Oracle Cloud Native Environment

Rook Module for Release 1.8

F87570-02
March 2026

Oracle Cloud Native Environment Rook Module for Release 1.8,

F87570-02

Contents

Preface

> **Warning**
>
> Oracle Cloud Native Environment 1.8 is now in Sustaining Support. See Oracle Open Source Support Policies for more information. New security patches and bug fixes are no longer provided for this software.
>
> Migrate applications and data to Oracle Cloud Native Environment Release 2.<latest> as soon as possible.

This document contains information about setting up and using the Rook module to provide persistent storage to Kubernetes applications in Oracle Cloud Native Environment. The Rook module deploys Ceph into the Kubernetes cluster to provide the storage. This document describes how to deploy the Rook module and set up a Ceph cluster, and provides a basic example to test the storage.

Documentation License

Conventions

The following text conventions are used in this document:

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

Documentation Accessibility

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

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Diversity and Inclusion

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

1

Introduction to the Rook Module

> **Warning**
>
> Oracle Cloud Native Environment 1.8 is now in Sustaining Support. See Oracle Open Source Support Policies for more information. New security patches and bug fixes are no longer provided for this software.
>
> Migrate applications and data to Oracle Cloud Native Environment Release 2.<latest> as soon as possible.

Rook is a container storage platform built on Ceph. Rook is deployed as a Kubernetes operator inside a Kubernetes cluster and automates the work required to provision Ceph-backed persistent storage using the Kubernetes Container Storage Interface (CSI).

The Rook module is used to perform the installation of Rook into a Kubernetes cluster running on Oracle Cloud Native Environment.

Ceph lets you set up various types of storage for Kubernetes applications:

- Block storage using a CephBlockPool. This provides raw block device volumes to pods.
- Shared file system storage using a CephFilesystem (CephFS). This provides mounting of a shared POSIX (Portable OS Interface) compliant folder into one or more pods. This is similar to NFS (Network File System) shared storage, or CIFS (Common Internet File System) shared folders.
- Object storage using the CephObjectStore. This provides blob (Binary Large Object) storage to pods.

A CephFilesystem volume can be used with ReadWriteMany persistent volumes, whereas a CephBlockPool block volume can't as it's ReadWriteOnce.

A Ceph cluster and storage can be set up with a configuration file that contains the Ceph-related Kubernetes custom resource definitions (CRDs) when you deploy the Rook module, or after using the CRDs with the `kubectl` command.

For more information about Rook, see the upstream Rook documentation.

2

Installing the Rook Module

> **Warning**
>
> Oracle Cloud Native Environment 1.8 is now in Sustaining Support. See Oracle Open Source Support Policies for more information. New security patches and bug fixes are no longer provided for this software.
>
> Migrate applications and data to Oracle Cloud Native Environment Release 2.<latest> as soon as possible.

This chapter discusses how to install the Rook module to set up dynamically provisioned persistent storage for Kubernetes applications using Ceph on Oracle Cloud Native Environment.

Prerequisites

This section contains the prerequisites for installing the Rook module.

Setting up the Worker Nodes

The Rook module deploys Ceph as containers to the Kubernetes worker nodes. You need at least three worker nodes in the Kubernetes cluster.

In addition, at least one of these local storage options must be available on the Kubernetes worker nodes:

- Raw devices (no partitions or formatted file systems).
- Raw partitions (no formatted file system).
- LVM Logical Volumes (no formatted file system).
- Persistent Volumes available from a storage class in `block` mode.

> **Tip**
>
> Use the `lsblk -f` command to ensure no file system is on the device or partition. If the `FSTYPE` field is empty, no file system is on the disk and it can be used with Ceph.

Creating a Rook Configuration File

If you deploy the Rook module without a configuration file, the Rook operator pod (`rook-ceph-operator`) is created. You can then create a Ceph cluster and storage using the `kubectl` command. You can optionally provide a Rook configuration file to set up a Ceph cluster and storage, which is set up for you when you deploy the Rook module.

You can provide a Rook configuration file on the operator node in YAML format. The configuration file contains the information to configure one or more Ceph clusters and storage

types. You use Ceph-related Kubernetes CRDs in the configuration file to perform the setup. Include as many CRDs in the configuration file as you need to set up Ceph clusters, storage options, and storage providers. For example:

```
---
apiVersion: ceph.rook.io/v1
kind: CephCluster
metadata:
  name: rook-ceph
  namespace: rook
spec:
...
---
apiVersion: ceph.rook.io/v1
kind: CephBlockPool
metadata:
  name: replicapool
  namespace: rook
spec:
...
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: rook-ceph-block
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: rook.rbd.csi.ceph.com
parameters:
...
```

The Platform API Server uses the information contained in the configuration file when creating the Rook module. Rook performs all the set up and configuration for Ceph, using the information you provide in this file.

Use the upstream documentation to create CRDs. For information on the options available to use in the configuration file, see the upstream Rook documentation for Ceph CRDs.

> **Important**
>
> The example CRDs in this section include CRDs to set up a basic Ceph cluster, storage types, and storage class providers. These are examples only and aren't recommended for a production environment.

CephCluster CRD

The CephCluster CRD is used to create a Ceph cluster. The following example configuration uses a Kubernetes cluster with 3 worker nodes that have a RAW disk attached to each node as `sdb`. This example uses a Ceph cluster name of `rook-ceph` in the `rook` namespace. Note that the Ceph image is pulled from the Oracle Container Registry.

```
---
apiVersion: ceph.rook.io/v1
kind: CephCluster
```

```
metadata:
  name: rook-ceph
  namespace: rook
spec:
  cephVersion:
    image: container-registry.oracle.com/olcne/ceph:v17.2.5
    imagePullPolicy: Always
  dataDirHostPath: /var/lib/rook
  mon:
    count: 3
    allowMultiplePerNode: false
  dashboard:
    enabled: false
  storage:
    useAllNodes: true
    useAllDevices: false
    deviceFilter: sdb
```

CephBlockPool CRD

Use a CephBlockPool CRD to create the Ceph storage pool. This example sets up a replica set of 3 in the CephBlockPool named `replicapool` in the `rook` namespace.

```
---
apiVersion: ceph.rook.io/v1
kind: CephBlockPool
metadata:
  name: replicapool
  namespace: rook
spec:
  failureDomain: host
  replicated:
    size: 3
    requireSafeReplicaSize: true
```

StorageClass CRD for CephBlockPool

To allow pods to access the Ceph block storage, you need to create a StorageClass. An example CRD for this follows:

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: rook-ceph-block
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: rook.rbd.csi.ceph.com
parameters:
  clusterID: rook
  pool: replicapool
  imageFormat: "2"
  imageFeatures: layering
  csi.storage.k8s.io/provisioner-secret-name: rook-csi-rbd-provisioner
  csi.storage.k8s.io/provisioner-secret-namespace: rook
```

```
  csi.storage.k8s.io/controller-expand-secret-name: rook-csi-rbd-provisioner
  csi.storage.k8s.io/controller-expand-secret-namespace: rook
  csi.storage.k8s.io/node-stage-secret-name: rook-csi-rbd-node
  csi.storage.k8s.io/node-stage-secret-namespace: rook
  csi.storage.k8s.io/fstype: ext4
allowVolumeExpansion: true
reclaimPolicy: Delete
```

CephFilesystem CRD

You might also want to set up a CephFilesystem. You do this by including the CephFilesystem CRD information in the configuration file. This example creates a CephFilesystem named `myfs` in the `rook` namespace, with a replica count of 3.

```
---
apiVersion: ceph.rook.io/v1
kind: CephFilesystem
metadata:
  name: myfs
  namespace: rook
spec:
  metadataPool:
    replicated:
      size: 3
  dataPools:
    - name: replicated
      replicated:
        size: 3
  preserveFilesystemOnDelete: true
  metadataServer:
    activeCount: 1
    activeStandby: true
```

StorageClass CRD for CephFilesystem

To allow pods to access the CephFilesystem storage, you need to create a StorageClass. An example CRD for this follows:

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: rook-cephfs
provisioner: rook.cephfs.csi.ceph.com
parameters:
  clusterID: rook
  fsName: myfs
  pool: myfs-replicated
  csi.storage.k8s.io/provisioner-secret-name: rook-csi-cephfs-provisioner
  csi.storage.k8s.io/provisioner-secret-namespace: rook
  csi.storage.k8s.io/controller-expand-secret-name: rook-csi-cephfs-
provisioner
  csi.storage.k8s.io/controller-expand-secret-namespace: rook
  csi.storage.k8s.io/node-stage-secret-name: rook-csi-cephfs-node
```

```
  csi.storage.k8s.io/node-stage-secret-namespace: rook
reclaimPolicy: Delete
```

CephObjectStore CRD

You might also want to set up a CephObjectStore. You do this by including the CephObjectStore CRD information in the configuration file. For example:

```
---
apiVersion: ceph.rook.io/v1
kind: CephObjectStore
metadata:
  name: my-store
  namespace: rook
spec:
  metadataPool:
    failureDomain: host
    replicated:
      size: 3
  dataPool:
    failureDomain: host
    erasureCoded:
      dataChunks: 2
      codingChunks: 1
  preservePoolsOnDelete: true
  gateway:
    sslCertificateRef:
    port: 80
    instances: 1
  healthCheck:
    startupProbe:
     disabled: false
    readinessProbe:
     disabled: false
     periodSeconds: 5
     failureThreshold: 2
```

StorageClass (bucket) CRD for CephObjectStore

To allow pods to access the CephObjectStorage storage, you need to create a StorageClass, which creates a bucket. An example CRD for this follows:

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
   name: rook-ceph-bucket
provisioner: rook-ceph.ceph.rook.io/bucket
reclaimPolicy: Delete
parameters:
  objectStoreName: my-store
  objectStoreNamespace: rook
```

Deploying the Rook Module

You can deploy all the modules required to set up Ceph storage for a Kubernetes cluster using a single `olcnectl module create` command. This method might be useful to deploy the Rook module at the same time as deploying a Kubernetes cluster.

If you have an existing deployment of the Kubernetes module, you can specify that instance when deploying the Rook module.

This section guides you through installing each component required to deploy the Rook module.

For the full list of the Platform CLI command options available when creating modules, see the `olcnectl module create` command in Platform Command-Line Interface.

To deploy the Rook module:

1. If you don't already have an environment set up, create one into which the modules can be deployed. For information on setting up an environment, see Installation. The name of the environment in this example is `myenvironment`.
2. If you don't already have a Kubernetes module set up or deployed, set one up. For information on adding a Kubernetes module to an environment, see Kubernetes Module. The name of the Kubernetes module in this example is `mycluster`.
3. Create a Rook module and associate it with the Kubernetes module named `mycluster` using the `--rook-kubernetes-module` option. In this example, the Rook module is named `myrook`.

   ```
   olcnectl module create \
   --environment-name myenvironment \
   --module rook \
   --name myrook \
   --rook-kubernetes-module mycluster \
   --rook-config rook-config.yaml
   ```

   The `--module` option sets the module type to create, which is `rook`. You define the name of the Rook module using the `--name` option, which in this case is `myrook`.

   The `--rook-kubernetes-module` option sets the name of the Kubernetes module.

   The `--rook-config` option sets the location of a YAML file that contains the configuration information for the Rook module. This is optional.

   If you don't include all the required options when adding the module, you're prompted to provide them.
4. Use the `olcnectl module install` command to install the Rook module. For example:

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

You can optionally use the `--log-level` option to set the level of logging displayed in the command output. By default, error messages are displayed. For example, you can set the logging level to show all messages when you include:

```
--log-level debug
```

The log messages are also saved as an operation log. You can view operation logs as commands are running, or when they've completed. For more information using operation logs, see Platform Command-Line Interface.

The Rook module is deployed into the Kubernetes cluster.

Verifying the Rook Module Deployment

You can verify the Rook module is deployed using the `olcnectl module instances` command on the operator node. For example:

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

The output looks similar to:

```
INSTANCE                  MODULE        STATE
mycluster                 kubernetes    installed
myrook                    rook          installed
...
```

Note the entry for `rook` in the `MODULE` column is in the `installed` state.

In addition, you can use the `olcnectl module report` command to review information about the module. For example, use the following command to review the Rook module named `myrook` in `myenvironment`:

```
olcnectl module report \
--environment-name myenvironment \
--name myrook \
--children
```

For more information on the syntax for the `olcnectl module report` command, see Platform Command-Line Interface.

On a control plane node, verify the `rook-ceph` deployments are running in the `rook` namespace. Confirm that the Ceph operator pod (`rook-ceph-operator`) deployment is running.

```
kubectl get deployments --namespace rook
```

The output looks similar to:

```
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
rook-ceph-operator            1/1     1            1           163m
```

If you used a configuration file to deploy extra Ceph objects, such as a Ceph cluster, storage, and storage class provisioners, you might have more deployments running. For example:

```
NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
csi-cephfsplugin-provisioner      2/2     2            2           159m
csi-rbdplugin-provisioner         2/2     2            2           159m
rook-ceph-operator                1/1     1            1           163m
rook-ceph-mgr-a                   1/1     1            1           163m
rook-ceph-mon-b                   1/1     1            1           163m
...
```

You can check the logs for the Ceph operator pod to ensure no errors occurred during the deployment:

```
kubectl logs --namespace rook rook-ceph-operator-...
```

On a control plane node, you can also verify any StorageClasses for the Ceph provisioner are created using the `kubectl get sc` command. These are only created if you used a configuration file to create them when deploying the module. For example:

```
kubectl get sc
```

The output looks similar to:

```
NAME                        PROVISIONER                  RECLAIMPOLICY
VOLUMEBINDINGMODE   ...
rook-ceph-block (default)   rook.rbd.csi.ceph.com        Delete
Immediate           ...
rook-cephfs                 rook.cephfs.csi.ceph.com     Delete
Immediate           ...
```

In this case, two StorageClasses exist, one named `rook-ceph-block`, which is the default provider. This is the provider for Ceph block storage. The other StorageClass is named `rook-cephfs`, which is the provider for CephFilesystem (CephFS).

You can get more details about a StorageClass using the `kubectl describe sc` command. For example:

```
kubectl describe sc rook-ceph-block
```

The output looks similar to:

```
Name:                  rook-ceph-block
IsDefaultClass:        Yes
Annotations:           storageclass.kubernetes.io/is-default-class=true
Provisioner:           rook.rbd.csi.ceph.com
Parameters:            clusterID=rook,csi.storage.k8s.io/controller-expand-
secret-name=rook- ...
AllowVolumeExpansion:  True
MountOptions:          <none>
ReclaimPolicy:         Delete
```

```
VolumeBindingMode:     Immediate
Events:                <none>
```

3
Using Ceph Storage

> **Warning**
>
> Oracle Cloud Native Environment 1.8 is now in Sustaining Support. See Oracle Open Source Support Policies for more information. New security patches and bug fixes are no longer provided for this software.
>
> Migrate applications and data to Oracle Cloud Native Environment Release 2.<latest> as soon as possible.

This chapter discusses how to use the Rook module to set up dynamically provisioned persistent storage using Ceph for Kubernetes applications in Oracle Cloud Native Environment.

Creating a CephCluster

This section contains a basic example on how to create a CephCluster.

If you don't create a Ceph cluster using a Rook configuration file, you can create one or more clusters after the Rook module is deployed. You do this using the `kubectl` command to deploy a CephCluster CRD.

For example, if you use the following CephCluster CRD in a YAML file:

```
---
apiVersion: ceph.rook.io/v1
kind: CephCluster
metadata:
  name: rook-ceph
  namespace: rook
spec:
  cephVersion:
    image: container-registry.oracle.com/olcne/ceph:v17.2.5
    imagePullPolicy: Always
  dataDirHostPath: /var/lib/rook
  mon:
    count: 3
    allowMultiplePerNode: false
  dashboard:
    enabled: false
  storage:
    useAllNodes: true
    useAllDevices: false
    deviceFilter: sdb
```

On a control plane node, use the `kubectl apply` command to create the CephCluster with the file:

```
kubectl apply -f filename.yaml
```

The CephCluster is created. You can verify the CephCluster is created using:

```
kubectl get cephcluster --namespace rook
```

The output looks similar to:

```
NAME        DATADIRHOSTPATH   MONCOUNT   AGE   PHASE
MESSAGE                        HEALTH ...
rook-ceph   /var/lib/rook     3          4m29s Ready   Cluster created
successfully   HEALTH ...
```

Creating CephBlockPool Storage

This section contains a basic test to verify you can create and use CephBlockPool storage to provide persistent block storage to applications running on Kubernetes.

If you don't create a CephBlockPool using a Rook configuration file, you can create one or more after the Rook module is deployed. You do this using the `kubectl` command to deploy a CephBlockPool CRD.

For example, if you use the following CephBlockPool CRD in a YAML file:

```
---
apiVersion: ceph.rook.io/v1
kind: CephBlockPool
metadata:
  name: replicapool
  namespace: rook
spec:
  failureDomain: host
  replicated:
    size: 3
    requireSafeReplicaSize: true
```

On a control plane node, use the `kubectl apply` command to create the CephBlockPool with the file:

```
kubectl apply -f filename.yaml
```

The CephBlockPool is created. You can verify the CephBlockPool is created using:

```
kubectl get cephblockpool --namespace rook
```

The output looks similar to:

```
NAME          PHASE
replicapool   Ready
```

If you don't create a StorageClass for the CephBlockPool using a Rook configuration file, you can create one after the Rook module is deployed.

For example, if you use the following StorageClass CRD in a YAML file:

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: rook-ceph-block
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: rook.rbd.csi.ceph.com
parameters:
  clusterID: rook
  pool: replicapool
  imageFormat: "2"
  imageFeatures: layering
  csi.storage.k8s.io/provisioner-secret-name: rook-csi-rbd-provisioner
  csi.storage.k8s.io/provisioner-secret-namespace: rook
  csi.storage.k8s.io/controller-expand-secret-name: rook-csi-rbd-provisioner
  csi.storage.k8s.io/controller-expand-secret-namespace: rook
  csi.storage.k8s.io/node-stage-secret-name: rook-csi-rbd-node
  csi.storage.k8s.io/node-stage-secret-namespace: rook
  csi.storage.k8s.io/fstype: ext4
allowVolumeExpansion: true
reclaimPolicy: Delete
```

On a control plane node, use the `kubectl apply` command to create the StorageClass with the file:

```
kubectl apply -f filename.yaml
```

The StorageClass is created. You can verify the StorageClass is created using:

```
kubectl get sc
```

The output looks similar to:

```
NAME                        PROVISIONER             RECLAIMPOLICY
VOLUMEBINDINGMODE   ...
rook-ceph-block (default)   rook.rbd.csi.ceph.com   Delete
Immediate           ...
```

To create a test application to use the CephBlockPool storage:

1. Create a Kubernetes PersistentVolumeClaim file. On a control plane node, create a file named `pvc-cephblock.yaml`. Copy the following into the file.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: myrook-block-pvc
spec:
 accessModes:
  - ReadWriteOnce
 resources:
  requests:
    storage: 1Gi
```

2. Create the Kubernetes PersistentVolumeClaim.

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

3. You can see the PersistentVolumeClaim is created using the `kubectl get pvc` command:

```
kubectl get pvc
```

The output looks similar to:

```
NAME               STATUS   VOLUME        CAPACITY   ACCESS MODES
STORAGECLASS      AGE
myrook-block-pvc   Bound    pvc-72da7...  1Gi        RWO            rook-
ceph-block   1h
```

You can get more details about the PersistentVolumeClaim using the `kubectl describe pvc` command. For example:

```
kubectl describe pvc myrook-block-pvc
```

The output looks similar to:

```
Name:          myrook-block-pvc
Namespace:     default
StorageClass:  rook-ceph-block
Status:        Bound
Volume:        pvc-72da7cbf-9e4e-49c9-92cf-65047e3780dd
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner:
rook.rbd.csi.ceph.com
               volume.kubernetes.io/storage-provisioner:
rook.rbd.csi.ceph.com
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
```

```
Used By:       <none>
Events:
...
```

4. Create a Kubernetes application that uses the PersistentVolumeClaim. Create a file named `nginx-block.yaml` and copy the following into the file.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    run: mynginx
  name: mynginx-block
spec:
  replicas: 1
  selector:
    matchLabels:
      run: mynginx
  template:
    metadata:
      labels:
        run: mynginx
    spec:
      containers:
      - image: container-registry.oracle.com/olcne/nginx:1.17.7
        name: mynginx
        ports:
        - containerPort: 80
        volumeMounts:
        - name: nginx-pvc
          mountPath: /usr/share/nginx/html
      volumes:
      - name: nginx-pvc
        persistentVolumeClaim:
          claimName: myrook-block-pvc
```

5. Start the application:

```
kubectl apply -f nginx-block.yaml
```

6. You can see the application is running using the `kubectl get deployment` command:

```
kubectl get deployment
```

The output looks similar to:

```
NAME            READY   UP-TO-DATE   AVAILABLE   AGE
mynginx-block   1/1     1            1           65s
```

7. You can see the application is using the PersistentVolumeClaim to provide persistent storage on CephBlockPool storage using the `kubectl describe deployment` command:

```
kubectl describe deployment mynginx-block
```

The output looks similar to:

```
...
Pod Template:
  Labels:  run=mynginx
  Containers:
   mynginx:
    Image:        container-registry.oracle.com/olcne/nginx:1.17.7
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:
      /usr/share/nginx/html from nginx-pvc (rw)
  Volumes:
   nginx-pvc:
    Type:       PersistentVolumeClaim (a reference to a
PersistentVolumeClaim in the same namespace)
    ClaimName:  myrook-block-pvc
    ReadOnly:   false
...
```

8. You can delete the test application using:

```
kubectl delete deployment mynginx-block
```

9. You can delete the PersistentVolumeClaim using:

```
kubectl delete pvc myrook-block-pvc
```

Creating CephFilesystem Storage

This section contains a basic test to verify you can use CephFilesystem storage to provide persistent storage to applications running on Kubernetes.

If you don't create a CephFilesystem using a Rook configuration file, you can create one or more after the Rook module is deployed. You do this using the `kubectl` command to deploy a CephFilesystem CRD.

For example, if you use the following CephFilesystem CRD in a YAML file:

```
---
apiVersion: ceph.rook.io/v1
kind: CephFilesystem
metadata:
  name: myfs
  namespace: rook
spec:
  metadataPool:
    replicated:
      size: 3
  dataPools:
    - name: replicated
      replicated:
        size: 3
  preserveFilesystemOnDelete: true
```

```
  metadataServer:
    activeCount: 1
    activeStandby: true
```

On a control plane node, use the `kubectl apply` command to create the CephFilesystem with the file:

```
kubectl apply -f filename.yaml
```

The CephFilesystem is created. You can verify the CephFilesystem is created using:

```
kubectl get cephfilesystem --namespace rook
```

The output looks similar to:

```
NAME   ACTIVEMDS   AGE   PHASE
myfs   1           18s   Ready
```

If you don't create a StorageClass for the CephFilesystem using a Rook configuration file, you can create one after the Rook module is deployed.

For example, if you use the following StorageClass CRD in a YAML file:

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: rook-cephfs
provisioner: rook.cephfs.csi.ceph.com
parameters:
  clusterID: rook
  fsName: myfs
  pool: myfs-replicated
  csi.storage.k8s.io/provisioner-secret-name: rook-csi-cephfs-provisioner
  csi.storage.k8s.io/provisioner-secret-namespace: rook
  csi.storage.k8s.io/controller-expand-secret-name: rook-csi-cephfs-
provisioner
  csi.storage.k8s.io/controller-expand-secret-namespace: rook
  csi.storage.k8s.io/node-stage-secret-name: rook-csi-cephfs-node
  csi.storage.k8s.io/node-stage-secret-namespace: rook
reclaimPolicy: Delete
```

On a control plane node, use the `kubectl apply` command to create the StorageClass with the file:

```
kubectl apply -f filename.yaml
```

The StorageClass is created. You can verify the StorageClass is created using:

```
kubectl get sc
```

The output looks similar to:

```
NAME                        PROVISIONER                RECLAIMPOLICY
VOLUMEBINDINGMODE   ...
rook-ceph-block (default)   rook.rbd.csi.ceph.com      Delete
Immediate           ...
rook-cephfs                 rook.cephfs.csi.ceph.com   Delete
Immediate           ...
```

To create a test application to use the CephFilesystem storage:

1. Create a Kubernetes PersistentVolumeClaim file. On a control plane node, create a file named `pvc-cephfs.yaml`. Copy the following into the file.

   ```
   apiVersion: v1
   kind: PersistentVolumeClaim
   metadata:
    name: myrook-pvc-fs
   spec:
    accessModes:
     - ReadWriteMany
    resources:
     requests:
       storage: 1Gi
    storageClassName: rook-cephfs
   ```

2. Create the Kubernetes PersistentVolumeClaim.

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

3. You can see the PersistentVolumeClaim is created using the `kubectl get pvc` command:

   ```
   kubectl get pvc
   ```

   The output looks similar to:

   ```
   NAME           STATUS    VOLUME       CAPACITY   ACCESS MODES
   STORAGECLASS     AGE
   myrook-pvc-fs  Bound     pvc-...      1Gi        RWX            rook-
   cephfs      18s
   ```

   You can get more details about the PersistentVolumeClaim using the `kubectl describe pvc` command. For example:

   ```
   kubectl describe pvc myrook-pvc-fs
   ```

   The output looks similar to:

   ```
   Name:          myrook-pvc-fs
   Namespace:     default
   StorageClass:  rook-cephfs
   Status:        Bound
   ```

```
Volume:        pvc-b98f9230-03d9-401d-9e19-81491eb785f9
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner:
rook.cephfs.csi.ceph.com
               volume.kubernetes.io/storage-provisioner:
rook.cephfs.csi.ceph.com
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWX
VolumeMode:    Filesystem
Used By:       <none>
Events:
  Type    Reason                Age
From                                                    ...
  ----    ------                ----
----                                                    ...
  Normal  ExternalProvisioning  106s  persistentvolume-
controller                          ...
  Normal  Provisioning          106s  rook.cephfs.csi.ceph.com_csi-
cephfsplugin-provisio...
  Normal  ProvisioningSucceeded 106s  rook.cephfs.csi.ceph.com_csi-
cephfsplugin-provisio...
```

4. Create a Kubernetes application that uses the PersistentVolumeClaim. Create a file named `nginx-cephfs.yaml` and copy the following into the file.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    run: mynginx
  name: mynginx-cephfs
spec:
  replicas: 1
  selector:
    matchLabels:
      run: mynginx
  template:
    metadata:
      labels:
        run: mynginx
    spec:
      containers:
      - image: container-registry.oracle.com/olcne/nginx:1.17.7
        name: mynginx
        ports:
        - containerPort: 80
        volumeMounts:
        - name: nginx-pvc
          mountPath: /usr/share/nginx/html
      volumes:
      - name: nginx-pvc
```

```
        persistentVolumeClaim:
          claimName: myrook-pvc-fs
```

5. Start the application:

   ```
   kubectl apply -f nginx-cephfs.yaml
   ```

6. You can see the application is running using the `kubectl get deployment` command:

   ```
   kubectl get deployment
   ```

   The output looks similar to:

   ```
   NAME             READY   UP-TO-DATE   AVAILABLE   AGE
   mynginx-cephfs   1/1     1            1           16s
   ```

7. You can see the application is using the PersistentVolumeClaim to provide persistent storage on CephFilesystem using the `kubectl describe deployment` command:

   ```
   kubectl describe deployment mynginx-cephfs
   ```

   The output looks similar to:

   ```
   ...
   Pod Template:
     Labels:  run=mynginx
     Containers:
      mynginx:
       Image:        container-registry.oracle.com/olcne/nginx:1.17.7
       Port:         80/TCP
       Host Port:    0/TCP
       Environment:  <none>
       Mounts:
         /usr/share/nginx/html from nginx-pvc (rw)
     Volumes:
      nginx-pvc:
       Type:       PersistentVolumeClaim (a reference to a
   PersistentVolumeClaim in the ...
       ClaimName:  myrook-pvc-fs
       ReadOnly:   false
   ...
   ```

8. You can delete the test application using:

   ```
   kubectl delete deployment mynginx-cephfs
   ```

9. You can delete the PersistentVolumeClaim using:

   ```
   kubectl delete pvc myrook-pvc-fs
   ```

Creating CephObjectStore Storage

This section contains a basic test to verify you can use CephObjectStore storage to provide object storage to applications running on Kubernetes.

The example in this section is based on the upstream Rook documentation example to create a CephObjectStore, and then test it. The content here is changed to use the `rook` Kubernetes namespace (the default namespace for Rook in Oracle Cloud Native Environment), but is otherwise the same. To create an application to test that you can `put` or `get` an object from the CephObjectStore, see the upstream documentation. The information here shows you how to set up the CephObjectStore and create an ObjectBucketClaim, but not how to create an application to test it.

If you don't create a CephObjectStore using a Rook configuration file, you can create one or more after the Rook module is deployed. You do this using the `kubectl` command to deploy a CephObjectStore CRD.

For example, if you use the following CephObjectStore CRD in a YAML file:

```
---
apiVersion: ceph.rook.io/v1
kind: CephObjectStore
metadata:
  name: my-store
  namespace: rook
spec:
  metadataPool:
    failureDomain: host
    replicated:
      size: 3
  dataPool:
    failureDomain: host
    erasureCoded:
      dataChunks: 2
      codingChunks: 1
  preservePoolsOnDelete: true
  gateway:
    sslCertificateRef:
    port: 80
    instances: 1
  healthCheck:
    startupProbe:
     disabled: false
    readinessProbe:
     disabled: false
     periodSeconds: 5
     failureThreshold: 2
```

On a control plane node, use the `kubectl apply` command to create the CephObjectStore with the file:

```
kubectl apply -f filename.yaml
```

The CephObjectStore is created. You can verify the CephObjectStore is created using:

```
kubectl get cephobjectstore --namespace rook
```

The output looks similar to:

```
NAME       PHASE
my-store   Ready
```

You can confirm the object store is configured by showing the pod is started:

```
kubectl get pod -l app=rook-ceph-rgw --namespace rook
```

The output looks similar to:

```
NAME                             READY   STATUS    RESTARTS   AGE
rook-ceph-rgw-my-store-a-...     1/1     Running   0          3m35s
```

If you don't create a StorageClass (bucket) for the CephObjectStore using a Rook configuration file, you can create one after the Rook module is deployed.

For example, if you use the following StorageClass CRD in a YAML file:

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
   name: rook-ceph-bucket
provisioner: rook-ceph.ceph.rook.io/bucket
reclaimPolicy: Delete
parameters:
  objectStoreName: my-store
  objectStoreNamespace: rook
```

On a control plane node, use the `kubectl apply` command to create the StorageClass with the file:

```
kubectl apply -f filename.yaml
```

The StorageClass is created. You can verify the StorageClass is created using:

```
kubectl get sc
```

The output looks similar to:

```
NAME                        PROVISIONER                     RECLAIMPOLICY
VOLUMEBIND ...
rook-ceph-block (default)   rook.rbd.csi.ceph.com           Delete
Immediate  ...
rook-ceph-bucket            rook-ceph.ceph.rook.io/bucket   Delete
Immediate  ...
```

```
rook-cephfs                rook.cephfs.csi.ceph.com        Delete
Immediate  ...
```

To create an ObjectBucketClaim to access the CephObjectStore storage:

1. Create a Kubernetes ObjectBucketClaim file. On a control plane node, create a file named `obc.yaml`. Copy the following into the file.

   ```
   apiVersion: objectbucket.io/v1alpha1
   kind: ObjectBucketClaim
   metadata:
     name: ceph-bucket
   spec:
     generateBucketName: ceph-bkt
     storageClassName: rook-ceph-bucket
   ```

2. Create the Kubernetes ObjectBucketClaim.

   ```
   kubectl apply -f obc.yaml
   ```

3. You can see the ObjectBucketClaim is created using the `kubectl get obc` command:

   ```
   kubectl get obc
   ```

   The output looks similar to:

   ```
   NAME          AGE
   ceph-bucket   31s
   ```

   You can get more details about the ObjectBucketClaim using the `kubectl describe obc` command. For example:

   ```
   kubectl describe obc ceph-bucket
   ```

   The output looks similar to:

   ```
   Name:         ceph-bucket
   Namespace:    default
   Labels:       <none>
   Annotations:  <none>
   API Version:  objectbucket.io/v1alpha1
   Kind:         ObjectBucketClaim
   Metadata:
     Creation Timestamp:  <date>
     Generation:          1
     Managed Fields:
       API Version:  objectbucket.io/v1alpha1
       Fields Type:  FieldsV1
       fieldsV1:
         f:metadata:
           f:annotations:
             .:
             f:kubectl.kubernetes.io/last-applied-configuration:
   ```

```
      f:spec:
        .:
        f:generateBucketName:
        f:storageClassName:
    Manager:         kubectl-client-side-apply
    Operation:       Update
    Time:            <date>
  Resource Version:  354339
  UID:               c53e5eb7-f460-435a-b31d-2eaab1bcddd3
Spec:
  Generate Bucket Name:  ceph-bkt
  Storage Class Name:    rook-ceph-bucket
Events:                  <none>
```

4. To create a Kubernetes application that uses the ObjectBucketClaim, follow the rest of the example in the upstream Rook documentation.
5. You can delete the ObjectBucketClaim using:

   ```
   kubectl delete obc ceph-bucket
   ```

4 Removing the Rook Module

> **⚠ Warning**
>
> Oracle Cloud Native Environment 1.8 is now in Sustaining Support. See Oracle Open Source Support Policies for more information. New security patches and bug fixes are no longer provided for this software.
>
> Migrate applications and data to Oracle Cloud Native Environment Release 2.<latest> as soon as possible.

You can remove a deployment of the Rook module and leave the Kubernetes cluster in place. To do this, you remove the Rook module from the environment.

If you used a Rook configuration file to deploy the Rook module, any Ceph configuration in that file is also removed, including Ceph clusters, storage, and StorageClasses. If you manually set up Ceph using the `kubectl` command with CRD files, that Ceph setup remains in place. Remove these manually before removing the Rook module.

Use the `olcnectl module uninstall` command to remove the Rook module. For example, to uninstall the Rook module named `myrook` in the environment named `myenvironment`:

```
olcnectl module uninstall \
--environment-name myenvironment \
--name myrook
```

The Rook module is removed from the environment.