Deploying Oracle Database Operator Cluster Add-on for Oracle Kubernetes Engine (OKE) on Compute Cloud@Customer or Private Cloud Appliance

Version 2.0

Purpose statement

The purpose of this document is to describe how the Oracle Database Operator cluster add-on for Oracle Kubernetes Engine (OKE) on Compute Cloud@Customer and Private Cloud Appliance enables automated, Kubernetes-native management of Oracle Database workloads.

This paper provides guidance for deploying and configuring the operator to help development, database, DevOps, and GitOps teams simplify operations, reduce deployment complexity, and accelerate the lifecycle management of Oracle databases on Kubernetes.

Disclaimer

Table of contents

Purpose statement **2**
Introduction **4**
Prerequisites **5**
Deployment **5**
OKE Cluster Access and Security Configuration **8**
Prerequisites 8
Certificate Management and Oracle Database Operator Status on OKE (Compute Cloud@Customer / PCA) 9
Enabling Role Bindings 10
Deploying SIDB (Single-Instance Database) with Oracle Database Operator **12**
Labeling Worker Nodes for Oracle Database Deployments 13
Single Instance Database Deployment Manifest 13
Single Instance Database Pod Creation and Initialization 15
Post-Deployment Validation and Database Connectivity Checks 20
Exposing the Oracle Database via Compute Cloud@Customer or Private Cloud Appliance OKE Load Balancer Service **22**
Conclusion **26**

Introduction

On Compute Cloud@Customer and Private Cloud Appliance, cluster add-ons are optional components that can be deployed on Kubernetes clusters to extend core Kubernetes capabilities and enhance cluster manageability and performance.

The Oracle Database Operator cluster add-on for Oracle Kubernetes Engine (OKE) on Compute Cloud@Customer and Private Cloud Appliance extends the Kubernetes API by introducing custom resources and controllers that automate the Oracle Database lifecycle. This enables developers, database administrators, DevOps, and GitOps teams to significantly reduce the time, effort, and operational complexity associated with deploying, operating, and managing Oracle databases on Kubernetes.

This solution paper provides detailed guidance on how to deploy and configure the Oracle Database Operator cluster add-on for OKE on Compute Cloud@Customer and Private Cloud Appliance.

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.

Prerequisites

Before deploying the Oracle Database Operator cluster add-on for Oracle Kubernetes Engine (OKE) on Compute Cloud@Customer or Private Cloud Appliance, ensure that the following prerequisites are met:

- Oracle Kubernetes Engine (OKE) deployed on Compute Cloud@Customer or Private Cloud Appliance.
- The Certificate Manager add-on must be in ACTIVE state before you can use the Database Operator add-on.

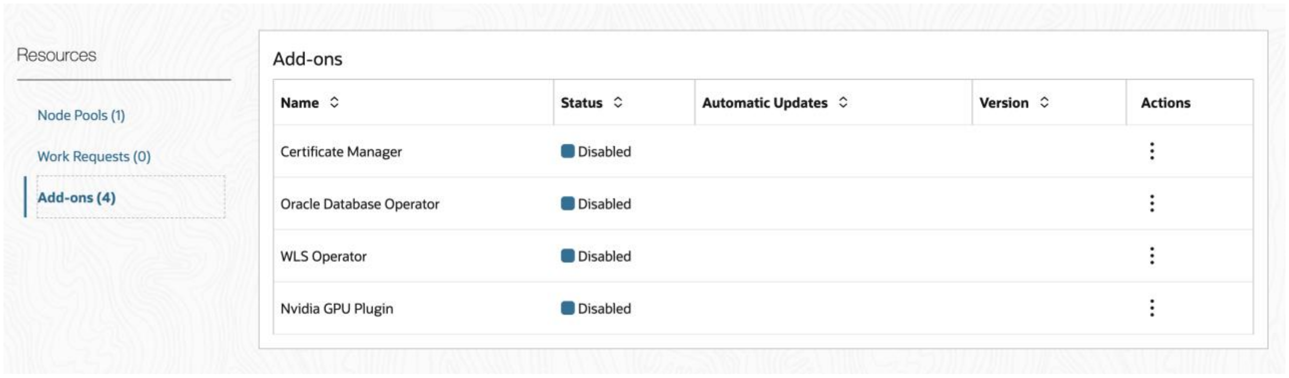
NOTE: Oracle Database Operator add-on can only be enabled on an existing OKE cluster.

Deployment

Complete these steps to deploy the Oracle Database Operator cluster add-on for Oracle Kubernetes Engine (OKE) on Compute Cloud@Customer or Private Cloud Appliance:

1. On Compute Cloud@Customer or Private Cloud Appliance management UI, navigate to **Dashboard**, then select **Containers**. Under the **Containers** section, click **Kubernetes Clusters (OKE)** and select your OKE cluster. This opens the cluster configuration page. Scroll to the bottom of the page and, under **Resources**, click **Add-ons** to display the list of available cluster add-ons.

NOTE: Since this is the first time configuring cluster add-ons on this Compute Cloud@Customer or Private Cloud Appliance, all add-ons will be listed as disabled by default.

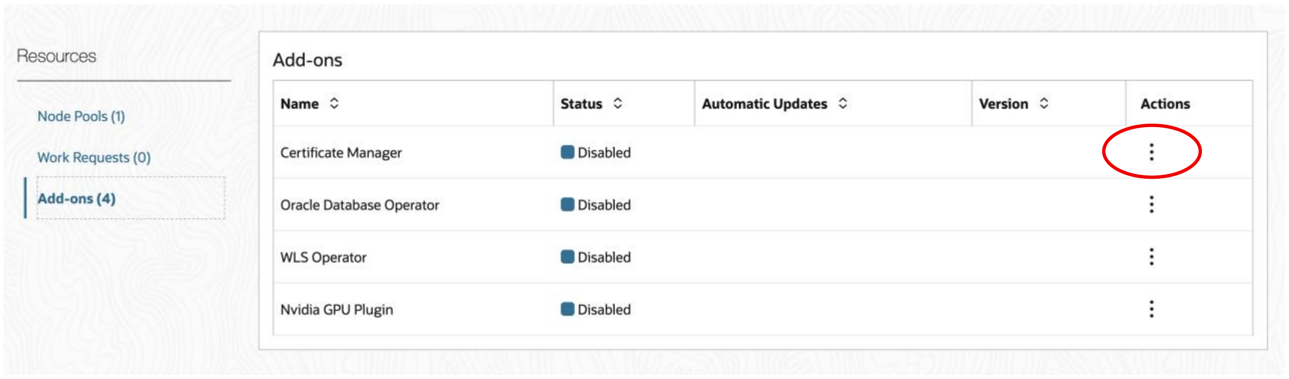


Add-ons

Name	Status	Automatic Updates	Version	Actions
Certificate Manager	Disabled			⋮
Oracle Database Operator	Disabled			⋮
WLS Operator	Disabled			⋮
Nvidia GPU Plugin	Disabled			⋮

Figure 1. List of OKE Add-ons available on Compute Cloud@Customer or Private Cloud Appliance.

2. Activate the Certificate Manager add-on before activating the Database Operator add-on. Click the Actions icon (three dots) in the Actions column.
3. Select **Enable** for the add-on that you want to deploy and enable for the cluster.



Add-ons

Name	Status	Automatic Updates	Version	Actions
Certificate Manager	🟦 Disabled			⋮
Oracle Database Operator	🟦 Disabled			⋮
WLS Operator	🟦 Disabled			⋮
Nvidia GPU Plugin	🟦 Disabled			⋮

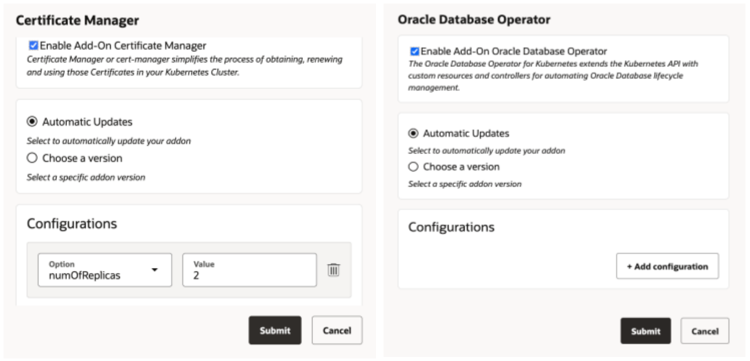
Figure 2. Activating Add-ons for OKE on Compute Cloud@Customer or Private Cloud Appliance.

4. **Add-on Version Update Options:** When enabling a cluster add-on on Compute Cloud@Customer, you must define how the add-on version is managed as new releases become available and as additional Kubernetes versions are supported by Oracle Kubernetes Engine (OKE). Two update strategies are supported: **Automatic Updates** or **Manual Version Selection**.

   - **Automatic Updates (Default):** The add-on is automatically deployed using the latest version compatible with the Kubernetes version running on the cluster. When newer add-on versions are released, the add-on is automatically updated, provided that the new version is compatible with both the cluster's Kubernetes version and the versions supported by OKE. Oracle recommends keeping Kubernetes clusters upgraded to supported versions to ensure continued compatibility and timely add-on updates.
   - **Manual Version Selection:** This option allows you to pin the add-on to a specific version, which remains in place until manually changed. The selected add-on version must be compatible with the Kubernetes version running on the cluster. When using the OCI Console, only compatible versions are displayed.

5. **Configuration:** When you enable the Certificate Manager cluster add-on, you can pass the following configuration in key/value pairs as arguments. This example uses two replicas only.
   a. Select **Add configuration** to select a configuration option and specify a value.
   b. Select **Add configuration** to set another configuration parameter.

The following screenshots show the Configuration selection and the list of configurable parameters that you can set on the Certificate Manager add-on and Oracle Database Operator.



Certificate Manager

[✓] Enable Add-On Certificate Manager
Certificate Manager or cert-manager simplifies the process of obtaining, renewing and using those Certificates in your Kubernetes Cluster.

[✓ Automatic Updates]
Select to automatically update your addon
Choose a version
Select a specific addon version

Configurations
Option: numOfReplicas
Value: 2

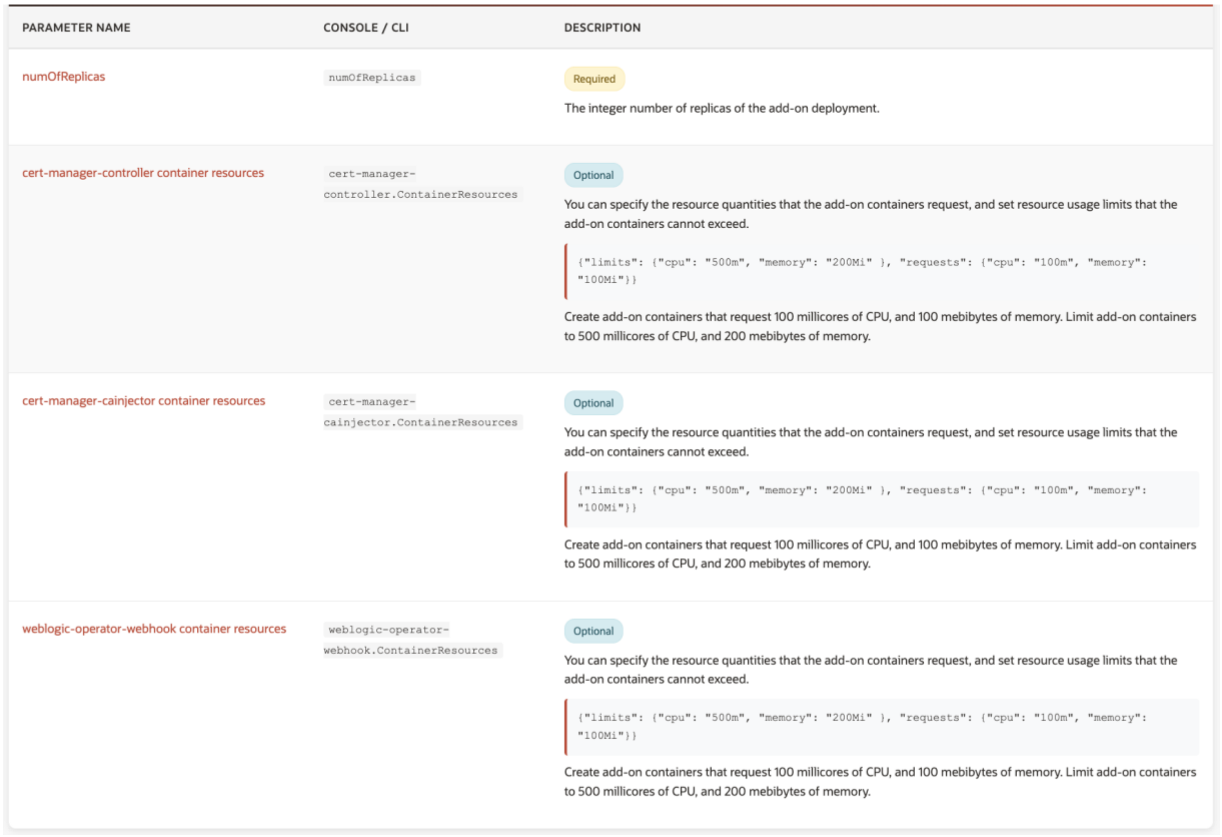
Oracle Database Operator

[✓] Enable Add-On Oracle Database Operator
The Oracle Database Operator for Kubernetes extends the Kubernetes API with custom resources and controllers for automating Oracle Database lifecycle management.

[✓ Automatic Updates]
Select to automatically update your addon
Choose a version
Select a specific addon version

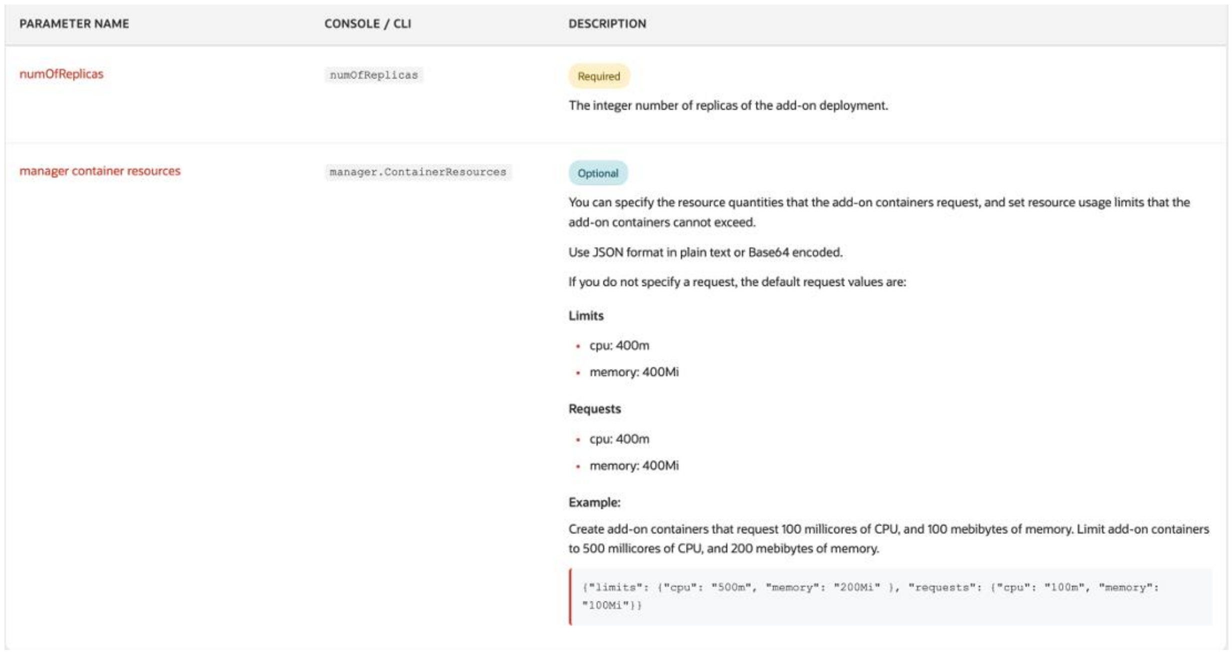
Configurations

Figure 3. Configuration for Certificate Manager and Oracle Database Operator Add-ons for OKE on Compute Cloud@Customer or Private Cloud Appliance.



PARAMETER NAME	CONSOLE / CLI	DESCRIPTION
numOfReplicas	`numOfReplicas`	Required The integer number of replicas of the add-on deployment.
cert-manager-controller container resources	`cert-manager-controller.ContainerResources`	Optional You can specify the resource quantities that the add-on containers request, and set resource usage limits that the add-on containers cannot exceed. `{"limits": {"cpu": "500m", "memory": "200Mi" }, "requests": {"cpu": "100m", "memory": "100Mi"}}` Create add-on containers that request 100 millicores of CPU, and 100 mebibytes of memory. Limit add-on containers to 500 millicores of CPU, and 200 mebibytes of memory.
cert-manager-cainjector container resources	`cert-manager-cainjector.ContainerResources`	Optional You can specify the resource quantities that the add-on containers request, and set resource usage limits that the add-on containers cannot exceed. `{"limits": {"cpu": "500m", "memory": "200Mi" }, "requests": {"cpu": "100m", "memory": "100Mi"}}` Create add-on containers that request 100 millicores of CPU, and 100 mebibytes of memory. Limit add-on containers to 500 millicores of CPU, and 200 mebibytes of memory.
weblogic-operator-webhook container resources	`weblogic-operator-webhook.ContainerResources`	Optional You can specify the resource quantities that the add-on containers request, and set resource usage limits that the add-on containers cannot exceed. `{"limits": {"cpu": "500m", "memory": "200Mi" }, "requests": {"cpu": "100m", "memory": "100Mi"}}` Create add-on containers that request 100 millicores of CPU, and 100 mebibytes of memory. Limit add-on containers to 500 millicores of CPU, and 200 mebibytes of memory.

Figure 4. Configuration Options for Certificate Manager Add-ons for OKE on Compute Cloud@Customer or Private Cloud Appliance.



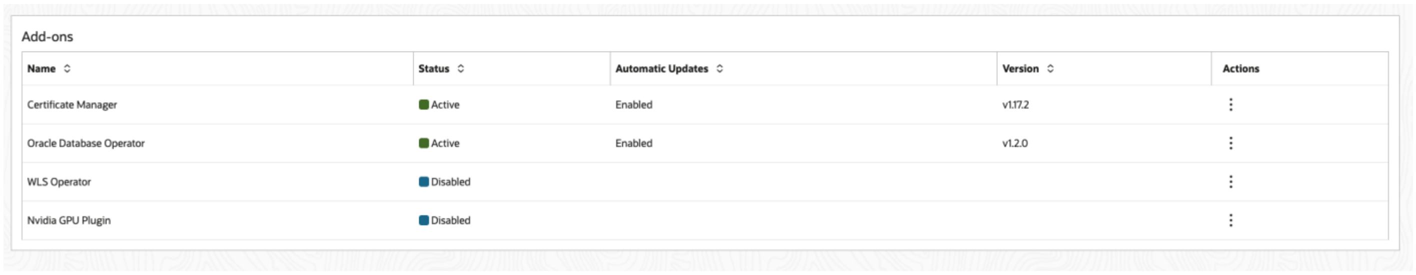
PARAMETER NAME	CONSOLE / CLI	DESCRIPTION
numOfReplicas	`numOfReplicas`	Required The integer number of replicas of the add-on deployment.
manager container resources	`manager.ContainerResources`	Optional You can specify the resource quantities that the add-on containers request, and set resource usage limits that the add-on containers cannot exceed. Use JSON format in plain text or Base64 encoded. If you do not specify a request, the default request values are: **Limits** • cpu: 400m • memory: 400Mi **Requests** • cpu: 400m • memory: 400Mi **Example:** Create add-on containers that request 100 millicores of CPU, and 100 mebibytes of memory. Limit add-on containers to 500 millicores of CPU, and 200 mebibytes of memory. `{"limits": {"cpu": "500m", "memory": "200Mi" }, "requests": {"cpu": "100m", "memory": "100Mi"}}`

Figure 5. Configuration Options for Oracle Database Operator Add-ons for OKE on Compute Cloud@Customer or Private Cloud Appliance.

(Default) Automatic Updates: If possible, this option automatically updates the add-on when new versions become available. This is the default behavior. The newest version of the add-on that supports the Kubernetes version that is specified for the cluster is deployed when you install the add-on. When a newer version of the add-on is released, the add-on is automatically updated if the new add-on version is compatible with the versions of Kubernetes that are supported by OKE at that time and the version of Kubernetes that the cluster is running. Best practice is to keep your clusters upgraded so that they are always running versions of Kubernetes that are listed as currently supported by OKE. See Supported Versions of Kubernetes in Kubernetes Engine (OKE) on Compute Cloud@Customer and Updating an OKE Cluster.

Choose a Version: This option keeps the add-on on the specific version that you select until you change it.
If you specify that you want to choose the version of the add-on do deploy, then the version that you choose is enabled. Ensure that the add-on version is compatible with the Kubernetes version that you have selected for the cluster or that is already running on the cluster. When you use the Console, you select the version from a list. All versions on the list are compatible with the Kubernetes version that you have selected for the cluster or that is already running on the cluster.

If you select Choose a Version, then you must select a version from the list.


Add-ons

Name	Status	Automatic Updates	Version	Actions
Certificate Manager	🟩 Active	Enabled	v1.17.2	⋮
Oracle Database Operator	🟩 Active	Enabled	v1.2.0	⋮
WLS Operator	🟦 Disabled			⋮
Nvidia GPU Plugin	🟦 Disabled			⋮

Figure 6. Certificate Manager and Oracle Database Operator Add-ons Status

OKE Cluster Access and Security Configuration

Prerequisites

Before accessing the Oracle Kubernetes Engine (OKE) cluster running on Compute Cloud@Customer (C3) or Oracle Private Cloud Appliance (PCA), ensure the following prerequisites are met on the client system (Oracle Linux, macOS, or Windows):

- OCI Command Line Interface (OCI CLI) is downloaded, installed, and configured with valid OCI IAM credentials (tenancy OCID, user OCID, API signing key, and region).
- Network connectivity from the client system to the C3 or PCA control plane endpoint.
- TLS trust configuration that allows the client to validate the Kubernetes API server certificate.
- kubectl installed and available in the system PATH.

Refer to the OCI CLI installation guides by operating system:

- Oracle Linux / Linux: https://docs.oracle.com/en-us/iaas/Content/API/SDKDocs/cliinstall.htm#InstallingCLI__linux
- macOS: https://docs.oracle.com/en-us/iaas/Content/API/SDKDocs/cliinstall.htm#InstallingCLI__macos_homebrew
- Windows: https://docs.oracle.com/en-us/iaas/Content/API/SDKDocs/cliinstall.htm#InstallingCLI__windows

During the initial configuration of Oracle Kubernetes Engine (OKE) on Compute Cloud@Customer (C3) or Oracle Private Cloud Appliance (PCA), the following commands perform two essential setup tasks required for deploying Oracle Database using the Oracle Database Operator.

The **curl** command is used to validate connectivity to the C3 or PCA control plane and to retrieve the certificate authority (CA) chain. This step ensures that the Kubernetes API endpoint is trusted by the client system and that TLS communication can be established securely. Run the following curl command line below and copy the content to a new file (~/.oci/ca.crt).

```
curl -vk https://iaas.<fqdn_of_your_c3_or_pca/cachain
```

The **oci ce cluster create-kubeconfig** trigger then generates and configures the local Kubernetes kubeconfig file, enabling authenticated access to the OKE cluster using OCI IAM–based authentication tokens. This configuration allows standard Kubernetes tooling, such as kubectl, to securely interact with the cluster. Run the following command line below to create a new kubeconfig file.

```
oci ce cluster create-kubeconfig --cluster-id ocid1.ccccluster.oc1.us-sanjose-
1.ivcyvpvq5wa.amaaaaaakdrwrhiam44wu43tmz2gk6tgmjzgenjzg5wdeylegryde3jrna2q --file $HOME/.kube/config --token-
version 2.0.0 --kube-endpoint PUBLIC_ENDPOINT --cert-bundle ~/.oci/ca.crt

New config written to the Kubeconfig file $HOME/.kube/config
```

Together, these steps establish secure, authenticated communication with the OKE cluster and prepare the environment for managing Oracle Database deployments through the Oracle Database Operator on C3 or PCA.

Certificate Management and Oracle Database Operator Status on OKE (Compute Cloud@Customer / PCA)

The **kubectl get all -n cert-manager** command provides a consolidated view of all Kubernetes resources deployed by cert-manager within the cert-manager namespace on an OKE cluster running on Oracle Compute Cloud@Customer or Private Cloud Appliance (PCA). The output confirms that all core cert-manager components are healthy and operational: the cert-manager controller (which reconciles Certificate and Issuer resources), the CA injector (which automatically injects trusted CA bundles into Kubernetes objects), and the webhook (which validates and mutates certificate-related resources). Each component is deployed as a Kubernetes Deployment with one ready Pod and exposed internally via ClusterIP Services, indicating a stable TLS automation framework for Kubernetes workloads.

Additionally, the **kubectl get pods -n oracle-database-operator-system** command confirms that the Oracle Database Operator controller is running successfully. This operator manages the full lifecycle of Oracle Databases on Kubernetes, including provisioning, configuration, and ongoing operations.

Together, these results demonstrate a healthy Kubernetes environment that supports secure TLS certificate automation and enterprise-grade database services on OKE running in customer-managed private cloud environments.

```
kubectl get all -n cert-manager
NAME                                         READY   STATUS    RESTARTS   AGE
pod/cert-manager-57547f5ddd-xz77p            1/1     Running   0          3d16h
pod/cert-manager-cainjector-7f5bcd98bf-lg99g   1/1     Running   0          3d16h
pod/cert-manager-webhook-6684bdd794-dznn8    1/1     Running   0          3d16h

NAME                              TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)            AGE
service/cert-manager              ClusterIP   10.136.86.242    <none>        9402/TCP           3d16h
service/cert-manager-cainjector   ClusterIP   10.138.67.127    <none>        9402/TCP           3d16h
service/cert-manager-webhook      ClusterIP   10.139.239.138   <none>        443/TCP,9402/TCP   3d16h

NAME                                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/cert-manager              1/1     1            1           3d16h
deployment.apps/cert-manager-cainjector   1/1     1            1           3d16h
deployment.apps/cert-manager-webhook      1/1     1            1           3d16h

NAME                                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/cert-manager-57547f5ddd              1         1         1       3d16h
replicaset.apps/cert-manager-cainjector-7f5bcd98bf   1         1         1       3d16h
replicaset.apps/cert-manager-webhook-6684bdd794      1         1         1       3d16h
ansouza@ansouza-mac .oci %
```

```
kubectl get pods -n oracle-database-operator-system
NAME                                                           READY   STATUS    RESTARTS   AGE
oracle-database-operator-controller-manager-5bfbbc869-m487g   1/1     Running   0          3d16h
```

Enabling Role Bindings

To enable proper operation of the Oracle Database Kubernetes Operator, cluster-level RBAC permissions are required. The operator is responsible for orchestrating database lifecycles that depend on cluster-scoped resources such as PersistentVolumes, Nodes, and DaemonSets, which are not confined to a single namespace. For this reason, a dedicated ClusterRole is defined to grant read and write access to these resources, and a corresponding ClusterRoleBinding associates these permissions with the ServiceAccount under which the operator runs. This RBAC configuration allows the operator to provision and manage database storage, understand node topology for high availability and RAC placement, and deploy node-level components required for database initialization and management. Without these cluster-level permissions, the operator would be unable to reconcile database state or provision storage correctly, resulting in failed or incomplete database deployments.

The following is an example of a role_bindings.yaml file, and how to apply the file. This file enables proper function of the Oracle Database Kubernetes Operator. Apply the role bindings.yaml file using the **kubectl apply -f role_bindings.yaml** command line listed below.

```
cat role_bindings.yaml

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pv-cluster-scope-role
rules:
- apiGroups:
  - ""
  resources:
  - persistentvolumes
  - daemonsets
  - nodes
  verbs:
  - get
  - list
  - watch
  - create
  - delete
  - update
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: pv-cluster-scope-rolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pv-cluster-scope-role
subjects:
- kind: ServiceAccount
  name: default
  namespace: oracle-database-operator-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: oracle-database-operator-oracle-database-operator-manager-rolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: oracle-database-operator-manager-role
subjects:
- kind: ServiceAccount
  name: default
  namespace: oracle-database-operator-system

kubectl apply -f role_bindings.yaml

clusterrole.rbac.authorization.k8s.io/pv-cluster-scope-role created
clusterrolebinding.rbac.authorization.k8s.io/pv-cluster-scope-rolebinding created
clusterrolebinding.rbac.authorization.k8s.io/oracle-database-operator-oracle-database-operator-manager-rolebinding
created
```

Because Oracle Database Operator is managed by a deployment, Oracle recommends that you restart the deployment. When you restart it, the new pod starts with the updated RBAC. Run the following command:

```
kubectl rollout restart deployment oracle-database-operator-controller-manager \
  -n oracle-database-operator-system

deployment.apps/oracle-database-operator-controller-manager restarted
```

Verify if the deployment has been restarted correctly:

```
kubectl get pods -n oracle-database-operator-system

NAME                                                           READY   STATUS    RESTARTS   AGE
oracle-database-operator-controller-manager-78bfbc698f-9knc5   1/1     Running   0          3m11s
```

Check operator logs for RBAC errors (should be clean)

```
kubectl logs -n oracle-database-operator-system \
 deployment/oracle-database-operator-controller-manager | grep -i rbac
```

NOTE: If there is no output from this command, then there are no reported errors.

Verify permissions explicitly

```
kubectl auth can-i get persistentvolumes --as=system:serviceaccount:oracle-database-operator-system:default

Expected output: yes
```

This output indicates the following:

- Controller initialized successfully
- Informers/watches were created
- RBAC permissions are now valid
- Operator is fully ready to reconcile databases

Deploying SIDB (Single-Instance Database) with Oracle Database Operator

With OKE cluster access and security configurations properly configured as described in the previous section, the environment is now prepared for deploying the Single Instance Database (SIDB) on OKE worker nodes. This section provides the detailed steps for executing this deployment.

During SIDB deployment, two Kubernetes secrets are created to support secure database provisioning. The first secret stores the Oracle database administrator password and is consumed by the Oracle Database Operator during database initialization. The second secret contains authentication credentials for Oracle Container Registry, enabling Kubernetes to securely pull the Oracle Database container images required to run the database. Together, these secrets allow the operator to provision and start the Single-Instance Database without embedding credentials directly in manifests or pod specifications.

1. **Set the database admin password:** The kubectl command line listed below creates a Kubernetes secret that stores the Oracle database administrator password. Internally the oracle_pwd is base64-encoded password format. For example:

   ```
   kubectl create secret generic db-admin-secret --from-literal=oracle_pwd=Oracle_26ai

   secret/db-admin-secret created
   ```

2. **Oracle Container Registry credentials**: Creates a Docker registry authentication secret that allows Kubernetes to pull Oracle Database container images from external registry. For example:

   ```
   kubectl create secret docker-registry oracle-container-registry-secret \
     --docker-server=container-registry.oracle.com \
     --docker-username=user@your-email.com\
     --docker-password=<OCR_AUTH_TOKEN> \
     --docker-email=user@your-email.com
   ```

```
secret/oracle-container-registry-secret created
```

NOTE: The <OCR_AUTH_TOKEN> is the token that you need to generate in Oracle Container Registry. Refer to https://container-registry.oracle.com/

3. Confirm that the secret has been added:

```
kubectl get secret
NAME                                TYPE                              DATA   AGE
db-admin-secret                     Opaque                            1      17m
oracle-container-registry-secret    kubernetes.io/dockerconfigjson    1      2m4s
```

Labeling Worker Nodes for Oracle Database Deployments

If your OKE cluster contains more than one node pool, then Oracle recommends that you label the worker nodes that will host Oracle Database workloads managed by the Oracle Database Kubernetes Operator. Node labeling ensures that database pods are scheduled only on the intended nodes, providing proper workload isolation, predictable performance, and operational consistency.

Before applying labels, you must first identify the correct node names in the cluster. This can be done using the following command:

```
kubectl get nodes
```

After you identify the appropriate nodes, apply a dedicated label (for example, node-role=database) to each node that will be used for Oracle Database deployments:

```
kubectl label node \
  oke-amaaaaaakdrwrhiamjydkm3gnrzgyzti-8cfct \
  oke-amaaaaaakdrwrhiamjydkm3gnrzgyzti-ncngb \
  oke-amaaaaaakdrwrhiamjydkm3gnrzgyzti-7mhsp \
  node-role=database
```

After labeling the nodes, verify that the labels have been applied correctly by filtering the nodes using the label selector:

```
kubectl get nodes -l node-role=database
```

A successful configuration will return only the nodes designated for Oracle Database workloads, confirming that they are ready to be used by Oracle Database Operator for scheduling Single Instance Database pods.

Single Instance Database Deployment Manifest

With the required secrets properly configured, the next step is to define the Kubernetes manifest `(.yaml file)` used to deploy a Single Instance Oracle Database using the Oracle Database Kubernetes Operator.

The `sidb-create-v1.yaml` file listed below, defines a Single Instance Oracle Database deployment on Kubernetes using the Oracle Database Operator. This manifest leverages Kubernetes Custom Resource Definitions (CRDs) to declaratively provision, configure, and manage the lifecycle of an Oracle Database instance running as a containerized workload on OKE clusters deployed on Oracle Compute Cloud@Customer or Private Cloud Appliance (PCA).

At a high level, this YAML file instructs the Oracle Database Operator to create a fully functional Oracle Database Enterprise Edition instance, including persistent storage, security credentials, database configuration parameters, and container image details. Listed below are the explanation of each section of the file.

API Version and Resource Type: The `apiVersion: database.oracle.com/v4` and `kind: SingleInstanceDatabase` specify that this resource is managed by Oracle Database Operator. The operator continuously reconciles this resource to ensure the database instance is created, configured, and maintained in the desired state.

Metadata: The metadata section defines the Kubernetes object name `(sidb-sample)` and namespace (default). This name uniquely identifies the database instance within the cluster and is used by the operator to track and manage the database lifecycle.

Database Identity and Configuration

- `sid: ORCL1` defines the Oracle System Identifier (SID) for the database instance. The SID uniquely identifies the database within the container.
- `edition: enterprise` specifies that Oracle Database Enterprise Edition is deployed, enabling advanced enterprise features.
- `charset: AL32UTF8` configures the database character set, which is the recommended Unicode character set for modern enterprise applications.
- `pdbName: orclpdb1` creates a Pluggable Database (PDB) inside the Container Database (CDB), enabling multitenant architecture and application isolation.

Security and Credentials: The `adminPassword` section references a Kubernetes Secret (`db-admin-secret`) that securely stores the database administrative password. This approach ensures sensitive credentials are never embedded directly in the YAML file, aligning with Kubernetes and enterprise security best practices.

Archivelog Configuration: The `archiveLog: true` parameter enables Oracle ArchiveLog mode. This is a critical setting for enterprise workloads, as it supports point-in-time recovery, backup strategies, and integration with disaster recovery and data protection solutions.

Container Image Configuration: The image block specifies the Oracle Database container image and version (23.5.0.0) pulled from the Oracle Container Registry. The `pullSecrets` field references a Kubernetes secret used to authenticate to the registry, which is especially important in restricted or private network environments such as Compute Cloud@Customer and PCA.

Persistent Storage Configuration: The persistence section defines how database data is stored:

- `size: 100Gi` allocates persistent storage capacity for database files.
- `storageClass: oci-bv` specifies the storage backend, mapping the database to OCI Block Volumes in environments where OCI storage is available (including C3 and PCA).
- `accessMode: ReadWriteOnce` ensures the volume is mounted by a single node, which is appropriate for a single-instance database deployment.

This configuration guarantees that database data remains persistent across pod restarts, upgrades, or rescheduling events.

Replica Count: The `replicas:1` setting explicitly deploys a single database pod, aligning with the Single Instance Database architecture. High availability at the infrastructure layer is typically handled through Kubernetes node resiliency and underlying storage durability rather than database-level clustering.

This YAML-driven approach enables fully automated, repeatable, and auditable Oracle Database deployments on Kubernetes running in customer-controlled environments. It aligns with modern GitOps and Infrastructure-as-Code practices while preserving enterprise-grade Oracle Database capabilities. The result is a consistent deployment model across OCI public regions, Compute Cloud@Customer, and Private Cloud Appliance—ideal for regulated, air-gapped, or sovereignty-sensitive workloads.

```
cat sidb-create-v1.yaml

apiVersion: database.oracle.com/v4
kind: SingleInstanceDatabase
metadata:
  name: sidb-sample
  namespace: default

spec:
  ## Use only alphanumeric characters for sid
  sid: ORCL1

  ## DB edition
  edition: enterprise

  ## Secret containing SIDB password mapped to secretKey
  adminPassword:
    secretName: db-admin-secret

  ## DB character set
  charset: AL32UTF8

  ## PDB name
  pdbName: orclpdb1

  ## Enable/Disable ArchiveLog
  archiveLog: true

  ## Database image details
  image:
    pullFrom: container-registry.oracle.com/database/enterprise:23.5.0.0
    pullSecrets: oracle-container-registry-secret

  ## Persistent storage configuration
  persistence:
    size: 100Gi
    # oci-bv applies to OCI block volumes.
    # Use "standard" for Minikube or other environments.
    storageClass: "oci-bv"
    accessMode: "ReadWriteOnce"

  ## Count of Database Pods
  replicas: 1
```

- Apply the sidb-create-v1.yaml file. To apply, run the following command line:

```
kubectl apply -f sidb-create-v1.yaml

singleinstancedatabase.database.oracle.com/sidb-sample created
```

Single Instance Database Pod Creation and Initialization

The `sidb-create-v1.yaml` example applied in the preceding section represents the successful application of the Single Instance Database (SIDB) Kubernetes manifest and the subsequent lifecycle actions orchestrated by Oracle Database Kubernetes Operator. The operator uses the SingleInstanceDatabase Custom Resource Definition (CRD) to continuously reconcile the desired state defined in the SIDB `.yaml` file. After the manifest is applied, the operator schedules a new database pod (`sidb-sample-xh30b`) on to a worker node labeled for database workloads and manages all database provisioning and configuration steps.

The pod initially enters the Init phase, during which the operator launches an init container (`init-wallet`) to prepare the required Oracle wallet and security artifacts on the persistent volume. After initialization completes, the operator starts the main database container and triggers the Oracle Database creation process. During this phase, the database pod remains in a Running but not Ready state, which is expected behavior. The readiness probe is intentionally blocked while the operator holds a database creation lock (`.ORCL1.create_lck`). The lock ensures that the database is fully provisioned before accepting client connections.

To monitor the status and progress of the database deployment in real time, the following command can be used:

```
kubectl get pods -l app=sidb-sample -w
```

This command continuously watches the database pod as it transitions through the initialization and creation phases.

The container logs show the complete operator-driven database creation workflow, including listener startup, datafile creation, instance initialization, pluggable database (PDB) creation, and post-configuration tasks. After database creation is complete, the operator releases the creation lock and replaces it with an existence lock, signaling that the database is fully initialized and operational. At this point, the readiness probe succeeds and the database becomes available for application connectivity. A successful readiness probe confirms that your SIDB deployment is successful on OKE running on Oracle Compute Cloud@Customer or on Private Cloud Appliance.

kubectl get pods -l app=sidb-sample -w

```
NAME                 READY   STATUS     RESTARTS   AGE
sidb-sample-xh30b    0/1     Init:0/1   0          10s
^C%
```

kubectl describe pods sidb-sample-xh30b

```
Name:             sidb-sample-xh30b
Namespace:        default
Priority:         0
Service Account:  default
Node:             oke-amaaaaaakdrwrhiamjydkm3gnrzgyzti-ncngb/171.31.8.6
Start Time:       Fri, 23 Jan 2026 14:00:17 -0700
Labels:           app=sidb-sample
                  version=
Annotations:      <none>
Status:           Running
IP:               10.244.8.68
IPs:
  IP:           10.244.8.68
Controlled By:  SingleInstanceDatabase/sidb-sample
Init Containers:
  init-wallet:
    Container ID:  cri-o://c9505aceae2983cba4c2117e489011840b8a70004c841d32f8defff6ece2558b
    Image:         container-registry.oracle.com/database/enterprise:21.3.0.0
    Image ID:      container-
registry.oracle.com/database/enterprise@sha256:c5ad975902cfe523a4ac9f046ec87cd0fd41c24118651ca0e7194f736ae4e3c7
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sh
    Args:
      -c
      if [ ! -f $ORACLE_BASE/oradata/.${ORACLE_SID}${CHECKPOINT_FILE_EXTN} ] || [ ! -f
${ORACLE_BASE}/oradata/dbconfig/$ORACLE_SID/.docker_enterprise ]; then while [ ! -f ${WALLET_DIR}/ewallet.p12 ] ||
pgrep -f $WALLET_CLI > /dev/null; do sleep 0.5; done; fi
    State:          Terminated
      Reason:       Completed
      Exit Code:    0
      Started:      Fri, 23 Jan 2026 14:00:34 -0700
      Finished:     Fri, 23 Jan 2026 14:00:42 -0700
    Ready:          True
    Restart Count:  0
    Environment:
      ORACLE_SID:  ORCL1
      WALLET_CLI:  mkstore
      WALLET_DIR:  /opt/oracle/oradata/dbconfig/${ORACLE_SID}/.wallet
    Mounts:
      /opt/oracle/oradata from datafiles-vol (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-2fdxg (ro)
Containers:
  sidb-sample:
    Container ID:   cri-o://17e75250ce5c26bde6c4506d20456dd59f009c72963c384bac76ce5f0a9e3508
    Image:          container-registry.oracle.com/database/enterprise:21.3.0.0
    Image ID:       container-
registry.oracle.com/database/enterprise@sha256:c5ad975902cfe523a4ac9f046ec87cd0fd41c24118651ca0e7194f736ae4e3c7
    Ports:          1521/TCP, 5500/TCP
    Host Ports:     0/TCP, 0/TCP
    State:          Running
      Started:      Fri, 23 Jan 2026 14:00:43 -0700
    Ready:          False
    Restart Count:  0
    Readiness:      exec [/bin/sh -c if [ -f $ORACLE_BASE/checkDBLockStatus.sh ]; then
$ORACLE_BASE/checkDBLockStatus.sh ; else $ORACLE_BASE/checkDBStatus.sh; fi ] delay=20s timeout=20s period=60s
#success=1 #failure=3
    Environment:
      SVC_HOST:             sidb-sample
      SVC_PORT:             1521
      CREATE_PDB:           true
      ORACLE_SID:           ORCL1
      WALLET_DIR:           /opt/oracle/oradata/dbconfig/${ORACLE_SID}/.wallet
      ORACLE_PDB:           orclpdb1
      ORACLE_CHARACTERSET:  AL32UTF8
      ORACLE_EDITION:       enterprise
      INIT_SGA_SIZE:
      INIT_PGA_SIZE:
      SKIP_DATAPATCH:       true
    Mounts:
      /opt/oracle/oradata from datafiles-vol (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-2fdxg (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       False
  ContainersReady             False
  PodScheduled                True
Volumes:
  datafiles-vol:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  sidb-sample
    ReadOnly:   false
  oracle-pwd-vol:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  db-admin-secret
    Optional:    true
  tls-secret-vol:
    Type:       EmptyDir (a temporary directory that shares a pod's lifetime)
    Medium:
    SizeLimit:  <unset>
  custom-scripts-vol:
    Type:       EmptyDir (a temporary directory that shares a pod's lifetime)
    Medium:
    SizeLimit:  <unset>
  kube-api-access-2fdxg:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    Optional:                false
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              node-role=database
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason                  Age   From                     Message
  ----    ------                  ----  ----                     -------
  Normal  Scheduled               41s   default-scheduler        Successfully assigned default/sidb-sample-xh30b
to oke-amaaaaaakdrwrhiamjydkm3gnrzgyzti-ncngb
  Normal  SuccessfulAttachVolume  27s   attachdetach-controller  AttachVolume.Attach succeeded for volume "csi-
94254826-91ef-4b77-a7fd-8e032c6ba693"
  Normal  Pulled                  24s   kubelet                  spec.initContainers{init-wallet}: Container image
"container-registry.oracle.com/database/enterprise:21.3.0.0" already present on machine
  Normal  Created                 24s   kubelet                  spec.initContainers{init-wallet}: Created
container: init-wallet
  Normal  Started                 24s   kubelet                  spec.initContainers{init-wallet}: Started
container init-wallet
  Normal  Pulled                  15s   kubelet                  spec.containers{sidb-sample}: Container image
"container-registry.oracle.com/database/enterprise:21.3.0.0" already present on machine
  Normal  Created                 15s   kubelet                  spec.containers{sidb-sample}: Created container:
sidb-sample
  Normal  Started                 15s   kubelet                  spec.containers{sidb-sample}: Started container
sidb-sample
```

In another terminal, check the logs and status of the database creation:

kubectl logs -f sidb-sample-xh30b -c sidb-sample

```
[2026:01:23 21:00:43]: Acquiring lock .ORCL1.create_lck with heartbeat 30 secs
[2026:01:23 21:00:43]: Lock acquired
[2026:01:23 21:00:43]: Starting heartbeat
[2026:01:23 21:00:43]: Lock held .ORCL1.create_lck
ORACLE EDITION: ENTERPRISE

LSNRCTL for Linux: Version 21.0.0.0.0 - Production on 23-JAN-2026 21:00:43

Copyright (c) 1991, 2021, Oracle.  All rights reserved.

Starting /opt/oracle/product/21c/dbhome_1/bin/tnslsnr: please wait...

TNSLSNR for Linux: Version 21.0.0.0.0 - Production
System parameter file is /opt/oracle/homes/OraDB21Home1/network/admin/listener.ora
Log messages written to /opt/oracle/diag/tnslsnr/sidb-sample-xh30b/listener/alert/log.xml
Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(KEY=EXTPROC1)))
Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=0.0.0.0)(PORT=1521)))

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=IPC)(KEY=EXTPROC1)))
STATUS of the LISTENER
------------------------
Alias                     LISTENER
Version                   TNSLSNR for Linux: Version 21.0.0.0.0 - Production
Start Date                23-JAN-2026 21:00:43
Uptime                    0 days 0 hr. 0 min. 0 sec
Trace Level               off
Security                  ON: Local OS Authentication
SNMP                      OFF
Listener Parameter File   /opt/oracle/homes/OraDB21Home1/network/admin/listener.ora
Listener Log File         /opt/oracle/diag/tnslsnr/sidb-sample-xh30b/listener/alert/log.xml
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(KEY=EXTPROC1)))
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=0.0.0.0)(PORT=1521)))
The listener supports no services
The command completed successfully
Prepare for db operation
8% complete
Copying database files
31% complete
Creating and starting Oracle instance
32% complete
36% complete
40% complete
43% complete
46% complete
Completing Database Creation
51% complete
54% complete
Creating Pluggable Databases
58% complete
77% complete
Executing Post Configuration Actions
100% complete
Database creation complete. For details check the logfiles at:
 /opt/oracle/cfgtoollogs/dbca/ORCL1.
Database Information:
Global Database Name:ORCL1
System Identifier(SID):ORCL1
Look at the log file "/opt/oracle/cfgtoollogs/dbca/ORCL1/ORCL1.log" for further details.

SQL*Plus: Release 21.0.0.0.0 - Production on Fri Jan 23 21:06:10 2026
Version 21.3.0.0.0

Copyright (c) 1982, 2021, Oracle.  All rights reserved.
```

```
Connected to:
Oracle Database 21c Enterprise Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL>
System altered.

SQL>
System altered.

SQL>
Pluggable database altered.

SQL>
PL/SQL procedure successfully completed.

SQL> SQL>
Session altered.

SQL>
User created.

SQL>
Grant succeeded.

SQL>
Grant succeeded.

SQL>
Grant succeeded.

SQL>
User altered.

SQL> SQL> Disconnected from Oracle Database 21c Enterprise Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0
The Oracle base remains unchanged with value /opt/oracle

Executing user defined scripts
/opt/oracle/runUserScripts.sh: running /opt/oracle/scripts/extensions/setup/swapLocks.sh
[2026:01:23 21:06:11]: Releasing lock .ORCL1.create_lck
[2026:01:23 21:06:11]: Lock released .ORCL1.create_lck
[2026:01:23 21:06:11]: Acquiring lock .ORCL1.exist_lck with heartbeat 30 secs
[2026:01:23 21:06:11]: Lock acquired
[2026:01:23 21:06:11]: Starting heartbeat
[2026:01:23 21:06:11]: Lock held .ORCL1.exist_lck

DONE: Executing user defined scripts

The Oracle base remains unchanged with value /opt/oracle
#########################
DATABASE IS READY TO USE!
#########################
```

Post-Deployment Validation and Database Connectivity Checks

After the database installation is complete, use the following commands to check the database status:

- Check the custom resource status:
  `kubectl describe singleinstancedatabase sidb-sample`

- Look for status updates:

```
kubectl get singleinstancedatabase sidb-sample -o yaml | grep -A 10 status
```

When the database status is Ready, the pod status changes to running status. For example:

```
kubectl get pod sidb-sample-xh30b

NAME                READY   STATUS    RESTARTS   AGE
sidb-sample-xh30b   1/1     Running   0          81m
```

Test Connection

When the database status is Ready, check the connection:

- Get the database service

  ```
  kubectl get svc sidb-sample
  ```

- Option 1: Connect from within the pod

  ```
  kubectl exec -it sidb-sample-xh30b -- sqlplus / as sysdba


  Defaulted container "sidb-sample" out of: sidb-sample, init-wallet (init)

  SQL*Plus: Release 21.0.0.0.0 - Production on Sat Jan 24 00:37:34 2026
  Version 21.3.0.0.0

  Copyright (c) 1982, 2021, Oracle.  All rights reserved.


  Connected to:
  Oracle Database 21c Enterprise Edition Release 21.0.0.0.0 - Production
  Version 21.3.0.0.0

  SQL>
  ```

- Option 2: Connect to the PDB

  ```
  kubectl exec -it sidb-sample-xh30b -- sqlplus sys/<your-password>@localhost:1521/ORCLPDB1 as sysdba
  ```

- Option 3: From another pod in the cluster

  ```
  kubectl run sqlplus-client --image=container-registry.oracle.com/database/instantclient:21 -it --rm -- \
    sqlplus sys/<your-password>@sidb-sample:1521/ORCL1 as sysdba
  ```

Verify Database Configuration

- Check database mode

  ```
  SELECT name, open_mode, log_mode FROM v$database;

  NAME      OPEN_MODE            LOG_MODE
  --------- -------------------- ------------
  ORCL1     READ WRITE           ARCHIVELOG
  ```

- Check PDBs

```
SELECT name, open_mode FROM v$pdbs;
NAME
--------------------------------------------------------------------------------
OPEN_MODE
----------
PDB$SEED
READ ONLY

ORCLPDB1
READ WRITE
```

- Check archive log destination

```
SHOW PARAMETER db_recovery_file_dest;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_recovery_file_dest                string      /opt/oracle/oradata/fast_recov
                                                 ery_area
db_recovery_file_dest_size           big integer 50G
```

- Check SGA/PGA

```
SHOW PARAMETER sga_target;
SHOW PARAMETER pga_aggregate_target;

EXIT;
EOF
```

Exposing the Oracle Database via Compute Cloud@Customer or Private Cloud Appliance OKE Load Balancer Service

By default, Oracle Database Operator deploys a single instance Database with a Kubernetes `ClusterIP` service, which restricts database listener access to internal cluster traffic only. This means that applications, database clients, and administrators outside the OKE cluster on Compute Cloud@Customer or Private Cloud Appliance cannot reach the database endpoint directly. To enable external connectivity, a Kubernetes Service of type `LoadBalancer` must be created. This service instructs the OKE platform to provision a Load Balancer on Compute Cloud@Customer or Private Cloud Appliance, exposing the Oracle Net `listener (port 1521)` through a stable, externally accessible IP address that serves as the primary database entry point for all external consumers.

Step 1: Verify the Existing `ClusterIP` Service

First, confirm the current database service and note the port it exposes:

```
kubectl get svc sidb-sample
NAME          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
sidb-sample   ClusterIP   10.129.215.236   <none>        1521/TCP   18d
```

The service exposes port `1521 (Oracle Net listener)` but only internally within the cluster.

Step 2: Create the Load Balancer Service Manifest

Create a `.yaml` manifest file that defines a `LoadBalancer` service pointing to the SIDB pods. The selector must match the label used by Oracle Database Operator for the SIDB deployment (`app: sidb-sample`).

NOTE: In this deployment, the Load Balancer is provisioned using the Dynamic Shape type with a Medium – 400 Mbps bandwidth. Dynamic Shapes provide predefined bandwidth tiers that simplify Load Balancer sizing by offering fixed throughput levels suited to common workload profiles. The 400 Mbps tier provides sufficient bandwidth for typical Oracle Database workloads, including SQL Net traffic, bulk data operations, and application connectivity. Depending on the production environment requirements, this bandwidth can be adjusted to higher tiers. Dynamic Shapes on Compute Cloud@Customer and Private Cloud Appliance supports bandwidth options up to 8,000 Mbps for high-throughput database environments. The oci-load-balancer-shape annotation in the manifest controls this setting.

```
Cat sidb-lb-service.yaml

apiVersion: v1
kind: Service
metadata:
  name: sidb-sample-lb
  namespace: default
  annotations:
    oci.oraclecloud.com/load-balancer-type: “lb”
    service.beta.kubernetes.io/oci-load-balancer-shape: “400Mbps”
spec:
  type: LoadBalancer
  selector:
    app: sidb-sample
  ports:
    - name: listener
      protocol: TCP
      port: 1521
      targetPort: 1521
```

NOTE: Before testing external connectivity to the database through the Load Balancer, ensure that the security list associated with the `service-lb` subnet on Compute Cloud@Customer or Private Cloud Appliance includes an ingress rule allowing TCP traffic on port `1521`. By default, the `service-lb` subnet security list may only permit ICMP and SSH (port 22) traffic. Without an explicit ingress rule for port 1521, external database connection attempts will time out even though the Load Balancer is active and the external IP is reachable via ICMP. To add this rule, navigate to the Networking section in the Compute Cloud@Customer or Private Cloud Appliance management UI, select the Virtual Cloud Network (VCN) associated with the OKE cluster, open the security list for the `service-lb` subnet, and create a new ingress rule with Source `0.0.0.0/0` (or a more restrictive CIDR block matching your client network), IP Protocol TCP, and Destination Port Range `1521`.

Step 3: Apply the Load Balancer Service

```
kubectl apply -f sidb-lb-service.yaml
```

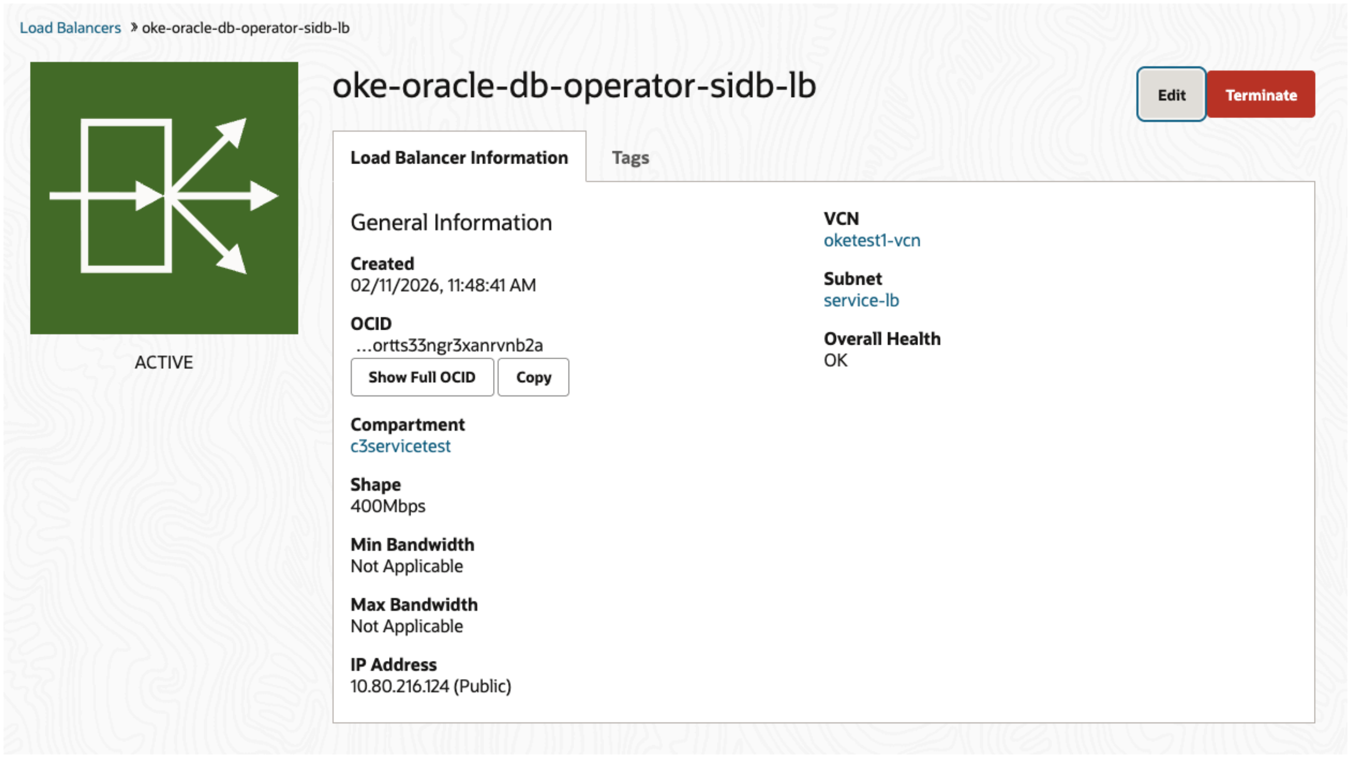
Step 4: Retrieve the External IP Address

After applying the manifest, the OKE platform provisions a Load Balancer on Compute Cloud@Customer or Private Cloud Appliance. The external IP may take a few seconds to be assigned. Monitor the service until the `EXTERNAL-IP` field is populated:

```
kubectl get svc sidb-sample-lb -w
NAME             TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
sidb-sample-lb   LoadBalancer   10.137.xx.xxx   <pending>     1521:31521/TCP   10s
sidb-sample-lb   LoadBalancer   10.137.xx.xxx   10.x.x.x     1521:31521/TCP   45s
```

After an external IP is assigned, the Load Balancer is ready.

On Compute Cloud@Customer or Private Cloud Appliance, the Load Balancer display name annotation may not be applied automatically during provisioning. If the Load Balancer is created with a default OCID-based name, navigate to the Load Balancer detail page in the Compute Cloud@Customer or Private Cloud Appliance management UI and click Edit to set a descriptive display name (for example, oke-oracle-db-operator-sidb-lb).


Load Balancers » oke-oracle-db-operator-sidb-lb
Edit
Terminate
Load Balancer Information
Tags
ACTIVE
Show Full OCID
Copy

oke-oracle-db-operator-sidb-lb

General Information

Created
02/11/2026, 11:48:41 AM

OCID
...ortts33ngr3xanrvnb2a

Compartment
c3servicetest

Shape
400Mbps

Min Bandwidth
Not Applicable

Max Bandwidth
Not Applicable

IP Address
10.80.216.124 (Public)

VCN
oketest1-vcn

Subnet
service-lb

Overall Health
OK

Figure 7. Compute Cloud@Customer or Private Cloud Appliance Load Balance configuration for Oracle Database on OKE

Step 5: Validate External Database Connectivity

Use the assigned external IP to connect from any client that has network access to the Compute Cloud@Customer or Private Cloud Appliance environment. For example, using SQL Plus from an external workstation:

Connect to the CDB:

```
sqlplus sys/<your_password>@<ip_address_of_the_loadbalance>:1521/ORCL1 as sysdba

SQL*Plus: Release 19.0.0.0.0 - Production on Wed Feb 11 12:34:16 2026
Version 19.8.0.0.0

Copyright (c) 1982, 2020, Oracle.  All rights reserved.


Connected to:
Oracle Database 21c Enterprise Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL>
```

Connect to the PDB:

```
sqlplus sys/<your_password>@<ip_address_of_the_loadbalance>:1521/ORCLPDB1 as sysdba

SQL*Plus: Release 19.0.0.0.0 - Production on Wed Feb 11 12:35:34 2026
Version 19.8.0.0.0

Copyright (c) 1982, 2020, Oracle.  All rights reserved.


Connected to:
Oracle Database 21c Enterprise Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL>
```

You can also use Oracle SQL Developer, JDBC thin connections, or any Oracle Net compatible client by using the following connection string:

```
<EXTERNAL-IP>:1521/ORCLPDB1
```

Conclusion

This solution paper demonstrated a complete end-to-end workflow for deploying and exposing an Oracle Database on Kubernetes using Oracle Database Operator cluster add-on for OKE on Compute Cloud@Customer or Private Cloud Appliance. The process covered enabling the Certificate Manager and Oracle Database Operator add-ons, configuring cluster access and RBAC permissions, provisioning a single instance Database through a declarative Kubernetes manifest, and exposing the database externally through an OKE Load Balancer service with Dynamic Shape bandwidth configuration.

The environment is now ready for application onboarding and Day-2 operations. Next steps include creating application schemas within the `ORCLPDB1` pluggable database, configuring backup policies to meet enterprise data protection requirements, setting up monitoring to ensure operational visibility, and establishing alerting for proactive performance management.

As workload demands evolve, the platform supports horizontal growth through additional pluggable databases (PDBs) for workload isolation and scalability. Load Balancer bandwidth can be scaled to higher Dynamic Shape tiers on Compute Cloud@Customer or Private Cloud Appliance to accommodate increased database traffic, and additional SIDB or multitenant deployments can be provisioned using the same operator-driven approach documented in this paper.

By combining the automation capabilities of Oracle Database Operator with the infrastructure services available on Compute Cloud@Customer or Private Cloud Appliance, organizations can deliver a fully operational, production-ready Oracle Database platform on Kubernetes, reducing deployment complexity, accelerating time to value, and maintaining enterprise-grade security and connectivity within customer-managed private cloud environments.

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