env-file ./soaserver1.env.list container-registry.oracle.com/middleware/soasuite:<REPLACE-WITH-RELEASE-VERSION> "/u01/oracle/container-scripts/startMS.sh"

The following lines indicate when the SOA Managed Server is ready to be used:

INFO: Managed Server is running

INFO: Managed Server has been started

Once the Managed Server container is created, you can view the server logs:

$ podman logs -f \<Managed Server container name\>

Create Oracle Service Bus Managed Server containers

Note: These steps are required only for the osb and soaosb domain type.
You can start containers to launch the Oracle Service Bus Managed Servers from the image created.

Create an environment variables file specific to each Managed Server in the cluster in the SOA domain. For example, osbserver1.env.list and osbserver2.env.list for an Oracle Service Bus cluster:

MANAGED_SERVER=<Managed Server name, either osb_server1 or osb_server2>
DOMAIN_NAME=soainfra
ADMIN_HOST=<Administration Server hostname>
ADMIN_PORT=<Node port number mapping Administration Server container port `7001`>
ADMIN_PASSWORD=<admin_password>
MANAGED_SERVER_CONTAINER=true
MANAGEDSERVER_PORT=<Container port number where Managed Server is running. Refer below note for details.>

IMPORTANT: In the Managed Servers environment variables file

- MANAGED_SERVER value must be osb_server1 or osb_server2 for the osb and soaosb domain type.
- ADMIN_PORT must match the **node** port mapping the Administration Server container port 7001 .
- MANAGEDSERVER_PORT:
    - For 12.2.1.4 must be 9001 for osb_server1 or 9002 for osb_server2
    - For 14.1.2 must be 8002 for osb_server1 or 8004 for osb_server2.

Example for 12.2.1.4 osbserver1.env.list is:

MANAGED_SERVER=osb_server1
DOMAIN_NAME=soainfra
ADMIN_HOST=<Administration Server hostname>

ADMIN_PORT=7001
ADMIN_PASSWORD= Your-password
MANAGED_SERVER_CONTAINER=true
MANAGEDSERVER_PORT=9001
Example for 14.1.2 osbserver1.env.list is:
MANAGED_SERVER=osb_server1
DOMAIN_NAME=soainfra
ADMIN_HOST=<Administration Server hostname>
ADMIN_PORT=7001
ADMIN_PASSWORD= Your-password
MANAGED_SERVER_CONTAINER=true
MANAGEDSERVER_PORT=8002

To start a Podman container for the Oracle Service Bus server (for osb_server1), you can use the podman run command passing osbserver1.env.list.

For example, the command for 12.2.1.4 will be:

$ podman run -it --name osbms1 --network=SOANet -p 9001:9001 -v soadomain_vol:/u01/oracle/user_projects --env-file ./osbserver1.env.list container-registry.oracle.com/middleware/soasuite:<REPLACE-WITH-RELEASE-VERSION> "/u01/oracle/container-scripts/startMS.sh"

And for 14.1.2 command will be:

$ podman run -it --name osbms1 --network=SOANet -p 8002:8002 -v soadomain_vol:/u01/oracle/user_projects --env-file ./osbserver1.env.list container-registry.oracle.com/middleware/soasuite:<REPLACE-WITH-RELEASE-VERSION> "/u01/oracle/container-scripts/startMS.sh"

The following lines indicate when the Oracle Service Bus Managed Server is ready to be used:

INFO: Managed Server is running

INFO: Managed Server has been started

Once the Managed Server container is created, you can view the server logs:

$ podman logs -f \<Managed Server container name\>

Access the Consoles

Now you can access the following Consoles:

For 12.2.1.4:

- Administration Server Web Console at http://<hostname>:7001/console with weblogic/Your-password credentials.
- EM Console at http://<hostname>:7001/em with weblogic/your-password credentials.
- SOA infra Console at http://<hostname>:8001/soa-infra with weblogic/your-password credentials.
- SOA infra Console at http://<hostname>:8002/soa-infra with weblogic/your-password credentials.
- Service Bus Console at http://<hostname>:7001/servicebus with weblogic/your-password credentials.

For 14.1.2:

- EM Console at http://<hostname>:7001/em with weblogic/your-password credentials.
- SOA infra Console at http://<hostname>:7003/soa-infra with weblogic/your-password credentials.
- SOA infra Console at http://<hostname>:7005/soa-infra with weblogic/your-password credentials.
- Service Bus Console at http://<hostname>:7001/servicebus with weblogic/your-password credentials.

Note: hostname is the FQDN of the host name where the container is running. Do not use 'localhost' for ADMIN_HOST. Use the actual FQDN name of the host as ADMIN_HOST.

Clean up the environment

1. Stop and remove all running containers from the node where the container is running:

   $ podman stop \<container name\>
   $ podman rm \<container name\>

   where containers are soadb, soaas, soams1, soams2, osbms1 and osbms2.

2. Clear the data volume:

   $ podman volume rm soadb_vol
   $ podman volume rm soadomain_vol

3. Remove the Podman network:

   $ podman network rm SOANet

Deploying NGINX Server from Docker Hub Repository

- Pull the NGINX docker image from the Docker Hub Repository

```
docker pull nginx

Using default tag: latest
Trying to pull repository docker.io/library/nginx ...
latest: Pulling from docker.io/library/nginx
61320b01ae5e: Pull complete
670a101d432b: Pull complete
405bd2df85b6: Pull complete
cc80efff8457: Pull complete
2b9310b2ee4b: Pull complete
6c4aa022e8e1: Pull complete
abddc69cb49d: Pull complete
Digest: sha256:fb39280b7b9eba5727c884a3c7810002e69e8f961cc373b89c92f14961d903a0
Status: Downloaded newer image for nginx:latest
nginx:latest
```

- Deploy the NGINX server in a container:

```
docker run -d --name nginx-server -p 8081:80 nginx

5bafad634e35ec7b3d631bc7d40a78e8d505687c90bc1611b65fb9da293315c9
```

Test locally: curl http://localhost:8081

Test via web browser:

http://Your-IP-address:8081/

Connect with us

Call +**1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at: **oracle.com/contact**.

blogs.oracle.com facebook.com/oracle twitter.com/oracle

Author: Anderson Souza