

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

Version 1.0

Copyright © 2026, Oracle and/or its affiliates

Public

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

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle software license and service agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This document is for informational purposes only and is intended solely to assist you in planning for the implementation and upgrade of the product features described. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described in this document remains at the sole discretion of Oracle. Due to the nature of the product architecture, it may not be possible to safely include all features described in this document without risking significant destabilization of the code.

This document may include some forward-looking content for illustrative purposes only. Some products and features discussed are indicative of the products and features of a prospective future launch in the United States only or elsewhere. Not all products and features discussed are currently offered for sale in the United States or elsewhere. Products and features of the actual offering may differ from those discussed in this document and may vary from country to country. Any timelines contained in this document are indicative only. Timelines and product features may depend on regulatory approvals or certification for individual products or features in the applicable country or region.

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 the 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

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.

- At least an 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

Follow the steps listed below 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.

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 on the three dots under the Actions column.
3. Select the Enable checkbox for the add-on you want to deploy and enable for this cluster.

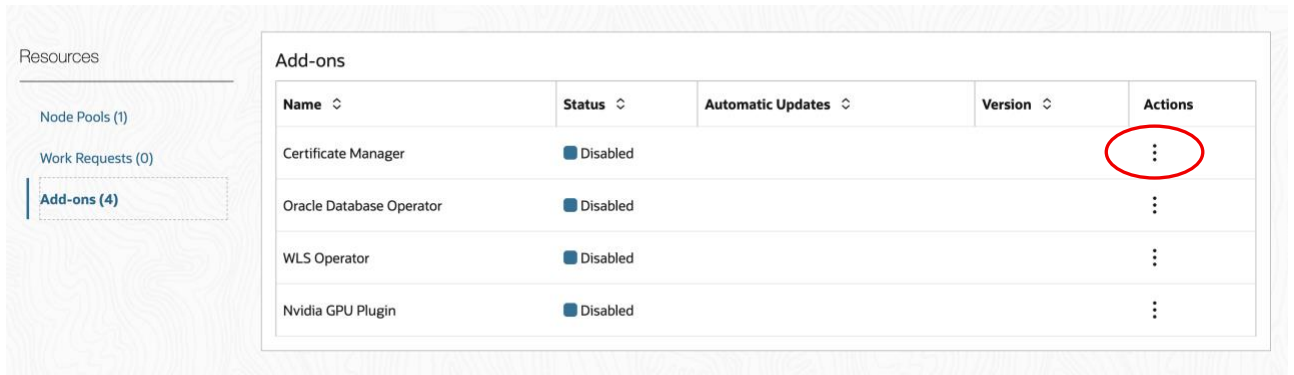


Figure 2. Activating Add-ons for OKE on Compute Cloud@Customer or Private Cloud Appliance.

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 the new version is compatible with both the cluster's Kubernetes version and the versions supported by OKE. We recommend 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. On this example, I am using 2 replicas only.
 - a. Select **Add configuration** to select a configuration option and specify a value.
 - b. Select **Add configuration** to set another configuration parameter.

Listed below are the screenshots with the Configuration part and the list of configurable parameters which can be set on Certificate Manager Add-on and Oracle DB Operator.

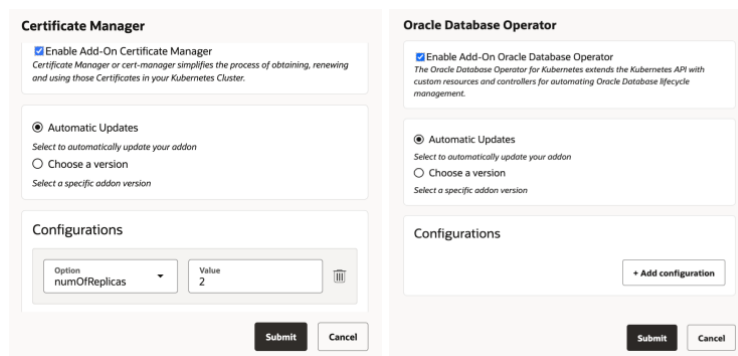


Figure 3. Configuration for Certificate Manager and Oracle Database Operator Add-ons for OKE on Compute Cloud@Customer or Private Cloud Appliance.

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

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	<code>numOfReplicas</code>	<p>Required</p> <p>The integer number of replicas of the add-on deployment.</p>
manager container resources	<code>manager.ContainerResources</code>	<p>Optional</p> <p>You can specify the resource quantities that the add-on containers request, and set resource usage limits that the add-on containers cannot exceed.</p> <p>Use JSON format in plain text or Base64 encoded.</p> <p>If you do not specify a request, the default request values are:</p> <p>Limits</p> <ul style="list-style-type: none"> cpu: 400m memory: 400Mi <p>Requests</p> <ul style="list-style-type: none"> cpu: 400m memory: 400Mi <p>Example:</p> <p>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.</p> <pre>{ "limits": { "cpu": "500m", "memory": "200Mi" }, "requests": { "cpu": "100m", "memory": "100Mi" } }</pre>

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 to deploy, the version 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	v1.7.2	⋮
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.cccccluster.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.

Listed below is a sample of a role bindings.yaml to enable proper operation of the Oracle Database Kubernetes Operator and how to apply the role_bindings.yaml file. 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
```

Since the Oracle Database Operator is managed by a deployment, you should restart the deployment, this way, new pod starts with the updated RBAC. Run the following command line listed below:

```
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-78bfb698f-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
```

(no output = good)

Verify permissions explicitly

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

Expected output: yes

The output above means:

- 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 the 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 you need to generate in Oracle Container Registry. Refer to <https://container-registry.oracle.com/>

3. Check if 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, it is recommended to 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
```

Once the appropriate nodes have been identified, apply a dedicated label (for example, node-role=database) to each node that will be used for Oracle Database deployments:

```
kubectl label node \
  oke-aaaaaaakdrwrhiamjydkm3gnrzgyzti-8cfct \
  oke-aaaaaaakdrwrhiamjydkm3gnrzgyzti-ncngb \
  oke-aaaaaaakdrwrhiamjydkm3gnrzgyzti-7mhsb \
  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 the 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 the 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 output below shown above represents the successful application of the Single Instance Database (SIDB) Kubernetes manifest and the subsequent lifecycle actions orchestrated by the 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. Once the manifest is applied, the operator schedules a new database pod (sidb-sample-xh30b) onto 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), ensuring 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. Once 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, confirming a successful SIDB deployment on OKE running on Oracle Compute Cloud@Customer or 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-aaaaaaaaakdrwrhiamjydkm3gnrzgzyti-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-amaaaaaakdrwhiamjydkm3gnrzgyzti-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:

```
kubect1 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

Once 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 Ready, the pod status will change to running status as listed below:

```
kubectl get pod sidb-sample-xh30b
```

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

Test Connection with the database once ready

- 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$pdb;
```

```
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_recovery_area
db_recovery_file_dest_size	big integer	50G

- Check SGA/PGA

```
SHOW PARAMETER sga_target;
```

```
SHOW PARAMETER pga_aggregate_target;
```

```
EXIT;
```

```
EOF
```

With the Single Instance Oracle Database successfully deployed and validated on OKE running on Oracle Compute Cloud@Customer or Private Cloud Appliance, the environment is now ready for application onboarding and day-2 operations. The next steps include creating application schemas and tables within the ORCLPDB1 pluggable database, configuring backup and recovery policies to meet enterprise data protection requirements, and setting up monitoring and alerting to ensure operational visibility and performance management.


As application demands grow, additional pluggable databases (PDBs) can also be created to support workload isolation and future scalability.

Together, these steps complete the transition from initial deployment to a fully operational, production-ready Oracle Database platform on Kubernetes.

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

Copyright © 2026, Oracle and/or its affiliates. This document is provided for information purposes only, and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document, and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Some regulatory certifications or registrations to products or services referenced on this website are held by Cerner Corporation. Cerner Corporation is a wholly-owned subsidiary of Oracle. Cerner Corporation is an ONC-certified health IT developer and a registered medical device manufacturer in the United States and other jurisdictions worldwide.

This document may include some forward-looking content for illustrative purposes only. Some products and features discussed are indicative of the products and features of a prospective future launch in the United States only or elsewhere. Not all products and features discussed are currently offered for sale in the United States or elsewhere. Products and features of the actual offering may differ from those discussed in this document and may vary from country to country. Any timelines contained in this document are indicative only. Timelines and product features may depend on regulatory approvals or certification for individual products or features in the applicable country or region.

Author: Anderson Souza