SUSE Enterprise Storage 7.1

Deploying and Administering SUSE Enterprise Storage with Rook

Deploying and Administering SUSE Enterprise Storage with Rook

SUSE Enterprise Storage 7.1

by Tomáš Bažant, Alexandra Settle, and Liam Proven

Deployment of containerized Ceph clusters on SUSE CaaS Platform is released under limited availability for the SUSE Enterprise Storage 7.1 release.

For more details about the SUSE CaaS Platform product, see https://documentation.suse.com/suse-caasp/4.5/.

Publication Date: 07 Nov 2024

https://documentation.suse.com

Contents

About this guide

Part of the SUSE Enterprise Storage family is the Rook deployment tool, which runs on SUSE CaaS Platform. Rook allows you to deploy and run Ceph on top of Kubernetes, in order to provide container workloads with all their storage needs.

Deployment using Rook is currently in *limited availability*, meaning that it is only available to nominated and approved customers. For information on how to get so nominated, contact your SUSE sales team.

Rook is a so-called storage operator: it automates many steps that you need to do manually in a "traditional" setup with cephadm. This guide explains how to install Rook after you installed SUSE CaaS Platform, and how to administer it.

SUSE Enterprise Storage 7.1 is an extension to SUSE Linux Enterprise Server 15 SP3. It combines the capabilities of the Ceph (http://ceph.com/) storage project with the enterprise engineering and support of SUSE. SUSE Enterprise Storage 7.1 provides IT organizations with the ability to deploy a distributed storage architecture that can support a number of use cases using commodity hardware platforms.

1 Available documentation

Note: Online documentation and latest updates

Documentation for our products is available at https://documentation.suse.com, where you can also find the latest updates, and browse or download the documentation in various formats. The latest documentation updates can be found in the English language version.

In addition, the product documentation is available in your installed system under `/usr/share/doc/manual`. It is included in an RPM package named `ses-manual_LANG_CODE`. Install it if it is not already on your system, for example:

```
# zypper install ses-manual_en
```

The following documentation is available for this product:

Deployment Guide (https://documentation.suse.com/ses/html/ses-all/book-storage-deployment.html)

This guide focuses on deploying a basic Ceph cluster, and how to deploy additional services. It also cover the steps for upgrading to SUSE Enterprise Storage 7.1 from the previous product version.

Administration and Operations Guide (https://documentation.suse.com/ses/html/ses-all/book-storage-admin.html)

This guide focuses on routine tasks that you as an administrator need to take care of after the basic Ceph cluster has been deployed (day 2 operations). It also describes all the supported ways to access data stored in a Ceph cluster.

Security Hardening Guide (https://documentation.suse.com/ses/html/ses-all/book-storage-security.html)

This guide focuses on how to ensure your cluster is secure.

Troubleshooting Guide (https://documentation.suse.com/ses/html/ses-all/book-storage-troubleshooting.html)

This guide takes you through various common problems when running SUSE Enterprise Storage 7.1 and other related issues to relevant components such as Ceph or Object Gateway.

SUSE Enterprise Storage for Windows Guide (https://documentation.suse.com/ses/html/ses-all/book-storage-windows.html)

This guide describes the integration, installation, and configuration of Microsoft Windows environments and SUSE Enterprise Storage using the Windows Driver.

2 Improving the documentation

Your feedback and contributions to this documentation are welcome. The following channels for giving feedback are available:

Service requests and support

For services and support options available for your product, see http://www.suse.com/support/.

To open a service request, you need a SUSE subscription registered at SUSE Customer Center. Go to https://scc.suse.com/support/requests, log in, and click *Create New*.

Bug reports

Report issues with the documentation at https://bugzilla.suse.com/. A Bugzilla account is required.

To simplify this process, you can use the *Report Documentation Bug* links next to headlines in the HTML version of this document. These preselect the right product and category in Bugzilla and add a link to the current section. You can start typing your bug report right away.

Contributions

To contribute to this documentation, use the *Edit Source* links next to headlines in the HTML version of this document. They take you to the source code on GitHub, where you can open a pull request. A GitHub account is required.

Note: *Edit Source* only available for English

The *Edit Source* links are only available for the English version of each document. For all other languages, use the *Report Documentation Bug* links instead.

For more information about the documentation environment used for this documentation, see the repository's README at https://github.com/SUSE/doc-ses.

Mail

You can also report errors and send feedback concerning the documentation to `doc-team@suse.com`. Include the document title, the product version, and the publication date of the document. Additionally, include the relevant section number and title (or provide the URL) and provide a concise description of the problem.

3 Documentation conventions

The following notices and typographic conventions are used in this document:

- `/etc/passwd`: Directory names and file names
- *`PLACEHOLDER`*: Replace *`PLACEHOLDER`* with the actual value
- `PATH`: An environment variable

- `ls`, `--help`: Commands, options, and parameters
- `user`: The name of user or group
- `package_name`: The name of a software package
- `Alt`, `Alt`–`F1`: A key to press or a key combination. Keys are shown in uppercase as on a keyboard.
- *File*, *File* › *Save As*: menu items, buttons
- `AMD/Intel` This paragraph is only relevant for the AMD64/Intel 64 architectures. The arrows mark the beginning and the end of the text block.
  `IBM Z, POWER` This paragraph is only relevant for the architectures `IBM Z` and `POWER`. The arrows mark the beginning and the end of the text block.
- *Chapter 1, "Example chapter"*: A cross-reference to another chapter in this guide.
- Commands that must be run with `root` privileges. Often you can also prefix these commands with the `sudo` command to run them as non-privileged user.

```
# command
> sudo command
```

- Commands that can be run by non-privileged users.

```
> command
```

- Notices

Warning: Warning notice

Vital information you must be aware of before proceeding. Warns you about security issues, potential loss of data, damage to hardware, or physical hazards.

Important: Important notice

Important information you should be aware of before proceeding.

Note: Note notice

Additional information, for example about differences in software versions.

Tip: Tip notice

Helpful information, like a guideline or a piece of practical advice.

- Compact Notices

Additional information, for example about differences in software versions.

Helpful information, like a guideline or a piece of practical advice.

4 Support

Find the support statement for SUSE Enterprise Storage and general information about technology previews below. For details about the product lifecycle, see https://www.suse.com/lifecycle.

If you are entitled to support, find details on how to collect information for a support ticket at https://documentation.suse.com/sles-15/html/SLES-all/cha-adm-support.html.

4.1 Support statement for SUSE Enterprise Storage

To receive support, you need an appropriate subscription with SUSE. To view the specific support offerings available to you, go to https://www.suse.com/support/ and select your product.

The support levels are defined as follows:

L1

Problem determination, which means technical support designed to provide compatibility information, usage support, ongoing maintenance, information gathering and basic troubleshooting using available documentation.

L2

Problem isolation, which means technical support designed to analyze data, reproduce customer problems, isolate problem area and provide a resolution for problems not resolved by Level 1 or prepare for Level 3.

L3

Problem resolution, which means technical support designed to resolve problems by engaging engineering to resolve product defects which have been identified by Level 2 Support.

For contracted customers and partners, SUSE Enterprise Storage is delivered with L3 support for all packages, except for the following:

- Technology previews.
- Sound, graphics, fonts, and artwork.
- Packages that require an additional customer contract.
- Some packages shipped as part of the module *Workstation Extension* are L2-supported only.
- Packages with names ending in `-devel` (containing header files and similar developer resources) will only be supported together with their main packages.

SUSE will only support the usage of original packages. That is, packages that are unchanged and not recompiled.

4.2 Technology previews

Technology previews are packages, stacks, or features delivered by SUSE to provide glimpses into upcoming innovations. Technology previews are included for your convenience to give you a chance to test new technologies within your environment. We would appreciate your feedback! If you test a technology preview, please contact your SUSE representative and let them know about your experience and use cases. Your input is helpful for future development.

Technology previews have the following limitations:

- Technology previews are still in development. Therefore, they may be functionally incomplete, unstable, or in other ways *not* suitable for production use.
- Technology previews are *not* supported.
- Technology previews may only be available for specific hardware architectures.

- Details and functionality of technology previews are subject to change. As a result, upgrading to subsequent releases of a technology preview may be impossible and require a fresh installation.
- SUSE may discover that a preview does not meet customer or market needs, or does not comply with enterprise standards. Technology previews can be removed from a product at any time. SUSE does not commit to providing a supported version of such technologies in the future.

For an overview of technology previews shipped with your product, see the release notes at https://www.suse.com/releasenotes/x86_64/SUSE-Enterprise-Storage/7.1.

5 Ceph contributors

The Ceph project and its documentation is a result of the work of hundreds of contributors and organizations. See https://ceph.com/contributors/ for more details.

6 Commands and command prompts used in this guide

As a Ceph cluster administrator, you will be configuring and adjusting the cluster behavior by running specific commands. There are several types of commands you will need:

6.1 Salt-related commands

These commands help you to deploy Ceph cluster nodes, run commands on several (or all) cluster nodes at the same time, or assist you when adding or removing cluster nodes. The most frequently used commands are **`ceph-salt`** and **`ceph-salt config`**. You need to run Salt commands on the Salt Master node as `root`. These commands are introduced with the following prompt:

```
root@master #
```

For example:

```
root@master # ceph-salt config ls
```

6.2 Ceph related commands

These are lower-level commands to configure and fine tune all aspects of the cluster and its gateways on the command line, for example **`ceph`**, **`cephadm`**, **`rbd`**, or **`radosgw-admin`**.

To run Ceph related commands, you need to have read access to a Ceph key. The key's capabilities then define your privileges within the Ceph environment. One option is to run Ceph commands as `root` (or via **`sudo`**) and use the unrestricted default keyring 'ceph.client.admin.key'.

The safer and recommended option is to create a more restrictive individual key for each administrator user and put it in a directory where the users can read it, for example:

```
~/.ceph/ceph.client.USERNAME.keyring
```

Tip: Path to Ceph keys

To use a custom admin user and keyring, you need to specify the user name and path to the key each time you run the **`ceph`** command using the `-n client.USER_NAME` and `--keyring PATH/TO/KEYRING` options.

To avoid this, include these options in the `CEPH_ARGS` variable in the individual users' `~/.bashrc` files.

Although you can run Ceph-related commands on any cluster node, we recommend running them on the Admin Node. This documentation uses the `cephuser` user to run the commands, therefore they are introduced with the following prompt:

```
cephuser@adm >
```

For example:

```
cephuser@adm > ceph auth list
```

Tip: Commands for specific nodes

If the documentation instructs you to run a command on a cluster node with a specific role, it will be addressed by the prompt. For example:

```
cephuser@mon >
```

6.2.1 Running `ceph-volume`

Starting with SUSE Enterprise Storage 7, Ceph services are running containerized. If you need to run `ceph-volume` on an OSD node, you need to prepend it with the `cephadm` command, for example:

```
cephuser@adm > cephadm ceph-volume simple scan
```

6.3 General Linux commands

Linux commands not related to Ceph, such as `mount`, `cat`, or `openssl`, are introduced either with the `cephuser@adm >` or `#` prompts, depending on which privileges the related command requires.

6.4 Additional information

For more information on Ceph key management, refer to *Book "Administration and Operations Guide", Chapter 30 "Authentication with* `cephx`*", Section 30.2 "Key management"*.

I Quick Start: Deploying and Upgrading Ceph on SUSE CaaS Platform

1 Quick start

SUSE Enterprise Storage is a distributed storage system designed for scalability, reliability, and performance, which is based on the Ceph technology. The traditional way to run a Ceph cluster is setting up a dedicated cluster to provide block, file, and object storage to a variety of clients.

Rook manages Ceph as a containerized application on Kubernetes and allows a hyper-converged setup, in which a single Kubernetes cluster runs applications and storage together. The primary purpose of SUSE Enterprise Storage deployed with Rook is to provide storage to other applications running in the Kubernetes cluster. This can be block, file, or object storage.

This chapter describes how to quickly deploy containerized SUSE Enterprise Storage 7.1 on top of a SUSE CaaS Platform 4.5 Kubernetes cluster.

1.1 Recommended hardware specifications

For SUSE Enterprise Storage deployed with Rook, the minimal configuration is preliminary, we will update it based on real customer needs.

For the purpose of this document, consider the following minimum configuration:

- A highly available Kubernetes cluster with 3 master nodes
- Four physical Kubernetes worker nodes, each with two OSD disks and 5GB of RAM per OSD disk
- Allow additional 4GB of RAM per additional daemon deployed on a node
- Dual-10 Gb ethernet as bonded network
- If you are running a hyper-converged infrastructure (HCI), ensure you add any additional requirements for your workloads.

1.2 Prerequisites

Ensure the following prerequisites are met before continuing with this quickstart guide:

- Installation of SUSE CaaS Platform 4.5. See the SUSE CaaS Platform documentation for more details on how to install: https://documentation.suse.com/en-us/suse-caasp/4.5/.
- Ensure `ceph-csi` (and required sidecars) are running in your Kubernetes cluster.

- Installation of the LVM2 package on the host where the OSDs are running.
- Ensure you have one of the following storage options to configure Ceph properly:
  - Raw devices (no partitions or formatted file systems)
  - Raw partitions (no formatted file system)
- Ensure the SUSE CaaS Platform 4.5 repository is enabled for the installation of Helm 3.

1.3 Getting started with Rook

Note

The following instructions are designed for a quick start deployment only. For more information on installing Helm, see https://documentation.suse.com/en-us/suse-caasp/4.5/.

1. Install Helm v3:

```
# zypper in helm
```

2. On a node with access to the Kubernetes cluster, execute the following:

```
> export HELM_EXPERIMENTAL_OCI=1
```

3. Create a local copy of the Helm chart to your local registry:

```
> helm chart pull registry.suse.com/ses/7.1/charts/rook-ceph:latest
```

If you are using a version of Helm >= 3.7, you do not need to specify the subcommand `chart`. The protocol is also explicit and the version is presumed to be `latest`.

```
> helm pull oci://registry.suse.com/ses/7.1/charts/rook-ceph
```

4. Export the Helm charts to a Rook-Ceph sub-directory under your current working directory:

```
> helm chart export registry.suse.com/ses/7.1/charts/rook-ceph:latest
```

For Helm versions >= 3.7, you just need to extract the tarball yourself:

```
> tar -xzf rook-ceph-1.8.6.tar.gz
```

5. Create a file named `myvalues.yaml` based off the `rook-ceph/values.yaml file`.
6. Set local parameters in `myvalues.yaml`.
7. Create the namespace:

```
kubectl@adm > kubectl create namespace rook-ceph
```

8. Install the helm charts:

```
> helm install -n rook-ceph rook-ceph ./rook-ceph/ -f myvalues.yaml
```

9. Verify the `rook-operator` is running:

```
kubectl@adm > kubectl -n rook-ceph get pod -l app=rook-ceph-operator
```

1.4 Deploying Ceph with Rook

1. You need to apply labels to your Kubernetes nodes before deploying your Ceph cluster. The key `node-role.rook-ceph/cluster` accepts one of the following values:

   - `any`
   - `mon`
   - `mon-mgr`
   - `mon-mgr-osd`

   Run the following the get the names of your cluster's nodes:

```
kubectl@adm > kubectl get nodes
```

2. On the Master node, run the following:

```
kubectl@adm > kubectl label nodes node-name label-key=label-value
```

   For example:

```
kubectl@adm > kubectl label node k8s-worker-node-1 node-role.rook-ceph/cluster=any
```

3. Verify the application of the label by re-running the following command:

```
kubectl@adm > kubectl get nodes --show-labels
```

You can also use the **`describe`** command to get the full list of labels given to the node. For example:

```
kubectl@adm > kubectl describe node node-name
```

4. Next, you need to apply a Ceph cluster manifest file, for example, `cluster.yaml`, to your Kubernetes cluster. You can apply the example `cluster.yaml` as is without any additional services or requirements from the Rook Helm chart.
   To apply the example Ceph cluster manifest to your Kubernetes cluster, run the following command:

```
> kubectl create -f rook-ceph/examples/cluster.yaml
```

1.5 Configuring the Ceph cluster

You can have two types of integration with your SUSE Enterprise Storage intregrated cluster. These types are: CephFS or RADOS Block Device (RBD).

Before you start the SUSE CaaS Platform and SUSE Enterprise Storage integration, ensure you have met the following prerequisites:

- The SUSE CaaS Platform cluster must have `ceph-common` and `xfsprogs` installed on all nodes. You can check this by running the **`rpm -q ceph-common`** command or the **`rpm -q xfsprogs`** command.
- That the SUSE Enterprise Storage cluster has a pool with a RBD device or CephFS enabled.

1.5.1 Configure CephFS

For more information on configuring CephFS, see https://documentation.suse.com/en-us/suse-caasp/4.5/ for steps and more information. This section will also provide the necessary procedure on attaching a pod to either an CephFS static or dynamic volume.

1.5.2 Configure RADOS block device

For instructions on configuring the RADOS Block Device (RBD) in a pod, see https://documentation.suse.com/en-us/suse-caasp/4.5/ for more information. This section will also provide the necessary procedure on attaching a pod to either an RBD static or dynamic volume.

1.6 Updating local images

1. To update your local image to the latest tag, apply the new parameters in `myvalues.yaml`:

```
image:
refix: rook
repository: registry.suse.com/ses/7.1/rook/ceph
tag: LATEST_TAG
pullPolicy: IfNotPresent
```

2. Re-pull a new local copy of the Helm chart to your local registry:

```
> helm3 chart pull REGISTRY_URL
```

3. Export the Helm charts to a Rook-Ceph sub-directory under your current working directory:

```
> helm3 chart export REGISTRY_URL
```

4. Upgrade the Helm charts:

```
> helm3 upgrade -n rook-ceph rook-ceph ./rook-ceph/ -f myvalues.yaml
```

1.7 Uninstalling

1. Delete any Kubernetes applications that are consuming Rook storage.
2. Delete all object, file, and block storage artifacts.
3. Remove the CephCluster:

```
kubectl@adm > >kubectl delete -f cluster.yaml
```

4. Uninstall the operator:

```
> helm uninstall REGISTRY_URL
```

   Or, if you are using Helm >= 3.7:

```
> helm uninstall -n rook-ceph rook-ceph
```

5. Delete any data on the hosts:

```
> rm -rf /var/lib/rook
```

6. Wipe the disks if necessary.

7. Delete the namespace:

```
> kubectl delete namespace rook-ceph
```

2 Updating Rook

This chapter describes how to update containerized SUSE Enterprise Storage 7 on top of a SUSE CaaS Platform 4.5 Kubernetes cluster.

This chapter takes you through the steps to update the software in a Rook-Ceph cluster from one version to the next. This includes both the Rook-Ceph Operator software itself as well as the Ceph cluster software.

Note

Version

2.1 Recommended hardware specifications

For SUSE Enterprise Storage deployed with Rook, the minimal configuration is preliminary, we will update it based on real customer needs.

For the purpose of this document, consider the following minimum configuration:

- A highly available Kubernetes cluster with 3 master nodes
- Four physical Kubernetes worker nodes, each with two OSD disks and 5 GB of RAM per OSD disk
- Allow additional 4 GB of RAM per additional daemon deployed on a node
- Dual-10 Gb ethernet as bonded network
- If you are running a hyper-converged infrastructure (HCI), ensure you add any additional requirements for your workloads.

2.2 Patch release upgrades

To update a patch release of Rook to another, you need to update the common resources and the image of the Rook Operator.

1. Get the latest common resource manifests that contain the relevant changes to the latest version:

```
> zypper in rook-k8s-yaml
> cd /usr/share/k8s-yaml/rook/ceph/
```

2. Apply the latest changes from the next version and update the Rook Operator image:

```
kubectl@adm > helm upgrade -n rook-ceph rook-ceph ./rook-ceph/ -f myvalues.yaml
kubectl@adm > kubectl -n rook-ceph set image deploy/rook-ceph-operator
 registry.suse.com/ses/7.1/rook/ceph:version-number
```

3. Upgrade the Ceph version:

```
kubectl@adm > kubectl -n ROOK_CLUSTER_NAMESPACE patch CephCluster CLUSTER_NAME --
type=merge -p "{\"spec\": {\"cephVersion\": {\"image\": \"registry.suse.com/ses/7.1/
ceph/ceph:version-number\"}}}"
```

4. Mark the Ceph cluster to only support the updated version:

```
# from a ceph-toolbox
# ceph osd require-osd-release version-name
```

Note

We recommend updating the Rook-Ceph common resources from the example manifests before any update. The common resources and CRDs might not be updated with every release, but K8s will only apply updates to the ones that changed.

2.3 Rook-Ceph Updates

This is a general guide for updating your Rook cluster. For detailed instructions on updating to each supported version, refer to the upstream Rook upgrade documentation: https://rook.io/docs/rook/v1.8/ceph-upgrade.html.

To successfully upgrade a Rook cluster, the following prerequisites must be met:

- Cluster health status is healthy with full functionality.
- All pods consuming Rook storage should be created, running, and in a steady state.

Each version upgrade has specific details outlined in the Rook documentation. Use the following steps as a base guideline.

Note

These methods should work for any number of Rook-Ceph clusters and Rook Operators as long as you parameterize the environment correctly. Merely repeat these steps for each Rook-Ceph cluster (`ROOK_CLUSTER_NAMESPACE`), and be sure to update the `ROOK_OPERATOR_NAMESPACE` parameter each time if applicable.

1. Update common resources and CRDs.
2. Update Ceph CSI versions.
3. Update the Rook Operator.
4. Wait for the upgrade to complete and verify the updated cluster.
5. Update `CephRBDMirror` and `CephBlockPool` configuration options.

II Administrating Ceph on SUSE CaaS Platform

3 Rook-Ceph administration

This part of the guide focuses on routine tasks that you as an administrator need to take care of after the basic Ceph cluster has been deployed ("day two operations"). It also describes all the supported ways to access data stored in a Ceph cluster.

The chapters in this part contain links to additional documentation resources. These include additional documentation that is available on the system, as well as documentation available on the Internet.

For an overview of the documentation available for your product and the latest documentation updates, refer to https://documentation.suse.com.

4 Ceph cluster administration

This chapter introduces tasks that are performed on the whole cluster.

4.1 Shutting down and restarting the cluster

To shut down the whole Ceph cluster for planned maintenance tasks, follow these steps:

1. Stop all clients that are using the cluster.
2. Verify that the cluster is in a healthy state. Use the following commands:

```
cephuser@adm > ceph status
cephuser@adm > ceph health
```

3. Set the following OSD flags:

```
    cephuser@adm > ceph osd set noout
    cephuser@adm > ceph osd set nobackfill
    cephuser@adm > ceph osd set norecover
```

4. Shutdown service nodes one by one (non-storage workers).
5. Shutdown Ceph Monitor nodes one by one (masters by default).
6. Shutdown Admin Node (masters).

After you finish the maintenance, you can start the cluster again by running the above procedure in reverse order.

5 Block Storage

Block Storage allows a single pod to mount storage. This guide shows how to create a simple, multi-tier web application on Kubernetes using persistent volumes enabled by Rook.

5.1 Provisioning Block Storage

Before Rook can provision storage, a `StorageClass` and a `CephBlockPool` need to be created. This will allow Kubernetes to interoperate with Rook when provisioning persistent volumes.

Note

This sample requires *at least one OSD per node*, with each OSD located on *three different nodes*.

Each OSD must be located on a different node, because the `failureDomain` (https://github.com/rook/rook/blob/master/Documentation/ceph-pool-crd.md#spec) is set to `host` and the `replicated.size` is set to `3`.

Note

This example uses the CSI driver, which is the preferred driver going forward for Kubernetes 1.13 and newer. Examples are found in the CSI RBD (https://github.com/rook/rook/tree/release-1.4/cluster/examples/kubernetes/ceph/csi/rbd) directory.

Save this `StorageClass` definition as `storageclass.yaml`:

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

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: rook-ceph-block
# Change "rook-ceph" provisioner prefix to match the operator namespace if needed
provisioner: rook-ceph.rbd.csi.ceph.com
parameters:
    # clusterID is the namespace where the rook cluster is running
    clusterID: rook-ceph
    # Ceph pool into which the RBD image shall be created
    pool: replicapool

    # RBD image format. Defaults to "2".
    imageFormat: "2"

    # RBD image features. Available for imageFormat: "2". CSI RBD currently supports only
 `layering` feature.
    imageFeatures: layering

    # The secrets contain Ceph admin credentials.
    csi.storage.k8s.io/provisioner-secret-name: rook-csi-rbd-provisioner
    csi.storage.k8s.io/provisioner-secret-namespace: rook-ceph
    csi.storage.k8s.io/controller-expand-secret-name: rook-csi-rbd-provisioner
    csi.storage.k8s.io/controller-expand-secret-namespace: rook-ceph
    csi.storage.k8s.io/node-stage-secret-name: rook-csi-rbd-node
    csi.storage.k8s.io/node-stage-secret-namespace: rook-ceph

    # Specify the filesystem type of the volume. If not specified, csi-provisioner
    # will set default as `ext4`. Note that `xfs` is not recommended due to potential
 deadlock
    # in hyperconverged settings where the volume is mounted on the same node as the
 osds.
    csi.storage.k8s.io/fstype: ext4

# Delete the rbd volume when a PVC is deleted
reclaimPolicy: Delete
```

If you have deployed the Rook operator in a namespace other than “rook-ceph”, change the prefix in the provisioner to match the namespace you used. For example, if the Rook operator is running in the namespace “my-namespace” the provisioner value should be “my-namespace.rbd.csi.ceph.com”.

Create the storage class.

```
kubectl@adm > kubectl create -f cluster/examples/kubernetes/ceph/csi/rbd/
storageclass.yaml
```

Note

As specified by Kubernetes (https://kubernetes.io/docs/concepts/storage/persistent-volumes/#retain), when using the `Retain` reclaim policy, any Ceph RBD image that is backed by a `PersistentVolume` will continue to exist even after the `PersistentVolume` has been deleted. These Ceph RBD images will need to be cleaned up manually using `rbd rm`.

5.2 Consuming storage: WordPress sample

In this example, we will create a sample application to consume the block storage provisioned by Rook with the classic WordPress and MySQL apps. Both of these applications will make use of block volumes provisioned by Rook.

Start MySQL and WordPress from the `cluster/examples/kubernetes` folder:

```
kubectl@adm > kubectl create -f mysql.yaml
kubectl create -f wordpress.yaml
```

Both of these applications create a block volume, and mount it to their respective pod. You can see the Kubernetes volume claims by running the following:

```
kubectl@adm > kubectl get pvc
NAME             STATUS    VOLUME                                     CAPACITY
 ACCESSMODES   AGE
mysql-pv-claim   Bound     pvc-95402dbc-efc0-11e6-bc9a-0cc47a3459ee   20Gi       RWO
      1m
wp-pv-claim      Bound     pvc-39e43169-efc1-11e6-bc9a-0cc47a3459ee   20Gi       RWO
      1m
```

Once the WordPress and MySQL pods are in the `Running` state, get the cluster IP of the WordPress app and enter it in your browser:

```
kubectl@adm > kubectl get svc wordpress
NAME        CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
wordpress   10.3.0.155   <pending>     80:30841/TCP   2m
```

You should see the WordPress application running.

If you are using Minikube, the WordPress URL can be retrieved with this one-line command:

```
kubectl@adm > echo http://$(minikube ip):$(kubectl get service wordpress -o
 jsonpath='{.spec.ports[0].nodePort}')
```

Note

When running in a Vagrant environment, there will be no external IP address to reach WordPress with. You will only be able to reach WordPress via the `CLUSTER-IP` from inside the Kubernetes cluster.

5.3 Consuming the storage: Toolbox

With the pool that was created above, we can also create a block image and mount it directly in a pod.

5.4 Teardown

To clean up all the artifacts created by the block-storage demonstration:

```
kubectl@adm > kubectl delete -f wordpress.yaml
kubectl@adm > kubectl delete -f mysql.yaml
kubectl@adm > kubectl delete -n rook-ceph cephblockpools.ceph.rook.io replicapool
kubectl@adm > kubectl delete storageclass rook-ceph-block
```

5.5 Advanced Example: Erasure-Coded Block Storage

If you want to use erasure-coded pools with RBD, your OSDs must use `bluestore` as their `storeType`. Additionally, the nodes that will mount the erasure-coded RBD block storage must have Linux kernel `4.11` or above.

This example requires *at least three bluestore OSDs*, with each OSD located on a *different node*.

The OSDs must be located on different nodes, because the `failureDomain` is set to `host` and the `erasureCoded` chunk settings require at least three different OSDs (two `dataChunks` plus one `codingChunk`).

To be able to use an erasure-coded pool, you need to create two pools (as seen below in the definitions): one erasure-coded, and one replicated.

5.5.1 Erasure coded CSI driver

The erasure-coded pool must be set as the `dataPool` parameter in `storageclass-ec.yaml` (https://github.com/rook/rook/blob/release-1.4/cluster/examples/kubernetes/ceph/csi/rbd/storageclass-ec.yaml) It is used for the data of the RBD images.

6 CephFS

6.1 Shared File System

A shared file system can be mounted with read/write permission from multiple pods. This may be useful for applications which can be clustered using a shared file system.

This example runs a shared file system for the kube-registry (https://github.com/kubernetes/kubernetes/tree/release-1.18/cluster/addons).

6.1.1 Prerequisites

This guide assumes you have created a Rook cluster as explained in the main guide: *Chapter 1, Quick start*.

Note

By default, only one shared file system can be created with Rook. Multiple file system support in Ceph is still considered experimental, and can be enabled with the environment variable `ROOK_ALLOW_MULTIPLE_FILESYSTEMS` defined in `operator.yaml`.

6.1.2 Creating the File System

Create the file system by specifying the desired settings for the metadata pool, data pools, and metadata server in the `CephFilesystem` CRD. In this example, we create the metadata pool with replication of three, and a single data pool with replication of three. For more options, see the documentation *Section 7.3, "Ceph shared file system CRD"*.

Save this shared file system definition as `filesystem.yaml`:

```
apiVersion: ceph.rook.io/v1
kind: CephFilesystem
metadata:
  name: myfs
  namespace: rook-ceph
spec:
  metadataPool:
    replicated:
```

```
      size: 3
    dataPools:
    - replicated:
        size: 3
    preservePoolsOnDelete: true
    metadataServer:
      activeCount: 1
      activeStandby: true
```

The Rook operator will create all the pools and other resources necessary to start the service. This may take a minute to complete.

Create the file system:

```
kubectl@adm > kubectl create -f filesystem.yaml
```

To confirm the file system is configured, wait for the MDS pods to start:

```
kubectl@adm > kubectl -n rook-ceph get pod -l app=rook-ceph-mds
NAME                                    READY     STATUS    RESTARTS   AGE
rook-ceph-mds-myfs-7d59fdfcf4-h8kw9     1/1       Running   0          12s
rook-ceph-mds-myfs-7d59fdfcf4-kgkjp     1/1       Running   0          12s
```

To see detailed status of the file system, start and connect to the Rook toolbox. A new line will be shown with **`ceph status`** for the `mds` service. In this example, there is one active instance of MDS which is up, with one MDS instance in `standby-replay` mode in case of failover.

```
cephuser@adm > ceph status
[...]
services:
mds: myfs-1/1/1 up {[myfs:0]=mzw58b=up:active}, 1 up:standby-replay
```

6.1.3 Provisioning Storage

Before Rook can start provisioning storage, a `StorageClass` needs to be created based on the file system. This is needed for Kubernetes to interoperate with the CSI driver to create persistent volumes.

Note

This example uses the CSI driver, which is the preferred driver going forward for Kubernetes 1.13 and newer.

Save this storage class definition as `storageclass.yaml`:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: rook-cephfs
# Change "rook-ceph" provisioner prefix to match the operator namespace if needed
provisioner: rook-ceph.cephfs.csi.ceph.com
parameters:
  # clusterID is the namespace where operator is deployed.
  clusterID: rook-ceph

  # CephFS file system name into which the volume shall be created
  fsName: myfs

  # Ceph pool into which the volume shall be created
  # Required for provisionVolume: "true"
  pool: myfs-data0

  # Root path of an existing CephFS volume
  # Required for provisionVolume: "false"
  # rootPath: /absolute/path

  # The secrets contain Ceph admin credentials. These are generated automatically by the
 operator
  # in the same namespace as the cluster.
  csi.storage.k8s.io/provisioner-secret-name: rook-csi-cephfs-provisioner
  csi.storage.k8s.io/provisioner-secret-namespace: rook-ceph
  csi.storage.k8s.io/controller-expand-secret-name: rook-csi-cephfs-provisioner
  csi.storage.k8s.io/controller-expand-secret-namespace: rook-ceph
  csi.storage.k8s.io/node-stage-secret-name: rook-csi-cephfs-node
  csi.storage.k8s.io/node-stage-secret-namespace: rook-ceph

reclaimPolicy: Delete
```

If you have deployed the Rook operator in a namespace other than “rook-ceph”, change the prefix in the provisioner to match the namespace you used. For example, if the Rook operator is running in “rook-op”, the provisioner value should be “rook-op.rbd.csi.ceph.com”.

Create the storage class:

```
kubectl@adm > kubectl create -f cluster/examples/kubernetes/ceph/csi/cephfs/
storageclass.yaml
```

Important

The CephFS CSI driver uses quotas to enforce the PVC size requested. Only newer kernels support CephFS quotas (kernel version of at least 4.17).

6.1.4 Consuming the Shared File System: K8s Registry Sample

As an example, we will start the kube-registry pod with the shared file system as the backing store. Save the following spec as `kube-registry.yaml`:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: cephfs-pvc
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: rook-cephfs
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kube-registry
  namespace: kube-system
  labels:
    k8s-app: kube-registry
    kubernetes.io/cluster-service: "true"
spec:
  replicas: 3
  selector:
    matchLabels:
      k8s-app: kube-registry
  template:
    metadata:
      labels:
        k8s-app: kube-registry
        kubernetes.io/cluster-service: "true"
    spec:
      containers:
      - name: registry
        image: registry:2
```

```
        imagePullPolicy: Always
        resources:
          limits:
            cpu: 100m
            memory: 100Mi
        env:
        # Configuration reference: https://docs.docker.com/registry/configuration/
        - name: REGISTRY_HTTP_ADDR
          value: :5000
        - name: REGISTRY_HTTP_SECRET
          value: "Ple4seCh4ngeThisN0tAVerySecretV4lue"
        - name: REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY
          value: /var/lib/registry
        volumeMounts:
        - name: image-store
          mountPath: /var/lib/registry
        ports:
        - containerPort: 5000
          name: registry
          protocol: TCP
        livenessProbe:
          httpGet:
            path: /
            port: registry
        readinessProbe:
          httpGet:
            path: /
            port: registry
      volumes:
      - name: image-store
        persistentVolumeClaim:
          claimName: cephfs-pvc
          readOnly: false
```

Create the Kube registry deployment:

```
kubectl@adm > kubectl create -f cluster/examples/kubernetes/ceph/csi/cephfs/kube-
registry.yaml
```

You now have a High-Availability Docker registry with persistent storage.

Note

If the Rook cluster has more than one file system and the application pod is scheduled to a node with kernel version older than 4.7, inconsistent results may arise, since kernels older than 4.7 do not support specifying file system namespaces.

6.1.5 Consuming the Shared File System: Toolbox

Once you have pushed an image to the registry, verify that `kube-registry` is using the file system that was configured above by mounting the shared file system in the toolbox pod.

6.1.5.1 Teardown

To clean up all the artifacts created by the file system demo:

```
kubectl@adm > kubectl delete -f kube-registry.yaml
```

To delete the file system components and backing data, delete the Filesystem CRD.

Note

WARNING: Data will be deleted if `preservePoolsOnDelete=false`.

```
kubectl@adm > kubectl -n rook-ceph delete cephfilesystem myfs
```

Note

If the “preservePoolsOnDelete” file system attribute is set to true, the above command will not delete the pools. Creating the file system again with the same CRD will reuse the previous pools.

7 Ceph cluster custom resource definitions

7.1 Ceph cluster CRD

Rook allows the creation and customization of storage clusters through Custom Resource Definitions (CRDs). There are two different methods of cluster creation, depending on whether the storage on which to base the Ceph cluster can be dynamically provisioned.

1. Specify the host paths and raw devices.
2. Specify the storage class Rook should use to consume storage via PVCs.

Examples for each of these approaches follow.

7.1.1 Host-based cluster

To get you started, here is a simple example of a CRD to configure a Ceph cluster with all nodes and all devices. In the next example, the MONs and OSDs are backed by PVCs.

Note

In addition to your CephCluster object, you need to create the namespace, service accounts, and RBAC rules for the namespace in which you will create the CephCluster. These resources are defined in the example `common.yaml` file.

```
apiVersion: ceph.rook.io/v1
kind: CephCluster
metadata:
  name: rook-ceph
  namespace: rook-ceph
spec:
  cephVersion:
    # see the "Cluster Settings" section below for more details on which image of Ceph to
 run
    image: ceph/ceph:v15.2.4
  dataDirHostPath: /var/lib/rook
  mon:
    count: 3
```

```
    allowMultiplePerNode: true
  storage:
    useAllNodes: true
    useAllDevices: true
```

7.1.2 PVC-based cluster

Note

Kubernetes version 1.13.0 or greater is required to provision OSDs on PVCs.

```
apiVersion: ceph.rook.io/v1
kind: CephCluster
metadata:
  name: rook-ceph
  namespace: rook-ceph
spec:
  cephVersion:
    # see the "Cluster Settings" section below for more details on which image of Ceph to
 run
    image: ceph/ceph:v15.2.4
  dataDirHostPath: /var/lib/rook
  mon:
    count: 3
    volumeClaimTemplate:
      spec:
        storageClassName: local-storage
        resources:
          requests:
            storage: 10Gi
  storage:
   storageClassDeviceSets:
    - name: set1
      count: 3
      portable: false
      tuneDeviceClass: false
      encrypted: false
      volumeClaimTemplates:
      - metadata:
          name: data
        spec:
          resources:
            requests:
```

```
              storage: 10Gi
          # IMPORTANT: Change the storage class depending on your environment (e.g.
local-storage, gp2)
          storageClassName: local-storage
          volumeMode: Block
          accessModes:
            - ReadWriteOnce
```

For more advanced scenarios, such as adding a dedicated device, please refer to *Section 7.1.4.8, "Dedicated metadata and WAL device for OSD on PVC"*.

7.1.3 Settings

Settings can be specified at the global level to apply to the cluster as a whole, while other settings can be specified at more fine-grained levels. If any setting is unspecified, a suitable default will be used automatically.

7.1.3.1 Cluster metadata

- `name`: The name that will be used internally for the Ceph cluster. Most commonly, the name is the same as the namespace since multiple clusters are not supported in the same namespace.
- `namespace`: The Kubernetes namespace that will be created for the Rook cluster. The services, pods, and other resources created by the operator will be added to this namespace. The common scenario is to create a single Rook cluster. If multiple clusters are created, they must not have conflicting devices or host paths.

7.1.3.2 Cluster settings

- `external`:
  - `enable`: if `true`, the cluster will not be managed by Rook but via an external entity. This mode is intended to connect to an existing cluster. In this case, Rook will only consume the external cluster. However, if an image is provided, Rook will be able to deploy various daemons in Kubernetes, such as object gateways, MDS and NFS. If an image is not provided, it will refuse. If this setting is enabled, **all** the other options will

be ignored except `cephVersion.image` and `dataDirHostPath`. See *Section 7.1.4.9, "External cluster"*. If `cephVersion.image` is left blank, Rook will refuse the creation of extra CRs such as object, file and NFS.

- `cephVersion`: The version information for launching the Ceph daemons.
  - `image`: The image used for running the Ceph daemons. For example, `ceph/ceph:v16.2.7` or `ceph/ceph:v15.2.4`. To ensure a consistent version of the image is running across all nodes in the cluster, we recommend to use a very specific image version. Tags also exist that would give the latest version, but they are only recommended for test environments. Using the `v14` or similar tag is not recommended in production because it may lead to inconsistent versions of the image running across different nodes in the cluster.
- `dataDirHostPath`: The path on the host where config and data should be stored for each of the services. If the directory does not exist, it will be created. Because this directory persists on the host, it will remain after pods are deleted. You **must not** use the following paths and any of their subpaths: **`/etc/ceph`**, **`/rook`** or **`/var/log/ceph`**.
  - On **Minikube** environments, use **`/data/rook`**. Minikube boots into a `tmpfs` but it provides some directories where files can persist across reboots. Using one of these directories will ensure that Rook's data and configuration files persist and that enough storage space is available.

Warning

WARNING: For test scenarios, if you delete a cluster and start a new cluster on the same hosts, the path used by `dataDirHostPath` must be deleted. Otherwise, stale keys and other configuration will remain from the previous cluster and the new MONs will fail to start. If this value is empty, each pod will get an ephemeral directory to store their config files that is tied to the lifetime of the pod running on that node.

- `continueUpgradeAfterChecksEvenIfNotHealthy`: if set to `true`, Rook will continue the OSD daemon upgrade process even if the PGs are not clean, or continue with the MDS upgrade even the file system is not healthy.
- `dashboard`: Settings for the Ceph Dashboard. To view the dashboard in your browser, see *Book "Administration and Operations Guide"*.

- `enabled`: Whether to enable the dashboard to view cluster status.
- `urlPrefix`: Allows serving the dashboard under a subpath (useful when you are accessing the dashboard via a reverse proxy).
- `port`: Allows changing the default port where the dashboard is served.
- `ssl`: Whether to serve the dashboard via SSL; ignored on Ceph versions older than `13.2.2`.

- `monitoring`: Settings for monitoring Ceph using Prometheus. To enable monitoring on your cluster, see the *Book "Administration and Operations Guide", Chapter 16 "Monitoring and alerting"*.
  - `enabled`: Whether to enable-Prometheus based monitoring for this cluster.
  - `rulesNamespace`: Namespace to deploy `prometheusRule`. If empty, the namespace of the cluster will be used. We recommend:
    - If you have a single Rook Ceph cluster, set the `rulesNamespace` to the same namespace as the cluster, or leave it empty.
    - If you have multiple Rook Ceph clusters in the same Kubernetes cluster, choose the same namespace to set `rulesNamespace` for all the clusters (ideally, namespace with Prometheus deployed). Otherwise, you will get duplicate alerts with duplicate alert definitions.
- `network`: For the network settings for the cluster, refer to *Section 7.1.3.5, "Network configuration settings"*.
- `mon`: contains MON related options *Section 7.1.3.3, "MON settings"*.
- `mgr`: manager top level section.
  - `modules`: is the list of Ceph Manager modules to enable.
- `crashCollector`: The settings for crash collector daemon(s).
  - `disable`: if set to `true`, the crash collector will not run on any node where a Ceph daemon runs.
- `annotations`: *Section 7.1.3.10, "Annotations and labels"*

- `labels`: *Section 7.1.3.10, "Annotations and labels"*
- `placement`: *Section 7.1.3.11, "Placement configuration settings"*
- `resources`: *Section 7.1.3.12, "Cluster-wide resources configuration settings"*
- `priorityClassNames`: *Section 7.1.3.14, "Priority class names configuration settings"*
- `storage`: Storage selection and configuration that will be used across the cluster. Note that these settings can be overridden for specific nodes.
    - `useAllNodes`: `true` or `false`, indicating if all nodes in the cluster should be used for storage according to the cluster level storage selection and configuration values. If individual nodes are specified under the `nodes` field, then `useAllNodes` must be set to `false`.
    - `nodes`: Names of individual nodes in the cluster that should have their storage included in accordance with either the cluster level configuration specified above or any node specific overrides described in the next section below. `useAllNodes` must be set to `false` to use specific nodes and their configuration. See *Section 7.1.3.6, "Node settings"* below.
    - `config`: Config settings applied to all OSDs on the node unless overridden by `devices`.
    - *Section 7.1.3.7, "Storage selection settings"*
    - *Section 7.1.3.8, "Storage class device sets"*
- `disruptionManagement`: The section for configuring management of daemon disruptions
    - `managePodBudgets`: if `true`, the operator will create and manage `PodDisruptionBudgets` for OSD, MON, RGW, and MDS daemons. The operator will block eviction of OSDs by default and unblock them safely when drains are detected.
    - `osdMaintenanceTimeout`: is a duration in minutes that determines how long an entire failure domain like `region/zone/host` will be held in `noout` (in addition to the default DOWN/OUT interval) when it is draining. This is only relevant when `managePodBudgets` is `true`. The default value is `30` minutes.

- `manageMachineDisruptionBudgets`: if `true`, the operator will create and manage `MachineDisruptionBudgets` to ensure OSDs are only fenced when the cluster is healthy. Only available on OpenShift.

- `machineDisruptionBudgetNamespace`: the namespace in which to watch the `MachineDisruptionBudgets`.

- `removeOSDsIfOutAndSafeToRemove`: If `true` the operator will remove the OSDs that are down and whose data has been restored to other OSDs.

- `cleanupPolicy`: *Section 7.1.4.10, "Cleanup policy"*

7.1.3.3 MON settings

- `count`: Set the number of MONs to be started. This should be an odd number between one and nine. If not specified, the default is set to three, and `allowMultiplePerNode` is also set to `true`.

- `allowMultiplePerNode`: Enable (`true`) or disable (`false`) the placement of multiple MONs on one node. Default is `false`.

- `volumeClaimTemplate`: A `PersistentVolumeSpec` used by Rook to create PVCs for monitor storage. This field is optional, and when not provided, HostPath volume mounts are used. The current set of fields from template that are used are `storageClassName` and the `storage` resource request and limit. The default storage size request for new PVCs is `10Gi`. Ensure that associated storage class is configured to use `volumeBindingMode: WaitForFirstConsumer`. This setting only applies to new monitors that are created when the requested number of monitors increases, or when a monitor fails and is recreated.

If these settings are changed in the CRD, the operator will update the number of MONs during a periodic check of the MON health, which by default is every 45 seconds.

To change the defaults that the operator uses to determine the MON health and whether to failover a MON, refer to the *Section 7.1.3.15, "Health settings"*. The intervals should be small enough that you have confidence the MONs will maintain quorum, while also being long enough to ignore network blips where MONs are failed over too often.

7.1.3.4 Ceph Manager settings

You can use the cluster CR to enable or disable any manager module. For example, this can be configured:

```
mgr:
  modules:
  - name: <name of the module>
    enabled: true
```

Some modules will have special configuration to ensure the module is fully functional after being enabled. Specifically, the `pg_autoscaler`—Rook will configure all new pools with PG autoscaling by setting: `osd_pool_default_pg_autoscale_mode = on`

7.1.3.5 Network configuration settings

If not specified, the default SDN will be used. Configure the network that will be enabled for the cluster and services.

- `provider`: Specifies the network provider that will be used to connect the network interface.
- `selectors`: List the network selector(s) that will be used associated by a key.

Note

Changing networking configuration after a Ceph cluster has been deployed is not supported and will result in a non-functioning cluster.

To use host networking, set `provider: host`.

7.1.3.6 Node settings

In addition to the cluster level settings specified above, each individual node can also specify configuration to override the cluster level settings and defaults. If a node does not specify any configuration, then it will inherit the cluster level settings.

- `name`: The name of the node, which should match its `kubernetes.io/hostname` label.
- `config`: Configuration settings applied to all OSDs on the node unless overridden by `devices`.
- *Section 7.1.3.7, "Storage selection settings"*

When `useAllNodes` is set to `true`, Rook attempts to make Ceph cluster management as hands-off as possible while still maintaining reasonable data safety. If a usable node comes online, Rook will begin to use it automatically. To maintain a balance between hands-off usability and data safety, nodes are removed from Ceph as OSD hosts only (1) if the node is deleted from Kubernetes itself or (2) if the node has its taints or affinities modified in such a way that the node is no longer usable by Rook. Any changes to taints or affinities, intentional or unintentional, may affect the data reliability of the Ceph cluster. In order to help protect against this somewhat, deletion of nodes by taint or affinity modifications must be confirmed by deleting the Rook-Ceph operator pod and allowing the operator deployment to restart the pod.

For production clusters, we recommend that `useAllNodes` is set to `false` to prevent the Ceph cluster from suffering reduced data reliability unintentionally due to a user mistake. When `useAllNodes` is set to `false`, Rook relies on the user to be explicit about when nodes are added to or removed from the Ceph cluster. Nodes are only added to the Ceph cluster if the node is added to the Ceph cluster resource. Similarly, nodes are only removed if the node is removed from the Ceph cluster resource.

7.1.3.6.1 Node updates

Nodes can be added and removed over time by updating the cluster CRD —for example, with the following command:

```
kubectl -n rook-ceph edit cephcluster rook-ceph
```

This will bring up your default text editor and allow you to add and remove storage nodes from the cluster. This feature is only available when `useAllNodes` has been set to `false`.

7.1.3.7 Storage selection settings

Below are the settings available, both at the cluster and individual node level, for selecting which storage resources will be included in the cluster.

- `useAllDevices`: `true` or `false`, indicating whether all devices found on nodes in the cluster should be automatically consumed by OSDs. This is **Not recommended** unless you have a very controlled environment where you will not risk formatting of devices with existing data. When `true`, all devices/partitions will be used. Is overridden by `deviceFilter` if specified.
- `deviceFilter`: A regular expression for short kernel names of devices (for example, `sda`) that allows selection of devices to be consumed by OSDs. If individual devices have been specified for a node then this filter will be ignored. For example:
  - `sdb`: Selects only the `sdb` device (if found).
  - `^sd`: Selects all devices starting with `sd`.
  - `^sd[a-d]`: Selects devices starting with `sda`, `sdb`, `sdc`, and `sdd` (if found).
  - `^s`: Selects all devices that start with `s`.
  - `^[^r]`: Selects all devices that do *not* start with `r`
- `devicePathFilter`: A regular expression for device paths (for example, `/dev/disk/by-path/pci-0:1:2:3-scsi-1`) that allows selection of devices to be consumed by OSDs. If individual devices or `deviceFilter` have been specified for a node then this filter will be ignored. For example:
  - `^/dev/sd.`: Selects all devices starting with `sd`
  - `^/dev/disk/by-path/pci-.*`: Selects all devices which are connected to PCI bus
- `devices`: A list of individual device names belonging to this node to include in the storage cluster.
  - `name`: The name of the device (for example, `sda`), or full udev path (such as, `/dev/disk/by-id/ata-ST4000DM004-XXXX` — this will not change after reboots).
  - `config`: Device-specific configuration settings.
- `storageClassDeviceSets`: Explained in *Section 7.1.3.8, "Storage class device sets"*.

7.1.3.8 Storage class device sets

The following are the settings for Storage Class Device Sets which can be configured to create OSDs that are backed by block mode PVs.

- `name`: A name for the set.
- `count`: The number of devices in the set.
- `resources`: The CPU and RAM requests or limits for the devices (optional).
- `placement`: The placement criteria for the devices (optional; default is no placement criteria).
  The syntax is the same as for *Section 7.1.3.11, "Placement configuration settings"*. It supports `nodeAffinity`, `podAffinity`, `podAntiAffinity` and `tolerations` keys.
  We recommend configuring the placement such that the OSDs will be as evenly spread across nodes as possible. At a minimum, anti-affinity should be added, so at least one OSD will be placed on each available node.
  However, if there are more OSDs than nodes, this anti-affinity will not be effective. Another placement scheme to consider is adding labels to the nodes in such a way that the OSDs can be grouped on those nodes, create multiple `storageClassDeviceSets`, and add node affinity to each of the device sets that will place the OSDs in those sets of nodes.
- `preparePlacement`: The placement criteria for the preparation of the OSD devices. Creating OSDs is a two-step process and the prepare job may require different placement than the OSD daemons. If the `preparePlacement` is not specified, the `placement` will instead be applied for consistent placement for the OSD prepare jobs and OSD deployments. The `preparePlacement` is only useful for `portable` OSDs in the device sets. OSDs that are not portable will be tied to the host where the OSD prepare job initially runs.
  - For example, provisioning may require topology spread constraints across zones, but the OSD daemons may require constraints across hosts within the zones.
- `portable`: If `true`, the OSDs will be allowed to move between nodes during failover. This requires a storage class that supports portability (for example, `aws-ebs`, but not the local storage provisioner). If `false`, the OSDs will be assigned to a node permanently. Rook will configure Ceph's CRUSH map to support the portability.
- `tuneDeviceClass`: If `true`, because the OSD can be on a slow device class, Rook will adapt to that by tuning the OSD process. This will make Ceph perform better under that slow device.

- `volumeClaimTemplates`: A list of PVC templates to use for provisioning the underlying storage devices.
  - `resources.requests.storage`: The desired capacity for the underlying storage devices.
  - `storageClassName`: The StorageClass to provision PVCs from. The default is to use the cluster-default StorageClass. This StorageClass should provide a raw block device, multipath device, or logical volume. Other types are not supported. If you want to use logical volumes, please see the known issue of OSD on LV-backed PVC: https://github.com/rook/rook/blob/master/Documentation/ceph-common-issues.md#lvm-metadata-can-be-corrupted-with-osd-on-lv-backed-pvc
  - `volumeMode`: The volume mode to be set for the PVC.
  - `accessModes`: The access mode for the PVC to be bound by OSD.
- `schedulerName`: Scheduler name for OSD pod placement (optional).
- `encrypted`: whether to encrypt all the OSDs in a given storageClassDeviceSet.

7.1.3.9 Storage selection via Ceph DriveGroups

Ceph DriveGroups allow for specifying highly advanced OSD layouts. Refer to *Book "Administration and Operations Guide", Chapter 13 "Operational tasks", Section 13.4.3 "Adding OSDs using DriveGroups specification"* for both general information and detailed specification of DriveGroups with useful examples.

Important

When managing a Rook/Ceph cluster's OSD layouts with DriveGroups, the `storage` configuration is mostly ignored. `storageClassDeviceSets` can still be used to create OSDs on PVC, but Rook will no longer use `storage` configurations for creating OSDs on a node's devices. To avoid confusion, we recommend using the `storage` configuration *or* `DriveGroups`, but never both. Because `storage` and `DriveGroups` should not be used simultaneously, Rook only supports provisioning OSDs with DriveGroups on new Rook-Ceph clusters.

DriveGroups are defined by a name, a Ceph DriveGroups spec, and a Rook placement.

- `name`: A name for the DriveGroups.
- `spec`: The Ceph DriveGroups spec. Some components of the spec are treated differently in the context of Rook as noted below.
  - Rook overrides Ceph's definition of `placement` in order to use Rook's `placement` below.
  - Rook overrides Ceph's `service_id` field to be the same as the DriveGroups `name` above.
- `placement`: The placement criteria for nodes to provision with the DriveGroups (optional; default is no placement criteria, which matches all untainted nodes). The syntax is the same as for *Section 7.1.3.11, "Placement configuration settings"*.

7.1.3.10 Annotations and labels

Annotations and Labels can be specified so that the Rook components will have those annotations or labels added to them.

You can set annotations and labels for Rook components for the list of key value pairs:

- `all`: Set annotations / labels for all components
- `mgr`: Set annotations / labels for MGRs
- `mon`: Set annotations / labels for MONs
- `osd`: Set annotations / labels for OSDs
- `prepareosd`: Set annotations / labels for OSD Prepare Jobs

When other keys are set, `all` will be merged together with the specific component.

7.1.3.11 Placement configuration settings

Placement configuration for the cluster services. It includes the following keys: `mgr`, `mon`, `osd`, `cleanup`, and `all`. Each service will have its placement configuration generated by merging the generic configuration under `all` with the most specific one (which will override any attributes).

Note

Placement of OSD pods is controlled using the *Section 7.1.3.8, "Storage class device sets"*, not the general `placement` configuration.

A placement configuration is specified (according to the Kubernetes PodSpec) as:

- `nodeAffinity`
- `podAffinity`
- `podAntiAffinity`
- `tolerations`
- `topologySpreadConstraints`

If you use `labelSelector` for OSD pods, you must write two rules both for `rook-ceph-osd` and `rook-ceph-osd-prepare`.

The Rook Ceph operator creates a job called `rook-ceph-detect-version` to detect the full Ceph version used by the given `cephVersion.image`. The placement from the MON section is used for the job except for the `PodAntiAffinity` field.

7.1.3.12 Cluster-wide resources configuration settings

Resources should be specified so that the Rook components are handled after Kubernetes Pod Quality of Service classes. This allows to keep Rook components running when for example a node runs out of memory and the Rook components are not killed depending on their Quality of Service class.

You can set resource requests/limits for Rook components through the *Section 7.1.3.13, "Resource requirements and limits"* structure in the following keys:

- `mgr`: Set resource requests/limits for MGRs.
- `mon`: Set resource requests/limits for MONs.
- `osd`: Set resource requests/limits for OSDs.
- `prepareosd`: Set resource requests/limits for OSD prepare job.

- `crashcollector`: Set resource requests and limits for crash. This pod runs wherever there is a Ceph pod running. It scrapes for Ceph daemon core dumps and sends them to the Ceph manager crash module so that core dumps are centralized and can be easily listed/accessed.
- `cleanup`: Set resource requests and limits for cleanup job, responsible for wiping cluster's data after uninstall.

In order to provide the best possible experience running Ceph in containers, Rook internally recommends minimum memory limits if resource limits are passed. If a user configures a limit or request value that is too low, Rook will still run the pod(s) and print a warning to the operator log.

- `mon`: 1024 MB
- `mgr`: 512 MB
- `osd`: 2048 MB
- `mds`: 4096 MB
- `prepareosd`: 50 MB
- `crashcollector`: 60MB

7.1.3.13 Resource requirements and limits

- `requests`: Requests for CPU or memory.
  - `cpu`: Request for CPU (example: one CPU core `1`, 50% of one CPU core `500m`).
  - `memory`: Limit for Memory (example: one gigabyte of memory `1Gi`, half a gigabyte of memory `512Mi`).
- `limits`: Limits for CPU or memory.
  - `cpu`: Limit for CPU (example: one CPU core `1`, 50% of one CPU core `500m`).
  - `memory`: Limit for Memory (example: one gigabyte of memory `1Gi`, half a gigabyte of memory `512Mi`).

7.1.3.14 Priority class names configuration settings

Priority class names can be specified so that the Rook components will have those priority class names added to them.

You can set priority class names for Rook components for the list of key value pairs:

- `all`: Set priority class names for MGRs, MONs, OSDs.
- `mgr`: Set priority class names for MGRs.
- `mon`: Set priority class names for MONs.
- `osd`: Set priority class names for OSDs.

The specific component keys will act as overrides to `all`.

7.1.3.15 Health settings

Rook-Ceph will monitor the state of the CephCluster on various components by default. The following CRD settings are available:

- `healthCheck`: main Ceph cluster health monitoring section

Currently three health checks are implemented:

- `mon`: health check on the Ceph monitors. Basic check as to whether monitors are members of the quorum. If after a certain timeout a given monitor has not rejoined the quorum, it will be failed over and replaced by a new monitor.
- `osd`: health check on the Ceph OSDs.
- `status`: Ceph health status check; periodically checks the Ceph health state, and reflects it in the CephCluster CR status field.

The liveness probe of each daemon can also be controlled via `livenessProbe`. The setting is valid for `mon`, `mgr` and `osd`. Here is a complete example for both `daemonHealth` and `livenessProbe`:

```
healthCheck:
  daemonHealth:
    mon:
      disabled: false
      interval: 45s
      timeout: 600s
```

```
    osd:
      disabled: false
      interval: 60s
    status:
      disabled: false
  livenessProbe:
    mon:
      disabled: false
    mgr:
      disabled: false
    osd:
      disabled: false
```

You can change the `mgr` probe by applying the following:

```
healthCheck:
  livenessProbe:
    mgr:
      disabled: false
      probe:
        httpGet:
          path: /
          port: 9283
        initialDelaySeconds: 3
        periodSeconds: 3
```

Changing the liveness probe is an advanced operation and should rarely be necessary. If you want to change these settings, start with the probe specification that Rook generates by default and then modify the desired settings.

7.1.4 Samples

Here are several samples for configuring Ceph clusters. Each of the samples must also include the namespace and corresponding access granted for management by the Ceph operator. See the common cluster resources below.

7.1.4.1 Storage configuration: All devices

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

```
spec:
  cephVersion:
    image: ceph/ceph:v15.2.4
  dataDirHostPath: /var/lib/rook
  mon:
    count: 3
    allowMultiplePerNode: true
  dashboard:
    enabled: true
  # cluster level storage configuration and selection
  storage:
    useAllNodes: true
    useAllDevices: true
    deviceFilter:
    config:
      metadataDevice:
      databaseSizeMB: "1024" # this value can be removed for environments with normal
 sized disks (100 GB or larger)
      journalSizeMB: "1024"  # this value can be removed for environments with normal
 sized disks (20 GB or larger)
      osdsPerDevice: "1"
```

7.1.4.2 Storage configuration: Specific devices

Individual nodes and their configurations can be specified so that only the named nodes below will be used as storage resources. Each node's "name" field should match their "kubernetes.io/hostname" label.

```
apiVersion: ceph.rook.io/v1
kind: CephCluster
metadata:
  name: rook-ceph
  namespace: rook-ceph
spec:
  cephVersion:
    image: ceph/ceph:v15.2.4
  dataDirHostPath: /var/lib/rook
  mon:
    count: 3
    allowMultiplePerNode: true
  dashboard:
    enabled: true
  # cluster level storage configuration and selection
  storage:
    useAllNodes: false
```

```
    useAllDevices: false
    deviceFilter:
    config:
      metadataDevice:
      databaseSizeMB: "1024" # this value can be removed for environments with normal
 sized disks (100 GB or larger)
    nodes:
    - name: "172.17.4.201"
      devices:             # specific devices to use for storage can be specified for
 each node
      - name: "sdb" # Whole storage device
      - name: "sdc1" # One specific partition. Should not have a file system on it.
      - name: "/dev/disk/by-id/ata-ST4000DM004-XXXX" # both device name and explicit udev
 links are supported
      config:         # configuration can be specified at the node level which overrides
 the cluster level config
        storeType: bluestore
    - name: "172.17.4.301"
      deviceFilter: "^sd."
```

7.1.4.3 Node affinity

To control where various services will be scheduled by Kubernetes, use the placement configuration sections below. The example under “all” would have all services scheduled on Kubernetes nodes labeled with “role=storage-node” and tolerate taints with a key of “storage-node”.

```
apiVersion: ceph.rook.io/v1
kind: CephCluster
metadata:
  name: rook-ceph
  namespace: rook-ceph
spec:
  cephVersion:
    image: ceph/ceph:v15.2.4
  dataDirHostPath: /var/lib/rook
  mon:
    count: 3
    allowMultiplePerNode: true
  # enable the Ceph dashboard for viewing cluster status
  dashboard:
    enabled: true
  placement:
    all:
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
```

```
          nodeSelectorTerms:
          - matchExpressions:
            - key: role
              operator: In
              values:
              - storage-node
      tolerations:
      - key: storage-node
        operator: Exists
    mgr:
      nodeAffinity:
      tolerations:
    mon:
      nodeAffinity:
      tolerations:
    osd:
      nodeAffinity:
      tolerations:
```

7.1.4.4 Resource requests and limits

To control how many resources the Rook components can request/use, you can set requests and limits in Kubernetes for them. You can override these requests and limits for OSDs per node when using `useAllNodes: false` in the `node` item in the `nodes` list.

Warning

Before setting resource requests/limits, review the Ceph documentation for hardware recommendations for each component.

```
apiVersion: ceph.rook.io/v1
kind: CephCluster
metadata:
  name: rook-ceph
  namespace: rook-ceph
spec:
  cephVersion:
    image: ceph/ceph:v15.2.4
  dataDirHostPath: /var/lib/rook
  mon:
    count: 3
    allowMultiplePerNode: true
  # enable the Ceph dashboard for viewing cluster status
```

```
  dashboard:
    enabled: true
  # cluster level resource requests/limits configuration
  resources:
  storage:
    useAllNodes: false
    nodes:
    - name: "172.17.4.201"
      resources:
        limits:
          cpu: "2"
          memory: "4096Mi"
        requests:
          cpu: "2"
          memory: "4096Mi"
```

7.1.4.5 OSD topology

The topology of the cluster is important in production environments where you want your data spread across failure domains. The topology can be controlled by adding labels to the nodes. When the labels are found on a node at first OSD deployment, Rook will add them to the desired level in the CRUSH map.

The complete list of labels in hierarchy order from highest to lowest is:

```
topology.kubernetes.io/region
topology.kubernetes.io/zone
topology.rook.io/datacenter
topology.rook.io/room
topology.rook.io/pod
topology.rook.io/pdu
topology.rook.io/row
topology.rook.io/rack
topology.rook.io/chassis
```

For example, if the following labels were added to a node:

```
kubectl label node mynode topology.kubernetes.io/zone=zone1
kubectl label node mynode topology.rook.io/rack=rack1
```

Note

For versions previous to K8s 1.17, use the topology key: `failure-domain.beta.kubernetes.io/zone` or region.

These labels would result in the following hierarchy for OSDs on that node (this command can be run in the Rook toolbox):

```
[root@mynode /]# ceph osd tree
ID CLASS WEIGHT  TYPE NAME                STATUS REWEIGHT PRI-AFF
-1       0.01358 root default
-5       0.01358     zone zone1
-4       0.01358         rack rack1
-3       0.01358             host mynode
 0   hdd 0.00679                 osd.0         up  1.00000 1.00000
 1   hdd 0.00679                 osd.1         up  1.00000 1.00000
```

Ceph requires unique names at every level in the hierarchy (CRUSH map). For example, you cannot have two racks with the same name that are in different zones. Racks in different zones must be named uniquely.

Note that the `host` is added automatically to the hierarchy by Rook. The host cannot be specified with a topology label. All topology labels are optional.

Tip

When setting the node labels prior to `CephCluster` creation, these settings take immediate effect. However, applying this to an already deployed `CephCluster` requires removing each node from the cluster first and then re-adding it with new configuration to take effect. Do this node by node to keep your data safe! Check the result with `ceph osd tree` from the *Chapter 9, Toolboxes*. The OSD tree should display the hierarchy for the nodes that already have been re-added.

To utilize the `failureDomain` based on the node labels, specify the corresponding option in the `CephBlockPool`.

```
apiVersion: ceph.rook.io/v1
kind: CephBlockPool
metadata:
  name: replicapool
  namespace: rook-ceph
spec:
  failureDomain: rack  # this matches the topology labels on nodes
  replicated:
    size: 3
```

This configuration will split the replication of volumes across unique racks in the data center setup.

7.1.4.6 Using PVC storage for monitors

In the CRD specification below three monitors are created each using a 10Gi PVC created by Rook using the `local-storage` storage class.

```
apiVersion: ceph.rook.io/v1
kind: CephCluster
metadata:
  name: rook-ceph
  namespace: rook-ceph
spec:
  cephVersion:
    image: ceph/ceph:v15.2.4
  dataDirHostPath: /var/lib/rook
  mon:
    count: 3
    allowMultiplePerNode: false
    volumeClaimTemplate:
      spec:
        storageClassName: local-storage
        resources:
          requests:
            storage: 10Gi
  dashboard:
    enabled: true
  storage:
    useAllNodes: true
    useAllDevices: true
    deviceFilter:
    config:
      metadataDevice:
      databaseSizeMB: "1024" # this value can be removed for environments with normal
 sized disks (100 GB or larger)
      journalSizeMB: "1024"  # this value can be removed for environments with normal
 sized disks (20 GB or larger)
      osdsPerDevice: "1"
```

7.1.4.7 Using StorageClassDeviceSets

In the CRD specification below, three OSDs (having specific placement and resource values) and three MONs with each using a 10Gi PVC, are created by Rook using the `local-storage` storage class.

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

```
metadata:
  name: rook-ceph
  namespace: rook-ceph
spec:
  dataDirHostPath: /var/lib/rook
  mon:
    count: 3
    allowMultiplePerNode: false
    volumeClaimTemplate:
      spec:
        storageClassName: local-storage
        resources:
          requests:
            storage: 10Gi
  cephVersion:
    image: ceph/ceph:v15.2.4
    allowUnsupported: false
  dashboard:
    enabled: true
  network:
    hostNetwork: false
  storage:
    storageClassDeviceSets:
    - name: set1
      count: 3
      portable: false
      tuneDeviceClass: false
      resources:
        limits:
          cpu: "500m"
          memory: "4Gi"
        requests:
          cpu: "500m"
          memory: "4Gi"
      placement:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: "rook.io/cluster"
                  operator: In
                  values:
                    - cluster1
                topologyKey: "topology.kubernetes.io/zone"
      volumeClaimTemplates:
```

```
    - metadata:
        name: data
      spec:
        resources:
          requests:
            storage: 10Gi
        storageClassName: local-storage
        volumeMode: Block
        accessModes:
          - ReadWriteOnce
```

7.1.4.8 Dedicated metadata and WAL device for OSD on PVC

In the simplest case, Ceph OSD BlueStore consumes a single (primary) storage device. BlueStore is the engine used by the OSD to store data.

The storage device is normally used as a whole, occupying the full device that is managed directly by BlueStore. It is also possible to deploy BlueStore across additional devices such as a DB device. This device can be used for storing BlueStore's internal metadata. BlueStore (or rather, the embedded RocksDB) will put as much metadata as it can on the DB device to improve performance. If the DB device fills up, metadata will spill back onto the primary device (where it would have been otherwise). Again, it is only helpful to provision a DB device if it is faster than the primary device.

You can have multiple `volumeClaimTemplates` where each might either represent a device or a metadata device. So just taking the `storage` section this will give something like:

```
 storage:
  storageClassDeviceSets:
   - name: set1
     count: 3
     portable: false
     tuneDeviceClass: false
     volumeClaimTemplates:
     - metadata:
         name: data
       spec:
         resources:
           requests:
             storage: 10Gi
         # IMPORTANT: Change the storage class depending on your environment (e.g.
local-storage, gp2)
         storageClassName: gp2
         volumeMode: Block
```

```
          accessModes:
            - ReadWriteOnce
      - metadata:
          name: metadata
        spec:
          resources:
            requests:
              # Find the right size https://docs.ceph.com/docs/master/rados/
configuration/bluestore-config-ref/#sizing
              storage: 5Gi
          # IMPORTANT: Change the storage class depending on your environment (e.g.
 local-storage, io1)
          storageClassName: io1
          volumeMode: Block
          accessModes:
            - ReadWriteOnce
```

Note

Rook only supports three naming conventions for a given template:

- *data*: represents the main OSD block device, where your data is being stored.
- *metadata:* represents the metadata (including `block.db` and `block.wal`) device used to store the Ceph Bluestore database for an OSD.
- "wal": represents the `block.wal` device used to store the Ceph BlueStore database for an OSD. If this device is set, "metadata" device will refer specifically to the `block.db` device. It is recommended to use a faster storage class for the metadata or wal device, with a slower device for the data. Otherwise, having a separate metadata device will not improve the performance.

The BlueStore partition has the following reference combinations supported by the ceph-volume utility:

- A single "data" device.

```
storage:
  storageClassDeviceSets:
  - name: set1
    count: 3
    portable: false
    tuneDeviceClass: false
```

```
      volumeClaimTemplates:
      - metadata:
          name: data
        spec:
          resources:
            requests:
              storage: 10Gi
          # IMPORTANT: Change the storage class depending on your environment (e.g.
 local-storage, gp2)
          storageClassName: gp2
          volumeMode: Block
          accessModes:
            - ReadWriteOnce
```

- A *data* device and a *metadata* device.

```
  storage:
    storageClassDeviceSets:
    - name: set1
      count: 3
      portable: false
      tuneDeviceClass: false
      volumeClaimTemplates:
      - metadata:
          name: data
        spec:
          resources:
            requests:
              storage: 10Gi
          # IMPORTANT: Change the storage class depending on your environment (e.g.
 local-storage, gp2)
          storageClassName: gp2
          volumeMode: Block
          accessModes:
            - ReadWriteOnce
      - metadata:
          name: metadata
        spec:
          resources:
            requests:
              # Find the right size https://docs.ceph.com/docs/master/rados/
configuration/bluestore-config-ref/#sizing
              storage: 5Gi
          # IMPORTANT: Change the storage class depending on your environment (e.g.
 local-storage, io1)
          storageClassName: io1
          volumeMode: Block
```

```
          accessModes:
            - ReadWriteOnce
```

- A *data* device and a *WAL* device. A WAL device can be used for BlueStore's internal journal or write-ahead log (`block.wal`). It is only useful to use a WAL device if the device is faster than the primary device (the data device). There is no separate *metadata* device in this case; the data of main OSD block and `block.db` are located in *data* device.

```
  storage:
    storageClassDeviceSets:
    - name: set1
      count: 3
      portable: false
      tuneDeviceClass: false
      volumeClaimTemplates:
      - metadata:
          name: data
        spec:
          resources:
            requests:
              storage: 10Gi
          # IMPORTANT: Change the storage class depending on your environment (e.g.
 local-storage, gp2)
          storageClassName: gp2
          volumeMode: Block
          accessModes:
            - ReadWriteOnce
      - metadata:
          name: wal
        spec:
          resources:
            requests:
              # Find the right size https://docs.ceph.com/docs/master/rados/
configuration/bluestore-config-ref/#sizing
              storage: 5Gi
          # IMPORTANT: Change the storage class depending on your environment (e.g.
 local-storage, io1)
          storageClassName: io1
          volumeMode: Block
          accessModes:
            - ReadWriteOnce
```

- A *data* device, a *metadata* device and a *wal* device.

```
  storage:
    storageClassDeviceSets:
```

```
    - name: set1
      count: 3
      portable: false
      tuneDeviceClass: false
      volumeClaimTemplates:
      - metadata:
          name: data
        spec:
          resources:
            requests:
              storage: 10Gi
          # IMPORTANT: Change the storage class depending on your environment (e.g.
 local-storage, gp2)
          storageClassName: gp2
          volumeMode: Block
          accessModes:
            - ReadWriteOnce
      - metadata:
          name: metadata
        spec:
          resources:
            requests:
              # Find the right size https://docs.ceph.com/docs/master/rados/
configuration/bluestore-config-ref/#sizing
              storage: 5Gi
          # IMPORTANT: Change the storage class depending on your environment (e.g.
 local-storage, io1)
          storageClassName: io1
          volumeMode: Block
          accessModes:
            - ReadWriteOnce
      - metadata:
          name: wal
        spec:
          resources:
            requests:
              # Find the right size https://docs.ceph.com/docs/master/rados/
configuration/bluestore-config-ref/#sizing
              storage: 5Gi
          # IMPORTANT: Change the storage class depending on your environment (e.g.
 local-storage, io1)
          storageClassName: io1
          volumeMode: Block
          accessModes:
            - ReadWriteOnce
```

With the present configuration, each OSD will have its main block allocated a 10 GB device as well a 5 GB device to act as a BlueStore database.

7.1.4.9 External cluster

The minimum supported Ceph version for the External Cluster is Luminous 12.2.x.

The features available from the external cluster will vary depending on the version of Ceph. The following table shows the minimum version of Ceph for some of the features:

FEATURE	CEPH VERSION
Dynamic provisioning RBD	12.2.X
Configure extra CRDs (object, file, NFS)[a]	13.2.3
Dynamic provisioning CephFS	14.2.3

a Configure an object store, shared file system, or NFS resources in the local cluster to connect to the external Ceph cluster

7.1.4.9.1 Prerequisites

In order to configure an external Ceph cluster with Rook, we need to inject some information in order to connect to that cluster. You can use the `cluster/examples/kubernetes/ceph/import-external-cluster.sh` script to achieve that. The script will look for the following populated environment variables:

- `NAMESPACE`: the namespace where the configmap and secrets should be injected
- `ROOK_EXTERNAL_FSID`: the FSID of the external Ceph cluster. This can be retrieved via the `ceph fsid` command.
- `ROOK_EXTERNAL_CEPH_MON_DATA`: this is a comma- separated list of running monitors' IP addresses along with their ports. For example, `a=172.17.0.4:6789,b=172.17.0.5:6789,c=172.17.0.6:6789`. You do not need to specify all the monitors; you can simply pass one, and the operator will discover the rest. The name of the monitor is the name that appears in the **`ceph status`** output.

Now, we need to give Rook a key to connect to the cluster in order to perform various operations, such as cluster health checks, CSI keys management, etc. We recommend generating keys with minimal access, so the admin key does not need to be used by the external cluster. In this case, the admin key is only needed to generate the keys that will be used by the external cluster. If the admin key is to be used by the external cluster, however, set the following variable:

- `ROOK_EXTERNAL_ADMIN_SECRET`: **OPTIONAL:** the external Ceph cluster admin secret key. This can be retrieved via the `ceph auth get-key client.admin` command.

Note

WARNING: If you plan to create CRs (pool, rgw, mds, nfs) in the external cluster, you **MUST** inject the client.admin keyring as well as injecting `cluster-external-management.yaml`

Example:

```
export NAMESPACE=rook-ceph-external
export ROOK_EXTERNAL_FSID=3240b4aa-ddbc-42ee-98ba-4ea7b2a61514
export ROOK_EXTERNAL_CEPH_MON_DATA=a=172.17.0.4:6789
export ROOK_EXTERNAL_ADMIN_SECRET=AQC6Ylxdja+NDBAAB7qy9MEAr4VLLq4dCIvxtg==
```

If the Ceph admin key is not provided, the following script needs to be executed on a machine that can connect to the Ceph cluster using the Ceph admin key. On that machine, run `cluster/examples/kubernetes/ceph/create-external-cluster-resources.sh`. The script will automatically create users and keys with the lowest possible privileges and populate the necessary environment variables for `cluster/examples/kubernetes/ceph/import-external-cluster.sh` to work correctly.

Finally, execute the script like this from a machine that has access to your Kubernetes cluster:

```
bash cluster/examples/kubernetes/ceph/import-external-cluster.sh
```

7.1.4.9.2 CephCluster example (consumer)

Assuming the above section has successfully completed, here is a CR example:

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

```
  namespace: rook-ceph-external
spec:
  external:
    enable: true
  crashCollector:
    disable: true
  # optionally, the ceph-mgr IP address can be pass to gather metric from the prometheus
 exporter
  #monitoring:
    #enabled: true
    #rulesNamespace: rook-ceph
    #externalMgrEndpoints:
      #- ip: 192.168.39.182
```

Choose the namespace carefully; if you have an existing cluster managed by Rook, you have likely already injected `common.yaml`. Additionally, you need to inject `common-external.yaml` too.

You can now create it like this:

```
kubectl create -f cluster/examples/kubernetes/ceph/cluster-external.yaml
```

If the previous section has not been completed, the Rook Operator will still acknowledge the CR creation but will wait forever to receive connection information.

Warning

If no cluster is managed by the current Rook Operator, you need to inject `common.yaml`, then modify `cluster-external.yaml` and specify `rook-ceph` as `namespace`.

If this is successful, you will see the CephCluster status as `connected`.

```
kubectl get CephCluster -n rook-ceph-external
NAME                 DATADIRHOSTPATH   MONCOUNT   AGE    STATE       HEALTH
rook-ceph-external   /var/lib/rook                162m   Connected   HEALTH_OK
```

Before you create a StorageClass with this cluster you will need to create a pool in your external Ceph Cluster.

7.1.4.9.3 Example StorageClass based on external Ceph pool

In the cluster, list the pools available:

```
rados df
```

```
POOL_NAME     USED OBJECTS CLONES COPIES MISSING_ON_PRIMARY UNFOUND DEGRADED RD_OPS  RD
 WR_OPS  WR USED COMPR UNDER COMPR
replicated_2g  0 B       0      0      0                  0       0        0      0 0 B
    0 0 B        0 B         0 B
```

Here is an example StorageClass configuration that uses the `replicated_2g` pool from the external cluster:

```
cat << EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
   name: rook-ceph-block-ext
# Change "rook-ceph" provisioner prefix to match the operator namespace if needed
provisioner: rook-ceph.rbd.csi.ceph.com
parameters:
    # clusterID is the namespace where the rook cluster is running
    clusterID: rook-ceph-external
    # Ceph pool into which the RBD image shall be created
    pool: replicated_2g

    # RBD image format. Defaults to "2".
    imageFormat: "2"

    # RBD image features. Available for imageFormat: "2". CSI RBD currently supports only
 `layering` feature.
    imageFeatures: layering

    # The secrets contain Ceph admin credentials.
    csi.storage.k8s.io/provisioner-secret-name: rook-csi-rbd-provisioner
    csi.storage.k8s.io/provisioner-secret-namespace: rook-ceph-external
    csi.storage.k8s.io/controller-expand-secret-name: rook-csi-rbd-provisioner
    csi.storage.k8s.io/controller-expand-secret-namespace: rook-ceph-external
    csi.storage.k8s.io/node-stage-secret-name: rook-csi-rbd-node
    csi.storage.k8s.io/node-stage-secret-namespace: rook-ceph-external

    # Specify the filesystem type of the volume. If not specified, csi-provisioner
    # will set default as `ext4`. Note that `xfs` is not recommended due to potential
 deadlock
    # in hyperconverged settings where the volume is mounted on the same node as the
 osds.
    csi.storage.k8s.io/fstype: ext4

# Delete the rbd volume when a PVC is deleted
reclaimPolicy: Delete
allowVolumeExpansion: true
EOF
```

You can now create a persistent volume based on this StorageClass.

7.1.4.9.4 CephCluster example (management)

The following CephCluster CR represents a cluster that will perform management tasks on the external cluster. It will not only act as a consumer, but will also allow the deployment of other CRDs such as CephFilesystem or CephObjectStore. As mentioned above, you would need to inject the admin keyring for that.

The corresponding YAML example:

```
apiVersion: ceph.rook.io/v1
kind: CephCluster
metadata:
  name: rook-ceph-external
  namespace: rook-ceph-external
spec:
  external:
    enable: true
  dataDirHostPath: /var/lib/rook
  cephVersion:
    image: ceph/ceph:v15.2.4 # Should match external cluster version
```

7.1.4.10 Cleanup policy

Rook has the ability to cleanup resources and data that were deployed when a **`delete ceph-cluster`** command is issued. The policy represents the confirmation that cluster data should be forcibly deleted. The `cleanupPolicy` should only be added to the cluster when the cluster is

about to be deleted. After the `confirmation` field of the cleanup policy is set, Rook will stop configuring the cluster as if the cluster is about to be destroyed in order to prevent these settings from being deployed unintentionally. The `cleanupPolicy` CR settings has different fields:

- `confirmation`: Only an empty string and `yes-really-destroy-data` are valid values for this field. If an empty string is set, Rook will only remove Ceph's metadata. A re-installation will not be possible unless the hosts are cleaned first. If `yes-really-destroy-data` the operator will automatically delete data on the hostpath of cluster nodes and clean devices with OSDs. The cluster can then be re-installed if desired with no further steps.
- `sanitizeDisks`: sanitizeDisks represents advanced settings that can be used to sanitize drives. This field only affects if `confirmation` is set to `yes-really-destroy-data`. However, the administrator might want to sanitize the drives in more depth with the following flags:
  - `method`: indicates whether the entire disk should be sanitized or Ceph metadata only. Possible choices are "quick" (default) or "complete".
  - `dataSource`: indicate where to get random bytes from to write on the disk. Possible choices are "zero" (default) or "random". Using random sources will consume entropy from the system and will take much more time then the zero source.
  - `iteration`: overwrite N times instead of the default (1). Takes an integer value.
- `allowUninstallWithVolumes`: If set to true, then the cephCluster deletion does not wait for the PVCs to be deleted. Default is `false`.

To automate activation of the cleanup, you can use the following command:

Warning

Data will be permanently deleted.

```
kubectl -n rook-ceph patch cephcluster rook-ceph --type merge \
 -p '{"spec":{"cleanupPolicy":{"confirmation":"yes-really-destroy-data"}}}'
```

Nothing will happen until the deletion of the CR is requested, so this can still be reverted. However, all new configuration by the operator will be blocked with this cleanup policy enabled.

Rook waits for the deletion of PVs provisioned using the CephCluster before proceeding to delete the CephCluster. To force deletion of the CephCluster without waiting for the PVs to be deleted, you can set the `allowUninstallWithVolumes` to `true` under `spec.CleanupPolicy`.

7.2 Ceph block pool CRD

Rook allows creation and customization of storage pools through the custom resource definitions (CRDs). The following settings are available for pools.

7.2.1 Samples

7.2.1.1 Replicated

For optimal performance, while also adding redundancy, this sample will configure Ceph to make three full copies of the data on multiple nodes.

Note

This sample requires at least one OSD per node, with each OSD located on three different nodes.

Each OSD must be located on a different node, because the `failureDomain` is set to `host` and the `replicated.size` is set to three.

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

7.2.1.2 Erasure coded

This sample will lower the overall storage capacity requirement, while also adding redundancy by using *Section 7.2.2.4, "Erasure coding"*.

Note

This sample requires at least three BlueStore OSDs.

The OSDs can be located on a single Ceph node or spread across multiple nodes, because the `failureDomain` is set to `osd` and the `erasureCoded` chunk settings require at least three different OSDs (two `dataChunks` + one `codingChunks`).

```
apiVersion: ceph.rook.io/v1
kind: CephBlockPool
metadata:
  name: ecpool
  namespace: rook-ceph
spec:
  failureDomain: osd
  erasureCoded:
    dataChunks: 2
    codingChunks: 1
  deviceClass: hdd
```

High performance applications typically will not use erasure coding due to the performance overhead of creating and distributing the chunks in the cluster.

When creating an erasure-coded pool, we recommend creating the pool when you have BlueStore OSDs in your cluster.

7.2.2 Pool settings

7.2.2.1 Metadata

- `name`: The name of the pool to create.
- `namespace`: The namespace of the Rook cluster where the pool is created.

7.2.2.2 Specification

- `replicated`: Settings for a replicated pool. If specified, `erasureCoded` settings must not be specified.

- `size`: The desired number of copies to make of the data in the pool.
- `requireSafeReplicaSize`: set to false if you want to create a pool with size one, setting pool size one could lead to data loss without recovery.

- `erasureCoded`: Settings for an erasure-coded pool. If specified, `replicated` settings must not be specified. See below for more details on *Section 7.2.2.4, "Erasure coding"*.
  - `dataChunks`: Number of chunks to divide the original object into
  - `codingChunks`: Number of coding chunks to generate
- `failureDomain`: The failure domain across which the data will be spread. This can be set to a value of either `osd` or `host`, with `host` being the default setting. A failure domain can also be set to a different type (for example, `rack`), if it is added as a `location` Storage Selection Settings. If a replicated pool of size three is configured and the `failureDomain` is set to `host`, all three copies of the replicated data will be placed on OSDs located on three different Ceph hosts. This case is guaranteed to tolerate a failure of two hosts without a loss of data. Similarly, a failure domain set to `osd`, can tolerate a loss of two OSD devices. If erasure coding is used, the data and coding chunks are spread across the configured failure domain.

Note

Neither Rook, nor Ceph, prevent the creation of a cluster where the replicated data (or erasure coded chunks) can be written safely. By design, Ceph will delay checking for suitable OSDs until a write request is made and this write can hang if there are not sufficient OSDs to satisfy the request.

- `deviceClass`: Sets up the CRUSH rule for the pool to distribute data only on the specified device class. If left empty or unspecified, the pool will use the cluster's default CRUSH root, which usually distributes data over all OSDs, regardless of their class.
- `crushRoot`: The root in the crush map to be used by the pool. If left empty or unspecified, the default root will be used. Creating a crush hierarchy for the OSDs currently requires the Rook toolbox to run the Ceph tools.

- `enableRBDStats`: Enables collecting RBD per-image IO statistics by enabling dynamic OSD performance counters. Defaults to `false`.
- `parameters`: Sets any parameters listed to the given pool
  - `target_size_ratio:` gives a hint (%) to Ceph in terms of expected consumption of the total cluster capacity of a given pool.
  - `compression_mode`: Sets up the pool for inline compression when using a BlueStore OSD. If left unspecified does not setup any compression mode for the pool. Values supported are the same as BlueStore inline compression modes, such as `none`, `passive`, `aggressive`, and `force`.

7.2.2.3 Add specific pool properties

With `poolProperties` you can set any pool property:

```
spec:
  parameters:
    <name of the parameter>: <parameter value>
```

For example:

```
spec:
  parameters:
    min_size: 1
```

7.2.2.4 Erasure coding

Erasure coding (http://docs.ceph.com/docs/master/rados/operations/erasure-code/) allows you to keep your data safe while reducing the storage overhead. Instead of creating multiple replicas of the data, erasure coding divides the original data into chunks of equal size, then generates extra chunks of that same size for redundancy.

For example, if you have an object of size 2 MB, the simplest erasure coding with two data chunks would divide the object into two chunks of size 1 MB each (data chunks). One more chunk (coding chunk) of size 1 MB will be generated. In total, 3 MB will be stored in the cluster. The object will be able to suffer the loss of any one of the chunks and still be able to reconstruct the original object.

The number of data and coding chunks you choose will depend on your resiliency to loss and how much storage overhead is acceptable in your storage cluster. Here are some examples to illustrate how the number of chunks affects the storage and loss toleration.

Data chunks (k)	Coding chunks (m)	Total storage	Losses Tolerated	OSDs required
2	1	1.5x	1	3
2	2	2x	2	4
4	2	1.5x	2	6
16	4	1.25x	4	20

The `failureDomain` must be also be taken into account when determining the number of chunks. The failure domain determines the level in the Ceph CRUSH hierarchy where the chunks must be uniquely distributed. This decision will impact whether node losses or disk losses are tolerated. There could also be performance differences of placing the data across nodes or OSDs.

- `host`: All chunks will be placed on unique hosts
- `osd`: All chunks will be placed on unique OSDs

If you do not have a sufficient number of hosts or OSDs for unique placement the pool can be created, writing to the pool will hang.

Rook currently only configures two levels in the CRUSH map. It is also possible to configure other levels such as `rack` with by adding topology labels to the nodes.

7.3 Ceph shared file system CRD

Rook allows creation and customization of shared file systems through the custom resource definitions (CRDs). The following settings are available for Ceph file systems.

7.3.1 Samples

7.3.1.1 Replicated

Note

This sample requires at least one OSD per node, with each OSD located on three different nodes.

Each OSD must be located on a different node, because both of the defined pools set the `failureDomain` to `host` and the `replicated.size` to three.

The `failureDomain` can also be set to another location type (for example, `rack`), if it has been added as a `location` in the Storage Selection Settings.

```
apiVersion: ceph.rook.io/v1
kind: CephFilesystem
metadata:
  name: myfs
  namespace: rook-ceph
spec:
  metadataPool:
    failureDomain: host
    replicated:
      size: 3
  dataPools:
    - failureDomain: host
      replicated:
        size: 3
  preservePoolsOnDelete: true
  metadataServer:
    activeCount: 1
    activeStandby: true
    # A key/value list of annotations
    annotations:
    #  key: value
    placement:
    #  nodeAffinity:
    #    requiredDuringSchedulingIgnoredDuringExecution:
    #      nodeSelectorTerms:
    #      - matchExpressions:
    #        - key: role
    #          operator: In
```

```
    #          values:
    #          - mds-node
    #  tolerations:
    #  - key: mds-node
    #    operator: Exists
    #  podAffinity:
    #  podAntiAffinity:
    #  topologySpreadConstraints:
    resources:
    #  limits:
    #    cpu: "500m"
    #    memory: "1024Mi"
    #  requests:
    #    cpu: "500m"
    #    memory: "1024Mi"
```

These definitions can be found in the `filesystem.yaml` file.

7.3.1.2 Erasure coded

Erasure coded pools require the OSDs to use BlueStore for the configured `storeType`. Additionally, erasure coded pools can only be used with `dataPools`. The `metadataPool` must use a replicated pool.

Note

This sample requires at least three BlueStore OSDs, with each OSD located on a different node.

The OSDs must be located on different nodes, because the `failureDomain` will be set to `host` by default, and the `erasureCoded` chunk settings require at least three different OSDs (two `dataChunks` + one `codingChunks`).

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

```
    - erasureCoded:
        dataChunks: 2
        codingChunks: 1
  metadataServer:
    activeCount: 1
    activeStandby: true
```

These definitions can also be found in the `filesystem-ec.yaml` file.

7.3.2 File system settings

7.3.2.1 Metadata

- `name`: The name of the file system to create, which will be reflected in the pool and other resource names.
- `namespace`: The namespace of the Rook cluster where the file system is created.

7.3.2.2 Pools

The pools allow all of the settings defined in the Pool CRD spec. In the example above, there must be at least three hosts (size three) and at least eight devices (six data + two coding chunks) in the cluster.

- `metadataPool`: The settings used to create the filesystem metadata pool. Must use replication.
- `dataPools`: The settings to create the file system data pools. If multiple pools are specified, Rook will add the pools to the file system. The data pools can use replication or erasure coding. If erasure coding pools are specified, the cluster must be running with BlueStore enabled on the OSDs.
- `preservePoolsOnDelete`: If it is set to `true` the pools used to support the file system will remain when the file system will be deleted. This is a security measure to avoid accidental loss of data. It is set to `false` by default. If not specified is also deemed as `false`.

7.3.3 Metadata server settings

The metadata server settings correspond to the MDS daemon settings.

- `activeCount`: The number of active MDS instances. As load increases, CephFS will automatically partition the file system across the MDS instances. Rook will create double the number of MDS instances as requested by the active count. The extra instances will be in standby mode for failover.
- `activeStandby`: If true, the extra MDS instances will be in active standby mode and will keep a warm cache of the file system metadata for faster failover. The instances will be assigned by CephFS in failover pairs. If false, the extra MDS instances will all be on passive standby mode and will not maintain a warm cache of the metadata.
- `annotations`: Key value pair list of annotations to add.
- `labels`: Key value pair list of labels to add.
- `placement`: The mds pods can be given standard Kubernetes placement restrictions with `nodeAffinity`, `tolerations`, `podAffinity`, and `podAntiAffinity` similar to placement defined for daemons configured by the cluster CRD.
- `resources`: Set resource requests and limits for the Filesystem MDS Pod(s).
- `priorityClassName`: Set priority class name for the File system MDS Pod(s)

8 Configuration

8.1 Ceph configuration

For almost any Ceph cluster, the user will want—and may need— to change some Ceph configurations. These changes often may be warranted in order to alter performance to meet SLAs, or to update default data resiliency settings.

Warning

Modify Ceph settings carefully, and review the Ceph configuration documentation before making any changes. Changing the settings could result in unhealthy daemons or even data loss if used incorrectly.

8.1.1 Required configurations

Rook and Ceph both strive to make configuration as easy as possible, but there are some configuration options which users are well advised to consider for any production cluster.

8.1.1.1 Default PG and PGP counts

The number of PGs and PGPs can be configured on a per-pool basis, but it is highly advised to set default values that are appropriate for your Ceph cluster. Appropriate values depend on the number of OSDs the user expects to have backing each pool.

Pools created prior to v1.1 will have a default PG count of 100. Pools created after v1.1 will have Ceph's default PG count.

As of the Ceph Octopus (v15.2.x) release, the PG auto-scaler mgr module is enabled by default.

With that setting, the autoscaler will be enabled for all new pools. If you do not desire to have the autoscaler enabled for all new pools, you will need to use the Rook toolbox to enable the module and enable the autoscaling on individual pools.

The autoscaler is not enabled for the existing pools after enabling the module. So if you want to enable the autoscaling for these existing pools, they must be configured from the toolbox.

8.1.2 Specifying configuration options

8.1.2.1 Toolbox and the Ceph CLI

The most recommended way of configuring Ceph is to set Ceph's configuration directly. The first method for doing so is to use Ceph's CLI from the Rook-Ceph toolbox pod. From the toolbox, the user can change Ceph configurations, enable manager modules, create users and pools, and much more.

8.1.2.2 Ceph Dashboard

The Ceph Dashboard is another way of setting some of Ceph's configuration directly. Configuration by the Ceph Dashboard is recommended with the same priority as configuration via the Ceph CLI (above).

8.1.2.3 Advanced configuration via `ceph.conf` overrides ConfigMap

Setting configuration options via Ceph's CLI requires that at least one MON be available for the configuration options to be set, and setting configuration options via dashboard requires at least one mgr to be available. Ceph may also have a small number of very advanced settings that are not able to be modified easily via CLI or dashboard. The **least** recommended method for configuring Ceph is intended as a last-resort fallback in situations like these.

9 Toolboxes

9.1 Rook toolbox

The Rook toolbox is a container with common tools used for rook debugging and testing. The toolbox is based on SUSE Linux Enterprise Server, so more tools of your choosing can be installed with `zypper`.

The toolbox can be run in two modes:

- *Section 9.1.1, "Interactive toolbox"*: Start a toolbox pod where you can connect and execute Ceph commands from a shell.
- *Section 9.1.2, "Running the toolbox job"*: Run a script with Ceph commands and collect the results from the job log.

Note

Prerequisite: Before running the toolbox you should have a running Rook cluster deployed.

9.1.1 Interactive toolbox

The Rook toolbox can run as a deployment in a Kubernetes cluster where you can connect and run arbitrary Ceph commands.

Save the tools spec as `toolbox.yaml`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rook-ceph-tools
  namespace: rook-ceph
  labels:
    app: rook-ceph-tools
spec:
  replicas: 1
  selector:
    matchLabels:
```

```
        app: rook-ceph-tools
    template:
      metadata:
        labels:
          app: rook-ceph-tools
      spec:
        dnsPolicy: ClusterFirstWithHostNet
        containers:
        - name: rook-ceph-tools
          image: registry.suse.com/ses/7.1/rook/ceph:LATEST_TAG
          command: ["/tini"]
          args: ["-g", "--", "/usr/bin/toolbox.sh"]
          imagePullPolicy: IfNotPresent
          env:
            - name: ROOK_CEPH_USERNAME
              valueFrom:
                secretKeyRef:
                  name: rook-ceph-mon
                  key: ceph-username
            - name: ROOK_CEPH_SECRET
              valueFrom:
                secretKeyRef:
                  name: rook-ceph-mon
                  key: ceph-secret
          volumeMounts:
            - mountPath: /etc/ceph
              name: ceph-config
            - name: mon-endpoint-volume
              mountPath: /etc/rook
        volumes:
          - name: mon-endpoint-volume
            configMap:
              name: rook-ceph-mon-endpoints
              items:
              - key: data
                path: mon-endpoints
          - name: ceph-config
            emptyDir: {}
        tolerations:
          - key: "node.kubernetes.io/unreachable"
            operator: "Exists"
            effect: "NoExecute"
            tolerationSeconds: 5
```

Launch the `rook-ceph-tools` pod:

```
kubectl@adm > kubectl create -f toolbox.yaml
```

Wait for the toolbox pod to download its container and get to the `running` state:

```
kubectl@adm > kubectl -n rook-ceph get pod -l "app=rook-ceph-tools"
```

Once the rook-ceph-tools pod is running, you can connect to it with:

```
kubectl@adm > kubectl -n rook-ceph exec -it $(kubectl -n rook-ceph get pod -l "app=rook-
ceph-tools" -o jsonpath='{.items[0].metadata.name}') bash
```

All available tools in the toolbox are ready for your troubleshooting needs.

Example:

- **`ceph status`**
- **`ceph osd status`**
- **`ceph df`**
- **`rados df`**

When you are done with the toolbox, you can remove the deployment:

```
kubectl@adm > kubectl -n rook-ceph delete deployment rook-ceph-tools
```

9.1.2 Running the toolbox job

If you want to run Ceph commands as a one-time operation and collect the results later from the logs, you can run a script as a Kubernetes job. The toolbox job will run a script that is embedded in the job specification. The script has the full flexibility of a bash script.

In this example, the **`ceph status`** command is executed when the job is created.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: rook-ceph-toolbox-job
  namespace: rook-ceph
  labels:
    app: ceph-toolbox-job
spec:
  template:
    spec:
      initContainers:
      - name: config-init
        image: registry.suse.com/ses/7.1/rook/ceph:LATEST_TAG
        command: ["/usr/bin/toolbox.sh"]
        args: ["--skip-watch"]
        imagePullPolicy: IfNotPresent
```

```
          env:
          - name: ROOK_CEPH_USERNAME
            valueFrom:
              secretKeyRef:
                name: rook-ceph-mon
                key: ceph-username
          - name: ROOK_CEPH_SECRET
            valueFrom:
              secretKeyRef:
                name: rook-ceph-mon
                key: ceph-secret
          volumeMounts:
          - mountPath: /etc/ceph
            name: ceph-config
          - name: mon-endpoint-volume
            mountPath: /etc/rook
        containers:
        - name: script
          image: registry.suse.com/ses/7.1/rook/ceph:LATEST_TAG
          volumeMounts:
          - mountPath: /etc/ceph
            name: ceph-config
            readOnly: true
          command:
          - "bash"
          - "-c"
          - |
            # Modify this script to run any ceph, rbd, radosgw-admin, or other commands
that could
            # be run in the toolbox pod. The output of the commands can be seen by
getting the pod log.
            #
            # example: print the ceph status
            ceph status
        volumes:
        - name: mon-endpoint-volume
          configMap:
            name: rook-ceph-mon-endpoints
            items:
            - key: data
              path: mon-endpoints
        - name: ceph-config
          emptyDir: {}
        restartPolicy: Never
```

Create the toolbox job:

```
kubectl@adm > kubectl create -f toolbox-job.yaml
```

After the job completes, see the results of the script:

```
kubectl@adm > kubectl -n rook-ceph logs -l job-name=rook-ceph-toolbox-job
```

10 Ceph OSD management

10.1 Ceph OSD management

Ceph Object Storage Daemons (OSDs) are the heart and soul of the Ceph storage platform. Each OSD manages a local device and together they provide the distributed storage. Rook will automate creation and management of OSDs to hide the complexity based on the desired state in the CephCluster CR as much as possible. This guide will walk through some of the scenarios to configure OSDs where more configuration may be required.

10.1.1 Analyzing OSD health

The `rook-ceph-tools` pod provides a simple environment to run Ceph tools. The Ceph commands mentioned in this document should be run from the toolbox.

Once created, connect to the pod to execute the **`ceph`** commands to analyze the health of the cluster, in particular the OSDs and placement groups (PGs). Some common commands to analyze OSDs include:

```
cephuser@adm > ceph status
cephuser@adm > ceph osd tree
cephuser@adm > ceph osd status
cephuser@adm > ceph osd df
cephuser@adm > ceph osd utilization
```

```
kubectl -n rook-ceph exec -it $(kubectl -n rook-ceph get pod -l "app=rook-ceph-tools" -o
 jsonpath='{.items[0].metadata.name}') bash
```

10.1.2 Adding an OSD

To add more OSDs, Rook automatically watches for new nodes and devices being added to your cluster. If they match the filters or other settings in the `storage` section of the cluster CR, the operator will create new OSDs.

10.1.3 Adding an OSD on a PVC

In more dynamic environments where storage can be dynamically provisioned with a raw block storage provider, the OSDs can be backed by PVCs.

To add more OSDs, you can either increase the `count` of the OSDs in an existing device set or you can add more device sets to the cluster CR. The operator will then automatically create new OSDs according to the updated cluster CR.

10.1.4 Removing an OSD

Removal of OSDs is intentionally not automated. Rook's charter is to keep your data safe, not to delete it. If you are sure you need to remove OSDs, it can be done. We just want you to be in control of this action.

To remove an OSD due to a failed disk or other re-configuration, consider the following to ensure the health of the data through the removal process:

1. Confirm you will have enough space on your cluster after removing your OSDs to properly handle the deletion.
2. Confirm the remaining OSDs and their placement groups (PGs) are healthy in order to handle the rebalancing of the data.
3. Do not remove too many OSDs at once, wait for rebalancing between removing multiple OSDs.
4. On host-based clusters, you may need to stop the Rook Operator while performing OSD removal steps in order to prevent Rook from detecting the old OSD and trying to re-create it before the disk is wiped or removed.

If all the PGs are `active+clean` and there are no warnings about being low on space, this means the data is fully replicated and it is safe to proceed. If an OSD is failing, the PGs will not be perfectly clean, and you will need to proceed anyway.

10.1.4.1 From the toolbox

1. Determine the OSD ID for the OSD to be removed. The OSD pod may be in an error state, such as `CrashLoopBackoff`, or the **`ceph`** commands in the toolbox may show which OSD is `down`.
2. Mark the OSD as `out` if not already marked as such by Ceph. This signals Ceph to start moving (backfilling) the data that was on that OSD to another OSD.

   ```
   ceph osd out osd.ID
   ```

   For example:

   ```
   cephuserceph osd out osd.23
   ```
3. Wait for the data to finish backfilling to other OSDs.
   `ceph status` will indicate the backfilling is done when all of the PGs are `active+clean`. It is safe to remove the disk after that.
4. Update your CephCluster CR such that the operator will not create an OSD on the device anymore. Depending on your CR settings, you may need to remove the device from the list or update the device filter. If you are using `useAllDevices: true`, no change to the CR is necessary.
5. Remove the OSD from the Ceph cluster:

   ```
   cephuser@adm > ceph osd purge ID --yes-i-really-mean-it
   ```
6. Verify the OSD is removed from the node in the CRUSH map:

   ```
   cephuser@adm > ceph osd tree
   ```

10.1.4.2 Removing the OSD deployment

The operator can automatically remove OSD deployments that are considered “safe-to-destroy” by Ceph. After the steps above, the OSD will be considered safe to remove since the data has all been moved to other OSDs. But this will only be done automatically by the operator if you have this setting in the cluster CR:

```
removeOSDsIfOutAndSafeToRemove: true
```

Otherwise, you will need to delete the deployment directly:

```
kubectl@adm > kubectl delete deployment -n rook-ceph rook-ceph-osd-ID
```

10.1.5 Replacing an OSD

To replace a disk that has failed:

1. Run the steps in the previous section to *Section 10.1.4, "Removing an OSD"*.
2. Replace the physical device and verify the new device is attached.
3. Check if your cluster CR will find the new device. If you are using `useAllDevices: true` you can skip this step. If your cluster CR lists individual devices or uses a device filter you may need to update the CR.
4. The operator ideally will automatically create the new OSD within a few minutes of adding the new device or updating the CR. If you do not see a new OSD automatically created, restart the operator (by deleting the operator pod) to trigger the OSD creation.
5. Verify if the OSD is created on the node by running **`ceph osd tree`** from the toolbox.

Note

The OSD might have a different ID than the previous OSD that was replaced.

10.1.6 Removing an OSD from a PVC

If you have installed your OSDs on top of PVCs and you desire to reduce the size of your cluster by removing OSDs:

1. Shrink the number of OSDs in the `storageClassDeviceSet` in the CephCluster CR.

   ```
   kubectl@adm > kubectl -n rook-ceph edit cephcluster rook-ceph
   ```

   Reduce the `count` of the OSDs to the desired number. Rook will not take any action to automatically remove the extra OSD(s), but will effectively stop managing the orphaned OSD.

2. Identify the orphaned PVC that belongs to the orphaned OSD.

   

   #### Note

   The orphaned PVC will have the highest index among the PVCs for the device set.

```
kubectl@adm > kubectl -n rook-ceph get pvc -l ceph.rook.io/DeviceSet=deviceSet
```

For example if the device set is named `set1` and the `count` was reduced from `3` to `2`, the orphaned PVC would have the index `2` and might be named `set1-2-data-vbwcf`

3. Identify the orphaned OSD.

Note

The OSD assigned to the PVC can be found in the labels on the PVC.

```
kubectl@adm > kubectl -n rook-ceph get pod -l ceph.rook.io/pvc=ORPHANED_PVC -o yaml
 | grep ceph-osd-id
```

For example, this might return:

```
cephuser@adm > ceph-osd-id: "0"
```

4. Now, proceed with the steps in the section above to *Section 10.1.4, "Removing an OSD"* for the orphaned OSD ID.

5. If desired, delete the orphaned PVC after the OSD is removed.

11 Ceph examples

11.1 Ceph examples

Configuration for Rook and Ceph can be configured in multiple ways to provide block devices, shared file system volumes, or object storage in a Kubernetes namespace. We have provided several examples to simplify storage setup, but remember there are many tunables and you will need to decide what settings work for your use case and environment.

11.1.1 Creating common resources

The first step to deploy Rook is to create the common resources. The configuration for these resources will be the same for most deployments. The `common.yaml` sets these resources up.

```
kubectl@adm > kubectl create -f common.yaml
```

The examples all assume the operator and all Ceph daemons will be started in the same namespace. If you want to deploy the operator in a separate namespace, see the comments throughout `common.yaml`.

11.1.2 Creating the operator

After the common resources are created, the next step is to create the Operator deployment.

- `operator.yaml`: The most common settings for production deployments

  ```
  kubectl@adm > kubectl create -f operator.yaml
  ```

- `operator-openshift.yaml`: Includes all of the operator settings for running a basic Rook cluster in an OpenShift environment.

  ```
  kubectl@adm > oc create -f operator-openshift.yaml
  ```

Settings for the operator are configured through environment variables on the operator deployment. The individual settings are documented in the `common.yaml`.

11.1.3 Creating the cluster CRD

Now that your operator is running, create your Ceph storage cluster. This CR contains the most critical settings that will influence how the operator configures the storage. It is important to understand the various ways to configure the cluster. These examples represent a very small set of the different ways to configure the storage.

- `cluster.yaml`: This file contains common settings for a production storage cluster. Requires at least three nodes.
- `cluster-test.yaml`: Settings for a test cluster where redundancy is not configured. Requires only a single node.
- `cluster-on-pvc.yaml`: This file contains common settings for backing the Ceph MONs and OSDs by PVs. Useful when running in cloud environments or where local PVs have been created for Ceph to consume.
- `cluster-with-drive-groups.yaml`: This file contains example configurations for creating advanced OSD layouts on nodes using Ceph Drive Groups.
- `cluster-external`: Connect to an external Ceph cluster with minimal access to monitor the health of the cluster and connect to the storage.
- `cluster-external-management`: Connect to an external Ceph cluster with the admin key of the external cluster to enable remote creation of pools and configure services such as an Object Storage or Shared file system.

11.1.4 Setting up consumable storage

Now we are ready to setup block, shared file system or object storage in the Rook Ceph cluster. These kinds of storage are respectively referred to as `CephBlockPool`, `Cephfilesystem` and `CephObjectStore` in the spec files.

11.1.4.1 Provisioning block devices

Ceph can provide raw block device volumes to pods. Each example below sets up a storage class which can then be used to provision a block device in Kubernetes pods.

- `storageclass.yaml`: This example illustrates replication of three for production scenarios and requires at least three nodes. Your data is replicated on three different Kubernetes worker nodes and intermittent or long-lasting single node failures will not result in data unavailability or loss.
- `storageclass-ec.yaml`: Configures erasure coding for data durability rather than replication.
- `storageclass-test.yaml`: Replication of one for test scenarios and it requires only a single node. Do not use this for applications that store valuable data or have high-availability storage requirements, since a single node failure can result in data loss.

The storage classes are found in different sub-directories depending on the driver:

- `csi/rbd`: The CSI driver for block devices.

11.1.4.2 Shared file system

CephFS (CephFS) allows the user to mount a shared POSIX-compliant folder into one or more hosts (pods in the container world). This storage is similar to NFS shared storage or CIFS shared folders.

File storage contains multiple pools that can be configured for different scenarios:

- `filesystem.yaml`: Replication of three for production scenarios. Requires at least three nodes.
- `filesystem-ec.yaml`: Erasure coding for production scenarios. Requires at least three nodes.
- `filesystem-test.yaml`: Replication of one for test scenarios. Requires only a single node.

Dynamic provisioning is possible with the CSI driver. The storage class for shared file systems is found in the `csi/cephfs` directory.

11.1.4.3 Object Storage

Ceph supports storing blobs of data called objects that support HTTP[S]-type get/put/post and delete semantics.

Object Storage contains multiple pools that can be configured for different scenarios:

- `object.yaml`: Replication of three for production scenarios. Requires at least three nodes.
- `object-openshift.yaml`: Replication of three with Object Gateway in a port range valid for OpenShift. Requires at least three nodes.
- `object-ec.yaml`: Erasure coding rather than replication for production scenarios. Requires at least three nodes.
- `object-test.yaml`: Replication of one for test scenarios. Requires only a single node.

11.1.4.4 Object Storage user

- `object-user.yaml`: Creates a simple object storage user and generates credentials for the S3 API.

11.1.4.5 Object Storage buckets

The Ceph operator also runs an object store bucket provisioner which can grant access to existing buckets or dynamically provision new buckets.

- `object-bucket-claim-retain.yaml`: Creates a request for a new bucket by referencing a StorageClass which saves the bucket when the initiating OBC is deleted.
- `object-bucket-claim-delete.yaml`: Creates a request for a new bucket by referencing a StorageClass which deletes the bucket when the initiating OBC is deleted.
- `storageclass-bucket-retain.yaml`: Creates a new StorageClass which defines the Ceph Object Store, a region, and retains the bucket after the initiating OBC is deleted.
- `storageclass-bucket-delete.yaml` Creates a new StorageClass which defines the Ceph Object Store, a region, and deletes the bucket after the initiating OBC is deleted.

12 Advanced configuration

12.1 Performing advanced configuration tasks

These examples show how to perform advanced configuration tasks on your Rook storage cluster.

12.1.1 Prerequisites

Most of the examples make use of the **`ceph`** client command. A quick way to use the Ceph client suite is from a Rook Toolbox container (https://github.com/rook/rook/blob/master/Documentation/ceph-toolbox.md).

The Kubernetes based examples assume Rook OSD pods are in the `rook-ceph` namespace. If you run them in a different namespace, modify **`kubectl -n rook-ceph [...]`** to fit your situation.

12.1.2 Using custom Ceph user and secret for mounting

Note

For extensive info about creating Ceph users, refer to *Book "Administration and Operations Guide", Chapter 30 "Authentication with `cephx`", Section 30.2.2 "Managing users"*

Using a custom Ceph user and secret key can be done for both file system and block storage.

Create a custom user in Ceph with read-write access in the `/bar` directory on CephFS (For Ceph Mimic or newer, use `data=POOL_NAME` instead of `pool=POOL_NAME`):

```
cephuser@adm > ceph auth get-or-create-key client.user1 mon \
 'allow r' osd 'allow rw tag cephfs pool=YOUR_FS_DATA_POOL' \
 mds 'allow r, allow rw path=/bar'
```

The command will return a Ceph secret key. This key should be added as a secret in Kubernetes like this:

```
kubectl@adm > kubectl create secret generic ceph-user1-secret --from-
literal=key=YOUR_CEPH_KEY
```

Note

This secret key must be created with the same name in each namespace where the StorageClass will be used.

In addition to this secret key, you must create a RoleBinding to allow the Rook Ceph agent to get the secret from each namespace. The RoleBinding is optional if you are using a ClusterRoleBinding for the Rook Ceph agent secret-key access. A ClusterRole which contains the permissions which are needed and used for the Bindings is shown as an example after the next step.

On a StorageClass `parameters` set the following options:

```
mountUser: user1
mountSecret: ceph-user1-secret
```

If you want the Rook-Ceph agent to require a `mountUser` and `mountSecret` to be set in StorageClasses using Rook, you need to set the environment variable `AGENT_MOUNT_SECURITY_MODE` to `Restricted` on the Rook-Ceph Operator deployment.

For more information on using the Ceph feature to limit access to CephFS paths, see http://docs.ceph.com/docs/mimic/cephfs/client-auth/#path-restriction.

12.1.2.1 Creating the `ClusterRole`

Note

When you are using the Helm chart to install the Rook-Ceph Operator, and have set `mountSecurityMode` to, for example, `Restricted`, then the below `ClusterRole` has already been created for you.

This `ClusterRole` is needed no matter whether you want to use one `RoleBinding` per namespace or a `ClusterRoleBinding`.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: rook-ceph-agent-mount
  labels:
    operator: rook
    storage-backend: ceph
rules:
- apiGroups:
  - ""
  resources:
  - secrets
  verbs:
  - get
```

12.1.2.2 Creating the `RoleBinding`

Note

You either need a `RoleBinding` in each namespace in which a mount secret resides in, or create a `ClusterRoleBinding` with which the Rook Ceph agent has access to Kubernetes secrets in all namespaces.

Create the `RoleBinding` shown here in each namespace for which the Rook Ceph agent should read secrets for mounting. The `RoleBinding` subjects' `namespace` must be the one the Rook-Ceph agent runs in (default `rook-ceph` for version 1.0 and newer; for previous versions, the default namespace was `rook-ceph-system`).

Replace `namespace: name-of-namespace-with-mountsecret` according to the name of all namespaces a `mountSecret` can be in.

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: rook-ceph-agent-mount
  namespace: name-of-namespace-with-mountsecret
  labels:
    operator: rook
    storage-backend: ceph
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: rook-ceph-agent-mount
subjects:
- kind: ServiceAccount
  name: rook-ceph-system
  namespace: rook-ceph
```

12.1.2.3 Creating the `ClusterRoleBinding`

This `ClusterRoleBinding` only needs to be created once, as it covers the whole cluster.

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: rook-ceph-agent-mount
  labels:
    operator: rook
    storage-backend: ceph
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: rook-ceph-agent-mount
subjects:
- kind: ServiceAccount
  name: rook-ceph-system
  namespace: rook-ceph
```

12.1.3 Collecting logs

All Rook logs can be collected in a Kubernetes environment with the following command:

```
for p in $(kubectl -n rook-ceph get pods -o jsonpath='{.items[*].metadata.name}')
do
  for c in $(kubectl -n rook-ceph get pod ${p} -o jsonpath='{.spec.containers[*].name}')
  do
    echo "BEGIN logs from pod: ${p} ${c}"
    kubectl -n rook-ceph logs -c ${c} ${p}
    echo "END logs from pod: ${p} ${c}"
  done
done
```

This gets the logs for every container in every Rook pod, and then compresses them into a `.gz` archive for easy sharing. Note that instead of `gzip`, you could instead pipe to **`less`** or to a single text file.

12.1.4 OSD information

Keeping track of OSDs and their underlying storage devices can be difficult. The following scripts will clear things up quickly.

12.1.4.1 Kubernetes

```
# Get OSD Pods
# This uses the example/default cluster name "rook"
OSD_PODS=$(kubectl get pods --all-namespaces -l \
app=rook-ceph-osd,rook_cluster=rook-ceph -o jsonpath='{.items[*].metadata.name}')

# Find node and drive associations from OSD pods
for pod in $(echo ${OSD_PODS})
do
  echo "Pod:  ${pod}"
  echo "Node: $(kubectl -n rook-ceph get pod ${pod} -o jsonpath='{.spec.nodeName}')"
  kubectl -n rook-ceph exec ${pod} -- sh -c '\
  for i in /var/lib/ceph/osd/ceph-*; do
    [ -f ${i}/ready ] || continue
    echo -ne "-$(basename ${i}) "
    echo $(lsblk -n -o NAME,SIZE ${i}/block 2> /dev/null || \
    findmnt -n -v -o SOURCE,SIZE -T ${i}) $(cat ${i}/type)
  done | sort -V
  echo'
```

```
done
```

The output should look as follows:

```
Pod:  osd-m2fz2
Node: node1.zbrbdl
-osd0  sda3  557.3G  bluestore
-osd1  sdf3  110.2G  bluestore
-osd2  sdd3  277.8G  bluestore
-osd3  sdb3  557.3G  bluestore
-osd4  sde3  464.2G  bluestore
-osd5  sdc3  557.3G  bluestore

Pod:  osd-nxxnq
Node: node3.zbrbdl
-osd6   sda3  110.7G  bluestore
-osd17  sdd3  1.8T    bluestore
-osd18  sdb3  231.8G  bluestore
-osd19  sdc3  231.8G  bluestore

Pod:  osd-tww1h
Node: node2.zbrbdl
-osd7   sdc3  464.2G  bluestore
-osd8   sdj3  557.3G  bluestore
-osd9   sdf3  66.7G   bluestore
-osd10  sdd3  464.2G  bluestore
-osd11  sdb3  147.4G  bluestore
-osd12  sdi3  557.3G  bluestore
-osd13  sdk3  557.3G  bluestore
-osd14  sde3  66.7G   bluestore
-osd15  sda3  110.2G  bluestore
-osd16  sdh3  135.1G  bluestore
```

12.1.5 Separate storage groups

Note

Instead of manually needing to set this, the `deviceClass` property can be used on pool structures in `CephBlockPool`, `CephFilesystem` and `CephObjectStore` CRD objects.

By default Rook-Ceph puts all storage under one replication rule in the CRUSH Map which provides the maximum amount of storage capacity for a cluster. If you would like to use different storage endpoints for different purposes, you need to create separate storage groups.

In the following example we will separate SSD drives from spindle-based drives, a common practice for those looking to target certain workloads onto faster (database) or slower (file archive) storage.

12.1.6 Configure pools

12.1.6.1 Sizing placement groups

Note

Since Ceph Nautilus (v14.x), you can use the Ceph Manager `pg_autoscaler` module to auto-scale the PGs as needed. If you want to enable this feature, refer to *Section 8.1.1.1, "Default PG and PGP counts"*.

The general rules for deciding how many PGs your pool(s) should contain is:

- Less than five OSDs: set `pg_num` to 128.
- Between 5 and 10 OSDs: set `pg_num` to 512.
- Between 10 and 50 OSDs: set `pg_num` to 1024.

If you have more than 50 OSDs, you need to know how to calculate the `pg_num` value by yourself. For calculating `pg_num` yourself, please make use of the *pgcalc* tool at http://ceph.com/pgcalc/.

If you are already using a pool, it is generally safe to set `pg_count` on the fly (see *Section 12.1.6.2, "Setting PG count"*). Decreasing the PG count is not recommended on a pool that is in use. The safest way to decrease the PG count is to back up the data, delete the pool, and recreate it.

12.1.6.2 Setting PG count

Be sure to read the *Section 12.1.6.1, "Sizing placement groups"* section before changing the number of PGs.

```
# Set the number of PGs in the rbd pool to 512
cephuser@adm > ceph osd pool set rbd pg_num 512
```

12.1.7 Creating custom `ceph.conf` settings

Warning

The advised method for controlling Ceph configuration is to manually use the Ceph CLI or the Ceph Dashboard, because this offers the most flexibility. We recommend that this is used only when absolutely necessary, and that the `config` is reset to an empty string if or when the configurations are no longer necessary. Configurations in the config file will make the Ceph cluster less configurable from the CLI and Ceph Dashboard and may make future tuning or debugging difficult.

Setting configs via Ceph's CLI requires that at least one MON is available for the configs to be set, and setting configs via Ceph Dashboard requires at least one MGR to be available. Ceph may also have a small number of very advanced settings that are not able to be modified easily via CLI or Ceph Dashboard. In order to set configurations before MONs are available or to set problematic configuration settings, the `rook-config-override` ConfigMap exists, and the `config` field can be set with the contents of a `ceph.conf` file. The contents will be propagated to all MON, MGR, OSD, MDS, and RGW daemons as an `/etc/ceph/ceph.conf` file.

Warning

Rook performs no validation on the config, so the validity of the settings is the user's responsibility.

If the `rook-config-override` ConfigMap is created before the cluster is started, the Ceph daemons will automatically pick up the settings. If you add the settings to the ConfigMap after the cluster has been initialized, each daemon will need to be restarted where you want the settings applied:

- MONs: ensure all three MONs are online and healthy before restarting each mon pod, one at a time.
- MGRs: the pods are stateless and can be restarted as needed, but note that this will disrupt the Ceph dashboard during restart.
- OSDs: restart your the pods by deleting them, one at a time, and running **`ceph -s`** between each restart to ensure the cluster goes back to “active/clean” state.
- RGW: the pods are stateless and can be restarted as needed.
- MDS: the pods are stateless and can be restarted as needed.

After the pod restart, the new settings should be in effect. Note that if the ConfigMap in the Ceph cluster's namespace is created before the cluster is created, the daemons will pick up the settings at first launch.

12.1.7.1 Custom `ceph.conf` example

In this example we will set the default pool `size` to two, and tell OSD daemons not to change the weight of OSDs on startup.

Warning

Modify Ceph settings carefully. You are leaving the sandbox tested by Rook. Changing the settings could result in unhealthy daemons or even data loss if used incorrectly.

When the Rook Operator creates a cluster, a placeholder ConfigMap is created that will allow you to override Ceph configuration settings. When the daemon pods are started, the settings specified in this ConfigMap will be merged with the default settings generated by Rook.

The default override settings are blank. Cutting out the extraneous properties, we would see the following defaults after creating a cluster:

```
kubectl@adm > kubectl -n rook-ceph get ConfigMap rook-config-override -o yaml
kind: ConfigMap
apiVersion: v1
metadata:
  name: rook-config-override
  namespace: rook-ceph
data:
  config: ""
```

To apply your desired configuration, you will need to update this ConfigMap. The next time the daemon pod(s) start, they will use the updated configs.

```
kubectl@adm > kubectl -n rook-ceph edit configmap rook-config-override
```

Modify the settings and save. Each line you add should be indented from the `config` property as such:

```
apiVersion: v1
kind: ConfigMap
```

```
metadata:
  name: rook-config-override
  namespace: rook-ceph
data:
  config: |
    [global]
    osd crush update on start = false
    osd pool default size = 2
```

12.1.8 OSD CRUSH settings

A useful view of the CRUSH Map (see *Book "Administration and Operations Guide", Chapter 17 "Stored data management"* for more details) is generated with the following command:

```
cephuser@adm > ceph osd tree
```

In this section we will be tweaking some of the values seen in the output.

12.1.8.1 OSD weight

The CRUSH weight controls the ratio of data that should be distributed to each OSD. This also means a higher or lower amount of disk I/O operations for an OSD with higher or lower weight, respectively.

By default, OSDs get a weight relative to their storage capacity, which maximizes overall cluster capacity by filling all drives at the same rate, even if drive sizes vary. This should work for most use-cases, but the following situations could warrant weight changes:

- Your cluster has some relatively slow OSDs or nodes. Lowering their weight can reduce the impact of this bottleneck.
- You are using BlueStore drives provisioned with Rook v0.3.1 or older. In this case, you may notice OSD weights did not get set relative to their storage capacity. Changing the weight can fix this and maximize cluster capacity.

This example sets the weight of `osd.0` which is 600 GiB.

```
cephuser@adm > ceph osd crush reweight osd.0 .600
```

12.1.8.2 OSD primary affinity

When pools are set with a size setting greater than one, data is replicated between nodes and OSDs. For every chunk of data a Primary OSD is selected to be used for reading that data to be sent to clients. You can control how likely it is for an OSD to become a Primary using the Primary Affinity setting. This is similar to the OSD weight setting, except it only affects reads on the storage device, not capacity or writes.

In this example, we will make sure `osd.0` is only selected as Primary if all other OSDs holding replica data are unavailable:

```
cephuser@adm >  osd primary-affinity osd.0 0
```

12.1.9 Removing phantom OSD

If you have OSDs in which are not showing any disks, you can remove those "Phantom OSDs" by following the instructions below. To check for "Phantom OSDs", you can run:

```
cephuser@adm > ceph osd tree
```

An example output looks like this:

```
ID  CLASS WEIGHT   TYPE NAME                STATUS REWEIGHT PRI-AFF
-1        57.38062 root default
-13        7.17258 host node1.example.com
2   hdd    3.61859      osd.2                up     1.00000  1.00000
-7              0  host node2.example.com   down   0        1.00000
```

The host `node2.example.com` in the output has no disks, so it is most likely a "Phantom OSD".

Now to remove it, use the ID in the first column of the output and replace `<ID>` with it. In the example output above the ID would be `-7`. The commands are:

```
cephuser@adm > ceph osd out ID
cephuser@adm > ceph osd crush remove osd.ID
cephuser@adm > ceph auth del osd.ID
cephuser@adm > ceph osd rm ID
```

To recheck that the phantom OSD was removed, re-run the following command and check if the OSD with the ID does not show up anymore:

```
ceph osd tree
```

12.1.10 Changing the failure domain

In Rook, it is now possible to indicate how the default CRUSH failure domain rule must be configured in order to ensure that replicas or erasure code shards are separated across hosts, and a single host failure does not affect availability. For instance, this is an example manifest of a block pool named `replicapool` configured with a `failureDomain` set to `osd`:

```
apiVersion: ceph.rook.io/v1
kind: CephBlockPool
metadata:
  name: replicapool
  namespace: rook
spec:
  # The failure domain will spread the replicas of the data across different failure
 zones
  failureDomain: osd
[...]
```

However, due to several reasons, we may need to change such failure domain to its other value: `host`. Unfortunately, changing it directly in the YAML manifest is not currently handled by Rook, so we need to perform the change directly using Ceph commands using the Rook tools pod, for instance:

```
cephuser@adm > ceph osd pool get replicapool crush_rule
crush_rule: replicapool
cephuser@adm > ceph osd crush rule create-replicated replicapool_host_rule default host
```

Notice that the suffix `host_rule` in the name of the rule is just for clearness about the type of rule we are creating here, and can be anything else as long as it is different from the existing one. Once the new rule has been created, we simply apply it to our block pool:

```
cephuser@adm > ceph osd pool set replicapool crush_rule replicapool_host_rule
```

And validate that it has been actually applied properly:

```
cephuser@adm > ceph osd pool get replicapool crush_rule
crush_rule: replicapool_host_rule
```

If the cluster's health was `HEALTH_OK` when we performed this change, immediately, the new rule is applied to the cluster transparently without service disruption.

Exactly the same approach can be used to change from `host` back to `osd`.

13 Object Storage

13.1 Object Storage

Object Storage exposes an S3 API to the storage cluster for applications to `put` and `get` data.

13.1.1 Configuring the Object Storage

Rook has the ability to either deploy an Object Storage in Kubernetes or to connect to an external Object Gateway service. Most commonly, the Object Storage will be configured locally by Rook.

13.1.1.1 Creating a local Object Storage

The below sample will create a `CephObjectStore` that starts the Object Gateway service in the cluster with an S3 API.

Note

This sample requires at least three BlueStore OSDs, with each OSD located on a different node.

The OSDs must be located on different nodes, because the `failureDomain` is set to `host` and the `erasureCoded` chunk settings require at least three different OSDs (two `dataChunks` + one `codingChunks`).

```
apiVersion: ceph.rook.io/v1
kind: CephObjectStore
metadata:
  name: my-store
  namespace: rook-ceph
spec:
  metadataPool:
    failureDomain: host
    replicated:
      size: 3
  dataPool:
    failureDomain: host
```

```
    erasureCoded:
      dataChunks: 2
      codingChunks: 1
  preservePoolsOnDelete: true
  gateway:
    type: s3
    sslCertificateRef:
    port: 80
    securePort:
    instances: 1
  healthCheck:
    bucket:
      disabled: false
      interval: 60s
```

After the `CephObjectStore` is created, the Rook Operator will then create all the pools and other resources necessary to start the service. This may take a minute to complete.

Create the object store:

```
kubectl@adm > kubectl create -f object.yaml
```

To confirm the object store is configured, wait for the rgw pod to start:

```
kubectl@adm > kubectl -n rook-ceph get pod -l app=rook-ceph-rgw
```

13.1.1.2 Connecting to an external Object Storage

Rook can connect to existing Object Gateway gateways to work in conjunction with the external mode of the CephCluster CRD. If you have an external CephCluster CR, you can instruct Rook to consume external gateways with the following:

```
apiVersion: ceph.rook.io/v1
kind: CephObjectStore
metadata:
  name: external-store
  namespace: rook-ceph
spec:
  gateway:
    port: 8080
    externalRgwEndpoints:
      - ip: 192.168.39.182
  healthCheck:
    bucket:
      enabled: true
```

```
        interval: 60s
```

You can use the existing `object-external.yaml` file. When ready the `ceph-object-controller` will output a message in the Operator log similar to this one:

```
ceph-object-controller: ceph object store gateway service running at 10.100.28.138:8080
```

You can now get and access the store via:

```
kubectl@adm > kubectl -n rook-ceph get svc -l app=rook-ceph-rgw
NAME                     TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
rook-ceph-rgw-my-store   ClusterIP   10.100.28.138     none        8080/TCP   6h59m
```

Any pod from your cluster can now access this endpoint:

```
kubectl@adm > curl 10.100.28.138:8080
```

It is also possible to use the internally registered DNS name:

```
kubectl@adm > curl rook-ceph-rgw-my-store.rook-ceph:8080
```

The DNS name is created with the following schema: `rook-ceph-rgw-$STORE_NAME.$NAMESPACE`.

13.1.2 Creating a bucket

Now that the object store is configured, next we need to create a bucket where a client can read and write objects. A bucket can be created by defining a storage class, similar to the pattern used by block and file storage. First, define the storage class that will allow object clients to create a bucket. The storage class defines the object storage system, the bucket retention policy, and other properties required by the administrator. Save the following as `storageclass-bucket-delete.yaml` (the example is named as such due to the `Delete` reclaim policy).

```
  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-ceph
    region: us-east-1
```

```
kubectl@adm > kubectl create -f storageclass-bucket-delete.yaml
```

Based on this storage class, an object client can now request a bucket by creating an Object Bucket Claim (OBC). When the OBC is created, the Rook-Ceph bucket provisioner will create a new bucket. Notice that the OBC references the storage class that was created above. Save the following as `object-bucket-claim-delete.yaml` (the example is named as such due to the `Delete` reclaim policy):

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

```
kubectl@adm > kubectl create -f object-bucket-claim-delete.yaml
```

Now that the claim is created, the operator will create the bucket as well as generate other artifacts to enable access to the bucket. A secret and ConfigMap are created with the same name as the OBC and in the same namespace. The secret contains credentials used by the application pod to access the bucket. The ConfigMap contains bucket endpoint information and is also consumed by the pod.

13.1.2.1 Client connections

The following commands extract key pieces of information from the secret and configmap:

```
#config-map, secret, OBC will part of default if no specific name space mentioned
export AWS_HOST=$(kubectl -n default get cm ceph-bucket -o yaml | grep BUCKET_HOST | awk
 '{print $2}')
export AWS_ACCESS_KEY_ID=$(kubectl -n default get secret ceph-bucket -o yaml | grep
 AWS_ACCESS_KEY_ID | awk '{print $2}' | base64 --decode)
export AWS_SECRET_ACCESS_KEY=$(kubectl -n default get secret ceph-bucket -o yaml | grep
 AWS_SECRET_ACCESS_KEY | awk '{print $2}' | base64 --decode)
```

13.1.3 Consuming the Object Storage

Now that you have the Object Storage configured and a bucket created, you can consume the object storage from an S3 client.

This section will guide you through testing the connection to the `CephObjectStore` and uploading and downloading from it. Run the following commands after you have connected to the Rook toolbox.

13.1.3.1 Setting environment variables

To simplify the S3 client commands, you will want to set the four environment variables for use by your client (for example, inside the toolbox). See above for retrieving the variables for a bucket created by an `ObjectBucketClaim`.

```
export AWS_HOST=HOST
export AWS_ENDPOINT=ENDPOINT
export AWS_ACCESS_KEY_ID=ACCESS_KEY
export AWS_SECRET_ACCESS_KEY=SECRET_KEY
```

- `Host`: The DNS host name where the Object Gateway service is found in the cluster. Assuming you are using the default `rook-ceph` cluster, it will be `rook-ceph-rgw-my-store.rook-ceph`.
- `Endpoint`: The endpoint where the Object Gateway service is listening. Run the following command and then combine the clusterIP and the port.

  ```
  kubectl@adm > kubectl -n rook-ceph get svc rook-ceph-rgw-my-store
  ```

- `Access key`: The user's `access_key` as printed above
- `Secret key`: The user's `secret_key` as printed above

The variables for the user generated in this example might be:

```
export AWS_HOST=rook-ceph-rgw-my-store.rook-ceph
export AWS_ENDPOINT=10.104.35.31:80
export AWS_ACCESS_KEY_ID=XEZDB3UJ6X7HVBE7X7MA
export AWS_SECRET_ACCESS_KEY=7yGIZON7EhFORz0I40BFniML36D2rl8CQQ5kXU6l
```

The access key and secret key can be retrieved as described in the section above on *Section 13.1.2.1, "Client connections"* or below in the section *Section 13.1.5, "Creating a user"* if you are not creating the buckets with an `ObjectBucketClaim`.

13.1.3.2 Installing the `s3cmd` package

To test the `CephObjectStore` we will install the `s3cmd` tool into the toolbox pod.

```
zypper --assumeyes install s3cmd
```

13.1.3.3 PUT or GET an object

Upload a file to the newly created bucket:

```
echo "Hello Rook" > /tmp/rookObj
s3cmd put /tmp/rookObj --no-ssl --host=${AWS_HOST} --host-bucket=  s3://rookbucket
```

Download and verify the file from the bucket:

```
s3cmd get s3://rookbucket/rookObj /tmp/rookObj-download --no-ssl --host=${AWS_HOST} --
host-bucket=
cat /tmp/rookObj-download
```

13.1.4 Setting up external access to the cluster

Rook sets up the object storage so pods will have access internal to the cluster. If your applications are running outside the cluster, you will need to setup an external service through a `NodePort`.

First, note the service that exposes RGW internal to the cluster. We will leave this service intact and create a new service for external access.

```
kubectl@adm > kubectl -n rook-ceph get service rook-ceph-rgw-my-store
NAME                     CLUSTER-IP   EXTERNAL-IP   PORT(S)     AGE
rook-ceph-rgw-my-store   10.3.0.177   none        80/TCP      2m
```

Save the external service as `rgw-external.yaml`:

```
  apiVersion: v1
  kind: Service
  metadata:
    name: rook-ceph-rgw-my-store-external
    namespace: rook-ceph
    labels:
      app: rook-ceph-rgw
      rook_cluster: rook-ceph
      rook_object_store: my-store
  spec:
    ports:
    - name: rgw
      port: 80
      protocol: TCP
```

```
    targetPort: 80
  selector:
    app: rook-ceph-rgw
    rook_cluster: rook-ceph
    rook_object_store: my-store
  sessionAffinity: None
  type: NodePort
```

Now, create the external service:

```
kubectl@adm > kubectl create -f rgw-external.yaml
```

See both Object Gateway services running and notice what port the external service is running on:

```
kubectl@adm > kubectl -n rook-ceph get service rook-ceph-rgw-my-store rook-ceph-rgw-my-
store-external
NAME                              TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
   AGE
rook-ceph-rgw-my-store            ClusterIP   10.104.82.228    none          80/TCP
   4m
rook-ceph-rgw-my-store-external   NodePort    10.111.113.237   none          80:31536/TCP
   39s
```

Internally the Object Gateway service is running on port 80. The external port in this case is 31536.

13.1.5 Creating a user

If you need to create an independent set of user credentials to access the S3 endpoint, create a `CephObjectStoreUser`. The user will be used to connect to the Object Gateway service in the cluster using the S3 API. The user will be independent of any object bucket claims that you might have created in the earlier instructions in this document.

```
apiVersion: ceph.rook.io/v1
kind: CephObjectStoreUser
metadata:
  name: my-user
  namespace: rook-ceph
spec:
  store: my-store
  displayName: "my display name"
```

When the `CephObjectStoreUser` is created, the Rook operator will then create the RGW user on the specified `CephObjectStore` and store the Access Key and Secret Key in a kubernetes secret in the same namespace as the `CephObjectStoreUser`.

Create the object store user:

```
kubectl@adm > kubectl create -f object-user.yaml
```

To confirm the object store user is configured, describe the secret:

```
kubectl@adm > kubectl -n rook-ceph describe secret rook-ceph-object-user-my-store-my-user
  Name:  rook-ceph-object-user-my-store-my-user
  Namespace: rook-ceph
  Labels:   app=rook-ceph-rgw
            rook_cluster=rook-ceph
            rook_object_store=my-store
  Annotations: none

  Type: kubernetes.io/rook

  Data
  ====
  AccessKey: 20 bytes
  SecretKey: 40 bytes
```

The `AccessKey` and `SecretKey` data fields can be mounted in a pod as an environment variable.

To directly retrieve the secrets:

```
kubectl@adm > kubectl -n rook-ceph get secret rook-ceph-object-user-my-store-my-user -o
 yaml \
 | grep AccessKey | awk '{print $2}' | base64 --decode
kubectl@adm > kubectl -n rook-ceph get secret rook-ceph-object-user-my-store-my-user -o
 yaml \
 | grep SecretKey | awk '{print $2}' | base64 --decode
```

13.2 Ceph Object Storage CRD

Rook allows creation and customization of object stores through the custom resource definitions (CRDs). The following settings are available for Ceph Object Storage.

13.2.1 Sample

13.2.1.1 Erasure code

Erasure coded pools require the OSDs to use `bluestore` for the configured `storeType`. Additionally, erasure coded pools can only be used with `dataPools`. The `metadataPool` must use a replicated pool.

Note

This sample requires at least three BlueStore OSDs, with each OSD located on a different node.

The OSDs must be located on different nodes, because the `failureDomain` is set to `host` and the `erasureCoded` chunk settings require at least three different OSDs (two `dataChunks` + one `codingChunks`).

```
apiVersion: ceph.rook.io/v1
kind: CephObjectStore
metadata:
  name: my-store
  namespace: rook-ceph
spec:
  metadataPool:
    failureDomain: host
    replicated:
      size: 3
  dataPool:
    failureDomain: host
    erasureCoded:
      dataChunks: 2
      codingChunks: 1
  preservePoolsOnDelete: true
  gateway:
    type: s3
    sslCertificateRef:
    port: 80
    securePort:
    instances: 1
    # A key/value list of annotations
    annotations:
    #  key: value
```

```
    placement:
    #  nodeAffinity:
    #    requiredDuringSchedulingIgnoredDuringExecution:
    #      nodeSelectorTerms:
    #      - matchExpressions:
    #        - key: role
    #          operator: In
    #          values:
    #          - rgw-node
    #  tolerations:
    #  - key: rgw-node
    #    operator: Exists
    #  podAffinity:
    #  podAntiAffinity:
    #  topologySpreadConstraints:
    resources:
    #  limits:
    #    cpu: "500m"
    #    memory: "1024Mi"
    #  requests:
    #    cpu: "500m"
    #    memory: "1024Mi"
  #zone:
    #name: zone-a
```

13.2.2 Object store settings

13.2.2.1 Metadata

- `name`: The name of the object store to create, which will be reflected in the pool and other resource names.
- `namespace`: The namespace of the Rook cluster where the object store is created.

13.2.2.2 Pools

The pools allow all of the settings defined in the pool CRD specification. In the example above, there must be at least three hosts (size 3) and at least three devices (two data + one coding chunks) in the cluster.

When the `zone` section is set, pools with the object store's name will not be created, since the object-store will the using the pools created by the ceph-object-zone.

- `metadataPool`: The settings used to create all of the object store metadata pools. Must use replication.
- `dataPool`: The settings to create the object store data pool. Can use replication or erasure coding.
- `preservePoolsOnDelete`: If it is set to "true", the pools used to support the object store will remain when the object store will be deleted. This is a security measure to avoid accidental loss of data. It is set to "false" by default. If it is not specified, this is also deemed as "false".

13.2.3 Creating gateway settings

The gateway settings correspond to the Object Gateway daemon settings.

- `type`: `S3` is supported
- `sslCertificateRef`: If the certificate is not specified, SSL will not be configured. If specified, this is the name of the Kubernetes secret that contains the SSL certificate to be used for secure connections to the object store. Rook will look in the secret provided at the `cert` key name. The value of the `cert` key must be in the format expected by the Object Gateway service: "The server key, server certificate, and any other CA or intermediate certificates be supplied in one file. Each of these items must be in pem form."
- `port`: The port on which the Object service will be reachable. If host networking is enabled, the Object Gateway daemons will also listen on that port. If running on SDN, the Object Gateway daemon listening port will be 8080 internally.
- `securePort`: The secure port on which Object Gateway pods will be listening. An SSL certificate must be specified.
- `instances`: The number of pods that will be started to load-balance this object store.
- `externalRgwEndpoints`: A list of IP addresses to connect to external existing Object Gateways (works with external mode). This setting will be ignored if the `CephCluster` does not have `external` spec enabled.
- `annotations`: Key-value pair list of annotations to add.

- `labels`: Key-value pair list of labels to add.
- `placement`: The Kubernetes placement settings to determine where the Object Gateway pods should be started in the cluster.
- `resources`: Set resource requests/limits for the Gateway Pod(s).
- `priorityClassName`: Set priority class name for the Gateway Pod(s).

Example of external Object Gateway endpoints to connect to:

```
gateway:
  port: 80
  externalRgwEndpoints:
    - ip: 192.168.39.182
```

This will create a service with the endpoint `192.168.39.182` on port `80`, pointing to the Ceph object external gateway. All the other settings from the gateway section will be ignored, except for `securePort`.

13.2.4 Zone settings

The zone (https://github.com/rook/rook/blob/master/Documentation/ceph-object-multisite.md) settings allow the object store to join custom created ceph-object-zone (https://github.com/rook/rook/blob/master/Documentation/ceph-object-multisite-crd.md).

- `name`: the name of the `ceph-object-zone` the object store will be in.

13.2.5 Runtime settings

13.2.5.1 MIME types

Rook provides a default `mime.types` file for each Ceph Object Storage. This file is stored in a Kubernetes ConfigMap with the name `rook-ceph-rgw-<STORE-NAME>-mime-types`. For most users, the default file should suffice, however, the option is available to users to edit the `mime.types` file in the ConfigMap as they desire. Users may have their own special file types, and particularly security conscious users may wish to pare down the file to reduce the possibility of a file type execution attack.

Rook will not overwrite an existing `mime.types` ConfigMap so that user modifications will not be destroyed. If the object store is destroyed and re-created, the ConfigMap will also be destroyed and re-created.

13.2.6 Health settings

Rook-Ceph will be default monitor the state of the object store endpoints. The following CRD settings are available:

- `healthCheck`: main object store health monitoring section

For example:

```
healthCheck:
  bucket:
    disabled: false
    interval: 60s
```

The endpoint health check procedure is the following:

1. Create an S3 user.
2. Create a bucket with that user.
3. PUT the file in the object store.
4. GET the file from the object store.
5. Verify object consistency.
6. Update CR health status check.

Rook-Ceph always keeps the bucket and the user for the health check; it just does a PUT and GET of an S3 object, since creating a bucket is an expensive operation.

13.3 Ceph object bucket claim

Rook supports the creation of new buckets and access to existing buckets via two custom resources:

- An `Object Bucket Claim (OBC)` is custom resource which requests a bucket (new or existing) and is described by a Custom Resource Definition (CRD) shown below.
- An `Object Bucket (OB)` is a custom resource automatically generated when a bucket is provisioned. It is a global resource, typically not visible to non-admin users, and contains information specific to the bucket. It is described by an OB CRD, also shown below.

An OBC references a storage class which is created by an administrator. The storage class defines whether the bucket requested is a new bucket or an existing bucket. It also defines the bucket retention policy. Users request a new or existing bucket by creating an OBC which is shown below. The ceph provisioner detects the OBC and creates a new bucket or grants access to an existing bucket, depending the the storage class referenced in the OBC. It also generates a Secret which provides credentials to access the bucket, and a ConfigMap which contains the bucket's endpoint. Application pods consume the information in the Secret and ConfigMap to access the bucket. Please note that to make provisioner watch the cluster namespace only you need to set `ROOK_OBC_WATCH_OPERATOR_NAMESPACE` to `true` in the operator manifest, otherwise it watches all namespaces.

13.3.1 Sample

13.3.1.1 OBC custom resource

```
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: ceph-bucket [1]
  namespace: rook-ceph [2]
spec:
  bucketName: [3]
  generateBucketName: photo-booth [4]
  storageClassName: rook-ceph-bucket [4]
  additionalConfig: [5]
    maxObjects: "1000"
```

```
    maxSize: "2G"
```

1. `name` of the `ObjectBucketClaim`. This name becomes the name of the Secret and ConfigMap.
2. `namespace` (optional) of the `ObjectBucketClaim`, which is also the namespace of the ConfigMap and Secret.
3. `bucketName` name of the `bucket`. **Not** recommended for new buckets, since names must be unique within an entire object store.
4. `generateBucketName` value becomes the prefix for a randomly-generated name; if supplied, then `bucketName` must be empty. If both `bucketName` and `generateBucketName` are supplied, then `BucketName` has precedence and `GenerateBucketName` is ignored. If both `bucketName` and `generateBucketName` are blank or omitted, then the storage class is expected to contain the name of an *existing* bucket. It is an error if all three bucket-related names are blank or omitted.
5. `storageClassName` which defines the StorageClass which contains the names of the bucket provisioner, the object store, and specifies the bucket-retention policy.
6. `additionalConfig` is an optional list of key-value pairs used to define attributes specific to the bucket being provisioned by this OBC. This information is typically tuned to a particular bucket provisioner, and may limit application portability. Options supported:
    - `maxObjects`: The maximum number of objects in the bucket
    - `maxSize`: The maximum size of the bucket, please note minimum recommended value is 4K.

13.3.1.2 OBC custom resource after bucket provisioning

```
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  creationTimestamp: "2019-10-18T09:54:01Z"
  generation: 2
  name: ceph-bucket
  namespace: default [1]
  resourceVersion: "559491"
spec:
```

```
  ObjectBucketName: obc-default-ceph-bucket [2]
  additionalConfig: null
  bucketName: photo-booth-c1178d61-1517-431f-8408-ec4c9fa50bee [3]
  cannedBucketAcl: ""
  ssl: false
  storageClassName: rook-ceph-bucket [4]
  versioned: false
status:
  Phase: bound [5]
```

1. `namespace` where OBC got created.
2. `ObjectBucketName` generated OB name created using name space and OBC name.
3. The generated (in this case), unique `bucket name` for the new bucket.
4. Name of the storage class from OBC got created.
5. Phases of bucket creation:
    - *Pending*: the operator is processing the request.
    - *Bound*: the operator finished processing the request and linked the OBC and OB
    - *Released*: the OB has been deleted, leaving the OBC unclaimed but unavailable.
    - *Failed*: not currently set.

13.3.1.3 App pod

```
apiVersion: v1
kind: Pod
metadata:
  name: app-pod
  namespace: dev-user
spec:
  containers:
  - name: mycontainer
    image: redis
    envFrom: [1]
    - configMapRef:
        name: ceph-bucket [2]
    - secretRef:
        name: ceph-bucket [3]
```

1. Use `env:` if mapping of the defined key names to the environment-variable names used by the app is needed.
2. Makes available to the pod as environment variables: `BUCKET_HOST`, `BUCKET_PORT`, `BUCKET_NAME`
3. makes available to the pod as environment variables: `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`

13.3.1.4 StorageClass

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: rook-ceph-bucket
  labels:
    aws-s3/object [1]
provisioner: rook-ceph.ceph.rook.io/bucket [2]
parameters: [3]
  objectStoreName: my-store
  objectStoreNamespace: rook-ceph
  region: us-west-1
  bucketName: ceph-bucket [4]
reclaimPolicy: Delete [5]
```

1. `label` (optional) here associates this `StorageClass` to a specific provisioner.
2. `provisioner` responsible for handling `OBCs` referencing this `StorageClass`.
3. **all** `parameter` required.
4. `bucketName` is required for access to existing buckets but is omitted when provisioning new buckets. Unlike greenfield provisioning, the brownfield bucket name appears in the `StorageClass`, not the `OBC`.
5. Rook-Ceph provisioner decides how to treat the `reclaimPolicy` when an `OBC` is deleted for the bucket.

- *Delete* = physically delete the bucket.
- *Retain* = do not physically delete the bucket.

13.4 Ceph Object Storage user custom resource definitions (CRD)

Rook allows creation and customization of object store users through the custom resource definitions (CRDs). The following settings are available for Ceph object store users.

13.4.1 Sample

```
apiVersion: ceph.rook.io/v1
kind: CephObjectStoreUser
metadata:
  name: my-user
  namespace: rook-ceph
spec:
  store: my-store
  displayName: my-display-name
```

13.4.2 Object Storage user settings

13.4.2.1 Metadata

- `name`: The name of the object store user to create, which will be reflected in the secret and other resource names.
- `namespace`: The namespace of the Rook cluster where the object store user is created.

13.4.2.2 Specification

- `store`: The object store in which the user will be created. This matches the name of the Object Storage CRD.
- `displayName`: The display name which will be passed to the `radosgw-admin user create` command.

14 Ceph Dashboard

14.1 Ceph Dashboard

The Ceph Dashboard is a helpful tool to give you an overview of the status of your Ceph cluster, including overall health, status of the MOPN quorum, status of the MGR, OSD, and other Ceph daemons, view pools and PG status, show logs for the daemons, and more. Rook makes it simple to enable the dashboard.

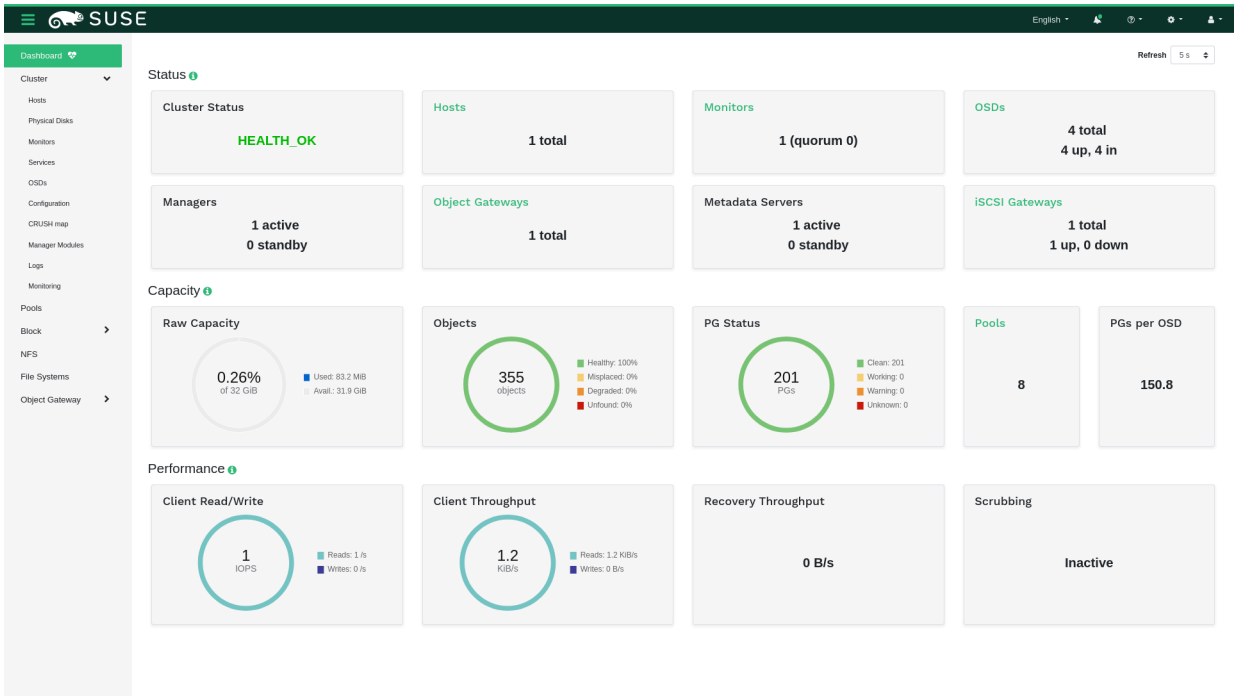

FIGURE 14.1: THE CEPH DASHBOARD

14.1.1 Enabling the Ceph Dashboard

The dashboard (http://docs.ceph.com/docs/mimic/mgr/dashboard/) can be enabled with settings in the CephCluster CRD. The CephCluster CRD must have the dashboard `enabled` setting set to `true`. This is the default setting in the example manifests.

```
spec:
  dashboard:
    enabled: true
```

The Rook operator will enable the `ceph-mgr` dashboard module. A service object will be created to expose that port inside the Kubernetes cluster. Rook will enable port 8443 for HTTPS access. This example shows that port 8443 was configured:

```
kubectl@adm > kubectl -n rook-ceph get service
NAME                         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
 AGE
rook-ceph-mgr                ClusterIP   10.108.111.192   <none>        9283/TCP
 3h
rook-ceph-mgr-dashboard      ClusterIP   10.110.113.240   <none>        8443/TCP
 3h
```

The first service is for reporting the Prometheus metrics, while the latter service is for the dashboard. If you are on a node in the cluster, you will be able to connect to the dashboard by using either the DNS name of the service at `https://rook-ceph-mgr-dashboard-https:8443` or by connecting to the cluster IP, in this example at `https://10.110.113.240:8443`.

Important

The dashboard will only be enabled for the first Ceph object store created by Rook.

14.1.1.1 Creating login credentials

After you connect to the dashboard, you will need to login for secure access. Rook creates a default user named `admin` and generates a secret called `rook-ceph-dashboard-admin-password` in the namespace where the Rook-Ceph cluster is running. To retrieve the generated password, you can run the following:

```
kubectl@adm > kubectl -n rook-ceph get secret rook-ceph-dashboard-password \
 -o jsonpath="{['data']['password']}" | base64 --decode && echo
```

14.1.2 Configuring the Ceph Dashboard

The following dashboard configuration settings are supported:

```
  spec:
    dashboard:
      urlPrefix: /ceph-dashboard
      port: 8443
```

```
    ssl: true
```

- `urlPrefix` If you are accessing the dashboard via a reverse proxy, you may wish to serve it under a URL prefix. To get the dashboard to use hyperlinks that include your prefix, you can set the `urlPrefix` setting.
- `port` The port that the dashboard is served on may be changed from the default using the `port` setting. The corresponding K8s service exposing the port will automatically be updated.
- `ssl` The dashboard may be served without SSL by setting the `ssl` option to false.

14.1.3 Viewing the Ceph Dashboard external to the cluster

Commonly, you will want to view the dashboard from outside the cluster. For example, on a development machine with the cluster running inside minikube, you will want to access the dashboard from the host.

There are several ways to expose a service, which will depend on the environment you are running in. You can use an Ingress Controller or other methods for exposing services such as NodePort, LoadBalancer, or ExternalIPs.

14.1.3.1 Node port

The simplest way to expose the service in minikube or similar environments is using the NodePort to open a port on the VM that can be accessed by the host. To create a service with the NodePort, save this YAML file as `dashboard-external-https.yaml`.

```
apiVersion: v1
kind: Service
metadata:
  name: rook-ceph-mgr-dashboard-external-https
  namespace: rook-ceph
  labels:
    app: rook-ceph-mgr
    rook_cluster: rook-ceph
spec:
  ports:
  - name: dashboard
    port: 8443
    protocol: TCP
    targetPort: 8443
```

```
    selector:
      app: rook-ceph-mgr
      rook_cluster: rook-ceph
    sessionAffinity: None
    type: NodePort
```

Now create the service:

```
kubectl@adm > kubectl create -f dashboard-external-https.yaml
```

You will see the new service `rook-ceph-mgr-dashboard-external-https` created:

```
kubectl@adm > kubectl -n rook-ceph get service
NAME                                  TYPE        CLUSTER-IP       EXTERNAL-IP
 PORT(S)          AGE
rook-ceph-mgr                         ClusterIP   10.108.111.192   <none>        9283/
TCP         4h
rook-ceph-mgr-dashboard               ClusterIP   10.110.113.240   <none>        8443/
TCP         4h
rook-ceph-mgr-dashboard-external-https  NodePort    10.101.209.6     <none>
 8443:31176/TCP   4h
```

In this example, port `31176` will be opened to expose port `8443` from the ceph-mgr pod. Find the IP address of the VM. If using minikube, you can run `minikube ip` to find the IP address. Now you can enter the URL in your browser such as `https://192.168.99.110:31176` and the dashboard will appear.

14.1.3.2 Creating the load balancer service

If you have a cluster on a cloud provider that supports load balancers, you can create a service that is provisioned with a public hostname. The yaml is the same as `dashboard-external-https.yaml` except for the following property:

```
  spec:
  [...]
    type: LoadBalancer
```

Now create the service:

```
kubectl@adm > kubectl create -f dashboard-loadbalancer.yaml
```

You will see the new service `rook-ceph-mgr-dashboard-loadbalancer` created:

```
kubectl@adm > kubectl -n rook-ceph get service
```

```
NAME                                    TYPE           CLUSTER-IP       EXTERNAL-IP
                                                         PORT(S)             AGE
rook-ceph-mgr                           ClusterIP      172.30.11.40     <none>
                                                         9283/TCP         4h
rook-ceph-mgr-dashboard                 ClusterIP      172.30.203.185   <none>
                                                         8443/TCP         4h
rook-ceph-mgr-dashboard-loadbalancer    LoadBalancer   172.30.27.242
 a7f23e8e2839511e9b7a5122b08f2038-1251669398.us-east-1.elb.amazonaws.com   8443:32747/TCP
      4h
```

Now you can enter the URL in your browser such as `https://a7f23e8e2839511e9b7a5122b08f2038-1251669398.us-east-1.elb.amazonaws.com:8443` and the dashboard will appear.

14.1.3.3 Ingress controller

If you have a cluster with an Nginx Ingress Controller and a certificate manager, then you can create an ingress like the one below. This example achieves four things:

1. Exposes the dashboard on the Internet (using an reverse proxy).
2. Issues an valid TLS Certificate for the specified domain name.
3. Tells the reverse proxy that the dashboard itself uses HTTPS.
4. Tells the reverse proxy that the dashboard itself does not have a valid certificate (it is self-signed).

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: rook-ceph-mgr-dashboard
  namespace: rook-ceph
  annotations:
    kubernetes.io/ingress.class: "nginx"
    kubernetes.io/tls-acme: "true"
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
    nginx.ingress.kubernetes.io/server-snippet: |
      proxy_ssl_verify off;
spec:
  tls:
   - hosts:
     - rook-ceph.example.com
     secretName: rook-ceph.example.com
  rules:
  - host: rook-ceph.example.com
```

```
      http:
        paths:
        - path: /
          backend:
            serviceName: rook-ceph-mgr-dashboard
            servicePort: https-dashboard
```

Customise the ingress resource to match your cluster. Replace the example domain name `rook-ceph.example.com` with a domain name that will resolve to your Ingress Controller (creating the DNS entry if required).

Now create the ingress:

```
kubectl@adm > kubectl create -f dashboard-ingress-https.yaml
```

You will see the new ingress `rook-ceph-mgr-dashboard` created:

```
kubectl@adm > kubectl -n rook-ceph get ingress
NAME                      HOSTS                     ADDRESS   PORTS     AGE
rook-ceph-mgr-dashboard   rook-ceph.example.com     80, 443   5m
```

And the new secret for the TLS certificate:

```
kubectl@adm > kubectl -n rook-ceph get secret rook-ceph.example.com
NAME                       TYPE                DATA      AGE
rook-ceph.example.com      kubernetes.io/tls   2         4m
```

You can now browse to `https://rook-ceph.example.com/` to log into the dashboard.

III Troubleshooting Ceph on SUSE CaaS Platform

15 Troubleshooting

15.1 Debugging Rook

There are a number of basic actions a user might need to take during debugging. These actions are defined here for reference when they are mentioned in more documentation below.

Important

This document is not devoted to an in-depth explanation of what Kubernetes is, what its features are, how it is used, how to navigate it, or how to debug applications that run on it. This document will use Kubernetes terms, and users are expected to know how to look up Kubernetes information they do not already have. This document will give an outline of how to use Kubernetes tools to get any information needed in the Rook-Ceph context and, when relevant, will briefly explain how Rook uses Kubernetes features.

15.1.1 Setting the operator log level to debug

In general, the first place to look when encountering a failure is to get logs for the `rook-ceph-operator` pod. To get the most informative logs possible, set the operator log level to `DEBUG`. To do this, modify Helm's `values.yaml` or modify the `operator.yaml` manifest. Regardless of the method chosen, the log level can always be set by editing the deployment directly with **`kubectl`**. For example:

```
kubectl@adm > kubectl --namespace rook-ceph set env deployment/rook-ceph-operator
 ROOK_LOG_LEVEL=DEBUG
```

After editing the deployment, the operator pod will restart automatically and will start outputting logs with the new log level.

Note

If you are experiencing a particular failure, it may take some time for the Rook operator to reach the failure location again to report debug logs.

15.1.2 Using the toolbox pod

Use the Rook toolbox pod to interface directly with the Ceph cluster via the CLI. For example:

```
kubectl@adm > kubectl --namespace rook-ceph exec -it deploy/rook-ceph-tools -- bash
```

If the `rook-ceph-tools` deployment does not exist, it should be created using the `toolbox.yaml` manifest.

Note

To set log levels for Ceph daemons, it is advised to use the Ceph CLI from the `toolbox` pod.

15.1.3 Using the SES supportutils plugin

The `supportutils` plugin for SUSE Enterprise Storage works with Rook clusters. It is installed by the `supportutils-plugin-ses` package. The plugin collects container logs and more information about a Rook-Ceph cluster, making collection of logs easy. Once the logs are collected, you can browse the collected information and logs without needing to progressively collect more detailed information at each step.

The `supportutils` plugin does not alter the Rook log level to `DEBUG`, and it is advised to set this to `DEBUG` before running the plugin. The plugin also does not change any Ceph log levels; also consider changing those if the failure merits it before running the plugin.

16 Common issues

16.1 Ceph common issues

Many of these problem cases are hard to summarize down to a short phrase that adequately describes the problem. Each problem will start with a bulleted list of symptoms. Keep in mind that all symptoms may not apply, depending on the configuration of Rook. If the majority of the symptoms are seen, then there is a fair chance that you are experiencing that problem.

16.1.1 Troubleshooting techniques

There are two main categories of information you will need to investigate issues in the cluster:

1. Kubernetes status and logs.
2. Ceph cluster status.

16.1.1.1 Running Ceph tools

After you verify the basic health of the running pods, next you will want to run Ceph tools for status of the storage components. There are two ways to run the Ceph tools, either in the Rook toolbox or inside other Rook pods that are already running.

- Logs on a specific node to find why a PVC is failing to mount: Rook agent errors around the attach and detach:

```
kubectl@adm > kubectl logs -n rook-ceph rook-ceph-agent-pod
```

- See the *Section 12.1.3, "Collecting logs"* for a script that will help you gather the logs.
- Other artifacts:
  - The monitors that are expected to be in quorum:

```
kubectl@adm > kubectl -n <cluster-namespace> get configmap rook-ceph-mon-
endpoints -o yaml | grep data
```

16.1.1.1.1 Using tools in the Rook toolbox

The `rook-ceph-tools pod` provides a simple environment to run Ceph tools. Once the pod is up and running, connect to the pod to execute Ceph commands to evaluate that current state of the cluster.

```
kubectl@adm > kubectl -n rook-ceph exec -it $(kubectl -n rook-ceph get pod -l "app=rook-
ceph-tools" -o jsonpath='{.items[0].metadata.name}') bash
```

16.1.1.1.2 Ceph commands

Here are some common commands to troubleshoot a Ceph cluster:

- **`ceph status`**
- **`ceph osd status`**
- **`ceph osd df`**
- **`ceph osd utilization`**
- **`ceph osd pool stats`**
- **`ceph osd tree`**
- **`ceph pg stat`**

The first two status commands provide the overall cluster health. The normal state for cluster operations is `HEALTH_OK`, but will still function when the state is in a `HEALTH_WARN` state. If you are in a `WARN` state, then the cluster is in a condition that it may enter the `HEALTH_ERROR` state at which point *all* disk I/O operations are halted. If a `HEALTH_WARN` state is observed, then one should take action to prevent the cluster from halting when it enters the `HEALTH_ERROR` state.

16.1.2 Cluster failing to service requests

16.1.2.1 Identifying symptoms

- Execution of the Ceph command hangs.
- `PersistentVolumes` are not being created.
- Large amount of slow requests are blocking.
- Large amount of stuck requests are blocking.
- One or more MONs are restarting periodically.

16.1.2.2 Investigating the current state of Ceph

Create a `rook-ceph-tools pod` to investigate the current state of Ceph. The following is an example of the output. In this case, the **`ceph status`** command would just hang and the process would need to be killed.

```
kubectl@adm > kubectl -n rook-ceph exec -it $(kubectl -n rook-ceph get pod -l "app=rook-
ceph-tools" -o jsonpath='{.items[0].metadata.name}') bash
cephuser@adm > ceph status
^CCluster connection interrupted or timed out
```

Another indication is when one or more of the MON pods restart frequently. Note the “mon107” that has only been up for 16 minutes in the following output.

```
kubectl@adm > kubectl -n rook-ceph get all -o wide --show-all
  NAME                                READY     STATUS    RESTARTS   AGE       IP
        NODE
  po/rook-ceph-mgr0-2487684371-gzlbq   1/1       Running   0          17h
 192.168.224.46   k8-host-0402
  po/rook-ceph-mon107-p74rj            1/1       Running   0          16m
 192.168.224.28   k8-host-0402
```

```
 rook-ceph-mon1-56fgm                 1/1       Running   0          2d
192.168.91.135   k8-host-0404
 rook-ceph-mon2-rlxcd                 1/1       Running   0          2d
192.168.123.33   k8-host-0403
 rook-ceph-osd-bg2vj                  1/1       Running   0          2d
192.168.91.177   k8-host-0404
 rook-ceph-osd-mwxdm                  1/1       Running   0          2d
192.168.123.31   k8-host-0403
```

16.1.2.3 Identifying the solution

What is happening here is that the MON pods are restarting and one or more of the Ceph daemons are not getting configured with the proper cluster information. This is commonly the result of not specifying a value for `dataDirHostPath` in your Cluster CRD.

The `dataDirHostPath` setting specifies a path on the local host for the Ceph daemons to store configuration and data. Setting this to a path like `/var/lib/rook`, reapplying your cluster CRD and restarting all the Ceph daemons (MON, MGR, OSD, RGW) should solve this problem. After the Object Gateway daemons have been restarted, it is advisable to restart the `rook-tools` pod.

16.1.3 Monitors are the only PODs running

16.1.3.1 Identifying symptoms

- Rook operator is running.
- Either a single mon starts or the MONs skip letters, specifically named `a`, `d`, and `f`.
- No MGR, OSD, or other daemons are created.

16.1.3.2 Investigating MON health

When the operator is starting a cluster, the operator will start one MON at a time and check that they are healthy before continuing to bring up all three MONs. If the first MON is not detected healthy, the operator will continue to check until it is healthy. If the first MON fails to start, a second and then a third MON may attempt to start. However, they will never form a quorum, and orchestration will be blocked from proceeding.

The likely causes for the MON health not being detected:

- The operator pod does not have network connectivity to the MON pod.
- The MON pod is failing to start.
- One or more MON pods are in running state, but are not able to form a quorum.

16.1.3.2.1 Failing to connect to the MON

Firstly, look at the logs of the operator to confirm if it is able to connect to the MONs.

```
kubectl@adm > kubectl -n rook-ceph logs -l app=rook-ceph-operator
```

Likely you will see an error similar to the following that the operator is timing out when connecting to the MON. The last command is **`ceph mon_status`**, followed by a timeout message five minutes later.

```
  2018-01-21 21:47:32.375833 I | exec: Running command: ceph mon_status --cluster=rook
 --conf=/var/lib/rook/rook-ceph/rook.config --keyring=/var/lib/rook/rook-ceph/
client.admin.keyring --format json --out-file /tmp/442263890
  2018-01-21 21:52:35.370533 I | exec: 2018-01-21 21:52:35.071462 7f96a3b82700  0
 monclient(hunting): authenticate timed out after 300
  2018-01-21 21:52:35.071462 7f96a3b82700  0 monclient(hunting): authenticate timed out
 after 300
  2018-01-21 21:52:35.071524 7f96a3b82700  0 librados: client.admin authentication error
 (110) Connection timed out
  2018-01-21 21:52:35.071524 7f96a3b82700  0 librados: client.admin authentication error
 (110) Connection timed out
  [errno 110] error connecting to the cluster
```

The error would appear to be an authentication error, but it is misleading. The real issue is a timeout.

16.1.3.2.2 Identifying the solution

If you see the timeout in the operator log, verify if the MON pod is running (see the next section). If the MON pod is running, check the network connectivity between the operator pod and the MON pod. A common issue is that the CNI is not configured correctly.

16.1.3.2.3 Failing MON pod

We need to verify if the MON pod started successfully.

```
kubectl@adm > kubectl -n rook-ceph get pod -l app=rook-ceph-mon
NAME                                READY     STATUS              RESTARTS   AGE
rook-ceph-mon-a-69fb9c78cd-58szd    1/1       CrashLoopBackOff    2          47s
```

If the MON pod is failing as in this example, you will need to look at the **`mon pod status`** or logs to determine the cause. If the pod is in a crash loop backoff state, you should see the reason by describing the pod.

The pod shows a termination status that the keyring does not match the existing keyring.

```
kubectl@adm > kubectl -n rook-ceph describe pod -l mon=rook-ceph-mon0
[...]
Last State:    Terminated
Reason:    Error
Message:    The keyring does not match the existing keyring in /var/lib/rook/rook-ceph-
mon0/data/keyring.
You may need to delete the contents of dataDirHostPath on the host from a previous
 deployment.
[...]
```

See the solution in the next section regarding cleaning up the `dataDirHostPath` on the nodes.

If you see the three mons running with the names `a`, `d`, and `f`, they likely did not form quorum even though they are running.

```
NAME                               READY   STATUS    RESTARTS   AGE
rook-ceph-mon-a-7d9fd97d9b-cdq7g   1/1     Running   0          10m
rook-ceph-mon-d-77df8454bd-r5jwr   1/1     Running   0          9m2s
rook-ceph-mon-f-58b4f8d9c7-89lgs   1/1     Running   0          7m38s
```

16.1.3.2.4 Identifying the solution

This is a common problem reinitializing the Rook cluster when the local directory used for persistence has **not** been purged. This directory is the `dataDirHostPath` setting in the cluster CRD, and is typically set to `/var/lib/rook`. To fix the issue, you will need to delete all components of Rook and then delete the contents of `/var/lib/rook` (or the directory specified by `dataDirHostPath`) on each of the hosts in the cluster. Then, when the cluster CRD is applied to start a new cluster, the rook-operator should start all the pods as expected.

Important

Deleting the `dataDirHostPath` folder is destructive to the storage. Only delete the folder if you are trying to permanently purge the Rook cluster.

16.1.4 PVCs stay in pending state

16.1.4.1 Identifying symptoms

When you create a PVC based on a Rook storage class, it stays pending indefinitely.

For the Wordpress example, you might see two PVCs in the pending state.

```
kubectl@adm > kubectl get pvc
NAME             STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS      AGE
mysql-pv-claim   Pending                                      rook-ceph-block   8s
wp-pv-claim      Pending                                      rook-ceph-block   16s
```

16.1.4.2 Investigating common causes

There are two common causes for the PVCs staying in the pending state:

1. There are no OSDs in the cluster.
2. The CSI provisioner pod is not running or is not responding to the request to provision the storage.

16.1.4.2.1 Confirming if there are OSDs

To confirm if you have OSDs in your cluster, connect to the Rook Toolbox and run the **`ceph status`** command. You should see that you have at least one OSD `up` and `in`. The minimum number of OSDs required depends on the `replicated.size` setting in the pool created for the storage class. In a "test" cluster, only one OSD is required (see `storageclass-test.yaml`). In the production storage class example (`storageclass.yaml`), three OSDs would be required.

```
cephuser@adm > ceph status
  cluster:
  id:     a0452c76-30d9-4c1a-a948-5d8405f19a7c
  health: HEALTH_OK

  services:
  mon: 3 daemons, quorum a,b,c (age 11m)
  mgr: a(active, since 10m)
  osd: 1 osds: 1 up (since 46s), 1 in (since 109m)
```

16.1.4.2.2 Preparing OSD logs

If you do not see the expected number of OSDs, investigate why they were not created. On each node where Rook looks for OSDs to configure, you will see an “osd prepare” pod.

```
kubectl@adm > kubectl -n rook-ceph get pod -l app=rook-ceph-osd-prepare
NAME                                ...  READY   STATUS      RESTARTS   AGE
rook-ceph-osd-prepare-minikube-9twvk   0/2     Completed   0          30m
```

See the section on *Section 16.1.6, “OSD pods are not created on my devices”* to investigate the logs.

16.1.4.2.3 Checking CSI driver

The CSI driver may not be responding to the requests. Look in the logs of the CSI provisioner pod to see if there are any errors during the provisioning.

There are two provisioner pods:

```
kubectl@adm > kubectl -n rook-ceph get pod -l app=csi-rbdplugin-provisioner
```

Get the logs of each of the pods. One of them should be the leader and be responding to requests.

```
kubectl@adm > kubectl -n rook-ceph logs csi-cephfsplugin-provisioner-d77bb49c6-q9hwq csi-
provisioner
```

16.1.4.2.4 Restarting the operator

Lastly, if you have OSDs `up` and `in`, the next step is to confirm the operator is responding to the requests. Look in the operator pod logs around the time when the PVC was created to confirm if the request is being raised. If the operator does not show requests to provision the block image, the operator may be stuck on some other operation. In this case, restart the operator pod to get things going again.

16.1.4.3 Identifying the solution

If the OSD prepare logs did not give you enough clues about why the OSDs were not being created, review your `cluster.yaml` configuration. The common mistakes include:

- If `useAllDevices: true`, Rook expects to find local devices attached to the nodes. If no devices are found, no OSDs will be created.
- If `useAllDevices: false`, OSDs will only be created if `deviceFilter` is specified.
- Only local devices attached to the nodes will be configurable by Rook. In other words, the devices must show up under `/dev`.
    - The devices must not have any partitions or file systems on them. Rook will only configure raw devices. Partitions are not yet supported.

16.1.5 OSD pods are failing to start

16.1.5.1 Identifying symptoms

- OSD pods are failing to start.
- You have started a cluster after tearing down another cluster.

16.1.5.2 Investigating configuration errors

When an OSD starts, the device or directory will be configured for consumption. If there is an error with the configuration, the pod will crash and you will see the `CrashLoopBackoff` status for the pod. Look in the OSD pod logs for an indication of the failure.

```
kubectl@adm > kubectl -n rook-ceph logs rook-ceph-osd-fl8fs
```

One common case for failure is that you have re-deployed a test cluster and some state may remain from a previous deployment. If your cluster is larger than a few nodes, you may get lucky enough that the monitors were able to start and form a quorum. However, now the OSDs pods may fail to start due to the old state. Looking at the OSD pod logs, you will see an error about the file already existing.

```
kubectl -n rook-ceph logs rook-ceph-osd-fl8fs
[...]
2017-10-31 20:13:11.187106 I | mkfs-osd0: 2017-10-31 20:13:11.186992 7f0059d62e00 -1
 bluestore(/var/lib/rook/osd0) _read_fsid unparsable uuid
2017-10-31 20:13:11.187208 I | mkfs-osd0: 2017-10-31 20:13:11.187026 7f0059d62e00 -1
 bluestore(/var/lib/rook/osd0) _setup_block_symlink_or_file failed to create block
 symlink to /dev/disk/by-partuuid/651153ba-2dfc-4231-ba06-94759e5ba273: (17) File exists
2017-10-31 20:13:11.187233 I | mkfs-osd0: 2017-10-31 20:13:11.187038 7f0059d62e00 -1
 bluestore(/var/lib/rook/osd0) mkfs failed, (17) File exists
2017-10-31 20:13:11.187254 I | mkfs-osd0: 2017-10-31 20:13:11.187042 7f0059d62e00 -1
 OSD::mkfs: ObjectStore::mkfs failed with error (17) File exists
2017-10-31 20:13:11.187275 I | mkfs-osd0: 2017-10-31 20:13:11.187121 7f0059d62e00 -1  **
 ERROR: error creating empty object store in /var/lib/rook/osd0: (17) File exists
```

16.1.5.3 Solution

If the error is from the file that already exists, this is a common problem reinitializing the Rook cluster when the local directory used for persistence has **not** been purged. This directory is the `dataDirHostPath` setting in the cluster CRD and is typically set to `/var/lib/rook`. To fix the issue you will need to delete all components of Rook and then delete the contents of `/var/lib/rook` (or the directory specified by `dataDirHostPath`) on each of the hosts in the cluster. Then when the cluster CRD is applied to start a new cluster, the rook-operator should start all the pods as expected.

16.1.6 OSD pods are not created on my devices

16.1.6.1 Identifying symptoms

- No OSD pods are started in the cluster.
- Devices are not configured with OSDs even though specified in the cluster CRD.
- One OSD pod is started on each node instead of multiple pods for each device.

16.1.6.2 Investigating

First, ensure that you have specified the devices correctly in the CRD. The cluster CRD has several ways to specify the devices that are to be consumed by the Rook storage:

- `useAllDevices: true`: Rook will consume all devices it determines to be available.
- `deviceFilter`: Consume all devices that match this regular expression.
- `devices`: Explicit list of device names on each node to consume.

Second, if Rook determines that a device is not available (has existing partitions or a formatted file system), Rook will skip consuming the devices. If Rook is not starting OSDs on the devices you expect, Rook may have skipped it for this reason. To see if a device was skipped, view the OSD preparation log on the node where the device was skipped. Note that it is completely normal and expected for OSD prepare pod to be in the `completed` state. After the job is complete, Rook leaves the pod around in case the logs need to be investigated.

Get the prepare pods in the cluster:

```
kubectl@adm > kubectl -n rook-ceph get pod -l app=rook-ceph-osd-prepare
NAME                                READY     STATUS      RESTARTS   AGE
rook-ceph-osd-prepare-node1-fvmrp   0/1       Completed   0          18m
rook-ceph-osd-prepare-node2-w9xv9   0/1       Completed   0          22m
rook-ceph-osd-prepare-node3-7rgnv   0/1       Completed   0          22m
```

View the logs for the node of interest in the "provision" container:

```
kubectl@adm > kubectl -n rook-ceph logs rook-ceph-osd-prepare-node1-fvmrp provision
```

Here are some key lines to look for in the log. A device will be skipped if Rook sees it has partitions or a file system:

```
2019-05-30 19:02:57.353171 W | cephosd: skipping device sda that is in use
2019-05-30 19:02:57.452168 W | skipping device "sdb5": ["Used by ceph-disk"]
```

Other messages about a disk being unusable by Ceph include:

```
Insufficient space (<5GB) on vgs
Insufficient space (<5GB)
LVM detected
Has BlueStore device label
locked
read-only
```

A device is going to be configured:

```
2019-05-30 19:02:57.535598 I | cephosd: device sdc to be configured by ceph-volume
```

For each device configured, you will see a report in the log:

```
2019-05-30 19:02:59.844642 I |   Type            Path
                LV Size         % of device
2019-05-30 19:02:59.844651 I |
 ----------------------------------------------------------------------------------------------------
2019-05-30 19:02:59.844677 I |   [data]          /dev/sdc
                7.00 GB         100%
```

16.1.6.3 Solution

Either update the CR with the correct settings, or clean the partitions or file system from your devices.

After the settings are updated or the devices are cleaned, trigger the operator to analyze the devices again by restarting the operator. Each time the operator starts, it will ensure all the desired devices are configured. The operator does automatically deploy OSDs in most scenarios, but an operator restart will cover any scenarios that the operator does not detect automatically.

Restart the operator to ensure devices are configured. A new pod will automatically be started when the current operator pod is deleted.

```
kubectl@adm > kubectl -n rook-ceph delete pod -l app=rook-ceph-operator
```

16.1.7 Rook agent modprobe exec format error

16.1.7.1 Identifying symptoms

- `PersistentVolumes` from Ceph fail or timeout to mount.
- Rook Agent logs contain `modinfo: ERROR: could not get modinfo from 'rbd': Exec format error` lines.

16.1.7.2 Solution

If it is feasible to upgrade your kernel, you should upgrade to 4.x, even better is 4.7 or above, due to a feature for CephFS added to the kernel.

If you are unable to upgrade the kernel, you need to go to each host that will consume storage and run:

```
modprobe rbd
```

This command inserts the `rbd` module into the kernel.

To persist this fix, you need to add the `rbd` kernel module to either `/etc/modprobe.d/` or `/etc/modules-load.d/`. For both paths create a file called `rbd.conf` with the following content:

```
rbd
```

Now when a host is restarted, the module should be loaded automatically.

16.1.8 Using multiple shared file systems (CephFS) is attempted on a kernel version older than 4.7

16.1.8.1 Identifying symptoms

- More than one shared file system (CephFS) has been created in the cluster.
- A pod attempts to mount any other shared file system besides the **first** one that was created.
- The pod incorrectly gets the first file system mounted instead of the intended file system.

16.1.8.2 Solution

The only solution to this problem is to upgrade your kernel to 4.7 or higher. This is due to a `mount` flag added in kernel version 4.7, which allows choosing the file system by name.

16.1.9 Activating log to file for a particular Ceph daemon

They are cases where looking at Kubernetes logs is not enough for various reasons, but just to name a few:

- Not everyone is familiar for Kubernetes logging and expects to find logs in traditional directories.
- Logs get eaten (buffer limit from the log engine) and thus not requestable from Kubernetes.

So for each daemon, `dataDirHostPath` is used to store logs, if logging is activated. Rook will bind-mount `dataDirHostPath` for every pod. As of Ceph Nautilus 14.2.1, it is possible to enable logging for a particular daemon on the fly. Let us say you want to enable logging for `mon.a`, but only for this daemon. Using the toolbox or from inside the operator run:

```
cephuser@adm > ceph config daemon mon.a log_to_file true
```

This will activate logging on the file system, you will be able to find logs in `dataDirHostPath/$NAMESPACE/log`, so typically this would mean `/var/lib/rook/rook-ceph/log`. You do not need to restart the pod, the effect will be immediate.

To disable the logging on file, simply set `log_to_file` to `false`.

For Ceph Luminous and Mimic releases, `mon_cluster_log_file` and `cluster_log_file` can be set to `/var/log/ceph/XXXX` in the config override ConfigMap to enable logging.

16.1.10 A worker node using RBD devices hangs up

16.1.10.1 Identifying symptoms

- There is no progress on I/O from/to one of RBD devices (`/dev/rbd*` or `/dev/nbd*`).
- After that, the whole worker node hangs up.

16.1.10.2 Investigating

This happens when the following conditions are satisfied.

- The problematic RBD device and the corresponding OSDs are co-located.
- There is an XFS file system on top of this device.

In addition, when this problem happens, you can see the following messages in `dmesg`.

```
dmesg
...
[51717.039319] INFO: task kworker/2:1:5938 blocked for more than 120 seconds.
[51717.039361]       Not tainted 4.15.0-72-generic #81-Ubuntu
[51717.039388] "echo 0 > /proc/sys/kernel/hung_task_timeout_secs" disables this message.
...
```

This is the so-called `hung_task` problem and means that there is a deadlock in the kernel.

16.1.10.3 Solution

You can bypass this problem by using ext4 or any other file systems rather than XFS. The file system type can be specified with `csi.storage.k8s.io/fstype` in StorageClass resource.

16.1.11 Too few PGs per OSD warning is shown

16.1.11.1 Identifying symptoms

- `ceph status` shows "too few PGs per OSD" warning as follows.

```
cephuser@adm > ceph status
cluster:
id:     fd06d7c3-5c5c-45ca-bdea-1cf26b783065
health: HEALTH_WARN
too few PGs per OSD (16 < min 30)
```

16.1.11.2 Solution

See *Book "Troubleshooting Guide", Chapter 5 "Troubleshooting placement groups (PGs)"* for more information.

16.1.12 LVM metadata can be corrupted with OSD on LV-backed PVC

16.1.12.1 Identifying symptoms

There is a critical flaw in OSD on LV-backed PVC. LVM metadata can be corrupted if both the host and OSD container modify it simultaneously. For example, the administrator might modify it on the host, while the OSD initialization process in a container could modify it too. In addition, if `lvmetad` is running, the possibility of occurrence gets higher. In this case, the change of LVM metadata in OSD container is not reflected to LVM metadata cache in host for a while.

If you still decide to configure an OSD on LVM, keep the following in mind to reduce the probability of this issue.

16.1.12.2 Solution

- Disable `lvmetad`.
- Avoid configuration of LVs from the host. In addition, do not touch the VGs and physical volumes that back these LVs.
- Avoid incrementing the `count` field of `storageClassDeviceSets` and create a new LV that backs a OSD simultaneously.

You can know whether the above-mentioned tag exists tag by running

```
#  lvs -o lv_name,lv_tags
```

If the `lv_tag` field is empty in an LV corresponding to the OSD lv_tags, this OSD encountered the problem. In this case, retire this OSD or replace with other new OSD before restarting.

A Ceph maintenance updates based on upstream 'Pacific' point releases

Several key packages in SUSE Enterprise Storage 7.1 are based on the Pacific release series of Ceph. When the Ceph project (https://github.com/ceph/ceph) publishes new point releases in the Pacific series, SUSE Enterprise Storage 7.1 is updated to ensure that the product benefits from the latest upstream bug fixes and feature backports.

This chapter contains summaries of notable changes contained in each upstream point release that has been—or is planned to be—included in the product.

Glossary

General

Admin node

The host from which you run the Ceph-related commands to administer cluster hosts.

Alertmanager

A single binary which handles alerts sent by the Prometheus server and notifies the end user.

archive sync module

Module that enables creating an Object Gateway zone for keeping the history of S3 object versions.

Bucket

A point that aggregates other nodes into a hierarchy of physical locations.

Ceph Client

The collection of Ceph components which can access a Ceph Storage Cluster. These include the Object Gateway, the Ceph Block Device, the CephFS, and their corresponding libraries, kernel modules, and FUSE clients.

Ceph Dashboard

A built-in Web-based Ceph management and monitoring application to administer various aspects and objects of the cluster. The dashboard is implemented as a Ceph Manager module.

Ceph Manager

Ceph Manager or MGR is the Ceph manager software, which collects all the state from the whole cluster in one place.

Ceph Monitor

Ceph Monitor or MON is the Ceph monitor software.

Ceph Object Storage

The object storage "product", service or capabilities, which consists of a Ceph Storage Cluster and a Ceph Object Gateway.

Ceph OSD Daemon

The `ceph-osd` daemon is the component of Ceph that is responsible for storing objects on a local file system and providing access to them over the network.

Ceph Storage Cluster

The core set of storage software which stores the user's data. Such a set consists of Ceph monitors and OSDs.

`ceph-salt`

Provides tooling for deploying Ceph clusters managed by cephadm using Salt.

cephadm

cephadm deploys and manages a Ceph cluster by connecting to hosts from the manager daemon via SSH to add, remove, or update Ceph daemon containers.

CephFS

The Ceph file system.

CephX

The Ceph authentication protocol. Cephx operates like Kerberos, but it has no single point of failure.

CRUSH rule

The CRUSH data placement rule that applies to a particular pool or pools.

CRUSH, CRUSH Map

Controlled Replication Under Scalable Hashing: An algorithm that determines how to store and retrieve data by computing data storage locations. CRUSH requires a map of the cluster to pseudo-randomly store and retrieve data in OSDs with a uniform distribution of data across the cluster.

DriveGroups

DriveGroups are a declaration of one or more OSD layouts that can be mapped to physical drives. An OSD layout defines how Ceph physically allocates OSD storage on the media matching the specified criteria.

Grafana

Database analytics and monitoring solution.

Metadata Server

Metadata Server or MDS is the Ceph metadata software.

Multi-zone

Node

Any single machine or server in a Ceph cluster.

Object Gateway

The S3/Swift gateway component for Ceph Object Store. Also known as the RADOS Gateway (RGW).

OSD

Object Storage Device: A physical or logical storage unit.

OSD node

A cluster node that stores data, handles data replication, recovery, backfilling, rebalancing, and provides some monitoring information to Ceph monitors by checking other Ceph OSD daemons.

PG

Placement Group: a sub-division of a *pool*, used for performance tuning.

Point Release

Any ad-hoc release that includes only bug or security fixes.

Pool

Logical partitions for storing objects such as disk images.

Prometheus

Systems monitoring and alerting toolkit.

RADOS Block Device (RBD)

The block storage component of Ceph. Also known as the Ceph block device.

Reliable Autonomic Distributed Object Store (RADOS)

The core set of storage software which stores the user's data (MON + OSD).

Routing tree

A term given to any diagram that shows the various routes a receiver can run.

Rule Set

Rules to determine data placement for a pool.

Samba

Windows integration software.

Samba Gateway

The Samba Gateway joins the Active Directory in the Windows domain to authenticate and authorize users.

zonegroup