Red Hat Ceph Storage 7

Block Device Guide

Managing, creating, configuring, and using Red Hat Ceph Storage Block Devices

Last Updated: 2024-03-08

Red Hat Ceph Storage 7 Block Device Guide

Managing, creating, configuring, and using Red Hat Ceph Storage Block Devices

Legal Notice

Abstract

This document describes how to manage, create, configure, and use Red Hat Ceph Storage Block Devices. Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message .

Table of Contents

CHAPTER 1. INTRODUCTION TO CEPH BLOCK DEVICES

A block is a set length of bytes in a sequence, for example, a 512-byte block of data. Combining many blocks together into a single file can be used as a storage device that you can read from and write to. Block-based storage interfaces are the most common way to store data with rotating media such as:

- Hard drives
- CD/DVD discs
- Floppy disks
- Traditional 9-track tapes

The ubiquity of block device interfaces makes a virtual block device an ideal candidate for interacting with a mass data storage system like Red Hat Ceph Storage.

Ceph block devices are thin-provisioned, resizable and store data striped over multiple Object Storage Devices (OSD) in a Ceph storage cluster. Ceph block devices are also known as Reliable Autonomic Distributed Object Store (RADOS) Block Devices (RBDs). Ceph block devices leverage RADOS capabilities such as:

- Snapshots
- Replication
- Data consistency

Ceph block devices interact with OSDs by using the **librbd** library.

Ceph block devices deliver high performance with infinite scalability to Kernel Virtual Machines (KVMs), such as Quick Emulator (QEMU), and cloud-based computing systems, like OpenStack, that rely on the **libvirt** and QEMU utilities to integrate with Ceph block devices. You can use the same storage cluster to operate the Ceph Object Gateway and Ceph block devices simultaneously.

IMPORTANT

To use Ceph block devices, requires you to have access to a running Ceph storage cluster. For details on installing a Red Hat Ceph Storage cluster, see the *Red Hat Ceph Storage Installation Guide*.

CHAPTER 2. CEPH BLOCK DEVICES

As a storage administrator, being familiar with Ceph's block device commands can help you effectively manage the Red Hat Ceph Storage cluster. You can create and manage block devices pools and images, along with enabling and disabling the various features of Ceph block devices.

Prerequisites

- A running Red Hat Ceph Storage cluster.

2.1. DISPLAYING THE COMMAND HELP

Display command, and sub-command online help from the command-line interface.

NOTE

The **-h** option still displays help for all available commands.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the client node.

Procedure

1. Use the **rbd help** command to display help for a particular **rbd** command and its subcommand:

   Syntax

   ```
   rbd help COMMAND SUBCOMMAND
   ```

2. To display help for the **snap list** command:

   ```
   [root@rbd-client ~]# rbd help snap list
   ```

2.2. CREATING A BLOCK DEVICE POOL

Before using the block device client, ensure a pool for **rbd** exists, is enabled and initialized.

NOTE

You **MUST** create a pool first before you can specify it as a source.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the client node.

Procedure

1. To create an **rbd** pool, execute the following:

   Syntax

   ```
   ceph osd pool create POOL_NAME PG_NUM
   ceph osd pool application enable POOL_NAME rbd
   rbd pool init -p POOL_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# ceph osd pool create pool1
   [root@rbd-client ~]# ceph osd pool application enable pool1 rbd
   [root@rbd-client ~]# rbd pool init -p pool1
   ```

Additional Resources

- See the *Pools* chapter in the *Red Hat Ceph Storage Storage Strategies Guide* for additional details.

2.3. CREATING A BLOCK DEVICE IMAGE

Before adding a block device to a node, create an image for it in the Ceph storage cluster.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the client node.

Procedure

1. To create a block device image, execute the following command:

   Syntax

   ```
   rbd create IMAGE_NAME --size MEGABYTES --pool POOL_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd create image1 --size 1024 --pool pool1
   ```

   This example creates a 1 GB image named **image1** that stores information in a pool named **pool1**.

   NOTE

   Ensure the pool exists before creating an image.

Additional Resources

- See the *Creating a block device pool* section in the *Red Hat Ceph Storage Block Device Guide* for additional details.

2.4. LISTING THE BLOCK DEVICE IMAGES

List the block device images.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the client node.

Procedure

1. To list block devices in the **rbd** pool, execute the following command:

   NOTE

   rbd is the default pool name.

   Example

   ```
   [root@rbd-client ~]# rbd ls
   ```

2. To list block devices in a specific pool:

   Syntax

   ```
   rbd ls POOL_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd ls pool1
   ```

2.5. RETRIEVING THE BLOCK DEVICE IMAGE INFORMATION

Retrieve information on the block device image.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the client node.

Procedure

1. To retrieve information from a particular image in the default **rbd** pool, run the following command:

   Syntax

   ```
   rbd --image IMAGE_NAME info
   ```

Example

```
[root@rbd-client ~]# rbd --image image1 info
```

2. To retrieve information from an image within a pool:

Syntax

```
rbd --image IMAGE_NAME -p POOL_NAME info
```

Example

```
[root@rbd-client ~]# rbd --image image1 -p pool1 info
```

2.6. RESIZING A BLOCK DEVICE IMAGE

Ceph block device images are thin-provisioned. They do not actually use any physical storage until you begin saving data to them. However, they do have a maximum capacity that you set with the **--size** option.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the client node.

Procedure

1. To increase or decrease the maximum size of a Ceph block device image for the default **rbd** pool:

Syntax

```
rbd resize --image IMAGE_NAME --size SIZE
```

Example

```
[root@rbd-client ~]# rbd resize --image image1 --size 1024
```

2. To increase or decrease the maximum size of a Ceph block device image for a specific pool:

Syntax

```
rbd resize --image POOL_NAME/IMAGE_NAME --size SIZE
```

Example

```
[root@rbd-client ~]# rbd resize --image pool1/image1 --size 1024
```

2.7. REMOVING A BLOCK DEVICE IMAGE

Remove a block device image.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the client node.

Procedure

1. To remove a block device from the default **rbd** pool:

   Syntax

   ```
   rbd rm IMAGE_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd rm image1
   ```

2. To remove a block device from a specific pool:

   Syntax

   ```
   rbd rm IMAGE_NAME -p POOL_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd rm image1 -p pool1
   ```

2.8. MOVING A BLOCK DEVICE IMAGE TO THE TRASH

RADOS Block Device (RBD) images can be moved to the trash using the **rbd trash** command. This command provides more options than the **rbd rm** command.

Once an image is moved to the trash, it can be removed from the trash at a later time. This helps to avoid accidental deletion.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the client node.

Procedure

1. To move an image to the trash execute the following:

   Syntax

   ```
   rbd trash mv [POOL_NAME/] IMAGE_NAME
   ```

Example

```
[root@rbd-client ~]# rbd trash mv pool1/image1
```

Once an image is in the trash, a unique image ID is assigned.

NOTE

You need this image ID to specify the image later if you need to use any of the trash options.

2. Execute the **rbd trash list *POOL_NAME*** for a list of IDs of the images in the trash. This command also returns the image's pre-deletion name. In addition, there is an optional **--image-id** argument that can be used with **rbd info** and **rbd snap** commands. Use **--image-id** with the **rbd info** command to see the properties of an image in the trash, and with **rbd snap** to remove an image's snapshots from the trash.

3. To remove an image from the trash execute the following:

Syntax

```
rbd trash rm [POOL_NAME/] IMAGE_ID
```

Example

```
[root@rbd-client ~]# rbd trash rm pool1/d35ed01706a0
```

IMPORTANT

Once an image is removed from the trash, it cannot be restored.

4. Execute the **rbd trash restore** command to restore the image:

Syntax

```
rbd trash restore [POOL_NAME/] IMAGE_ID
```

Example

```
[root@rbd-client ~]# rbd trash restore pool1/d35ed01706a0
```

5. To remove all expired images from trash:

Syntax

```
rbd trash purge POOL_NAME
```

Example

```
[root@rbd-client ~]# rbd trash purge pool1
Removing images: 100% complete...done.
```

2.9. DEFINING AN AUTOMATIC TRASH PURGE SCHEDULE

You can schedule periodic trash purge operations on a pool.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the client node.

Procedure

1. To add a trash purge schedule, execute:

   Syntax

   ```
   rbd trash purge schedule add --pool POOL_NAME INTERVAL
   ```

   Example

   ```
   [ceph: root@host01 /]# rbd trash purge schedule add --pool pool1 10m
   ```

2. To list the trash purge schedule, execute:

   Syntax

   ```
   rbd trash purge schedule ls --pool POOL_NAME
   ```

   Example

   ```
   [ceph: root@host01 /]# rbd trash purge schedule ls --pool pool1
   every 10m
   ```

3. To know the status of trash purge schedule, execute:

   Example

   ```
   [ceph: root@host01 /]# rbd trash purge schedule status
   POOL   NAMESPACE   SCHEDULE  TIME
   pool1              2021-08-02 11:50:00
   ```

4. To remove the trash purge schedule, execute:

   Syntax

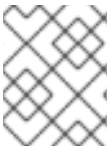
   ```
   rbd trash purge schedule remove --pool POOL_NAME INTERVAL
   ```

   Example

   ```
   [ceph: root@host01 /]# rbd trash purge schedule remove --pool pool1 10m
   ```

2.10. ENABLING AND DISABLING IMAGE FEATURES

The block device images, such as **fast-diff**, **exclusive-lock**, **object-map**, or **deep-flatten**, are enabled by default. You can enable or disable these image features on already existing images.



NOTE

The **deep flatten** feature can be only disabled on already existing images but not enabled. To use **deep flatten**, enable it when creating images.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the client node.

Procedure

1. Retrieve information from a particular image in a pool:

   Syntax

   ```
   rbd --image POOL_NAME/IMAGE_NAME info
   ```

   Example

   ```
   [ceph: root@host01 /]# rbd --image pool1/image1 info
   ```

2. Enable a feature:

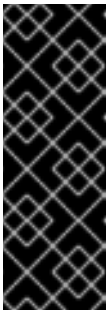
   Syntax

   ```
   rbd feature enable POOL_NAME/IMAGE_NAME FEATURE_NAME
   ```

   a. To enable the **exclusive-lock** feature on the **image1** image in the **pool1** pool:

      Example

      ```
      [ceph: root@host01 /]# rbd feature enable pool1/image1 exclusive-lock
      ```

      IMPORTANT

      If you enable the **fast-diff** and **object-map** features, then rebuild the object map:

      + .Syntax

      ```
      rbd object-map rebuild POOL_NAME/IMAGE_NAME
      ```

3. Disable a feature:

Syntax

```
rbd feature disable POOL_NAME/IMAGE_NAME FEATURE_NAME
```

a. To disable the **fast-diff** feature on the **image1** image in the **pool1** pool:

Example

```
[ceph: root@host01 /]# rbd feature disable pool1/image1 fast-diff
```

2.11. WORKING WITH IMAGE METADATA

Ceph supports adding custom image metadata as key-value pairs. The pairs do not have any strict format.

Also, by using metadata, you can set the RADOS Block Device (RBD) configuration parameters for particular images.

Use the **rbd image-meta** commands to work with metadata.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the client node.

Procedure

1. To set a new metadata key-value pair:

   Syntax

   ```
   rbd image-meta set POOL_NAME/IMAGE_NAME KEY VALUE
   ```

   Example

   ```
   [ceph: root@host01 /]# rbd image-meta set pool1/image1 last_update 2021-06-06
   ```

   This example sets the **last_update** key to the **2021-06-06** value on the **image1** image in the **pool1** pool.

2. To view a value of a key:

   Syntax

   ```
   rbd image-meta get POOL_NAME/IMAGE_NAME KEY
   ```

   Example

   ```
   [ceph: root@host01 /]# rbd image-meta get pool1/image1 last_update
   ```

   This example views the value of the **last_update** key.

3. To show all metadata on an image:

   Syntax

   ```
   rbd image-meta list POOL_NAME/IMAGE_NAME
   ```

   Example

   ```
   [ceph: root@host01 /]# rbd image-meta list pool1/image1
   ```

   This example lists the metadata set for the **image1** image in the **pool1** pool.

4. To remove a metadata key-value pair:

   Syntax

   ```
   rbd image-meta remove POOL_NAME/IMAGE_NAME KEY
   ```

   Example

   ```
   [ceph: root@host01 /]# rbd image-meta remove pool1/image1 last_update
   ```

   This example removes the **last_update** key-value pair from the **image1** image in the **pool1** pool.

5. To override the RBD image configuration settings set in the Ceph configuration file for a particular image:

   Syntax

   ```
   rbd config image set POOL_NAME/IMAGE_NAME  PARAMETER VALUE
   ```

   Example

   ```
   [ceph: root@host01 /]# rbd config image set pool1/image1 rbd_cache false
   ```

   This example disables the RBD cache for the **image1** image in the **pool1** pool.

Additional Resources

- See the *Block device general options* section in the *Red Hat Ceph Storage Block Device Guide* for a list of possible configuration options.

2.12. MOVING IMAGES BETWEEN POOLS

You can move RADOS Block Device (RBD) images between different pools within the same cluster.

During this process, the source image is copied to the target image with all snapshot history and optionally with link to the source image's parent to help preserve sparseness. The source image is read only, the target image is writable. The target image is linked to the source image while the migration is in progress.

You can safely run this process in the background while the new target image is in use. However, stop all clients using the target image before the preparation step to ensure that clients using the image are updated to point to the new target image.

IMPORTANT

The **krbd** kernel module does not support live migration at this time.

Prerequisites

- Stop all clients that use the source image.
- Root-level access to the client node.

Procedure

1. Prepare for migration by creating the new target image that cross-links the source and target images:

   Syntax

   ```
   rbd migration prepare SOURCE_IMAGE TARGET_IMAGE
   ```

   Replace:

   - *SOURCE_IMAGE* with the name of the image to be moved. Use the *POOL/IMAGE_NAME* format.
   - *TARGET_IMAGE* with the name of the new image. Use the *POOL/IMAGE_NAME* format.

   Example

   ```
   [root@rbd-client ~]# rbd migration prepare pool1/image1 pool2/image2
   ```

2. Verify the state of the new target image, which is supposed to be **prepared**:

   Syntax

   ```
   rbd status TARGET_IMAGE
   ```

   Example

   ```
   [root@rbd-client ~]# rbd status pool2/image2
   Watchers: none
   Migration:
           source: pool1/image1 (5e2cba2f62e)
           destination: pool2/image2 (5e2ed95ed806)
           state: prepared
   ```

3. Optionally, restart the clients using the new target image name.
4. Copy the source image to target image:

Syntax

```
rbd migration execute TARGET_IMAGE
```

Example

```
[root@rbd-client ~]# rbd migration execute pool2/image2
```

5. Ensure that the migration is completed:

Example

```
[root@rbd-client ~]# rbd status pool2/image2
Watchers:
   watcher=1.2.3.4:0/3695551461 client.123 cookie=123
Migration:
        source: pool1/image1 (5e2cba2f62e)
        destination: pool2/image2 (5e2ed95ed806)
        state: executed
```

6. Commit the migration by removing the cross-link between the source and target images, and this also removes the source image:

Syntax

```
rbd migration commit TARGET_IMAGE
```

Example

```
[root@rbd-client ~]# rbd migration commit pool2/image2
```

If the source image is a parent of one or more clones, use the **--force** option after ensuring that the clone images are not in use:

Example

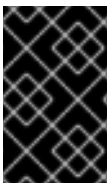
```
[root@rbd-client ~]# rbd migration commit pool2/image2 --force
```

7. If you did not restart the clients after the preparation step, restart them using the new target image name.

2.13. MIGRATING POOLS

You can migrate or copy RADOS Block Device (RBD) images.

During this process, the source image is exported and then imported.



IMPORTANT

Use this migration process if the workload contains **only** RBD images. No **rados cppool** images can exist in the workload. If **rados cppool** images exist in the workload, see Migrating a pool in the *Storage Strategies Guide*.

IMPORTANT

While running the export and import commands, be sure that there is no active I/O in the related RBD images. It is recommended to take production down during this pool migration time.

Prerequisites

- Stop all active I/O in the RBD images which are being exported and imported.
- Root-level access to the client node.

Procedure

- Migrate the volume.

  Syntax

  ```
  rbd export volumes/VOLUME_NAME - | rbd import --image-format 2 -
  volumes_new/VOLUME_NAME
  ```

  Example

  ```
  [root@rbd-client ~]# rbd export volumes/volume-3c4c63e3-3208-436f-9585-fee4e2a3de16 - |
  rbd import --image-format 2 - volumes_new/volume-3c4c63e3-3208-436f-9585-
  fee4e2a3de16
  ```

- If using the local drive for import or export is necessary, the commands can be divided, first exporting to a local drive and then importing the files to a new pool.

  Syntax

  ```
  rbd export volume/VOLUME_NAME FILE_PATH
  rbd import --image-format 2 FILE_PATH volumes_new/VOLUME_NAME
  ```

  Example

  ```
  [root@rbd-client ~]# rbd export volumes/volume-3c4c63e3-3208-436f-9585-fee4e2a3de16
  <path of export file>
  [root@rbd-client ~]# rbd import --image-format 2 <path> volumes_new/volume-3c4c63e3-
  3208-436f-9585-fee4e2a3de16
  ```

2.14. THE RBDMAP SERVICE

The **systemd** unit file, **rbdmap.service**, is included with the **ceph-common** package. The **rbdmap.service** unit executes the **rbdmap** shell script.

This script automates the mapping and unmapping of RADOS Block Devices (RBD) for one or more RBD images. The script can be ran manually at any time, but the typical use case is to automatically mount RBD images at boot time, and unmount at shutdown. The script takes a single argument, which can be either **map**, for mounting or **unmap**, for unmounting RBD images. The script parses a configuration file, the default is **/etc/ceph/rbdmap**, but can be overridden using an environment variable called **RBDMAPFILE**. Each line of the configuration file corresponds to an RBD image.

The format of the configuration file format is as follows:

IMAGE_SPEC RBD_OPTS

Where *IMAGE_SPEC* specifies the *POOL_NAME / IMAGE_NAME*, or just the *IMAGE_NAME*, in which case the *POOL_NAME* defaults to **rbd**. The *RBD_OPTS* is an optional list of options to be passed to the underlying **rbd map** command. These parameters and their values should be specified as a comma-separated string:

OPT1=VAL1,OPT2=VAL2,...,OPT_N=VAL_N

This will cause the script to issue an **rbd map** command like the following:

Syntax

```
rbd map POOLNAME/IMAGE_NAME --OPT1 VAL1 --OPT2 VAL2
```

NOTE

For options and values which contain commas or equality signs, a simple apostrophe can be used to prevent replacing them.

When successful, the **rbd map** operation maps the image to a **/dev/rbdX** device, at which point a **udev** rule is triggered to create a friendly device name symlink, for example, **/dev/rbd/*POOL_NAME*/*IMAGE_NAME***, pointing to the real mapped device. For mounting or unmounting to succeed, the friendly device name must have a corresponding entry in **/etc/fstab** file. When writing **/etc/fstab** entries for RBD images, it is a good idea to specify the **noauto** or **nofail** mount option. This prevents the init system from trying to mount the device too early, before the device exists.

Additional Resources

- See the **rbd** manpage for a full list of possible options.

2.15. CONFIGURING THE RBDMAP SERVICE

To automatically map and mount, or unmap and unmount, RADOS Block Devices (RBD) at boot time, or at shutdown respectively.

Prerequisites

- Root-level access to the node doing the mounting.
- Installation of the **ceph-common** package.

Procedure

1. Open for editing the **/etc/ceph/rbdmap** configuration file.
2. Add the RBD image or images to the configuration file:

   ### Example

   ```
   foo/bar1    id=admin,keyring=/etc/ceph/ceph.client.admin.keyring
   ```

```
foo/bar2
id=admin,keyring=/etc/ceph/ceph.client.admin.keyring,options='lock_on_read,queue_depth=102
4'
```

3. Save changes to the configuration file.
4. Enable the RBD mapping service:

   Example

   ```
   [root@client ~]# systemctl enable rbdmap.service
   ```

Additional Resources

- See the *The rbdmap service* section of the *Red Hat Ceph Storage Block Device Guide* for more details on the RBD system service.

2.16. PERSISTENT WRITE LOG CACHE

In a Red Hat Ceph Storage cluster, Persistent Write Log (PWL) cache provides a persistent, fault-tolerant write-back cache for librbd-based RBD clients.

PWL cache uses a log-ordered write-back design which maintains checkpoints internally so that writes that get flushed back to the cluster are always crash consistent. If the client cache is lost entirely, the disk image is still consistent but the data appears stale. You can use PWL cache with persistent memory (PMEM) or solid-state disks (SSD) as cache devices.

For PMEM, the cache mode is replica write log (RWL) and for SSD, the cache mode is (SSD). Currently, PWL cache supports RWL and SSD modes and is disabled by default.

Primary benefits of PWL cache are:

- PWL cache can provide high performance when the cache is not full. The larger the cache, the longer the duration of high performance.
- PWL cache provides persistence and is not much slower than RBD cache. RBD cache is faster but volatile and cannot guarantee data order and persistence.
- In a steady state, where the cache is full, performance is affected by the number of I/Os in flight. For example, PWL can provide higher performance at low io_depth, but at high io_depth, such as when the number of I/Os is greater than 32, the performance is often worse than that in cases without cache.

Use cases for PMEM caching are:

- Different from RBD cache, PWL cache has non-volatile characteristics and is used in scenarios where you do not want data loss and need performance.
- RWL mode provides low latency. It has a stable low latency for burst I/Os and it is suitable for those scenarios with high requirements for stable low latency.
- RWL mode also has high continuous and stable performance improvement in scenarios with low I/O depth or not too much inflight I/O.

Use case for SSD caching is:

- The advantages of SSD mode are similar to RWL mode. SSD hardware is relatively cheap and popular, but its performance is slightly lower than PMEM.

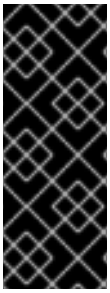
2.17. PERSISTENT WRITE LOG CACHE LIMITATIONS

When using Persistent Write Log (PWL) cache, there are several limitations that should be considered.

- The underlying implementation of persistent memory (PMEM) and solid-state disks (SSD) is different, with PMEM having higher performance. At present, PMEM can provide "persist on write" and SSD is "persist on flush or checkpoint". In future releases, these two modes will be configurable.
- When users switch frequently and open and close images repeatedly, Ceph displays poor performance. If PWL cache is enabled, the performance is worse. It is not recommended to set **num_jobs** in a Flexible I/O (fio) test, but instead setup multiple jobs to write different images.

2.18. ENABLING PERSISTENT WRITE LOG CACHE

You can enable persistent write log cache (PWL) on a Red Hat Ceph Storage cluster by setting the Ceph RADOS block device (RBD) **rbd_persistent_cache_mode** and **rbd_plugins** options.



IMPORTANT

The exclusive-lock feature must be enabled to enable persistent write log cache. The cache can be loaded only after the exclusive-lock is acquired. Exclusive-locks are enabled on newly created images by default unless overridden by the **rbd_default_features** configuration option or the **--image-feature** flag for the **rbd create** command. See the *Enabling and disabling image features* section for more details on the **exclusive-lock** feature.

Set the persistent write log cache options at the host level by using the **ceph config set** command. Set the persistent write log cache options at the pool or image level by using the **rbd config pool set** or the **rbd config image set** commands.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the monitor node.
- The exclusive-lock feature is enabled.
- Client-side disks are persistent memory (PMEM) or solid-state disks (SSD).
- RBD cache is disabled.

Procedure

1. Enable PWL cache:
   a. At the host level, use the **ceph config set** command:

      Syntax

```
ceph config set client rbd_persistent_cache_mode CACHE_MODE
ceph config set client rbd_plugins pwl_cache
```

Replace *CACHE_MODE* with **rwl** or **ssd**.

Example

```
[ceph: root@host01 /]# ceph config set client rbd_persistent_cache_mode ssd
[ceph: root@host01 /]# ceph config set client rbd_plugins pwl_cache
```

b. At the pool level, use the **rbd config pool set** command:

Syntax

```
rbd config pool set POOL_NAME rbd_persistent_cache_mode CACHE_MODE
rbd config pool set POOL_NAME rbd_plugins pwl_cache
```

Replace *CACHE_MODE* with **rwl** or **ssd**.

Example

```
[ceph: root@host01 /]# rbd config pool set pool1 rbd_persistent_cache_mode ssd
[ceph: root@host01 /]# rbd config pool set pool1 rbd_plugins pwl_cache
```

c. At the image level, use the **rbd config image set** command:

Syntax

```
rbd config image set POOL_NAME/IMAGE_NAME rbd_persistent_cache_mode
CACHE_MODE
rbd config image set POOL_NAME/IMAGE_NAME rbd_plugins pwl_cache
```

Replace *CACHE_MODE* with **rwl** or **ssd**.

Example

```
[ceph: root@host01 /]# rbd config image set pool1/image1 rbd_persistent_cache_mode
ssd
[ceph: root@host01 /]# rbd config image set pool1/image1 rbd_plugins pwl_cache
```

2. Optional: Set the additional RBD options at the host, the pool, or the image level:

Syntax

```
rbd_persistent_cache_mode CACHE_MODE
rbd_plugins pwl_cache
rbd_persistent_cache_path /PATH_TO_CACHE_DIRECTORY 1
rbd_persistent_cache_size PERSISTENT_CACHE_SIZE 2
```

1 **rbd_persistent_cache_path** - A file folder to cache data that must have direct access (DAX) enabled when using the **rwl** mode to avoid performance degradation.

2 **rbd_persistent_cache_size** - The cache size per image, with a minimum cache size of 1 GB. The larger the cache size, the better the performance.

a. Setting additional RBD options for **rwl** mode:

Example

```
rbd_cache false
rbd_persistent_cache_mode rwl
rbd_plugins pwl_cache
rbd_persistent_cache_path /mnt/pmem/cache/
rbd_persistent_cache_size 1073741824
```

b. Setting additional RBD options for **ssd** mode:

Example

```
rbd_cache false
rbd_persistent_cache_mode ssd
rbd_plugins pwl_cache
rbd_persistent_cache_path /mnt/nvme/cache
rbd_persistent_cache_size 1073741824
```

Additional Resources

- See the *Direct Access for files* article on *kernel.org* for more details on using DAX.

2.19. CHECKING PERSISTENT WRITE LOG CACHE STATUS

You can check the status of the Persistent Write Log (PWL) cache. The cache is used when an exclusive lock is acquired, and when the exclusive-lock is released, the persistent write log cache is closed. The cache status shows information about the cache size, location, type, and other cache-related information. Updates to the cache status are done when the cache is opened and closed.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the monitor node.
- A running process with PWL cache enabled.

Procedure

- View the PWL cache status:

Syntax

```
rbd status POOL_NAME/IMAGE_NAME
```

Example

```
[ceph: root@host01 /]# rbd status pool1/image1
Watchers:
        watcher=10.10.0.102:0/1061883624 client.25496 cookie=140338056493088
Persistent cache state:
        host: host02
        path: /mnt/nvme0/rbd-pwl.rbd.101e5824ad9a.pool
        size: 1 GiB
        mode: ssd
        stats_timestamp: Mon Apr 18 13:26:32 2022
        present: true    empty: false    clean: false
        allocated: 509 MiB
        cached: 501 MiB
        dirty: 338 MiB
        free: 515 MiB
        hits_full: 1450 / 61%
        hits_partial: 0 / 0%
        misses: 924
        hit_bytes: 192 MiB / 66%
        miss_bytes: 97 MiB
```

2.20. FLUSHING PERSISTENT WRITE LOG CACHE

You can flush the cache file with the **rbd** command, specifying **persistent-cache flush**, the pool name, and the image name before discarding the persistent write log (PWL) cache. The **flush** command can explicitly write cache files back to the OSDs. If there is a cache interruption or the application dies unexpectedly, all the entries in the cache are flushed to the OSDs so that you can manually flush the data and then **invalidate** the cache.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the monitor node.
- PWL cache is enabled.

Procedure

- Flush the PWL cache:

  #### Syntax

  ```
  rbd persistent-cache flush POOL_NAME/IMAGE_NAME
  ```

  #### Example

  ```
  [ceph: root@host01 /]# rbd persistent-cache flush pool1/image1
  ```

Additional Resources

- See the *Discarding persistent write log cache* section in the *Red Hat Ceph Storage Block Device Guide* for more details.

2.21. DISCARDING PERSISTENT WRITE LOG CACHE

You might need to manually discard the Persistent Write Log (PWL) cache, for example, if the data in the cache has expired. You can discard a cache file for an image by using the **rbd persistent-cache invalidate** command. The command removes the cache metadata for the specified image, disables the cache feature, and deletes the local cache file, if it exists.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the monitor node.
- PWL cache is enabled.

Procedure

- Discard PWL cache:

  #### Syntax

  ```
  rbd persistent-cache invalidate POOL_NAME/IMAGE_NAME
  ```

  #### Example

  ```
  [ceph: root@host01 /]# rbd persistent-cache invalidate pool1/image1
  ```

2.22. MONITORING PERFORMANCE OF CEPH BLOCK DEVICES USING THE COMMAND-LINE INTERFACE

Starting with Red Hat Ceph Storage 4.1, a performance metrics gathering framework is integrated within the Ceph OSD and Manager components. This framework provides a built-in method to generate and process performance metrics upon which other Ceph Block Device performance monitoring solutions are built.

A new Ceph Manager module,**rbd_support**, aggregates the performance metrics when enabled. The **rbd** command has two new actions: **iotop** and **iostat**.

NOTE

The initial use of these actions can take around 30 seconds to populate the data fields.

Prerequisites

- User-level access to a Ceph Monitor node.

Procedure

1. Ensure the **rbd_support** Ceph Manager module is enabled:

   #### Example

```
[ceph: root@host01 /]# ceph mgr module ls

{
      "always_on_modules": [
        "balancer",
         "crash",
         "devicehealth",
         "orchestrator",
         "pg_autoscaler",
         "progress",
         "rbd_support",   <--
         "status",
         "telemetry",
         "volumes"
}
```

2. To display an "iotop"-style of images:

   Example

   ```
   [user@mon ~]$ rbd perf image iotop
   ```

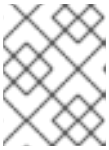
   NOTE

   The write ops, read-ops, write-bytes, read-bytes, write-latency, and read-latency columns can be sorted dynamically by using the right and left arrow keys.

3. To display an "iostat"-style of images:

   Example

   ```
   [user@mon ~]$ rbd perf image iostat
   ```

   NOTE

   The output from this command can be in JSON or XML format, and then can be sorted using other command-line tools.

CHAPTER 3. LIVE MIGRATION OF IMAGES

As a storage administrator, you can live-migrate RBD images between different pools or even with the same pool, within the same storage cluster. You can migrate between different images formats and layouts and even from external data sources. When live migration is initiated, the source image is deep copied to the destination image, pulling all snapshot history while preserving the sparse allocation of data where possible.

IMPORTANT

Currently, the **krbd** kernel module does not support live migration.

Prerequisites

- A running Red Hat Ceph Storage cluster.

3.1. THE LIVE MIGRATION PROCESS

By default, during the live migration of the RBD images with the same storage cluster, the source image is marked read-only. All clients redirect the Input/Output (I/O) to the new target image. Additionally, this mode can preserve the link to the source image's parent to preserve sparseness, or it can flatten the image during the migration to remove the dependency on the source image's parent. You can use the live migration process in an import-only mode, where the source image remains unmodified. You can link the target image to an external data source, such as a backup file, HTTP(s) file, or an S3 object. The live migration copy process can safely run in the background while the new target image is being used.

The live migration process consists of three steps:

Prepare Migration: The first step is to create new target image and link the target image to the source image. If the import-only mode is not configured, the source image will also be linked to the target image and marked read-only. Attempts to read uninitialized data extents within the target image will internally redirect the read to the source image, and writes to uninitialized extents within the target image will internally deep copy, the overlapping source image extents to the target image.

Execute Migration: This is a background operation that deep-copies all initialized blocks from the source image to the target. You can run this step when clients are actively using the new target image.

Finish Migration: You can commit or abort the migration, once the background migration process is completed. Committing the migration removes the cross-links between the source and target images, and will remove the source image if not configured in the import-only mode. Aborting the migration remove the cross-links, and will remove the target image.

3.2. FORMATS

You can use the **native** format to describe a native RBD image within a Red Hat Ceph Storage cluster as the source image. The **source-spec** JSON document is encoded as:

Syntax

```
{
    "type": "native",
    "pool_name": "POOL_NAME",
    ["pool_id": "POOL_ID",] (optional, alternative to "POOL_NAME" key)
    ["pool_namespace": "POOL_NAMESPACE",] (optional)
```

```
    "image_name": "IMAGE_NAME>",
    ["image_id": "IMAGE_ID",] (optional, useful if image is in trash)
    "snap_name": "SNAP_NAME",
    ["snap_id": "SNAP_ID",] (optional, alternative to "SNAP_NAME" key)
}
```

Note that the **native** format does not include the stream object since it utilizes native Ceph operations. For example, to import from the image **rbd/ns1/image1@snap1**, the **source-spec** could be encoded as:

Example

```
{
"type": "native",
"pool_name": "rbd",
"pool_namespace": "ns1",
"image_name": "image1",
"snap_name": "snap1"
}
```

You can use the **qcow** format to describe a QEMU copy-on-write (QCOW) block device. Both the QCOW v1 and v2 formats are currently supported with the exception of advanced features such as compression, encryption, backing files, and external data files. You can link the **qcow** format data to any supported stream source:

Example

```
{
    "type": "qcow",
    "stream": {
      "type": "file",
  "file_path": "/mnt/image.qcow"
    }
}
```

You can use the **raw** format to describe a thick-provisioned, raw block device export that is **rbd export –export-format 1 *SNAP_SPEC***. You can link the **raw** format data to any supported stream source:

Example

```
{
    "type": "raw",
    "stream": {
      "type": "file",
      "file_path": "/mnt/image-head.raw"
    },
    "snapshots": [
        {
            "type": "raw",
            "name": "snap1",
            "stream": {
              "type": "file",
       "file_path": "/mnt/image-snap1.raw"
            }
```

```
        },
    ] (optional oldest to newest ordering of snapshots)
}
```

The inclusion of the **snapshots** array is optional and currently only supports thick-provisioned raw snapshot exports.

3.3. STREAMS

File stream

You can use the **file** stream to import from a locally accessible POSIX file source.

Syntax

```
{
    <format unique parameters>
    "stream": {
        "type": "file",
        "file_path": "FILE_PATH"
    }
}
```

For example, to import a raw-format image from a file located at **/mnt/image.raw**, the **source-spec** JSON file is:

Example

```
{
    "type": "raw",
    "stream": {
        "type": "file",
        "file_path": "/mnt/image.raw"
    }
}
```

HTTP stream

You can use the **HTTP** stream to import from a remote HTTP or HTTPS web server.

Syntax

```
{
    <format unique parameters>
    "stream": {
        "type": "http",
        "url": "URL_PATH"
    }
}
```

For example, to import a raw-format image from a file located at **http://download.ceph.com/image.raw**, the **source-spec** JSON file is:

Example

```
{
    "type": "raw",
    "stream": {
        "type": "http",
        "url": "http://download.ceph.com/image.raw"
    }
}
```

S3 stream

You can use the **s3** stream to import from a remote S3 bucket.

Syntax

```
{
    <format unique parameters>
    "stream": {
        "type": "s3",
        "url": "URL_PATH",
        "access_key": "ACCESS_KEY",
        "secret_key": "SECRET_KEY"
    }
}
```

For example, to import a raw-format image from a file located at **http://s3.ceph.com/bucket/image.raw**, its source-spec JSON is encoded as follows:

Example

```
{
    "type": "raw",
    "stream": {
        "type": "s3",
        "url": "http://s3.ceph.com/bucket/image.raw",
        "access_key": "NX5QOQKC6BH2IDN8HC7A",
        "secret_key": "LnEsqNNqZIpkzauboDcLXLcYaWwLQ3Kop0zAnKIn"
    }
}
```

3.4. PREPARING THE LIVE MIGRATION PROCESS

You can prepare the default live migration process for RBD images within the same Red Hat Ceph Storage cluster. The **rbd migration prepare** command accepts all the same layout options as the **rbd create** command. The **rbd create** command allows changes to the on-disk layout of the immutable image. If you only want to change the on-disk layout and want to keep the original image name, skip the **migration_target** argument. All clients using the source image must be stopped before preparing a live migration. The **prepare** step will fail if it finds any running clients with the image open in read/write mode. You can restart the clients using the new target image once the **prepare** step is completed.

NOTE

You cannot restart the clients using the source image as it will result in a failure.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Two block device pools.
- One block device image.

Procedure

1. Prepare the live migration within the storage cluster:

   Syntax

   ```
   rbd migration prepare SOURCE_POOL_NAME/SOURCE_IMAGE_NAME
   TARGET_POOL_NAME/SOURCE_IMAGE_NAME
   ```

   Example

   ```
   [ceph: root@rbd-client /]# rbd migration prepare sourcepool1/sourceimage1
   targetpool1/sourceimage1
   ```

   OR

   If you want to rename the source image:

   Syntax

   ```
   rbd migration prepare SOURCE_POOL_NAME/SOURCE_IMAGE_NAME
   TARGET_POOL_NAME/NEW_SOURCE_IMAGE_NAME
   ```

   Example

   ```
   [ceph: root@rbd-client /]# rbd migration prepare sourcepool1/sourceimage1
   targetpool1/newsourceimage1
   ```

   In the example, **newsourceimage1** is the renamed source image.

2. You can check the current state of the live migration process with the following command:

   Syntax

   ```
   rbd status TARGET_POOL_NAME/SOURCE_IMAGE_NAME
   ```

   Example

   ```
   [ceph: root@rbd-client /]# rbd status targetpool1/sourceimage1
   Watchers: none
   Migration:
   source: sourcepool1/sourceimage1 (adb429cb769a)
   destination: targetpool2/testimage1 (add299966c63)
   state: prepared
   ```

IMPORTANT

During the migration process, the source image is moved into the RBD trash to prevent mistaken usage.

Example

```
[ceph: root@rbd-client /]# rbd info sourceimage1
rbd: error opening image sourceimage1: (2) No such file or directory
```

Example

```
[ceph: root@rbd-client /]# rbd trash ls --all sourcepool1
adb429cb769a sourceimage1
```

3.5. PREPARING IMPORT-ONLY MIGRATION

You can initiate the **import-only** live migration process by running the **rbd migration prepare** command with the **--import-only** and either, **--source-spec** or **--source-spec-path** options, passing a JSON document that describes how to access the source image data directly on the command line or from a file.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- A bucket and an S3 object are created.

Procedure

1. Create a JSON file:

   Example

   ```
   [ceph: root@rbd-client /]# cat testspec.json
    {
      "type": "raw",
       "stream": {
           "type": "s3",
           "url": "http:10.74.253.18:80/testbucket1/image.raw",
          "access_key": "RLJOCP6345BGB38YQXI5",
          "secret_key": "oahWRB2ote2rnLy4dojYjDrsvaBADriDDgtSfk6o"
    }
   ```

2. Prepare the **import-only** live migration process:

   Syntax

   ```
   rbd migration prepare --import-only --source-spec-path "JSON_FILE"
   TARGET_POOL_NAME
   ```

   Example

```
[ceph: root@rbd-client /]# rbd migration prepare --import-only --source-spec-path
"testspec.json" targetpool1
```

NOTE

The **rbd migration prepare** command accepts all the same image options as the **rbd create** command.

3. You can check the status of the **import-only** live migration:

Example

```
[ceph: root@rbd-client /]# rbd status targetpool1/sourceimage1
Watchers: none
Migration:
source: {"stream":
{"access_key":"RLJOCP6345BGB38YQXI5","secret_key":"oahWRB2ote2rnLy4dojYjDrsvaBAD
riDDgtSfk6o","type":"s3","url":"http://10.74.253.18:80/testbucket1/image.raw"},"type":"raw"}
destination: targetpool1/sourceimage1 (b13865345e66)
state: prepared
```

3.6. EXECUTING THE LIVE MIGRATION PROCESS

After you prepare for the live migration, you must copy the image blocks from the source image to the target image.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Two block device pools.
- One block device image.

Procedure

1. Execute the live migration:

Syntax

```
rbd migration execute TARGET_POOL_NAME/SOURCE_IMAGE_NAME
```

Example

```
[ceph: root@rbd-client /]# rbd migration execute targetpool1/sourceimage1
Image migration: 100% complete...done.
```

2. You can check the feedback on the progress of the migration block deep-copy process:

Syntax

```
rbd status TARGET_POOL_NAME/SOURCE_IMAGE_NAME
```

-

Example

```
[ceph: root@rbd-client /]# rbd status targetpool1/sourceimage1
Watchers: none
Migration:
source: sourcepool1/testimage1 (adb429cb769a)
destination: targetpool1/testimage1 (add299966c63)
state: executed
```

3.7. COMMITTING THE LIVE MIGRATION PROCESS

You can commit the migration, once the live migration has completed deep-copying all the data blocks from the source image to the target image.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Two block device pools.
- One block device image.

Procedure

1. Commit the migration, once deep-copying is completed:

   Syntax

   ```
   rbd migration commit TARGET_POOL_NAME/SOURCE_IMAGE_NAME
   ```

   Example

   ```
   [ceph: root@rbd-client /]# rbd migration commit targetpool1/sourceimage1
   Commit image migration: 100% complete...done.
   ```

Verification

Committing the live migration will remove the cross-links between the source and target images, and also removes the source image from the source pool:

Example

```
[ceph: root@rbd-client /]# rbd trash list --all sourcepool1
```

3.8. ABORTING THE LIVE MIGRATION PROCESS

You can revert the live migration process. Aborting live migration reverts the prepare and execute steps.

NOTE

You can abort only if you have not committed the live migration.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Two block device pools.
- One block device image.

Procedure

1. Abort the live migration process:

   Syntax

   ```
   rbd migration abort TARGET_POOL_NAME/SOURCE_IMAGE_NAME
   ```

   Example

   ```
   [ceph: root@rbd-client /]# rbd migration abort targetpool1/sourceimage1
   Abort image migration: 100% complete...done.
   ```

Verification

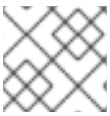
When the live migration process is aborted, the target image is deleted and access to the original source image is restored in the source pool:

Example

```
[ceph: root@rbd-client /]# rbd ls sourcepool1
sourceimage1
```

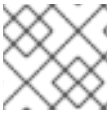
CHAPTER 4. IMAGE ENCRYPTION

As a storage administrator, you can set a secret key that is used to encrypt a specific RBD image. Image level encryption is handled internally by RBD clients.



NOTE

The **krbd** module does not support image level encryption.



NOTE

You can use external tools such as **dm-crypt** or **QEMU** to encrypt an RBD image.

Prerequisites

- A running Red Hat Ceph Storage 7 cluster.
- **root** level permissions.

4.1. ENCRYPTION FORMAT

RBD images are not encrypted by default. You can encrypt an RBD image by formatting to one of the supported encryption formats. The format operation persists the encryption metadata to the RBD image. The encryption metadata includes information such as the encryption format and version, cipher algorithm and mode specifications, as well as the information used to secure the encryption key.

The encryption key is protected by a user kept secret that is a passphrase, which is never stored as persistent data in the RBD image. The encryption format operation requires you to specify the encryption format, cipher algorithm, and mode specification as well as a passphrase. The encryption metadata is stored in the RBD image, currently as an encryption header that is written at the start of the raw image. This means that the effective image size of the encrypted image would be lower than the raw image size.

NOTE

Unless explicitly (re-)formatted, clones of an encrypted image are inherently encrypted using the same format and secret.

NOTE

Any data written to the RBD image before formatting might become unreadable, even though it might still occupy storage resources. RBD images with the journal feature enabled cannot be encrypted.

4.2. ENCRYPTION LOAD

By default, all RBD APIs treat encrypted RBD images the same way as unencrypted RBD images. You can read or write raw data anywhere in the image. Writing raw data into the image might risk the integrity of the encryption format. For example, the raw data could override the encryption metadata located at the beginning of the image. To safely perform encrypted Input/Output(I/O) or maintenance operations on the encrypted RBD image, an additional encryption load operation must be applied immediately after opening the image.

The encryption load operation requires you to specify the encryption format and a passphrase for unlocking the encryption key for the image itself and each of its explicitly formatted ancestor images. All I/Os for the opened RBD image are encrypted or decrypted, for a cloned RBD image, this includes IOs for the parent images. The encryption key is stored in memory by the RBD client until the image is closed.

NOTE

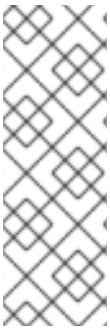
Once the encryption is loaded on the RBD image, no other encryption load or format operation can be applied. Additionally, API calls for retrieving the RBD image size and the parent overlap using the opened image context returns the effective image size and the effective parent overlap respectively. The encryption is loaded automatically when mapping the RBD images as block devices through **rbd-nbd**.

NOTE

API calls for retrieving the image size and the parent overlap using the opened image context returns the effective image size and the effective parent overlap.

NOTE

If a clone of an encrypted image is explicitly formatted, flattening or shrinking of the cloned image ceases to be transparent since the parent data must be re-encrypted according to the cloned image format as it is copied from the parent snapshot. If encryption is not loaded before the flatten operation is issued, any parent data that was previously accessible in the cloned image might become unreadable.



NOTE

If a clone of an encrypted image is explicitly formatted, the operation of shrinking the cloned image ceases to be transparent. This is because, in scenarios such as the cloned image containing snapshots or the cloned image being shrunk to a size that is not aligned with the object size, the action of copying some data from the parent snapshot, similar to flattening is involved. If encryption is not loaded before the shrink operation is issued, any parent data that was previously accessible in the cloned image might become unreadable.

4.3. SUPPORTED FORMATS

Both Linux Unified Key Setup (LUKS) 1 and 2 are supported. The data layout is fully compliant with the LUKS specification. External LUKS compatible tools such as **dm-crypt** or **QEMU** can safely perform encrypted Input/Output (I/O) on encrypted RBD images. Additionally, you can import existing LUKS images created by external tools, by copying the raw LUKS data into the RBD image.

Currently, only Advanced Encryption Standards (AES) 128 and 256 encryption algorithms are supported. xts-plain64 is currently the only supported encryption mode.

To use the LUKS format, format the RBD image with the following command:

NOTE

You need to create a file named **passphrase.txt** and enter a passphrase. You can randomly generate the passphrase, which might contain NULL characters. If the passphrase ends with a newline character, it is stripped off.

Syntax

```
rbd encryption format POOL_NAME/LUKS_IMAGE luks1|luks2 PASSPHRASE_FILE
```

Example

```
[ceph: root@host01 /]# rbd encryption format pool1/luksimage1 luks1 passphrase.bin
```

NOTE

You can select either **luks1** or **luks** encryption format.

The encryption format operation generates a LUKS header and writes it at the start of the RBD image. A single keyslot is appended to the header. The keyslot holds a randomly generated encryption key, and is protected by the passphrase read from the passphrase file. By default, AES-256 in xts-plain64 mode, which is the current recommended mode and the default for other LUKS tools, is used. Adding or removing additional passphrases is currently not supported natively, but can be achieved using LUKS tools such as **cryptsetup**. The LUKS header size can vary that is upto 136MiB in LUKS, but it is usually upto 16MiB, dependent on the version of **libcryptsetup** installed. For optimal performance, the encryption format sets the data offset to be aligned with the image object size. For example, expect a minimum overhead of 8MiB if using an image configured with an 8MiB object size.

In LUKS1, sectors, which are the minimal encryption units, are fixed at 512 bytes. LUKS2 supports larger sectors, and for better performance, the default sector size is set to the maximum of 4KiB. Writes which are either smaller than a sector, or are not aligned to a sector start, trigger a guarded **read-modify-write** chain on the client, with a considerable latency penalty. A batch of such unaligned writes can lead to I/O races which further deteriorates performance. Red Hat recommends to avoid using RBD encryption in cases where incoming writes cannot be guaranteed to be LUKS sector aligned.

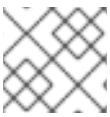
To map a LUKS encrypted image, run the following command:

Syntax

```
rbd device map -t nbd -o encryption-format=luks1|luks2,encryption-passphrase-file=passphrase.txt
POOL_NAME/LUKS_IMAGE
```

Example

```
[ceph: root@host01 /]# rbd device map -t nbd -o encryption-format=luks1,encryption-passphrase-
file=passphrase.txt pool1/luksimage1
```



NOTE

You can select either **luks1** or **luks2** encryption format.

NOTE

For security reasons, both the encryption format and encryption load operations are CPU-intensive, and might take a few seconds to complete. For encrypted I/O, assuming AES-NI is enabled, a relatively small microseconds latency might be added, as well as a small increase in CPU utilization.

4.4. ADDING ENCRYPTION FORMAT TO IMAGES AND CLONES

Layered-client-side encryption is supported. The cloned images can be encrypted with their own format and passphrase, potentially different from that of the parent image.

Add encryption format to images and clones with the **rbd encryption format** command. Given a LUKS2-formatted image, you can create both a LUKS2-formatted clone and a LUKS1-formatted clone.

Prerequisites

- A running Red Hat Ceph Storage cluster with Block Device (RBD) configured.
- Root-level access to the node.

Procedure

1. Create a LUKS2-formatted image:

   #### Syntax

   ```
   rbd create --size SIZE POOL_NAME/LUKS_IMAGE
   rbd encryption format POOL_NAME/LUKS_IMAGE luks1|luks2 PASSPHRASE_FILE
   rbd resize --size 50G --encryption-passphrase-file PASSPHRASE_FILE
   POOL_NAME/LUKS_IMAGE
   ```

   #### Example

   ```
   [ceph: root@host01 /]# rbd create --size 50G mypool/myimage
   [ceph: root@host01 /]# rbd encryption format mypool/myimage luks2 passphrase.txt
   [ceph: root@host01 /]# rbd resize --size 50G --encryption-passphrase-file passphrase.txt
   mypool/myimage
   ```

   The **rbd resize** command grows the image to compensate for the overhead associated with the LUKS2 header.

2. With the LUKS2-formatted image, create a LUKS2-formatted clone with the same effective size:

   #### Syntax

   ```
   rbd snap create POOL_NAME/IMAGE_NAME@SNAP_NAME
   rbd snap protect POOL_NAME/IMAGE_NAME@SNAP_NAME
   rbd clone POOL_NAME/IMAGE_NAME@SNAP_NAME POOL_NAME/CLONE_NAME
   rbd encryption format POOL_NAME/CLONE_NAME luks1 CLONE_PASSPHRASE_FILE
   ```

   #### Example

   ```
   [ceph: root@host01 /]# rbd snap create mypool/myimage@snap
   [ceph: root@host01 /]# rbd snap protect mypool/myimage@snap
   [ceph: root@host01 /]# rbd clone mypool/myimage@snap mypool/myclone
   [ceph: root@host01 /]# rbd encryption format mypool/myclone luks1 clone-passphrase.bin
   ```

3. With the LUKS2-formatted image, create a LUKS1-formatted clone with the same effective size:

Syntax

```
rbd snap create POOL_NAME/IMAGE_NAME@SNAP_NAME
rbd snap protect POOL_NAME/IMAGE_NAME@SNAP_NAME
rbd clone POOL_NAME/IMAGE_NAME@SNAP_NAME POOL_NAME/CLONE_NAME
rbd encryption format POOL_NAME/CLONE_NAME luks1 CLONE_PASSPHRASE_FILE
rbd resize --size SIZE --allow-shrink --encryption-passphrase-file
CLONE_PASSPHRASE_FILE --encryption-passphrase-file PASSPHRASE_FILE
POOL_NAME/CLONE_NAME
```

Example

```
[ceph: root@host01 /]# rbd snap create mypool/myimage@snap
[ceph: root@host01 /]# rbd snap protect mypool/myimage@snap
[ceph: root@host01 /]# rbd clone mypool/myimage@snap mypool/myclone
[ceph: root@host01 /]# rbd encryption format mypool/myclone luks1 clone-passphrase.bin
[ceph: root@host01 /]# rbd resize --size 50G --allow-shrink --encryption-passphrase-file
clone-passphrase.bin --encryption-passphrase-file passphrase.bin mypool/myclone
```

Since LUKS1 header is usually smaller than LUKS2 header, the **rbd resize** command at the end shrinks the cloned image to get rid of unwanted space allowance.

4. With the LUKS-1-formatted image, create a LUKS2-formatted clone with the same effective size:

Syntax

```
rbd resize --size SIZE POOL_NAME/LUKS_IMAGE
rbd snap create POOL_NAME/IMAGE_NAME@SNAP_NAME
rbd snap protect POOL_NAME/IMAGE_NAME@SNAP_NAME
rbd clone POOL_NAME/IMAGE_NAME@SNAP_NAME POOL_NAME/CLONE_NAME
rbd encryption format POOL_NAME/CLONE_NAME luks2 CLONE_PASSPHRASE_FILE
rbd resize --size SIZE --allow-shrink --encryption-passphrase-file PASSPHRASE_FILE
POOL_NAME/LUKS_IMAGE
rbd resize --size SIZE --allow-shrink --encryption-passphrase-file
CLONE_PASSPHRASE_FILE --encryption-passphrase-file PASSPHRASE_FILE
POOL_NAME_/CLONE_NAME
```

Example

```
[ceph: root@host01 /]# rbd resize --size 51G mypool/myimage
[ceph: root@host01 /]# rbd snap create mypool/myimage@snap
[ceph: root@host01 /]# rbd snap protect mypool/myimage@snap
[ceph: root@host01 /]# rbd clone mypool/my-image@snap mypool/myclone
[ceph: root@host01 /]# rbd encryption format mypool/myclone luks2 clone-passphrase.bin
[ceph: root@host01 /]# rbd resize --size 50G --allow-shrink --encryption-passphrase-file
passphrase.bin mypool/myimage
[ceph: root@host01 /]# rbd resize --size 50G --allow-shrink --encryption-passphrase-file
clone-passphrase.bin --encryption-passphrase-file passphrase.bin mypool/myclone
```

Since LUKS2 header is usually bigger than LUKS1 header, the **rbd resize** command at the

beginning temporarily grows the parent image to reserve some extra space in the parent snapshot and consequently the cloned image. This is necessary to make all parent data accessible in the cloned image. The **rbd resize** command at the end shrinks the parent image back to its original size and does not impact the parent snapshot and the cloned image to get rid of the unused reserved space

The same applies to creating a formatted clone of an unformatted image, since an unformatted image does not have a header at all.

Additional Resources

- See the *Configuring Ansible inventory location* section in the *Red Hat Ceph Storage Installation Guide* for more details on adding clients to the **cephadm-ansible** inventory.

CHAPTER 5. MANAGING SNAPSHOTS

As a storage administrator, being familiar with Ceph's snapshotting feature can help you manage the snapshots and clones of images stored in the Red Hat Ceph Storage cluster.

Prerequisites

- A running Red Hat Ceph Storage cluster.

5.1. CEPH BLOCK DEVICE SNAPSHOTS

A snapshot is a read-only copy of the state of an image at a particular point in time. One of the advanced features of Ceph block devices is that you can create snapshots of the images to retain a history of an image's state. Ceph also supports snapshot layering, which allows you to clone images quickly and easily, for example a virtual machine image. Ceph supports block device snapshots using the **rbd** command and many higher level interfaces, including **QEMU**, **libvirt**, OpenStack and CloudStack.

NOTE

If a snapshot is taken while **I/O** is occurring, then the snapshot might not get the exact or latest data of the image and the snapshot might have to be cloned to a new image to be mountable. Red Hat recommends stopping **I/O** before taking a snapshot of an image. If the image contains a filesystem, the filesystem must be in a consistent state before taking a snapshot. To stop **I/O** you can use **fsfreeze** command. For virtual machines, the **qemu-guest-agent** can be used to automatically freeze filesystems when creating a snapshot.

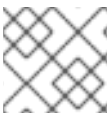
Figure 5.1. Ceph Block device snapshots

Additional Resources

- See the **fsfreeze(8)** man page for more details.

5.2. THE CEPH USER AND KEYRING

When **cephx** is enabled, you must specify a user name or ID and a path to the keyring containing the corresponding key for the user.



NOTE

cephx is enabled by default.

You might also add the **CEPH_ARGS** environment variable to avoid re-entry of the following parameters:

Syntax

```
rbd --id USER_ID --keyring=/path/to/secret [commands]
rbd --name USERNAME --keyring=/path/to/secret [commands]
```

Example

```
[root@rbd-client ~]# rbd --id admin --keyring=/etc/ceph/ceph.keyring [commands]
[root@rbd-client ~]# rbd --name client.admin --keyring=/etc/ceph/ceph.keyring [commands]
```

TIP

Add the user and secret to the **CEPH_ARGS** environment variable so that you do not need to enter them each time.

5.3. CREATING A BLOCK DEVICE SNAPSHOT

Create a snapshot of a Ceph block device.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. Specify the **snap create** option, the pool name and the image name:
   - Method 1:

     Syntax

     ```
     rbd --pool POOL_NAME snap create --snap SNAP_NAME IMAGE_NAME
     ```

     Example

     ```
     [root@rbd-client ~]# rbd --pool pool1 snap create --snap snap1 image1
     ```

   - Method 2:

     Syntax

     ```
     rbd snap create POOL_NAME/IMAGE_NAME@SNAP_NAME
     ```

     Example

     ```
     [root@rbd-client ~]# rbd snap create pool1/image1@snap1
     ```

5.4. LISTING THE BLOCK DEVICE SNAPSHOTS

List the block device snapshots.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. Specify the pool name and the image name:

   Syntax

   ```
   rbd --pool POOL_NAME --image IMAGE_NAME snap ls
   rbd snap ls POOL_NAME/IMAGE_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd --pool pool1 --image image1 snap ls
   [root@rbd-client ~]# rbd snap ls pool1/image1
   ```

5.5. ROLLING BACK A BLOCK DEVICE SNAPSHOT

Rollback a block device snapshot.

NOTE

Rolling back an image to a snapshot means overwriting the current version of the image with data from a snapshot. The time it takes to execute a rollback increases with the size of the image. It is **faster to clone** from a snapshot **than to rollback** an image to a snapshot, and it is the preferred method of returning to a pre-existing state.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. Specify the **snap rollback** option, the pool name, the image name and the snap name:

   Syntax

   ```
   rbd --pool POOL_NAME snap rollback --snap SNAP_NAME IMAGE_NAME
   rbd snap rollback POOL_NAME/IMAGE_NAME@SNAP_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd --pool pool1 snap rollback --snap snap1 image1
   [root@rbd-client ~]# rbd snap rollback pool1/image1@snap1
   ```

5.6. DELETING A BLOCK DEVICE SNAPSHOT

Delete a snapshot for Ceph block devices.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. To delete a block device snapshot, specify the **snap rm** option, the pool name, the image name and the snapshot name:

   #### Syntax

   ```
   rbd --pool POOL_NAME snap rm --snap SNAP_NAME IMAGE_NAME
   rbd snap rm POOL_NAME-/IMAGE_NAME@SNAP_NAME
   ```

   #### Example

   ```
   [root@rbd-client ~]# rbd --pool pool1 snap rm --snap snap2 image1
   [root@rbd-client ~]# rbd snap rm pool1/image1@snap1
   ```

IMPORTANT

If an image has any clones, the cloned images retain reference to the parent image snapshot. To delete the parent image snapshot, you must flatten the child images first.

NOTE

Ceph OSD daemons delete data asynchronously, so deleting a snapshot does not free up the disk space immediately.

Additional Resources

- See the *Flattening cloned images* in the *Red Hat Ceph Storage Block Device Guide* for details.

5.7. PURGING THE BLOCK DEVICE SNAPSHOTS

Purge block device snapshots.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. Specify the **snap purge** option and the image name on a specific pool:

   #### Syntax

```
rbd --pool POOL_NAME snap purge IMAGE_NAME
rbd snap purge POOL_NAME/IMAGE_NAME
```

Example

```
[root@rbd-client ~]# rbd --pool pool1 snap purge image1
[root@rbd-client ~]# rbd snap purge pool1/image1
```

5.8. RENAMING A BLOCK DEVICE SNAPSHOT

Rename a block device snapshot.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. To rename a snapshot:

   Syntax

   ```
   rbd snap rename POOL_NAME/IMAGE_NAME@ORIGINAL_SNAPSHOT_NAME
   POOL_NAME/IMAGE_NAME@NEW_SNAPSHOT_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd snap rename data/dataset@snap1 data/dataset@snap2
   ```

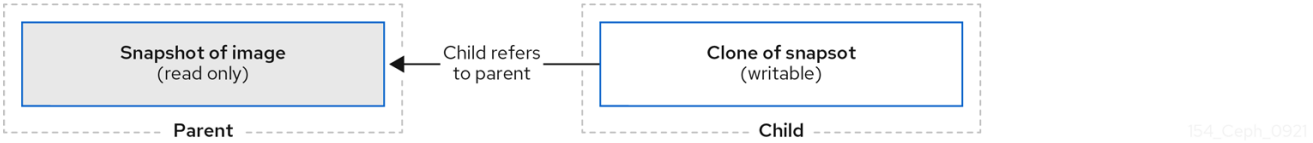
   This renames **snap1** snapshot of the **dataset** image on the **data** pool to **snap2**.

2. Execute the **rbd help snap rename** command to display additional details on renaming snapshots.

5.9. CEPH BLOCK DEVICE LAYERING

Ceph supports the ability to create many copy-on-write (COW) or copy-on-read (COR) clones of a block device snapshot. Snapshot layering enables Ceph block device clients to create images very quickly. For example, you might create a block device image with a Linux VM written to it. Then, snapshot the image, protect the snapshot, and create as many clones as you like. A snapshot is read-only, so cloning a snapshot simplifies semantics—making it possible to create clones rapidly.

Figure 5.2. Ceph Block device layering



NOTE

The terms **parent** and **child** mean a Ceph block device snapshot, parent, and the corresponding image cloned from the snapshot, child. These terms are important for the command line usage below.

Each cloned image, the child, stores a reference to its parent image, which enables the cloned image to open the parent snapshot and read it. This reference is removed when the clone is **flattened** that is, when information from the snapshot is completely copied to the clone.

A clone of a snapshot behaves exactly like any other Ceph block device image. You can read to, write from, clone, and resize the cloned images. There are no special restrictions with cloned images. However, the clone of a snapshot refers to the snapshot, so you **MUST** protect the snapshot before you clone it.

A clone of a snapshot can be a copy-on-write (COW) or copy-on-read (COR) clone. Copy-on-write (COW) is always enabled for clones while copy-on-read (COR) has to be enabled explicitly. Copy-on-write (COW) copies data from the parent to the clone when it writes to an unallocated object within the clone. Copy-on-read (COR) copies data from the parent to the clone when it reads from an unallocated object within the clone. Reading data from a clone will only read data from the parent if the object does not yet exist in the clone. Rados block device breaks up large images into multiple objects. The default is set to 4 MB and all copy-on-write (COW) and copy-on-read (COR) operations occur on a full object, that is writing 1 byte to a clone will result in a 4 MB object being read from the parent and written to the clone if the destination object does not already exist in the clone from a previous COW/COR operation.

Whether or not copy-on-read (COR) is enabled, any reads that cannot be satisfied by reading an underlying object from the clone will be rerouted to the parent. Since there is practically no limit to the number of parents, meaning that you can clone a clone, this reroute continues until an object is found or you hit the base parent image. If copy-on-read (COR) is enabled, any reads that fail to be satisfied directly from the clone result in a full object read from the parent and writing that data to the clone so that future reads of the same extent can be satisfied from the clone itself without the need of reading from the parent.

This is essentially an on-demand, object-by-object flatten operation. This is specially useful when the clone is in a high-latency connection away from it's parent, that is the parent in a different pool, in another geographical location. Copy-on-read (COR) reduces the amortized latency of reads. The first few reads will have high latency because it will result in extra data being read from the parent, for example, you read 1 byte from the clone but now 4 MB has to be read from the parent and written to the clone, but all future reads will be served from the clone itself.

To create copy-on-read (COR) clones from snapshot you have to explicitly enable this feature by adding **rbd_clone_copy_on_read = true** under **[global]** or **[client]** section in the **ceph.conf** file.

Additional Resources

- For more information on **flattening**, see the *Flattening cloned images* section in the *Red Hat Ceph Storage Block Device Gudie*.

5.10. PROTECTING A BLOCK DEVICE SNAPSHOT

Clones access the parent snapshots. All clones would break if a user inadvertently deleted the parent snapshot.

You can set the **set-require-min-compat-client** parameter to greater than or equal to mimic versions of Ceph.

Example

```
ceph osd set-require-min-compat-client mimic
```

This creates clone v2, by default. However, clients older than mimic cannot access those block device images.

NOTE

Clone v2 does not require protection of snapshots.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

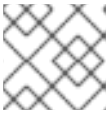
1. Specify ***POOL_NAME***, ***IMAGE_NAME***, and ***SNAP_SHOT_NAME*** in the following command:

   #### Syntax

   ```
   rbd --pool POOL_NAME snap protect --image IMAGE_NAME --snap SNAPSHOT_NAME
   rbd snap protect POOL_NAME/IMAGE_NAME@SNAPSHOT_NAME
   ```

   #### Example

   ```
   [root@rbd-client ~]# rbd --pool pool1 snap protect --image image1 --snap snap1
   [root@rbd-client ~]# rbd snap protect pool1/image1@snap1
   ```

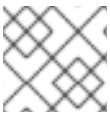
   

   NOTE

   You cannot delete a protected snapshot.

5.11. CLONING A BLOCK DEVICE SNAPSHOT

Clone a block device snapshot to create a read or write child image of the snapshot within the same pool or in another pool. One use case would be to maintain read-only images and snapshots as templates in one pool, and writable clones in another pool.



NOTE

Clone v2 does not require protection of snapshots.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. To clone a snapshot, you need to specify the parent pool, snapshot, child pool and image name:

   Syntax

   ```
   rbd snap --pool POOL_NAME --image PARENT_IMAGE --snap SNAP_NAME --dest-pool
   POOL_NAME --dest CHILD_IMAGE_NAME
   rbd clone POOL_NAME/PARENT_IMAGE@SNAP_NAME
   POOL_NAME/CHILD_IMAGE_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd clone --pool pool1 --image image1 --snap snap2 --dest-pool pool2 --
   dest childimage1
   [root@rbd-client ~]# rbd clone pool1/image1@snap1 pool1/childimage1
   ```

5.12. UNPROTECTING A BLOCK DEVICE SNAPSHOT

Before you can delete a snapshot, you must unprotect it first. Additionally, you may *NOT* delete snapshots that have references from clones. You must flatten each clone of a snapshot, before you can delete the snapshot.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. Run the following commands:

   Syntax

   ```
   rbd --pool POOL_NAME snap unprotect --image IMAGE_NAME --snap SNAPSHOT_NAME
   rbd snap unprotect POOL_NAME/IMAGE_NAME@SNAPSHOT_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd --pool pool1 snap unprotect --image image1 --snap snap1

   [root@rbd-client ~]# rbd snap unprotect pool1/image1@snap1
   ```

5.13. LISTING THE CHILDREN OF A SNAPSHOT

List the children of a snapshot.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. To list the children of a snapshot, execute the following:

   Syntax

   ```
   rbd --pool POOL_NAME children --image IMAGE_NAME --snap SNAP_NAME
   rbd children POOL_NAME/IMAGE_NAME@SNAPSHOT_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd --pool pool1 children --image image1 --snap snap1
   [root@rbd-client ~]# rbd children pool1/image1@snap1
   ```

5.14. FLATTENING CLONED IMAGES

Cloned images retain a reference to the parent snapshot. When you remove the reference from the child clone to the parent snapshot, you effectively "flatten" the image by copying the information from the snapshot to the clone. The time it takes to flatten a clone increases with the size of the snapshot. Because a flattened image contains all the information from the snapshot, a flattened image will use more storage space than a layered clone.

NOTE

If the **deep flatten** feature is enabled on an image, the image clone is dissociated from its parent by default.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. To delete a parent image snapshot associated with child images, you must flatten the child images first:

   Syntax

   ```
   rbd --pool POOL_NAME flatten --image IMAGE_NAME
   rbd flatten POOL_NAME/IMAGE_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd --pool pool1 flatten --image childimage1
   [root@rbd-client ~]# rbd flatten pool1/childimage1
   ```

CHAPTER 6. MIRRORING CEPH BLOCK DEVICES

As a storage administrator, you can add another layer of redundancy to Ceph block devices by mirroring data images between Red Hat Ceph Storage clusters. Understanding and using Ceph block device mirroring can provide you protection against data loss, such as a site failure. There are two configurations for mirroring Ceph block devices, one-way mirroring or two-way mirroring, and you can configure mirroring on pools and individual images.

Prerequisites

- A minimum of two healthy running Red Hat Ceph Storage clusters.
- Network connectivity between the two storage clusters.
- Access to a Ceph client node for each Red Hat Ceph Storage cluster.
- A CephX user with administrator-level capabilities.

6.1. CEPH BLOCK DEVICE MIRRORING

RADOS Block Device (RBD) mirroring is a process of asynchronous replication of Ceph block device images between two or more Ceph storage clusters. By locating a Ceph storage cluster in different geographic locations, RBD Mirroring can help you recover from a site disaster. Journal-based Ceph block device mirroring ensures point-in-time consistent replicas of all changes to an image, including reads and writes, block device resizing, snapshots, clones and flattening.

RBD mirroring uses exclusive locks and the journaling feature to record all modifications to an image in the order in which they occur. This ensures that a crash-consistent mirror of an image is available.

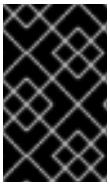
IMPORTANT

The CRUSH hierarchies supporting primary and secondary pools that mirror block device images must have the same capacity and performance characteristics, and must have adequate bandwidth to ensure mirroring without excess latency. For example, if you have X MB/s average write throughput to images in the primary storage cluster, the network must support N * X throughput in the network connection to the secondary site plus a safety factor of Y% to mirror N images.

The **rbd-mirror** daemon is responsible for synchronizing images from one Ceph storage cluster to another Ceph storage cluster by pulling changes from the remote primary image and writes those changes to the local, non-primary image. The **rbd-mirror** daemon can run either on a single Ceph storage cluster for one-way mirroring or on two Ceph storage clusters for two-way mirroring that participate in the mirroring relationship.

For RBD mirroring to work, either using one-way or two-way replication, a couple of assumptions are made:

- A pool with the same name exists on both storage clusters.
- A pool contains journal-enabled images you want to mirror.



IMPORTANT

In one-way or two-way replication, each instance of **rbd-mirror** must be able to connect to the other Ceph storage cluster simultaneously. Additionally, the network must have sufficient bandwidth between the two data center sites to handle mirroring.

One-way Replication

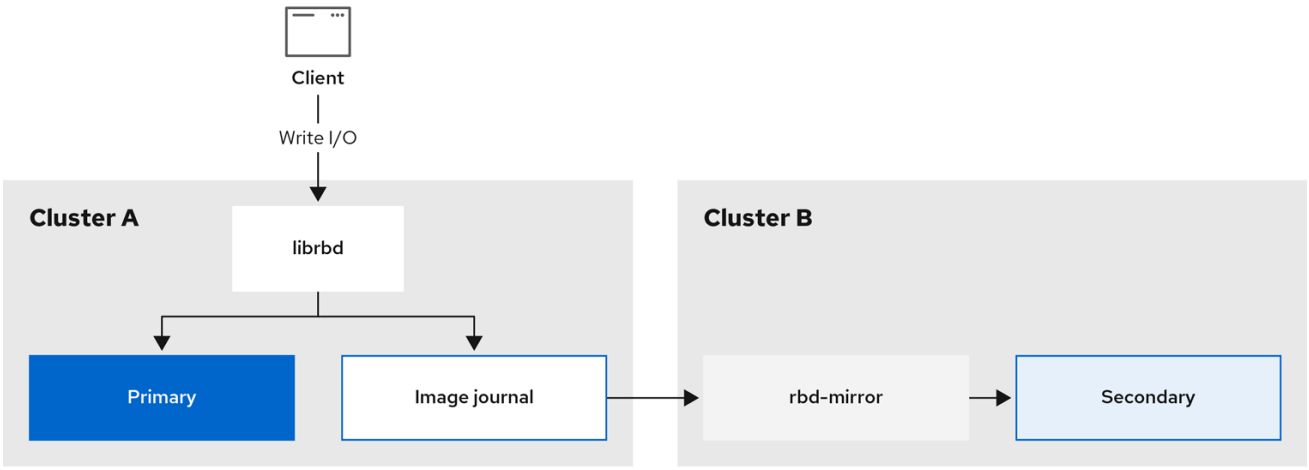
One-way mirroring implies that a primary image or pool of images in one storage cluster gets replicated to a secondary storage cluster. One-way mirroring also supports replicating to multiple secondary storage clusters.

On the secondary storage cluster, the image is the non-primary replicate; that is, Ceph clients cannot write to the image. When data is mirrored from a primary storage cluster to a secondary storage cluster, the **rbd-mirror** runs ONLY on the secondary storage cluster.

For one-way mirroring to work, a couple of assumptions are made:

- You have two Ceph storage clusters and you want to replicate images from a primary storage cluster to a secondary storage cluster.
- The secondary storage cluster has a Ceph client node attached to it running the **rbd-mirror** daemon. The **rbd-mirror** daemon will connect to the primary storage cluster to sync images to the secondary storage cluster.

Figure 6.1. One-way mirroring



Two-way Replication

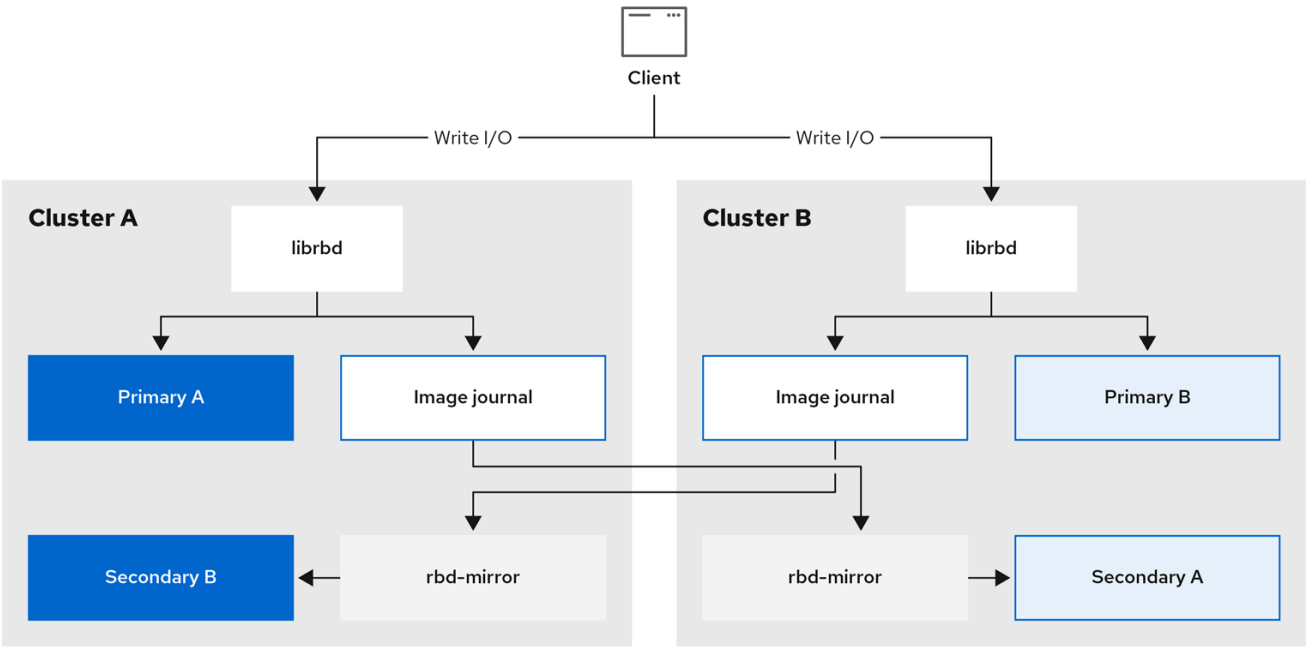
Two-way replication adds an **rbd-mirror** daemon on the primary cluster so images can be demoted on it and promoted on the secondary cluster. Changes can then be made to the images on the secondary cluster and they will be replicated in the reverse direction, from secondary to primary. Both clusters must have **rbd-mirror** running to allow promoting and demoting images on either cluster. Currently, two-way replication is only supported between two sites.

For two-way mirroring to work, a couple of assumptions are made:

- You have two storage clusters and you want to be able to replicate images between them in either direction.
- Both storage clusters have a client node attached to them running the **rbd-mirror** daemon. The

rbd-mirror daemon running on the secondary storage cluster will connect to the primary storage cluster to synchronize images to secondary, and the **rbd-mirror** daemon running on the primary storage cluster will connect to the secondary storage cluster to synchronize images to primary.

Figure 6.2. Two-way mirroring



Mirroring Modes

Mirroring is configured on a per-pool basis with mirror peering storage clusters. Ceph supports two mirroring modes, depending on the type of images in the pool.

Pool Mode

All images in a pool with the journaling feature enabled are mirrored.

Image Mode

Only a specific subset of images within a pool are mirrored. You must enable mirroring for each image separately.

Image States

Whether or not an image can be modified depends on its state:

- Images in the primary state can be modified.
- Images in the non-primary state cannot be modified.

Images are automatically promoted to primary when mirroring is first enabled on an image. The promotion can happen:

- Implicitly by enabling mirroring in pool mode.
- Explicitly by enabling mirroring of a specific image.

It is possible to demote primary images and promote non-primary images.

Additional Resources

- See the *Enabling mirroring on a pool* section of the *Red Hat Ceph Storage Block Device Guide* for more details.
- See the *Enabling image mirroring* section of the *Red Hat Ceph Storage Block Device Guide* for more details.
- See the *Image promotion and demotion* section of the *Red Hat Ceph Storage Block Device Guide* for more details.

6.1.1. An overview of journal-based and snapshot-based mirroring

RADOS Block Device (RBD) images can be asynchronously mirrored between two Red Hat Ceph Storage clusters through two modes:

Journal-based mirroring

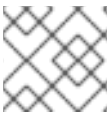
This mode uses the RBD journaling image feature to ensure point-in-time and crash consistent replication between two Red Hat Ceph Storage clusters. The actual image is not modified until every write to the RBD image is first recorded to the associated journal. The remote cluster reads from this journal and replays the updates to its local copy of the image. Because each write to the RBD images results in two writes to the Ceph cluster, write latencies nearly double with the usage of the RBD journaling image feature.

Snapshot-based mirroring

This mode uses periodic scheduled or manually created RBD image mirror snapshots to replicate crash consistent RBD images between two Red Hat Ceph Storage clusters. The remote cluster determines any data or metadata updates between two mirror snapshots and copies the deltas to its local copy of the image. The RBD **fast-diff** image feature enables the quick determination of updated data blocks without the need to scan the full RBD image. The complete delta between two snapshots needs to be synchronized prior to use during a failover scenario. Any partially applied set of deltas are rolled back at moment of failover.

6.2. CONFIGURING ONE-WAY MIRRORING USING THE COMMAND-LINE INTERFACE

This procedure configures one-way replication of a pool from the primary storage cluster to a secondary storage cluster.



NOTE

When using one-way replication you can mirror to multiple secondary storage clusters.

NOTE

Examples in this section will distinguish between two storage clusters by referring to the primary storage cluster with the primary images as **site-a**, and the secondary storage cluster you are replicating the images to, as **site-b**. The pool name used in these examples is called **data**.

Prerequisites

- A minimum of two healthy and running Red Hat Ceph Storage clusters.

- Root-level access to a Ceph client node for each storage cluster.
- A CephX user with administrator-level capabilities.

Procedure

1. Log into the **cephadm** shell on both the sites:

   Example

   ```
   [root@site-a ~]# cephadm shell
   [root@site-b ~]# cephadm shell
   ```

2. On **site-b**, schedule the deployment of mirror daemon on the secondary cluster:

   Syntax

   ```
   ceph orch apply rbd-mirror --placement=NODENAME
   ```

   Example

   ```
   [ceph: root@site-b /]# ceph orch apply rbd-mirror --placement=host04
   ```

   NOTE

   The **nodename** is the host where you want to configure mirroring in the secondary cluster.

3. Enable journaling features on an image on **site-a**.

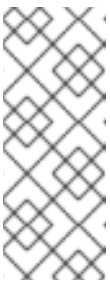
   a. For **new images**, use the **--image-feature** option:

      Syntax

      ```
      rbd create IMAGE_NAME --size MEGABYTES --pool POOL_NAME --image-feature
      FEATURE FEATURE
      ```

      Example

      ```
      [ceph: root@site-a /]# rbd create image1 --size 1024 --pool data --image-feature
      exclusive-lock,journaling
      ```

      NOTE

      If **exclusive-lock** is already enabled, use **journaling** as the only argument, else it returns the following error:

      ```
      one or more requested features are already enabled
      (22) Invalid argument
      ```

   b. For **existing images**, use the **rbd feature enable** command:

Syntax

```
rbd feature enable POOL_NAME/IMAGE_NAME FEATURE, FEATURE
```

Example

```
[ceph: root@site-a /]# rbd feature enable data/image1 exclusive-lock, journaling
```

c. To enable journaling on all new images by default, set the configuration parameter using **ceph config set** command:

Example

```
[ceph: root@site-a /]# ceph config set global rbd_default_features 125
[ceph: root@site-a /]# ceph config show mon.host01 rbd_default_features
```

4. Choose the mirroring mode, either pool or image mode, on both the storage clusters.

a. Enabling **pool mode**:

Syntax

```
rbd mirror pool enable POOL_NAME MODE
```

Example

```
[ceph: root@site-a /]# rbd mirror pool enable data pool
[ceph: root@site-b /]# rbd mirror pool enable data pool
```

This example enables mirroring of the whole pool named **data**.

b. Enabling **image mode**:

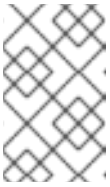
Syntax

```
rbd mirror pool enable POOL_NAME MODE
```

Example

```
[ceph: root@site-a /]# rbd mirror pool enable data image
[ceph: root@site-b /]# rbd mirror pool enable data image
```

This example enables image mode mirroring on the pool named **data**.



NOTE

To enable mirroring on specific images in a pool, see the *Enabling image mirroring* section in the *Red Hat Ceph Storage Block Device Guide* for more details.

c. Verify that mirroring has been successfully enabled at both the sites:

Syntax

```
rbd mirror pool info POOL_NAME
```

Example

```
[ceph: root@site-a /]# rbd mirror pool info data
Mode: pool
Site Name: c13d8065-b33d-4cb5-b35f-127a02768e7f

Peer Sites: none

[ceph: root@site-b /]# rbd mirror pool info data
Mode: pool
Site Name: a4c667e2-b635-47ad-b462-6faeeee78df7

Peer Sites: none
```

5. On a Ceph client node, bootstrap the storage cluster peers.

   a. Create Ceph user accounts, and register the storage cluster peer to the pool:

   Syntax

   ```
   rbd mirror pool peer bootstrap create --site-name PRIMARY_LOCAL_SITE_NAME POOL_NAME > PATH_TO_BOOTSTRAP_TOKEN
   ```

   Example

   ```
   [ceph: root@rbd-client-site-a /]# rbd mirror pool peer bootstrap create --site-name site-a data > /root/bootstrap_token_site-a
   ```

   NOTE

   This example bootstrap command creates the **client.rbd-mirror.site-a** and the **client.rbd-mirror-peer** Ceph users.

   b. Copy the bootstrap token file to the **site-b** storage cluster.

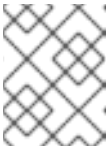
   c. Import the bootstrap token on the **site-b** storage cluster:

   Syntax

   ```
   rbd mirror pool peer bootstrap import --site-name SECONDARY_LOCAL_SITE_NAME --direction rx-only POOL_NAME PATH_TO_BOOTSTRAP_TOKEN
   ```

   Example

   ```
   [ceph: root@rbd-client-site-b /]# rbd mirror pool peer bootstrap import --site-name site-b --direction rx-only data /root/bootstrap_token_site-a
   ```



NOTE

For one-way RBD mirroring, you must use the **--direction rx-only** argument, as two-way mirroring is the default when bootstrapping peers.

6. To verify the mirroring status, run the following command from a Ceph Monitor node on the primary and secondary sites:

Syntax

```
rbd mirror image status POOL_NAME/IMAGE_NAME
```

Example

```
[ceph: root@mon-site-a /]# rbd mirror image status data/image1
image1:
  global_id:   c13d8065-b33d-4cb5-b35f-127a02768e7f
  state:       up+stopped
  description: remote image is non-primary
  service:     host03.yuoosv on host03
  last_update: 2021-10-06 09:13:58
```

Here, **up** means the **rbd-mirror** daemon is running, and **stopped** means this image is not the target for replication from another storage cluster. This is because the image is primary on this storage cluster.

Example

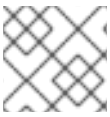
```
[ceph: root@mon-site-b /]# rbd mirror image status data/image1
image1:
  global_id:   c13d8065-b33d-4cb5-b35f-127a02768e7f
```

Additional Resources

- See the *Ceph block device mirroring* section in the *Red Hat Ceph Storage Block Device Guide* for more details.
- See the *User Management* section in the *Red Hat Ceph Storage Administration Guide* for more details on Ceph users.

6.3. CONFIGURING TWO-WAY MIRRORING USING THE COMMAND-LINE INTERFACE

This procedure configures two-way replication of a pool between the primary storage cluster, and a secondary storage cluster.



NOTE

When using two-way replication you can only mirror between two storage clusters.

NOTE

Examples in this section will distinguish between two storage clusters by referring to the primary storage cluster with the primary images as **site-a**, and the secondary storage cluster you are replicating the images to, as **site-b**. The pool name used in these examples is called **data**.

Prerequisites

- A minimum of two healthy and running Red Hat Ceph Storage clusters.
- Root-level access to a Ceph client node for each storage cluster.
- A CephX user with administrator-level capabilities.

Procedure

1. Log into the **cephadm** shell on both the sites:

   Example

   ```
   [root@site-a ~]# cephadm shell
   [root@site-b ~]# cephadm shell
   ```

2. On the **site-a** primary cluster, run the following command:

   Example

   ```
   [ceph: root@site-a /]# ceph orch apply rbd-mirror --placement=host01
   ```

   NOTE

   The **nodename** is the host where you want to configure mirroring.

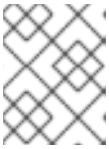
3. On **site-b**, schedule the deployment of mirror daemon on the secondary cluster:

   Syntax

   ```
   ceph orch apply rbd-mirror --placement=NODENAME
   ```

   Example

   ```
   [ceph: root@site-b /]# ceph orch apply rbd-mirror --placement=host04
   ```

   NOTE

   The **nodename** is the host where you want to configure mirroring in the secondary cluster.

4. Enable journaling features on an image on **site-a**.

   a. For **new images**, use the **--image-feature** option:

Syntax

```
rbd create IMAGE_NAME --size MEGABYTES --pool POOL_NAME --image-feature
FEATURE FEATURE
```

Example

```
[ceph: root@site-a /]# rbd create image1 --size 1024 --pool data --image-feature
exclusive-lock,journaling
```

NOTE

If **exclusive-lock** is already enabled, use **journaling** as the only argument, else it returns the following error:

```
one or more requested features are already enabled
(22) Invalid argument
```

b. For **existing images**, use the **rbd feature enable** command:

Syntax

```
rbd feature enable POOL_NAME/IMAGE_NAME FEATURE, FEATURE
```

Example

```
[ceph: root@site-a /]# rbd feature enable data/image1 exclusive-lock, journaling
```

c. To enable journaling on all new images by default, set the configuration parameter using **ceph config set** command:

Example

```
[ceph: root@site-a /]# ceph config set global rbd_default_features 125
[ceph: root@site-a /]# ceph config show mon.host01 rbd_default_features
```

5. Choose the mirroring mode, either pool or image mode, on both the storage clusters.

a. Enabling **pool mode**:

Syntax

```
rbd mirror pool enable POOL_NAME MODE
```

Example

```
[ceph: root@site-a /]# rbd mirror pool enable data pool
[ceph: root@site-b /]# rbd mirror pool enable data pool
```

This example enables mirroring of the whole pool named **data**.

b. Enabling **image mode**:

Syntax

```
rbd mirror pool enable POOL_NAME MODE
```

Example

```
[ceph: root@site-a /]# rbd mirror pool enable data image
[ceph: root@site-b /]# rbd mirror pool enable data image
```

This example enables image mode mirroring on the pool named **data**.

NOTE

To enable mirroring on specific images in a pool, see the *Enabling image mirroring* section in the *Red Hat Ceph Storage Block Device Guide* for more details.

c. Verify that mirroring has been successfully enabled at both the sites:

Syntax

```
rbd mirror pool info POOL_NAME
```

Example

```
[ceph: root@site-a /]# rbd mirror pool info data
Mode: pool
Site Name: c13d8065-b33d-4cb5-b35f-127a02768e7f

Peer Sites: none

[ceph: root@site-b /]# rbd mirror pool info data
Mode: pool
Site Name: a4c667e2-b635-47ad-b462-6faeeee78df7

Peer Sites: none
```

6. On a Ceph client node, bootstrap the storage cluster peers.

a. Create Ceph user accounts, and register the storage cluster peer to the pool:

Syntax

```
rbd mirror pool peer bootstrap create --site-name PRIMARY_LOCAL_SITE_NAME
POOL_NAME > PATH_TO_BOOTSTRAP_TOKEN
```

Example

```
[ceph: root@rbd-client-site-a /]# rbd mirror pool peer bootstrap create --site-name site-a
data > /root/bootstrap_token_site-a
```

NOTE

This example bootstrap command creates the **client.rbd-mirror.site-a** and the **client.rbd-mirror-peer** Ceph users.

b. Copy the bootstrap token file to the **site-b** storage cluster.

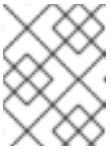
c. Import the bootstrap token on the **site-b** storage cluster:

Syntax

```
rbd mirror pool peer bootstrap import --site-name SECONDARY_LOCAL_SITE_NAME --direction rx-tx POOL_NAME PATH_TO_BOOTSTRAP_TOKEN
```

Example

```
[ceph: root@rbd-client-site-b /]# rbd mirror pool peer bootstrap import --site-name site-b --direction rx-tx data /root/bootstrap_token_site-a
```



NOTE

The **--direction** argument is optional, as two-way mirroring is the default when bootstrapping peers.

7. To verify the mirroring status, run the following command from a Ceph Monitor node on the primary and secondary sites:

Syntax

```
rbd mirror image status POOL_NAME/IMAGE_NAME
```

Example

```
[ceph: root@mon-site-a /]# rbd mirror image status data/image1
image1:
  global_id:   a4c667e2-b635-47ad-b462-6faeeee78df7
  state:       up+stopped
  description: local image is primary
  service:     host03.glsdbv on host03.ceph.redhat.com
  last_update: 2021-09-16 10:55:58
  peer_sites:
    name: a
    state: up+stopped
    description: replaying,
{"bytes_per_second":0.0,"entries_behind_primary":0,"entries_per_second":0.0,"non_primary_position":{"entry_tid":3,"object_number":3,"tag_tid":1},"primary_position":
{"entry_tid":3,"object_number":3,"tag_tid":1}}
    last_update: 2021-09-16 10:55:50
```

Here, **up** means the **rbd-mirror** daemon is running, and **stopped** means this image is not the target for replication from another storage cluster. This is because the image is primary on this storage cluster.

Example

```
[ceph: root@mon-site-b /]# rbd mirror image status data/image1
image1:
  global_id:   a4c667e2-b635-47ad-b462-6faeeee78df7
  state:       up+replaying
  description: replaying,
{"bytes_per_second":0.0,"entries_behind_primary":0,"entries_per_second":0.0,"non_primary_p
osition":{"entry_tid":3,"object_number":3,"tag_tid":1},"primary_position":
{"entry_tid":3,"object_number":3,"tag_tid":1}}
  service:     host05.dtisty on host05
  last_update: 2021-09-16 10:57:20
  peer_sites:
    name: b
    state: up+stopped
    description: local image is primary
    last_update: 2021-09-16 10:57:28
```

If images are in the state **up+replaying**, then mirroring is functioning properly. Here, **up** means the **rbd-mirror** daemon is running, and **replaying** means this image is the target for replication from another storage cluster.

NOTE

Depending on the connection between the sites, mirroring can take a long time to sync the images.

Additional Resources

- See the *Ceph block device mirroring* section in the *Red Hat Ceph Storage Block Device Guide* for more details.
- See the *User Management* section in the *Red Hat Ceph Storage Administration Guide* for more details on Ceph users.

6.4. ADMINISTRATION FOR MIRRORING CEPH BLOCK DEVICES

As a storage administrator, you can do various tasks to help you manage the Ceph block device mirroring environment. You can do the following tasks:

- Viewing information about storage cluster peers.
- Add or remove a storage cluster peer.
- Getting mirroring status for a pool or image.
- Enabling mirroring on a pool or image.
- Disabling mirroring on a pool or image.
- Delaying block device replication.
- Promoting and demoting an image.

Prerequisites

- A minimum of two healthy running Red Hat Ceph Storage cluster.
- Root-level access to the Ceph client nodes.
- A one-way or two-way Ceph block device mirroring relationship.
- A CephX user with administrator-level capabilities.

6.4.1. Viewing information about peers

View information about storage cluster peers.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. To view information about the peers:

   Syntax

   ```
   rbd mirror pool info POOL_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd mirror pool info data
   Mode: pool
   Site Name: a

   Peer Sites:

   UUID: 950ddadf-f995-47b7-9416-b9bb233f66e3
   Name: b
   Mirror UUID: 4696cd9d-1466-4f98-a97a-3748b6b722b3
   Direction: rx-tx
   Client: client.rbd-mirror-peer
   ```

6.4.2. Enabling mirroring on a pool

Enable mirroring on a pool by running the following commands on both peer clusters.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. To enable mirroring on a pool:

   Syntax

   ```
   rbd mirror pool enable POOL_NAME MODE
   ```

   Example

   ```
   [root@rbd-client ~]# rbd mirror pool enable data pool
   ```

   This example enables mirroring of the whole pool named **data**.

   Example

   ```
   [root@rbd-client ~]# rbd mirror pool enable data image
   ```

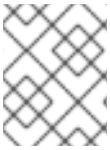
   This example enables image mode mirroring on the pool named **data**.

Additional Resources

- See the *Mirroring Ceph block devices* section in the *Red Hat Ceph Storage Block Device Guide* for details.

6.4.3. Disabling mirroring on a pool

Before disabling mirroring, remove the peer clusters.



NOTE

When you disable mirroring on a pool, you also disable it on any images within the pool for which mirroring was enabled separately in image mode.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. To disable mirroring on a pool:

   Syntax

   ```
   rbd mirror pool disable POOL_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd mirror pool disable data
   ```

   This example disables mirroring of a pool named **data**.

6.4.4. Enabling image mirroring

Enable mirroring on the whole pool in image mode on both peer storage clusters.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. Enable mirroring for a specific image within the pool:

   Syntax

   ```
   rbd mirror image enable POOL_NAME/IMAGE_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd mirror image enable data/image2
   ```

   This example enables mirroring for the **image2** image in the **data** pool.

Additional Resources

- See the *Enabling mirroring on a pool* section in the *Red Hat Ceph Storage Block Device Guide* for details.

6.4.5. Disabling image mirroring

You can disable Ceph Block Device mirroring on images.

Prerequisites

- A running Red Hat Ceph Storage cluster with snapshot-based mirroring configured.
- Root-level access to the node.

Procedure

1. To disable mirroring for a specific image:

   Syntax

   ```
   rbd mirror image disable POOL_NAME/IMAGE_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd mirror image disable data/image2
   ```

   This example disables mirroring of the **image2** image in the **data** pool.

Additional Resources

- See the *Configuring Ansible inventory location* section in the *Red Hat Ceph Storage Installation Guide* for more details on adding clients to the **cephadm-ansible** inventory.

6.4.6. Image promotion and demotion

You can promote or demote an image in a pool.

NOTE

Do not force promote non-primary images that are still syncing, because the images will not be valid after the promotion.

Prerequisites

- A running Red Hat Ceph Storage cluster with snapshot-based mirroring configured.
- Root-level access to the node.

Procedure

1. To demote an image to non-primary:

   Syntax

   ```
   rbd mirror image demote POOL_NAME/IMAGE_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd mirror image demote data/image2
   ```

   This example demotes the **image2** image in the **data** pool.

2. To promote an image to primary:

   Syntax

   ```
   rbd mirror image promote POOL_NAME/IMAGE_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd mirror image promote data/image2
   ```

   This example promotes **image2** in the **data** pool.

   Depending on which type of mirroring you are using, see either *Recover from a disaster with one-way mirroring* or *Recover from a disaster with two-way mirroring* for details.

   Syntax

   ```
   rbd mirror image promote --force POOL_NAME/IMAGE_NAME
   ```

Example

```
[root@rbd-client ~]# rbd mirror image promote --force data/image2
```

Use forced promotion when the demotion cannot be propagated to the peer Ceph storage cluster. For example, because of cluster failure or communication outage.

Additional Resources

- See the *Failover after a non-orderly shutdown* section in the *Red Hat Ceph Storage Block Device Guide* for details.

6.4.7. Image resynchronization

You can re-synchronize an image. In case of an inconsistent state between the two peer clusters, the **rbd-mirror** daemon does not attempt to mirror the image that is causing the inconsistency.

Prerequisites

- A running Red Hat Ceph Storage cluster with snapshot-based mirroring configured.
- Root-level access to the node.

Procedure

1. To request a re-synchronization to the primary image:

   #### Syntax

   ```
   rbd mirror image resync POOL_NAME/IMAGE_NAME
   ```

   #### Example

   ```
   [root@rbd-client ~]# rbd mirror image resync data/image2
   ```

   This example requests resynchronization of **image2** in the **data** pool.

Additional Resources

- To recover from an inconsistent state because of a disaster, see either *Recover from a disaster with one-way mirroring* or *Recover from a disaster with two-way mirroring* for details.

6.4.8. Adding a storage cluster peer

Add a storage cluster peer for the **rbd-mirror** daemon to discover its peer storage cluster. For example, to add the **site-a** storage cluster as a peer to the **site-b** storage cluster, then follow this procedure from the client node in the **site-b** storage cluster.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. Register the peer to the pool:

   Syntax

   ```
   rbd --cluster CLUSTER_NAME mirror pool peer add POOL_NAME
   PEER_CLIENT_NAME@PEER_CLUSTER_NAME -n CLIENT_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd --cluster site-b mirror pool peer add data client.site-a@site-a -n
   client.site-b
   ```

6.4.9. Removing a storage cluster peer

Remove a storage cluster peer by specifying the peer UUID.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. Specify the pool name and the peer Universally Unique Identifier (UUID).

   Syntax

   ```
   rbd mirror pool peer remove POOL_NAME PEER_UUID
   ```

   Example

   ```
   [root@rbd-client ~]# rbd mirror pool peer remove data 7e90b4ce-e36d-4f07-8cbc-
   42050896825d
   ```

   TIP

   To view the peer UUID, use the **rbd mirror pool info** command.

6.4.10. Getting mirroring status for a pool

You can get the mirror status for a pool on the storage clusters.

Prerequisites

- A running Red Hat Ceph Storage cluster with snapshot-based mirroring configured.
- Root-level access to the node.

Procedure

1. To get the mirroring pool summary:

   Syntax

   ```
   rbd mirror pool status POOL_NAME
   ```

   Example

   ```
   [root@site-a ~]# rbd mirror pool status data
   health: OK
   daemon health: OK
   image health: OK
   images: 1 total
       1 replaying
   ```

   TIP

   To output status details for every mirroring image in a pool, use the **--verbose** option.

6.4.11. Getting mirroring status for a single image

You can get the mirror status for an image by running the **mirror image status** command.

Prerequisites

- A running Red Hat Ceph Storage cluster with snapshot-based mirroring configured.
- Root-level access to the node.

Procedure

1. To get the status of a mirrored image:

   Syntax

   ```
   rbd mirror image status POOL_NAME/IMAGE_NAME
   ```

   Example

   ```
   [root@site-a ~]# rbd mirror image status data/image2
   image2:
     global_id:   1e3422a2-433e-4316-9e43-1827f8dbe0ef
     state:       up+unknown
     description: remote image is non-primary
     service:     pluto008.yuoosv on pluto008
     last_update: 2021-10-06 09:37:58
   ```

   This example gets the status of the **image2** image in the **data** pool.

6.4.12. Delaying block device replication

Whether you are using one- or two-way replication, you can delay replication between RADOS Block

Device (RBD) mirroring images. You might want to implement delayed replication if you want a window of cushion time in case an unwanted change to the primary image needs to be reverted before being replicated to the secondary image.

NOTE

Delaying block device replication is only applicable with journal-based mirroring.

To implement delayed replication, the **rbd-mirror** daemon within the destination storage cluster should set the **rbd_mirroring_replay_delay = *MINIMUM_DELAY_IN_SECONDS*** configuration option. This setting can either be applied globally within the **ceph.conf** file utilized by the **rbd-mirror** daemons, or on an individual image basis.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. To utilize delayed replication for a specific image, on the primary image, run the following **rbd** CLI command:

   Syntax

   ```
   rbd image-meta set POOL_NAME/IMAGE_NAME conf_rbd_mirroring_replay_delay MINIMUM_DELAY_IN_SECONDS
   ```

   Example

   ```
   [root@rbd-client ~]# rbd image-meta set vms/vm-1 conf_rbd_mirroring_replay_delay 600
   ```

   This example sets a 10 minute minimum replication delay on image **vm-1** in the **vms** pool.

6.4.13. Converting journal-based mirroring to snapshot-based mirrorring

You can convert journal-based mirroring to snapshot-based mirroring by disabling mirroring and enabling snapshot.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. Log into the Cephadm shell:

   Example

   ```
   [root@rbd-client ~]# cephadm shell
   ```

2. Disable mirroring for a specific image within the pool:

   Syntax

   ```
   rbd mirror image disable POOL_NAME/IMAGE_NAME
   ```

   Example

   ```
   [ceph: root@rbd-client /]# rbd mirror image disable mirror_pool/mirror_image
   Mirroring disabled
   ```

3. Enable snapshot-based mirroring for the image:

   Syntax

   ```
   rbd mirror image enable POOL_NAME/IMAGE_NAME snapshot
   ```

   Example

   ```
   [ceph: root@rbd-client /]# rbd mirror image enable mirror_pool/mirror_image snapshot
   Mirroring enabled
   ```

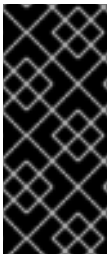
   This example enables snapshot-based mirroring for the **mirror_image** image in the **mirror_pool** pool.

6.4.14. Creating an image mirror-snapshot

Create an image mirror-snapshot when it is required to mirror the changed contents of an RBD image when using snapshot-based mirroring.

Prerequisites

- A minimum of two healthy running Red Hat Ceph Storage clusters.
- Root-level access to the Ceph client nodes for the Red Hat Ceph Storage clusters.
- A CephX user with administrator-level capabilities.
- Access to the Red Hat Ceph Storage cluster where a snapshot mirror will be created.



IMPORTANT

By default, a maximum of 5 image mirror-snapshots are retained. The most recent image mirror-snapshot is automatically removed if the limit is reached. If required, the limit can be overridden through the **rbd_mirroring_max_mirroring_snapshots** configuration. Image mirror-snapshots are automatically deleted when the image is removed or when mirroring is disabled.

Procedure

- To create an image-mirror snapshot:

  Syntax

Syntax

```
rbd --cluster CLUSTER_NAME mirror image snapshot POOL_NAME/IMAGE_NAME
```

Example

```
[root@site-a ~]# rbd mirror image snapshot data/image1
```

Additional Resources

- See the *Mirroring Ceph block devices* section in the *Red Hat Ceph Storage Block Device Guide* for details.

6.4.15. Scheduling mirror-snapshots

Mirror-snapshots can be automatically created when mirror-snapshot schedules are defined. The mirror-snapshot can be scheduled globally, per-pool or per-image levels. Multiple mirror-snapshot schedules can be defined at any level but only the most specific snapshot schedules that match an individual mirrored image will run.

6.4.15.1. Creating a mirror-snapshot schedule

You can create a mirror-snapshot schedule using the **snapshot schedule** command.

Prerequisites

- A minimum of two healthy running Red Hat Ceph Storage clusters.
- Root-level access to the Ceph client nodes for the Red Hat Ceph Storage clusters.
- A CephX user with administrator-level capabilities.
- Access to the Red Hat Ceph Storage cluster where the mirror image needs to be scheduled.

Procedure

1. To create a mirror-snapshot schedule:

Syntax

```
rbd --cluster CLUSTER_NAME mirror snapshot schedule add --pool POOL_NAME --image IMAGE_NAME INTERVAL [START_TIME]
```

The *CLUSTER_NAME* should be used only when the cluster name is different from the default name **ceph**. The interval can be specified in days, hours, or minutes using d, h, or m suffix respectively. The optional START_TIME can be specified using the ISO 8601 time format.

Example

```
[root@site-a ~]# rbd mirror snapshot schedule add --pool data --image image1 6h
```

Example

```
[root@site-a ~]# rbd mirror snapshot schedule add --pool data --image image1 24h 14:00:00-05:00
```

Additional Resources

- See the *Mirroring Ceph block devices* section in the *Red Hat Ceph Storage Block Device Guide* for details.

6.4.15.2. Listing all snapshot schedules at a specific level

You can list all snapshot schedules at a specific level.

Prerequisites

- A minimum of two healthy running Red Hat Ceph Storage clusters.
- Root-level access to the Ceph client nodes for the Red Hat Ceph Storage clusters.
- A CephX user with administrator-level capabilities.
- Access to the Red Hat Ceph Storage cluster where the mirror image needs to be scheduled.

Procedure

1. To list all snapshot schedules for a specific global, pool or image level, with an optional pool or image name:

   Syntax

   ```
   rbd --cluster site-a mirror snapshot schedule ls --pool POOL_NAME --recursive
   ```

   Additionally, the **--recursive** option can be specified to list all schedules at the specified level as shown below:

   Example

   ```
   [root@rbd-client ~]# rbd mirror snapshot schedule ls --pool data --recursive
   POOL       NAMESPACE IMAGE  SCHEDULE
   data       -         -      every 1d starting at 14:00:00-05:00
   data       -         image1 every 6h
   ```

Additional Resources

- See the *Mirroring Ceph block devices* section in the *Red Hat Ceph Storage Block Device Guide* for details.

6.4.15.3. Removing a mirror-snapshot schedule

You can remove a mirror-snapshot schedule using the **snapshot schedule remove** command.

Prerequisites

- A minimum of two healthy running Red Hat Ceph Storage clusters.

- Root-level access to the Ceph client nodes for the Red Hat Ceph Storage clusters.
- A CephX user with administrator-level capabilities.
- Access to the Red Hat Ceph Storage cluster where the mirror image needs to be scheduled.

Procedure

1. To remove a mirror-snapshot schedule:

   Syntax

   ```
   rbd --cluster CLUSTER_NAME mirror snapshot schedule remove --pool POOL_NAME --image IMAGE_NAME INTERVAL START_TIME
   ```

   The interval can be specified in days, hours, or minutes using d, h, m suffix respectively. The optional START_TIME can be specified using the ISO 8601 time format.

   Example

   ```
   [root@site-a ~]# rbd mirror snapshot schedule remove --pool data --image image1 6h
   ```

   Example

   ```
   [root@site-a ~]# rbd mirror snapshot schedule remove --pool data --image image1 24h 14:00:00-05:00
   ```

Additional Resources

- See the *Mirroring Ceph block devices* section in the *Red Hat Ceph Storage Block Device Guide* for details.

6.4.15.4. Viewing the status for the next snapshots to be created

You can view the status for the next snapshots to be created for snapshot-based mirroring RBD images.

Prerequisites

- A minimum of two healthy running Red Hat Ceph Storage clusters.
- Root-level access to the Ceph client nodes for the Red Hat Ceph Storage clusters.
- A CephX user with administrator-level capabilities.
- Access to the Red Hat Ceph Storage cluster where the mirror image needs to be scheduled.

Procedure

1. To view the status for the next snapshots to be created:

   Syntax

```
rbd --cluster site-a mirror snapshot schedule status [--pool POOL_NAME] [--image
IMAGE_NAME]
```

Example

```
[root@rbd-client ~]# rbd mirror snapshot schedule status
SCHEDULE    TIME       IMAGE
2021-09-21 18:00:00 data/image1
```

Additional Resources

- See the *Mirroring Ceph block devices* section in the *Red Hat Ceph Storage Block Device Guide* for details.

6.5. RECOVER FROM A DISASTER

As a storage administrator, you can be prepared for eventual hardware failure by knowing how to recover the data from another storage cluster where mirroring was configured.

In the examples, the primary storage cluster is known as the **site-a**, and the secondary storage cluster is known as the **site-b**. Additionally, the storage clusters both have a **data** pool with two images, **image1** and **image2**.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- One-way or two-way mirroring was configured.

6.5.1. Disaster recovery

Asynchronous replication of block data between two or more Red Hat Ceph Storage clusters reduces downtime and prevents data loss in the event of a significant data center failure. These failures have a widespread impact, also referred as a **large blast radius**, and can be caused by impacts to the power grid and natural disasters.

Customer data needs to be protected during these scenarios. Volumes must be replicated with consistency and efficiency and also within Recovery Point Objective (RPO) and Recovery Time Objective (RTO) targets. This solution is called a Wide Area Network- Disaster Recovery (WAN-DR).

In such scenarios it is hard to restore the primary system and the data center. The quickest way to recover is to failover the applications to an alternate Red Hat Ceph Storage cluster (disaster recovery site) and make the cluster operational with the latest copy of the data available. The solutions that are used to recover from these failure scenarios are guided by the application:

- **Recovery Point Objective (RPO)**: The amount of data loss, an application tolerate in the worst case.
- **Recovery Time Objective (RTO)**: The time taken to get the application back on line with the latest copy of the data available.

Additional Resources

- See the *Mirroring Ceph block devices* Chapter in the *Red Hat Ceph Storage Block Device Guide* for details.
- See the *Encryption in transit* section in the *Red Hat Ceph Storage Data Security and Hardening Guide* to know more about data transmission over the wire in an encrypted state.

6.5.2. Recover from a disaster with one-way mirroring

To recover from a disaster when using one-way mirroring use the following procedures. They show how to fail over to the secondary cluster after the primary cluster terminates, and how to fail back. The shutdown can be orderly or non-orderly.

IMPORTANT

One-way mirroring supports multiple secondary sites. If you are using additional secondary clusters, choose one of the secondary clusters to fail over to. Synchronize from the same cluster during fail back.

6.5.3. Recover from a disaster with two-way mirroring

To recover from a disaster when using two-way mirroring use the following procedures. They show how to fail over to the mirrored data on the secondary cluster after the primary cluster terminates, and how to failback. The shutdown can be orderly or non-orderly.

6.5.4. Failover after an orderly shutdown

Failover to the secondary storage cluster after an orderly shutdown.

Prerequisites

- Minimum of two running Red Hat Ceph Storage clusters.
- Root-level access to the node.
- Pool mirroring or image mirroring configured with one-way mirroring.

Procedure

1. Stop all clients that use the primary image. This step depends on which clients use the image. For example, detach volumes from any OpenStack instances that use the image.
2. Demote the primary images located on the **site-a** cluster by running the following commands on a monitor node in the **site-a** cluster:

   Syntax

   ```
   rbd mirror image demote POOL_NAME/IMAGE_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd mirror image demote data/image1
   [root@rbd-client ~]# rbd mirror image demote data/image2
   ```

3. Promote the non-primary images located on the **site-b** cluster by running the following commands on a monitor node in the **site-b** cluster:

   Syntax

   ```
   rbd mirror image promote POOL_NAME/IMAGE_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd mirror image promote data/image1
   [root@rbd-client ~]# rbd mirror image promote data/image2
   ```

4. After some time, check the status of the images from a monitor node in the **site-b** cluster. They should show a state of **up+stopped** and be listed as primary:

   ```
   [root@rbd-client ~]# rbd mirror image status data/image1
   image1:
     global_id:   08027096-d267-47f8-b52e-59de1353a034
     state:       up+stopped
     description: local image is primary
     last_update: 2019-04-17 16:04:37
   [root@rbd-client ~]# rbd mirror image status data/image2
   image2:
     global_id:   596f41bc-874b-4cd4-aefe-4929578cc834
     state:       up+stopped
     description: local image is primary
     last_update: 2019-04-17 16:04:37
   ```

5. Resume the access to the images. This step depends on which clients use the image.

Additional Resources

- See the *Block Storage and Volumes* chapter in the *Red Hat OpenStack Platform Storage Guide* .

6.5.5. Failover after a non-orderly shutdown

Failover to secondary storage cluster after a non-orderly shutdown.

Prerequisites

- Minimum of two running Red Hat Ceph Storage clusters.
- Root-level access to the node.
- Pool mirroring or image mirroring configured with one-way mirroring.

Procedure

1. Verify that the primary storage cluster is down.
2. Stop all clients that use the primary image. This step depends on which clients use the image. For example, detach volumes from any OpenStack instances that use the image.

3. Promote the non-primary images from a Ceph Monitor node in the **site-b** storage cluster. Use the **--force** option, because the demotion cannot be propagated to the **site-a** storage cluster:

   Syntax

   ```
   rbd mirror image promote --force POOL_NAME/IMAGE_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd mirror image promote --force data/image1
   [root@rbd-client ~]# rbd mirror image promote --force data/image2
   ```

4. Check the status of the images from a Ceph Monitor node in the **site-b** storage cluster. They should show a state of **up+stopping_replay**. The description should say **force promoted**, meaning it is in the intermittent state. Wait until the state comes to **up+stopped** to validate the site is successfully promoted.

   Example

   ```
   [root@rbd-client ~]# rbd mirror image status data/image1
   image1:
     global_id:   08027096-d267-47f8-b52e-59de1353a034
     state:       up+stopping_replay
     description: force promoted
     last_update: 2023-04-17 13:25:06

   [root@rbd-client ~]# rbd mirror image status data/image1
   image1:
     global_id:   08027096-d267-47f8-b52e-59de1353a034
     state:       up+stopped
     description: force promoted
     last_update: 2023-04-17 13:25:06
   ```

Additional Resources

- See the *Block Storage and Volumes* chapter in the *Red Hat OpenStack Platform Storage Guide* .

6.5.6. Prepare for fail back

If two storage clusters were originally configured only for one-way mirroring, in order to fail back, configure the primary storage cluster for mirroring in order to replicate the images in the opposite direction.

During failback scenario, the existing peer that is inaccessible must be removed before adding a new peer to an existing cluster.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the client node.

Procedure

1. Log into the Cephadm shell:

   Example

   ```
   [root@rbd-client ~]# cephadm shell
   ```

2. On the **site-a** storage cluster , run the following command:

   Example

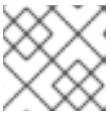
   ```
   [ceph: root@rbd-client /]# ceph orch apply rbd-mirror --placement=host01
   ```

3. Remove any inaccessible peers.

   IMPORTANT

   This step must be run on the peer site which is up and running.

   NOTE

   Multiple peers are supported only for one way mirroring.

   a. Get the pool UUID:

      Syntax

      ```
      rbd mirror pool info POOL_NAME
      ```

      Example

      ```
      [ceph: root@host01 /]# rbd mirror pool info pool_failback
      ```

   b. Remove the inaccessible peer:

      Syntax

      ```
      rbd mirror pool peer remove POOL_NAME PEER_UUID
      ```

      Example

      ```
      [ceph: root@host01 /]# rbd mirror pool peer remove pool_failback f055bb88-6253-4041-923d-08c4ecbe799a
      ```

4. Create a block device pool with a name same as its peer mirror pool.

   a. To create an rbd pool, execute the following:

      Syntax

```
ceph osd pool create POOL_NAME PG_NUM
ceph osd pool application enable POOL_NAME rbd
rbd pool init -p POOL_NAME
```

Example

```
[root@rbd-client ~]# ceph osd pool create pool1
[root@rbd-client ~]# ceph osd pool application enable pool1 rbd
[root@rbd-client ~]# rbd pool init -p pool1
```

5. On a Ceph client node, bootstrap the storage cluster peers.

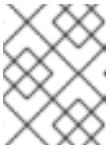
   a. Create Ceph user accounts, and register the storage cluster peer to the pool:

   Syntax

   ```
   rbd mirror pool peer bootstrap create --site-name LOCAL_SITE_NAME POOL_NAME > PATH_TO_BOOTSTRAP_TOKEN
   ```

   Example

   ```
   [ceph: root@rbd-client-site-a /]# rbd mirror pool peer bootstrap create --site-name site-a data > /root/bootstrap_token_site-a
   ```

   NOTE

   This example bootstrap command creates the **client.rbd-mirror.site-a** and the **client.rbd-mirror-peer** Ceph users.

   b. Copy the bootstrap token file to the **site-b** storage cluster.

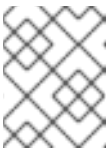
   c. Import the bootstrap token on the **site-b** storage cluster:

   Syntax

   ```
   rbd mirror pool peer bootstrap import --site-name LOCAL_SITE_NAME --direction rx-only POOL_NAME PATH_TO_BOOTSTRAP_TOKEN
   ```

   Example

   ```
   [ceph: root@rbd-client-site-b /]# rbd mirror pool peer bootstrap import --site-name site-b --direction rx-only data /root/bootstrap_token_site-a
   ```

   NOTE

   For one-way RBD mirroring, you must use the **--direction rx-only** argument, as two-way mirroring is the default when bootstrapping peers.

6. From a monitor node in the **site-a** storage cluster, verify the **site-b** storage cluster was successfully added as a peer:

Example

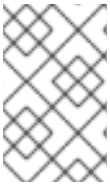
```
[ceph: root@rbd-client /]# rbd mirror pool info -p data
Mode: image
Peers:
  UUID                                 NAME   CLIENT
  d2ae0594-a43b-4c67-a167-a36c646e8643 site-b client.site-b
```

Additional Resources

- For detailed information, see the *User Management* chapter in the *Red Hat Ceph Storage Administration Guide*.

6.5.6.1. Fail back to the primary storage cluster

When the formerly primary storage cluster recovers, fail back to the primary storage cluster.



NOTE

If you have scheduled snapshots at the image level, then you need to re-add the schedule as image resync operations changes the RBD Image ID and the previous schedule becomes obsolete.

Prerequisites

- Minimum of two running Red Hat Ceph Storage clusters.
- Root-level access to the node.
- Pool mirroring or image mirroring configured with one-way mirroring.

Procedure

1. Check the status of the images from a monitor node in the **site-b** cluster again. They should show a state of **up-stopped** and the description should say **local image is primary**:

   Example

   ```
   [root@rbd-client ~]# rbd mirror image status data/image1
   image1:
     global_id:   08027096-d267-47f8-b52e-59de1353a034
     state:       up+stopped
     description: local image is primary
     last_update: 2019-04-22 17:37:48
   [root@rbd-client ~]# rbd mirror image status data/image2
   image2:
     global_id:   08027096-d267-47f8-b52e-59de1353a034
     state:       up+stopped
     description: local image is primary
     last_update: 2019-04-22 17:38:18
   ```

2. From a Ceph Monitor node on the **site-a** storage cluster determine if the images are still primary:

Syntax

```
rbd mirror pool info POOL_NAME/IMAGE_NAME
```

Example

```
[root@rbd-client ~]# rbd info data/image1
[root@rbd-client ~]# rbd info data/image2
```

In the output from the commands, look for **mirroring primary: true** or **mirroring primary: false**, to determine the state.

3. Demote any images that are listed as primary by running a command like the following from a Ceph Monitor node in the **site-a** storage cluster:

Syntax

```
rbd mirror image demote POOL_NAME/IMAGE_NAME
```

Example

```
[root@rbd-client ~]# rbd mirror image demote data/image1
```

4. Resynchronize the images ONLY if there was a non-orderly shutdown. Run the following commands on a monitor node in the **site-a** storage cluster to resynchronize the images from **site-b** to **site-a**:

Syntax

```
rbd mirror image resync POOL_NAME/IMAGE_NAME
```

Example

```
[root@rbd-client ~]# rbd mirror image resync data/image1
Flagged image for resync from primary
[root@rbd-client ~]# rbd mirror image resync data/image2
Flagged image for resync from primary
```

5. After some time, ensure resynchronization of the images is complete by verifying they are in the **up+replaying** state. Check their state by running the following commands on a monitor node in the **site-a** storage cluster:

Syntax

```
rbd mirror image status POOL_NAME/IMAGE_NAME
```

Example

```
[root@rbd-client ~]# rbd mirror image status data/image1
[root@rbd-client ~]# rbd mirror image status data/image2
```

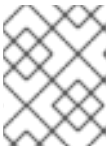
6. Demote the images on the **site-b** storage cluster by running the following commands on a Ceph Monitor node in the **site-b** storage cluster:

   Syntax

   ```
   rbd mirror image demote POOL_NAME/IMAGE_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd mirror image demote data/image1
   [root@rbd-client ~]# rbd mirror image demote data/image2
   ```

   NOTE

   If there are multiple secondary storage clusters, this only needs to be done from the secondary storage cluster where it was promoted.

7. Promote the formerly primary images located on the **site-a** storage cluster by running the following commands on a Ceph Monitor node in the **site-a** storage cluster:

   Syntax

   ```
   rbd mirror image promote POOL_NAME/IMAGE_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd mirror image promote data/image1
   [root@rbd-client ~]# rbd mirror image promote data/image2
   ```

8. Check the status of the images from a Ceph Monitor node in the **site-a** storage cluster. They should show a status of **up+stopped** and the description should say **local image is primary**:

   Syntax

   ```
   rbd mirror image status POOL_NAME/IMAGE_NAME
   ```

   Example

   ```
   [root@rbd-client ~]# rbd mirror image status data/image1
   image1:
     global_id:   08027096-d267-47f8-b52e-59de1353a034
     state:       up+stopped
     description: local image is primary
     last_update: 2019-04-22 11:14:51
   [root@rbd-client ~]# rbd mirror image status data/image2
   image2:
     global_id:   596f41bc-874b-4cd4-aefe-4929578cc834
     state:       up+stopped
     description: local image is primary
     last_update: 2019-04-22 11:14:51
   ```

6.5.7. Remove two-way mirroring

After fail back is complete, you can remove two-way mirroring and disable the Ceph block device mirroring service.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. Remove the **site-b** storage cluster as a peer from the **site-a** storage cluster:

   Example

   ```
   [root@rbd-client ~]# rbd mirror pool peer remove data client.remote@remote --cluster local
   [root@rbd-client ~]# rbd --cluster site-a mirror pool peer remove data client.site-b@site-b -n client.site-a
   ```

2. Stop and disable the **rbd-mirror** daemon on the **site-a** client:

   Syntax

   ```
   systemctl stop ceph-rbd-mirror@CLIENT_ID
   systemctl disable ceph-rbd-mirror@CLIENT_ID
   systemctl disable ceph-rbd-mirror.target
   ```

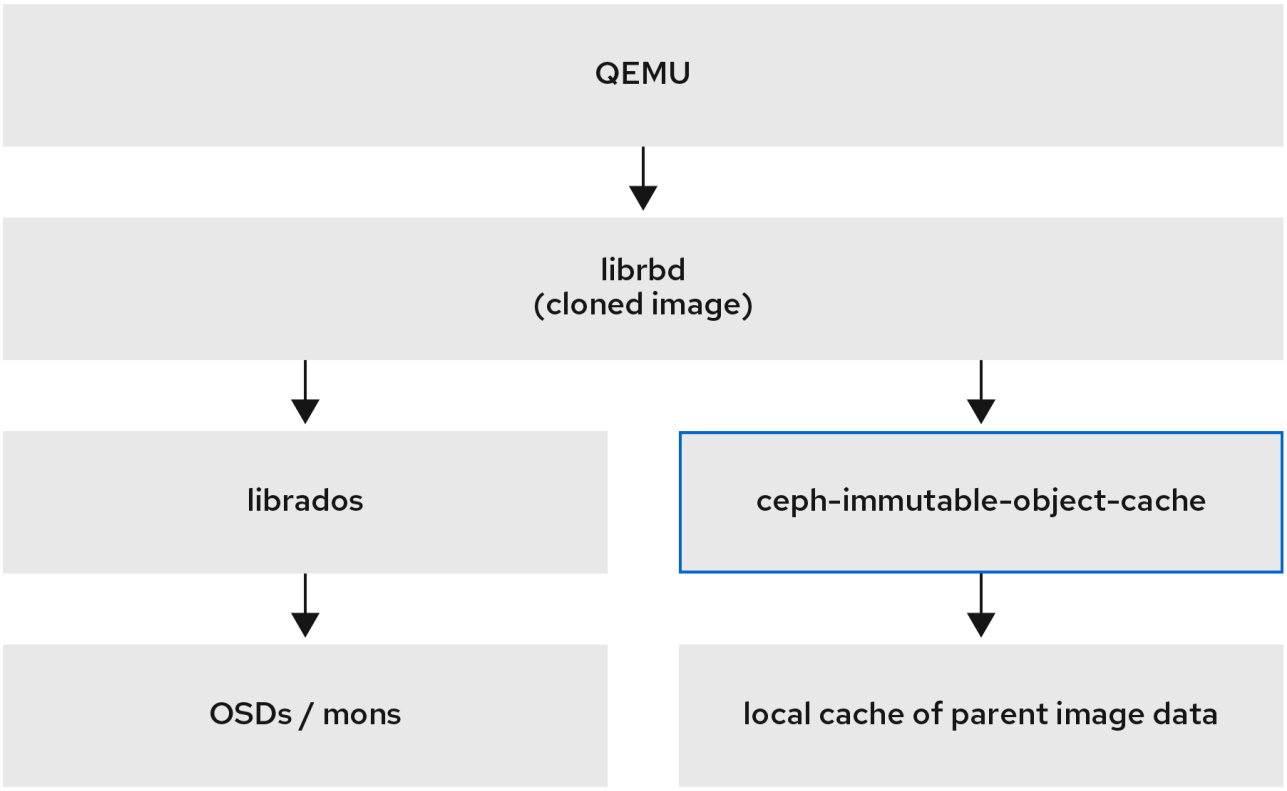
   Example

   ```
   [root@rbd-client ~]# systemctl stop ceph-rbd-mirror@site-a
   [root@rbd-client ~]# systemctl disable ceph-rbd-mirror@site-a
   [root@rbd-client ~]# systemctl disable ceph-rbd-mirror.target
   ```

CHAPTER 7. MANAGEMENT OF CEPH-IMMUTABLE-OBJECT-CACHE DAEMONS

As a storage administrator, use the **ceph-immutable-object-cache** daemons to cache the parent image content on the local disk. This cache is in the local caching directory. Future reads on that data use the local cache.

Figure 7.1. Ceph immutable cache daemon



7.1. EXPLANATION OF CEPH-IMMUTABLE-OBJECT-CACHE DAEMONS

Cloned Block device images usually modify only a small fraction of the parent image. For example, in a virtual desktop interface (VDI), the virtual machines are cloned from the same base image and initially differ only by the hostname and the IP address. During the bootup, if you use a local cache of the parent image, this speeds up reads on the caching host. This change reduces the client to cluster network traffic.

Reasons to use ceph-immutable-object-cache daemons

The **ceph-immutable-object-cache** daemon is a part of Red Hat Ceph Storage. It is a scalable, open-source, and distributed storage system. It connects to local clusters with the RADOS protocol, relying on default search paths to find **ceph.conf** files, monitor addresses and authentication information for them such as **/etc/ceph/*CLUSTER*.conf**, **/etc/ceph/*CLUSTER*.keyring**, and **/etc/ceph/*CLUSTER.NAME*.keyring**, where *CLUSTER* is the human-friendly name of the cluster, and *NAME* is the RADOS user to connect as an example, **client.ceph-immutable-object-cache**.

Key components of the daemon

The **ceph-immutable-object-cache** daemon has the following parts:

- Domain socket based inter-process communication (IPC): The daemon listens on a local domain socket on start-up and waits for connections from **librbd** clients.
- Least recently used (LRU) based promotion or demotion policy: The daemon maintains in-memory statistics of cache-hits on each cache file. It demotes the cold cache if capacity reaches to the configured threshold.
- File-based caching store: The daemon maintains a simple file based cache store. On promotion the RADOS objects are fetched from RADOS cluster and stored in the local caching directory.

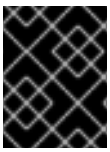
When you open each cloned RBD image, **librbd** tries to connect to the cache daemon through its Unix domain socket. Once successfully connected, **librbd** coordinates with the daemon on the subsequent reads. If there is a read that is not cached, the daemon promotes the RADOS object to the local caching directory, so the next read on that object is serviced from cache. The daemon also maintains simple LRU statistics so that under capacity pressure it evicts cold cache files as needed.

NOTE

For better performance, use SSDs as the underlying storage.

7.2. CONFIGURING THE CEPH-IMMUTABLE-OBJECT-CACHE DAEMON

The **ceph-immutable-object-cache** is a daemon for object cache of RADOS objects among Ceph clusters.



IMPORTANT

To use the **ceph-immutable-object-cache** daemon, you must be able to connect RADOS clusters.

The daemon promotes the objects to a local directory. These cache objects service the future reads. You can configure the daemon by installing the **ceph-immutable-object-cache** package.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- At least one SSD for the cache.

Procedure

1. Enable the RBD shared read only parent image cache. Add the following parameters under **[client]** in the **/etc/ceph/ceph.conf** file:

   Example

   ```
   [root@ceph-host01 ~]# vi /etc/ceph/ceph.conf

   [client]
   rbd parent cache enabled = true
   rbd plugins = parent_cache
   ```

   Restart the cluster.

2. Install the **ceph-immutable-object-cache** package:

   Example

   ```
   [root@ceph-host1 ~]# dnf install ceph-immutable-object-cache
   ```

3. Create a unique Ceph user ID, the keyring:

   Syntax

   ```
   ceph auth get-or-create client.ceph-immutable-object-cache.USER_NAME mon 'profile rbd'
   osd 'profile rbd-read-only'
   ```

   Example

   ```
   [root@ceph-host1 ~]# ceph auth get-or-create client.ceph-immutable-object-cache.user mon
   'profile rbd' osd 'profile rbd-read-only'

   [client.ceph-immutable-object-cache.user]
    key = AQCVPH1gFgHRAhAAp8ExRIsoxQK4QSYSRoVJLw==
   ```

   Copy this keyring.

4. In the **/etc/ceph** directory, create a file and paste the keyring:

   Example

   ```
   [root@ceph-host1 ]# vi /etc/ceph/ceph.client.ceph-immutable-object-cache.user.keyring

   [client.ceph-immutable-object-cache.user]
    key = AQCVPH1gFgHRAhAAp8ExRIsoxQK4QSYSRoVJLw
   ```

5. Enable the daemon:

   Syntax

   ```
   systemctl enable ceph-immutable-object-cache@ceph-immutable-object-
   cache.USER_NAME
   ```

   Specify the *USER_NAME* as the daemon instance.

   Example

   ```
   [root@ceph-host1 ~]# systemctl enable ceph-immutable-object-cache@ceph-immutable-
   object-cache.user

   Created symlink /etc/systemd/system/ceph-immutable-object-cache.target.wants/ceph-
   immutable-object-cache@ceph-immutable-object-cache.user.service →
   /usr/lib/systemd/system/ceph-immutable-object-cache@.service.
   ```

6. Start the **ceph-immutable-object-cache** daemon:

   Syntax

```
systemctl start ceph-immutable-object-cache@ceph-immutable-object-cache.USER_NAME
```

Example

```
[root@ceph-host1 ~]# systemctl start ceph-immutable-object-cache@ceph-immutable-object-cache.user
```

Verification

- Check the status of the configuration:

  Syntax

  ```
  systemctl status ceph-immutable-object-cache@ceph-immutable-object-cache.USER_NAME
  ```

  Example

  ```
  [root@ceph-host1 ~]# systemctl status ceph-immutable-object-cache@ceph-immutable-object-cache.user

  ● ceph-immutable-object-cache@ceph-immutable-object-cache.user>
    Loaded: loaded (/usr/lib/systemd/system/ceph-immutable-objec>
    Active: active (running) since Mon 2021-04-19 13:49:06 IST; >
  Main PID: 85020 (ceph-immutable-)
     Tasks: 15 (limit: 49451)
    Memory: 8.3M
    CGroup: /system.slice/system-ceph\x2dimmutable\x2dobject\x2d>
            └─85020 /usr/bin/ceph-immutable-object-cache -f --cl>
  ```

7.3. GENERIC SETTINGS OF CEPH-IMMUTABLE-OBJECT-CACHE DAEMONS

A few important generic settings of **ceph-immutable-object-cache** daemons are listed.

immutable_object_cache_sock

Description

The path to the domain socket used for communication between librbd clients and the ceph-immutable-object-cache daemon.

Type

String

Default

/var/run/ceph/immutable_object_cache_sock

immutable_object_cache_path

Description

The immutable object cache data directory.

Type

String

Default

/tmp/ceph_immutable_object_cache

immutable_object_cache_max_size

Description

The maximum size for immutable cache.

Type

Size

Default

1G

immutable_object_cache_watermark

Description

The high-water mark for the cache. The value is between zero and one. If the cache size reaches this threshold the daemon starts to delete cold cache based on LRU statistics.

Type

Float

Default

0.9

7.4. QOS SETTINGS OF CEPH-IMMUTABLE-OBJECT-CACHE DAEMONS

The **ceph-immutable-object-cache** daemons supports throttling which supports the settings described.

immutable_object_cache_qos_schedule_tick_min

Description

Minimum schedule tick for immutable object cache.

Type

Milliseconds

Default

50

immutable_object_cache_qos_iops_limit

Description

User-defined immutable object cache IO operations limit per second.

Type

Integer

Default

0

immutable_object_cache_qos_iops_burst

Description

User-defined burst limit of immutable object cache IO operations.

Type

Integer

Default

0

immutable_object_cache_qos_iops_burst_seconds

Description

User-defined burst duration in seconds of immutable object cache IO operations.

Type

Seconds

Default

1

immutable_object_cache_qos_bps_limit

Description

User-defined immutable object cache IO bytes limit per second.

Type

Integer

Default

0

immutable_object_cache_qos_bps_burst

Description

User-defined burst limit of immutable object cache IO bytes.

Type

Integer

Default

0

immutable_object_cache_qos_bps_burst_seconds

Description

The desired burst limit of read operations.

Type

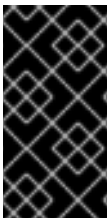
Seconds

Default

1

CHAPTER 8. THE RBD KERNEL MODULE

As a storage administrator, you can access Ceph block devices through the **rbd** kernel module. You can map and unmap a block device, and displaying those mappings. Also, you can get a list of images through the **rbd** kernel module.



IMPORTANT

Kernel clients on Linux distributions other than Red Hat Enterprise Linux (RHEL) are permitted but not supported. If issues are found in the storage cluster when using these kernel clients, Red Hat will address them, but if the root cause is found to be on the kernel client side, the issue will have to be addressed by the software vendor.

Prerequisites

- A running Red Hat Ceph Storage cluster.

8.1. CREATE A CEPH BLOCK DEVICE AND USE IT FROM A LINUX KERNEL MODULE CLIENT

As a storage administrator, you can create a Ceph Block Device for a Linux kernel module client in the Red Hat Ceph Storage Dashboard. As a system administrator, you can map that block device on a Linux client, and partition, format, and mount it, using the command line. After this, you can read and write files to it.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- A Red Hat Enterprise Linux client.

8.1.1. Creating a Ceph block device for a Linux kernel module client using dashboard

You can create a Ceph block device specifically for a Linux kernel module client using the dashboard web interface by enabling only the features it supports.

Kernel module client supports features like Deep flatten, Layering, Exclusive lock, Object map, and Fast diff.

Object map, Fast diff, and Deep flatten features require Red Hat Enterprise Linux 8.2 and later.

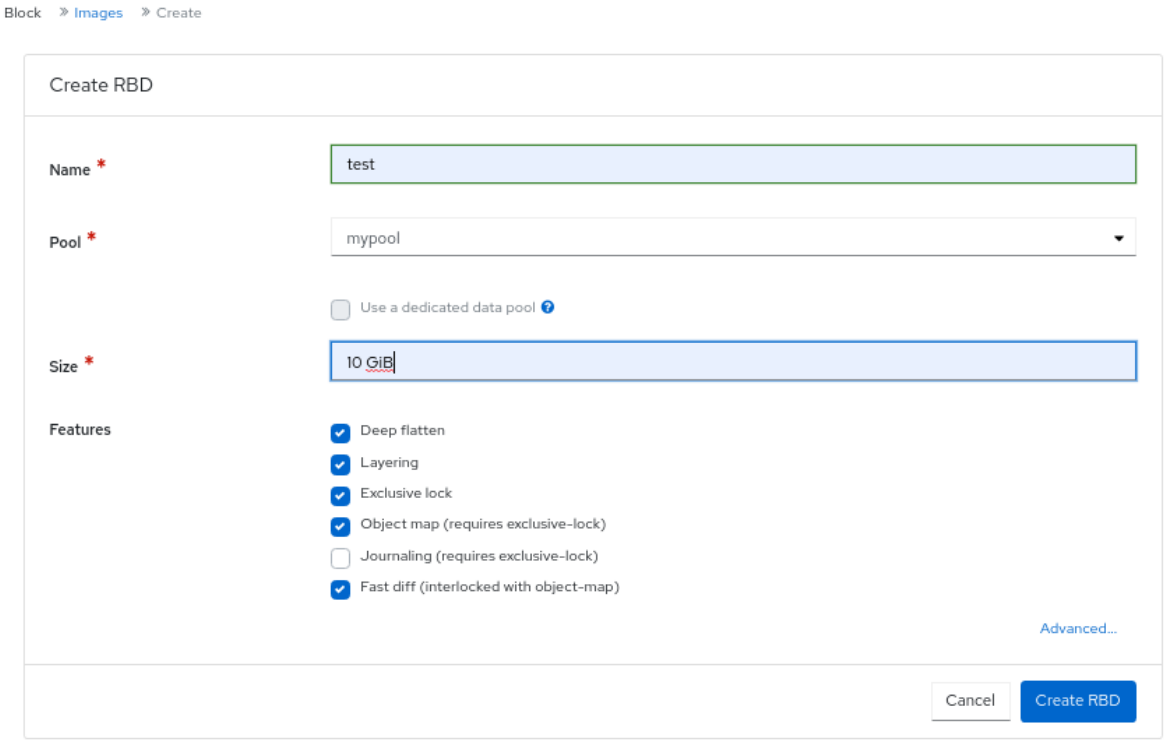
Prerequisites

- A running Red Hat Ceph Storage cluster.
- A replicated RBD pool created and enabled.

Procedure

1. From the *Block* drop-down menu, select *Images*.
2. Click *Create*.

3. In the *Create RBD* window, enter a image name, select the RBD enabled pool, select the supported features:



4. Click *Create RBD*.

Verification

- You will get a notification that the image is created successfully.

Additional Resources

- For more information, see *Map and mount a Ceph Block Device on Linux using the command line* in the *Red Hat Ceph Storage Block Device Guide*.
- For more information, see the *Red Hat Ceph Storage Dashboard Guide*.

8.1.2. Map and mount a Ceph Block Device on Linux using the command line

You can map a Ceph Block Device from a Red Hat Enterprise Linux client using the Linux **rbd** kernel module. After mapping it, you can partition, format, and mount it, so you can write files to it.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- A Ceph block device for a Linux kernel module client using the dashboard is created.
- A Red Hat Enterprise Linux client.

Procedure

1. On the Red Hat Enterprise Linux client node, enable the Red Hat Ceph Storage 6 Tools repository:

```
[root@rbd-client ~]# subscription-manager repos --enable=rhceph-6-tools-for-rhel-9-x86_64-
rpms
```

2. Install the **ceph-common** RPM package:

```
[root@rbd-client ~]# dnf install ceph-common
```

3. Copy the Ceph configuration file from a Monitor node to the Client node:

Syntax

```
scp root@MONITOR_NODE:/etc/ceph/ceph.conf /etc/ceph/ceph.conf
```

Example

```
[root@rbd-client ~]# scp root@cluster1-node2:/etc/ceph/ceph.conf /etc/ceph/ceph.conf
root@192.168.0.32's password:
ceph.conf                                     100%  497   724.9KB/s   00:00
[root@client1 ~]#
```

4. Copy the key file from a Monitor node to the Client node:

Syntax

```
scp root@MONITOR_NODE:/etc/ceph/ceph.client.admin.keyring
/etc/ceph/ceph.client.admin.keyring
```

Example

```
[root@rbd-client ~]# scp root@cluster1-node2:/etc/ceph/ceph.client.admin.keyring
/etc/ceph/ceph.client.admin.keyring
root@192.168.0.32's password:
ceph.client.admin.keyring                     100%  151   265.0KB/s   00:00
[root@client1 ~]#
```

5. Map the image:

Syntax

```
rbd map --pool POOL_NAME IMAGE_NAME --id admin
```

Example

```
[root@rbd-client ~]# rbd map --pool block-device-pool image1 --id admin
/dev/rbd0
[root@client1 ~]#
```

6. Create a partition table on the block device:

Syntax

```
parted /dev/MAPPED_BLOCK_DEVICE mklabel msdos
```

Example

```
[root@rbd-client ~]# parted /dev/rbd0 mklabel msdos
Information: You may need to update /etc/fstab.
```

7. Create a partition for an XFS file system:

Syntax

```
parted /dev/MAPPED_BLOCK_DEVICE mkpart primary xfs 0% 100%
```

Example

```
[root@rbd-client ~]# parted /dev/rbd0 mkpart primary xfs 0% 100%
Information: You may need to update /etc/fstab.
```

8. Format the partition:

Syntax

```
mkfs.xfs /dev/MAPPED_BLOCK_DEVICE_WITH_PARTITION_NUMBER
```

Example

```
[root@rbd-client ~]# mkfs.xfs /dev/rbd0p1
meta-data=/dev/rbd0p1            isize=512    agcount=16, agsize=163824 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=1, sparse=1, rmapbt=0
         =                       reflink=1
data     =                       bsize=4096   blocks=2621184, imaxpct=25
         =                       sunit=16     swidth=16 blks
naming   =version 2              bsize=4096   ascii-ci=0, ftype=1
log      =internal log           bsize=4096   blocks=2560, version=2
         =                       sectsz=512   sunit=16 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

9. Create a directory to mount the new file system on:

Syntax

```
mkdir PATH_TO_DIRECTORY
```

Example

```
[root@rbd-client ~]# mkdir /mnt/ceph
```

10. Mount the file system:

Syntax

```
mount /dev/MAPPED_BLOCK_DEVICE_WITH_PARTITION_NUMBER
PATH_TO_DIRECTORY
```

Example

```
[root@rbd-client ~]# mount /dev/rbd0p1 /mnt/ceph/
```

11. Verify that the file system is mounted and showing the correct size:

Syntax

```
df -h PATH_TO_DIRECTORY
```

Example

```
[root@rbd-client ~]# df -h /mnt/ceph/
Filesystem      Size  Used Avail Use% Mounted on
/dev/rbd0p1      10G  105M  9.9G   2% /mnt/ceph
```

Additional Resources

- For more information, see *Creating a Ceph Block Device for a Linux kernel module client using Dashboard*.
- For more information, see Managing file systems for Red Hat Enterprise Linux 8.
- For more information, see Storage Administration Guide for Red Hat Enterprise Linux 7.

8.2. MAPPING A BLOCK DEVICE

Use **rbd** to map an image name to a kernel module. You must specify the image name, the pool name and the user name. **rbd** will load the RBD kernel module if it is not already loaded.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. Return a list of the images:

Example

```
[root@rbd-client ~]# rbd list
```

2. Following are the two options to map the image:
   - Map an image name to a kernel module:

Syntax

```
rbd device map POOL_NAME/IMAGE_NAME --id USER_NAME
```

Example

```
[root@rbd-client ~]# rbd device map rbd/myimage --id admin
```

- Specify a secret when using **cephx** authentication by either the keyring or a file containing the secret:

  Syntax

  ```
  [root@rbd-client ~]# rbd device map POOL_NAME/IMAGE_NAME --id USER_NAME --keyring PATH_TO_KEYRING
  ```

  or

  ```
  [root@rbd-client ~]# rbd device map POOL_NAME/IMAGE_NAME --id USER_NAME --keyfile PATH_TO_FILE
  ```

8.3. DISPLAYING MAPPED BLOCK DEVICES

You can display which block device images are mapped to the kernel module with the **rbd** command.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. Display the mapped block devices:

   ```
   [root@rbd-client ~]# rbd device list
   ```

8.4. UNMAPPING A BLOCK DEVICE

You can unmap a block device image with the **rbd** command, by using the **unmap** option and providing the device name.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.
- An image that is mapped.

Procedure

1. Get the specification of the device.

Example

```
[root@rbd-client ~]# rbd device list
```

2. Unmap the block device image:

Syntax

```
rbd device unmap /dev/rbd/POOL_NAME/IMAGE_NAME
```

Example

```
[root@rbd-client ~]# rbd device unmap /dev/rbd/pool1/image1
```

8.5. SEGREGATING IMAGES WITHIN ISOLATED NAMESPACES WITHIN THE SAME POOL

When using Ceph Block Devices directly without a higher-level system, such as OpenStack or OpenShift Container Storage, it was not possible to restrict user access to specific block device images. When combined with CephX capabilities, users can be restricted to specific pool namespaces to restrict access to the images.

You can use RADOS namespaces, a new level of identity to identify an object, to provide isolation between rados clients within a pool. For example, a client can only have full permissions on a namespace specific to them. This makes using a different RADOS client for each tenant feasible, which is particularly useful for a block device where many different tenants are accessing their own block device images.

You can segregate block device images within isolated namespaces within the same pool.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Upgrade all the kernels to 4x and to librbd and librados on all clients.
- Root-level access to the monitor and client nodes.

Procedure

1. Create an **rbd** pool:

Syntax

```
ceph osd pool create POOL_NAME PG_NUM
```

Example

```
[ceph: root@host01 /]# ceph osd pool create mypool 100
pool 'mypool' created
```

2. Associate the **rbd** pool with the RBD application:

Syntax

```
ceph osd pool application enable POOL_NAME rbd
```

Example

```
[ceph: root@host01 /]# ceph osd pool application enable mypool rbd
enabled application 'rbd' on pool 'mypool'
```

3. Initialize the pool with the RBD application:

Syntax

```
rbd pool init -p POOL_NAME
```

Example

```
[ceph: root@host01 /]# rbd pool init -p mypool
```

4. Create two namespaces:

Syntax

```
rbd namespace create --namespace NAMESPACE
```

Example

```
[ceph: root@host01 /]# rbd namespace create --namespace namespace1

[ceph: root@host01 /]# rbd namespace create --namespace namespace2

[ceph: root@host01 /]# rbd namespace ls --format=json
[{"name":"namespace2"},{"name":"namespace1"}]
```

5. Provide access to the namespaces for two users:

Syntax

```
ceph auth get-or-create client.USER_NAME mon 'profile rbd' osd 'profile rbd pool=rbd
namespace=NAMESPACE' -o /etc/ceph/client.USER_NAME.keyring
```

Example

```
[ceph: root@host01 /]# ceph auth get-or-create client.testuser mon 'profile rbd' osd 'profile
rbd pool=rbd namespace=namespace1' -o /etc/ceph/client.testuser.keyring

[ceph: root@host01 /]# ceph auth get-or-create client.newuser mon 'profile rbd' osd 'profile
rbd pool=rbd namespace=namespace2' -o /etc/ceph/client.newuser.keyring
```

6. Get the key of the clients:

Syntax

```
ceph auth get client.USER_NAME
```

Example

```
[ceph: root@host01 /]# ceph auth get client.testuser

[client.testuser]
 key = AQDMp61hBf5UKRAAgjQ2In0Z3uwAase7mrlKnQ==
 caps mon = "profile rbd"
 caps osd = "profile rbd pool=rbd namespace=namespace1"
exported keyring for client.testuser

[ceph: root@host01 /]# ceph auth get client.newuser

[client.newuser]
 key = AQDfp61hVfLFHRAA7D80ogmZl80ROY+AUG4A+Q==
 caps mon = "profile rbd"
 caps osd = "profile rbd pool=rbd namespace=namespace2"
exported keyring for client.newuser
```

7. Create the block device images and use the pre-defined namespace within a pool:

Syntax

```
rbd create --namespace NAMESPACE IMAGE_NAME --size SIZE_IN_GB
```

Example

```
[ceph: root@host01 /]# rbd create --namespace namespace1 image01 --size 1G

[ceph: root@host01 /]# rbd create --namespace namespace2 image02 --size 1G
```

8. Optional: Get the details of the namespace and the associated image:

Syntax

```
rbd --namespace NAMESPACE ls --long
```

Example

```
[ceph: root@host01 /]#  rbd --namespace namespace1 ls --long
NAME    SIZE  PARENT FMT PROT LOCK
image01 1 GiB          2

[ceph: root@host01 /]# rbd --namespace namespace2 ls --long
NAME    SIZE  PARENT FMT PROT LOCK
image02 1 GiB          2
```

9. Copy the Ceph configuration file from the Ceph Monitor node to the client node:

```
scp /etc/ceph/ceph.conf root@CLIENT_NODE:/etc/ceph/
```

Example

```
[ceph: root@host01 /]# scp /etc/ceph/ceph.conf root@host02:/etc/ceph/

root@host02's password:
ceph.conf                                     100%  497   724.9KB/s   00:00
```

10. Copy the admin keyring from the Ceph Monitor node to the client node:

Syntax

```
scp /etc/ceph/ceph.client.admin.keyring root@CLIENT_NODE:/etc/ceph
```

Example

```
[ceph: root@host01 /]# scp /etc/ceph/ceph.client.admin.keyring root@host02:/etc/ceph/

root@host02's password:
ceph.client.admin.keyring                     100%  151   265.0KB/s   00:00
```

11. Copy the keyrings of the users from the Ceph Monitor node to the client node:

Syntax

```
scp /etc/ceph/ceph.client.USER_NAME.keyring root@CLIENT_NODE:/etc/ceph/
```

Example

```
[ceph: root@host01 /]# scp /etc/ceph/client.newuser.keyring root@host02:/etc/ceph/

[ceph: root@host01 /]# scp /etc/ceph/client.testuser.keyring root@host02:/etc/ceph/
```

12. Map the block device image:

Syntax

```
rbd map --name NAMESPACE IMAGE_NAME -n client.USER_NAME --keyring
/etc/ceph/client.USER_NAME.keyring
```

Example

```
[ceph: root@host01 /]# rbd map --namespace namespace1 image01  -n  client.testuser --
keyring=/etc/ceph/client.testuser.keyring

/dev/rbd0

[ceph: root@host01 /]# rbd map --namespace namespace2 image02 -n client.newuser --
keyring=/etc/ceph/client.newuser.keyring

/dev/rbd1
```

- This does not allow access to users in the other namespaces in the same pool.

Example

```
[ceph: root@host01 /]# rbd map --namespace namespace2 image02  -n  client.testuser --
keyring=/etc/ceph/client.testuser.keyring

rbd: warning: image already mapped as /dev/rbd1
rbd: sysfs write failed
rbd: error asserting namespace: (1) Operation not permitted
In some cases useful info is found in syslog - try "dmesg | tail".
2021-12-06 02:49:08.106 7f8d4fde2500 -1 librbd::api::Namespace: exists: error asserting
namespace: (1) Operation not permitted
rbd: map failed: (1) Operation not permitted

[ceph: root@host01 /]# rbd map --namespace namespace1 image01 -n client.newuser --
keyring=/etc/ceph/client.newuser.keyring

rbd: warning: image already mapped as /dev/rbd0
rbd: sysfs write failed
rbd: error asserting namespace: (1) Operation not permitted
In some cases useful info is found in syslog - try "dmesg | tail".
2021-12-03 12:16:24.011 7fcad776a040 -1 librbd::api::Namespace: exists: error asserting
namespace: (1) Operation not permitted
rbd: map failed: (1) Operation not permitted
```

13. Verify the device:

Example

```
[ceph: root@host01 /]# rbd showmapped

id pool namespace      image   snap device
0  rbd  namespace1  image01 -  /dev/rbd0
1  rbd  namespace2   image02 -  /dev/rbd1
```

CHAPTER 9. USING THE CEPH BLOCK DEVICE PYTHON MODULE

The **rbd** python module provides file-like access to Ceph block device images. In order to use this built-in tool, import the **rbd** and **rados** Python modules.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the node.

Procedure

1. Connect to RADOS and open an IO context:

```
cluster = rados.Rados(conffile='my_ceph.conf')
cluster.connect()
ioctx = cluster.open_ioctx('mypool')
```

2. Instantiate an **:class:rbd.RBD** object, which you use to create the image:

```
rbd_inst = rbd.RBD()
size = 4 * 1024**3  # 4 GiB
rbd_inst.create(ioctx, 'myimage', size)
```

3. To perform I/O on the image, instantiate an **:class:rbd.Image** object:

```
image = rbd.Image(ioctx, 'myimage')
data = 'foo' * 200
image.write(data, 0)
```

This writes 'foo' to the first 600 bytes of the image. Note that data cannot be **:type:unicode** - **librbd** does not know how to deal with characters wider than a **:c:type:char**.

4. Close the image, the IO context and the connection to RADOS:

```
image.close()
ioctx.close()
cluster.shutdown()
```

To be safe, each of these calls must to be in a separate **:finally** block:

```
import rados
import rbd

cluster = rados.Rados(conffile='my_ceph_conf')
try:
    ioctx = cluster.open_ioctx('my_pool')
    try:
        rbd_inst = rbd.RBD()
        size = 4 * 1024**3  # 4 GiB
        rbd_inst.create(ioctx, 'myimage', size)
```

```
        image = rbd.Image(ioctx, 'myimage')
        try:
            data = 'foo' * 200
            image.write(data, 0)
        finally:
            image.close()
    finally:
        ioctx.close()
finally:
    cluster.shutdown()
```

This can be cumbersome, so the **Rados**, **Ioctx**, and **Image** classes can be used as context managers that close or shut down automatically. Using them as context managers, the above example becomes:

```
with rados.Rados(conffile='my_ceph.conf') as cluster:
    with cluster.open_ioctx('mypool') as ioctx:
        rbd_inst = rbd.RBD()
        size = 4 * 1024**3  # 4 GiB
        rbd_inst.create(ioctx, 'myimage', size)
        with rbd.Image(ioctx, 'myimage') as image:
            data = 'foo' * 200
            image.write(data, 0)
```

APPENDIX A. CEPH BLOCK DEVICE CONFIGURATION REFERENCE

As a storage administrator, you can fine tune the behavior of Ceph block devices through the various options that are available. You can use this reference for viewing such things as the default Ceph block device options, and Ceph block device caching options.

Prerequisites

- A running Red Hat Ceph Storage cluster.

A.1. BLOCK DEVICE DEFAULT OPTIONS

It is possible to override the default settings for creating an image. Ceph will create images with format **2** and no striping.

rbd_default_format

Description

The default format (**2**) if no other format is specified. Format **1** is the original format for a new image, which is compatible with all versions of **librbd** and the kernel module, but does not support newer features like cloning. Format **2** is supported by **librbd** and the kernel module since version 3.11 (except for striping). Format **2** adds support for cloning and is more easily extensible to allow more features in the future.

Type

Integer

Default

2

rbd_default_order

Description

The default order if no other order is specified.

Type

Integer

Default

22

rbd_default_stripe_count

Description

The default stripe count if no other stripe count is specified. Changing the default value requires striping v2 feature.

Type

64-bit Unsigned Integer

Default

0

rbd_default_stripe_unit

Description

The default stripe unit if no other stripe unit is specified. Changing the unit from **0** (that is, the object size) requires the striping v2 feature.

Type

64-bit Unsigned Integer

Default

0

rbd_default_features

Description

The default features enabled when creating an block device image. This setting only applies to format 2 images. The settings are:
1: Layering support. Layering enables you to use cloning.

2: Striping v2 support. Striping spreads data across multiple objects. Striping helps with parallelism for sequential read/write workloads.

4: Exclusive locking support. When enabled, it requires a client to get a lock on an object before making a write.

8: Object map support. Block devices are thin-provisioned—meaning, they only store data that actually exists. Object map support helps track which objects actually exist (have data stored on a drive). Enabling object map support speeds up I/O operations for cloning, or importing and exporting a sparsely populated image.

16: Fast-diff support. Fast-diff support depends on object map support and exclusive lock support. It adds another property to the object map, which makes it much faster to generate diffs between snapshots of an image, and the actual data usage of a snapshot much faster.

32: Deep-flatten support. Deep-flatten makes **rbd flatten** work on all the snapshots of an image, in addition to the image itself. Without it, snapshots of an image will still rely on the parent, so the parent will not be delete-able until the snapshots are deleted. Deep-flatten makes a parent independent of its clones, even if they have snapshots.

64: Journaling support. Journaling records all modifications to an image in the order they occur. This ensures that a crash-consistent mirror of the remote image is available locally

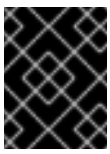
The enabled features are the sum of the numeric settings.

Type

Integer

Default

61 - layering, exclusive-lock, object-map, fast-diff, and deep-flatten are enabled



IMPORTANT

The current default setting is not compatible with the RBD kernel driver nor older RBD clients.

rbd_default_map_options

Description

Most of the options are useful mainly for debugging and benchmarking. See **man rbd** under **Map Options** for details.

Type

String

Default

""

A.2. BLOCK DEVICE GENERAL OPTIONS

rbd_op_threads

Description

The number of block device operation threads.

Type

Integer

Default

1

WARNING

Do not change the default value of **rbd_op_threads** because setting it to a number higher than **1** might cause data corruption.

rbd_op_thread_timeout

Description

The timeout (in seconds) for block device operation threads.

Type

Integer

Default

60

rbd_non_blocking_aio

Description

If **true**, Ceph will process block device asynchronous I/O operations from a worker thread to prevent blocking.

Type

Boolean

Default

true

rbd_concurrent_management_ops

Description

The maximum number of concurrent management operations in flight (for example, deleting or resizing an image).

Type

Integer

Default

10

rbd_request_timed_out_seconds

Description

The number of seconds before a maintenance request times out.

Type

Integer

Default

30

rbd_clone_copy_on_read

Description

When set to **true**, copy-on-read cloning is enabled.

Type

Boolean

Default

false

rbd_enable_alloc_hint

Description

If **true**, allocation hinting is enabled, and the block device will issue a hint to the OSD back end to indicate the expected size object.

Type

Boolean

Default

true

rbd_skip_partial_discard

Description

If **true**, the block device will skip zeroing a range when trying to discard a range inside an object.

Type

Boolean

Default

false

rbd_tracing

Description

Set this option to **true** to enable the Linux Trace Toolkit Next Generation User Space Tracer (LTTng-UST) tracepoints. See Tracing RADOS Block Device (RBD) Workloads with the RBD Replay Feature for details.

Type

Boolean

Default

false

rbd_validate_pool

Description

Set this option to **true** to validate empty pools for RBD compatibility.

Type

Boolean

Default

true

rbd_validate_names

Description

Set this option to **true** to validate image specifications.

Type

Boolean

Default

true

A.3. BLOCK DEVICE CACHING OPTIONS

The user space implementation of the Ceph block device, that is, **librbd**, cannot take advantage of the Linux page cache, so it includes its own in-memory caching, called **RBD caching**. Ceph block device caching behaves just like well-behaved hard disk caching. When the operating system sends a barrier or a flush request, all dirty data is written to the Ceph OSDs. This means that using write-back caching is just as safe as using a well-behaved physical hard disk with a virtual machine that properly sends flushes, that is, Linux kernel version 2.6.32 or higher. The cache uses a Least Recently Used (LRU) algorithm, and in write-back mode it can coalesce contiguous requests for better throughput.

Ceph block devices support write-back caching. To enable write-back caching, set **rbd_cache = true** to the **[client]** section of the Ceph configuration file. By default, **librbd** does not perform any caching. Writes and reads go directly to the storage cluster, and writes return only when the data is on disk on all replicas. With caching enabled, writes return immediately, unless there are more than **rbd_cache_max_dirty** unflushed bytes. In this case, the write triggers write-back and blocks until enough bytes are flushed.

Ceph block devices support write-through caching. You can set the size of the cache, and you can set targets and limits to switch from write-back caching to write-through caching. To enable write-through mode, set **rbd_cache_max_dirty** to 0. This means writes return only when the data is on disk on all replicas, but reads may come from the cache. The cache is in memory on the client, and each Ceph block device image has its own. Since the cache is local to the client, there is no coherency if there are others accessing the image. Running other file systems, such as GFS or OCFS, on top of Ceph block devices will not work with caching enabled.

The Ceph configuration settings for Ceph block devices must be set in the **[client]** section of the Ceph configuration file, by default, **/etc/ceph/ceph.conf**.

The settings include:

rbd_cache

Description

Enable caching for RADOS Block Device (RBD).

Type

Boolean

Required

No

Default

true

rbd_cache_size

Description

The RBD cache size in bytes.

Type

64-bit Integer

Required

No

Default

32 MiB

rbd_cache_max_dirty

Description

The **dirty** limit in bytes at which the cache triggers write-back. If **0**, uses write-through caching.

Type

64-bit Integer

Required

No

Constraint

Must be less than **rbd cache size**.

Default

24 MiB

rbd_cache_target_dirty

Description

The **dirty target** before the cache begins writing data to the data storage. Does not block writes to the cache.

Type

64-bit Integer

Required

No

Constraint

Must be less than **rbd cache max dirty**.

Default

16 MiB

rbd_cache_max_dirty_age

Description

The number of seconds dirty data is in the cache before writeback starts.

Type

Float

Required

No

Default

1.0

rbd_cache_max_dirty_object

Description

The dirty limit for objects - set to **0** for auto calculate from **rbd_cache_size**.

Type

Integer

Default

0

rbd_cache_block_writes_upfront

Description

If **true**, it will block writes to the cache before the **aio_write** call completes. If **false**, it will block before the **aio_completion** is called.

Type

Boolean

Default

false

rbd_cache_writethrough_until_flush

Description

Start out in write-through mode, and switch to write-back after the first flush request is received. Enabling this is a conservative but safe setting in case VMs running on rbd are too old to send flushes, like the virtio driver in Linux before 2.6.32.

Type

Boolean

Required

No

Default

true

A.4. BLOCK DEVICE PARENT AND CHILD READ OPTIONS

rbd_balance_snap_reads

Description

Ceph typically reads objects from the primary OSD. Since reads are immutable, you may enable this feature to balance snap reads between the primary OSD and the replicas.

Type

Boolean

Default

false

rbd_localize_snap_reads

Description

Whereas **rbd_balance_snap_reads** will randomize the replica for reading a snapshot. If you enable **rbd_localize_snap_reads**, the block device will look to the CRUSH map to find the closest or local OSD for reading the snapshot.

Type

Boolean

Default

false

rbd_balance_parent_reads

Description

Ceph typically reads objects from the primary OSD. Since reads are immutable, you may enable this feature to balance parent reads between the primary OSD and the replicas.

Type

Boolean

Default

false

rbd_localize_parent_reads

Description

Whereas **rbd_balance_parent_reads** will randomize the replica for reading a parent. If you enable **rbd_localize_parent_reads**, the block device will look to the CRUSH map to find the closest or local OSD for reading the parent.

Type

Boolean

Default

true

A.5. BLOCK DEVICE READ AHEAD OPTIONS

RBD supports read-ahead/prefetching to optimize small, sequential reads. This should normally be handled by the guest OS in the case of a VM, but boot loaders may not issue efficient reads. Read-ahead is automatically disabled if caching is disabled.

rbd_readahead_trigger_requests

Description

Number of sequential read requests necessary to trigger read-ahead.

Type

Integer

Required

No

Default

10

rbd_readahead_max_bytes

Description

Maximum size of a read-ahead request. If zero, read-ahead is disabled.

Type

64-bit Integer

Required

No

Default

512 KiB

rbd_readahead_disable_after_bytes

Description

After this many bytes have been read from an RBD image, read-ahead is disabled for that image until it is closed. This allows the guest OS to take over read-ahead once it is booted. If zero, read-ahead stays enabled.

Type

64-bit Integer

Required

No

Default

50 MiB

A.6. BLOCK DEVICE BLOCKLIST OPTIONS

rbd_blocklist_on_break_lock

Description

Whether to blocklist clients whose lock was broken.

Type

Boolean

Default

true

rbd_blocklist_expire_seconds

Description

The number of seconds to blocklist - set to 0 for OSD default.

Type

Integer

Default

0

A.7. BLOCK DEVICE JOURNAL OPTIONS

rbd_journal_order

Description

The number of bits to shift to compute the journal object maximum size. The value is between **12** and **64**.

Type

32-bit Unsigned Integer

Default

24

rbd_journal_splay_width

Description

The number of active journal objects.

Type

32-bit Unsigned Integer

Default

4

rbd_journal_commit_age

Description

The commit time interval in seconds.

Type

Double Precision Floating Point Number

Default

5

rbd_journal_object_flush_interval

Description

The maximum number of pending commits per a journal object.

Type

Integer

Default

0

rbd_journal_object_flush_bytes

Description

The maximum number of pending bytes per a journal object.

Type

Integer

Default

0

rbd_journal_object_flush_age

Description

The maximum time interval in seconds for pending commits.

Type

Double Precision Floating Point Number

Default

0

rbd_journal_pool

Description

Specifies a pool for journal objects.

Type

String

Default

""

A.8. BLOCK DEVICE CONFIGURATION OVERRIDE OPTIONS

Block device configuration override options for global and pool levels.

Global level

Available keys

rbd_qos_bps_burst

Description

The desired burst limit of IO bytes.

Type

Integer

Default

0

rbd_qos_bps_limit

Description

The desired limit of IO bytes per second.

Type

Integer

Default

0

rbd_qos_iops_burst

Description

The desired burst limit of IO operations.

Type

Integer

Default

0

rbd_qos_iops_limit

Description

The desired limit of IO operations per second.

Type

Integer

Default

0

rbd_qos_read_bps_burst

Description

The desired burst limit of read bytes.

Type

Integer

Default

0

rbd_qos_read_bps_limit

Description

The desired limit of read bytes per second.

Type

Integer

Default

0

rbd_qos_read_iops_burst

Description

The desired burst limit of read operations.

Type

Integer

Default

0

rbd_qos_read_iops_limit

Description

The desired limit of read operations per second.

Type

Integer

Default

0

rbd_qos_write_bps_burst

Description

The desired burst limit of write bytes.

Type

Integer

Default

0

rbd_qos_write_bps_limit

Description

The desired limit of write bytes per second.

Type

Integer

Default

0

rbd_qos_write_iops_burst

Description

The desired burst limit of write operations.

Type

Integer

Default

0

rbd_qos_write_iops_limit

Description

The desired burst limit of write operations per second.

Type

Integer

Default

0

The above keys can be used for the following:

rbd config global set *CONFIG_ENTITY KEY VALUE*

Description

Set a global level configuration override.

rbd config global get *CONFIG_ENTITY KEY*

Description

Get a global level configuration override.

rbd config global list *CONFIG_ENTITY*

Description

List the global level configuration overrides.

rbd config global remove *CONFIG_ENTITY KEY*

Description

Remove a global level configuration override.

Pool level

rbd config pool set *POOL_NAME KEY VALUE*

Description

Set a pool level configuration override.

rbd config pool get *POOL_NAME KEY*

Description

Get a pool level configuration override.

rbd config pool list *POOL_NAME*

Description

List the pool level configuration overrides.

rbd config pool remove *POOL_NAME KEY*

Description

Remove a pool level configuration override.

NOTE

CONFIG_ENTITY is global, client or client id. ***KEY*** is the config key. ***VALUE*** is the config value. ***POOL_NAME*** is the name of the pool.

A.9. BLOCK DEVICE INPUT AND OUTPUT OPTIONS

General input and output options for Red Hat Ceph Storage.

rbd_compression_hint

Description

Hint to send to the OSDs on write operations. If set to **compressible** and the OSD

bluestore_compression_mode setting is **passive**, the OSD attempts to compress data. If set to **incompressible** and the OSD **bluestore_compression_mode** setting is **aggressive**, the OSD will not attempt to compress data.

Type

Enum

Required

No

Default

none

Values

none, **compressible**, **incompressible**

rbd_read_from_replica_policy

Description

Policy for determining which OSD receives read operations. If set to **default**, each PG's primary OSD will always be used for read operations. If set to **balance**, read operations will be sent to a randomly selected OSD within the replica set. If set to **localize**, read operations will be sent to the closest OSD as determined by the CRUSH map and the **crush_location** configuration option, where the **crush_location** is denoted using **key=value**. The **key** aligns with the CRUSH map keys.

NOTE

This feature requires the storage cluster to be configured with a minimum compatible OSD release of the latest version of Red Hat Ceph Storage.

Type

Enum

Required

No

Default

default

Values

default, **balance**, **localize**