Red Hat Ceph Storage Block Devices

Learn to manage, create, configure, and use Red Hat Ceph Storage 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

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:** Using Ceph Block Devices requires access to a running Ceph storage cluster. For more information about installing a Red Hat Ceph Storage cluster, see Initial installation.

Using Ceph Block Devices

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

Displaying 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
```

Creating block device pools

Learn how to create a block device pool.

PREREQUISITE

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.

Before beginning this procedure, ensure that you have the following:

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

For more information, see Pools.

- Create a Ceph Block Device (`rbd`) pool.

```
ceph osd pool create POOL_NAME PG_NUM

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

For 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
```

Creating images

Before adding a block device to a node, create a Ceph Block Device 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, run 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.

Reference

for more information, see “Creating block device pools” on page 2.

Listing images

List the Ceph Block Device images with the **`rbd ls`** command.

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, run 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
   ```

Retrieving 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
   ```

Resizing images

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.

PREREQUISITE

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

1. To increase 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 decrease the maximum size of a Ceph Block Device image for the default `rbd` pool:
   Syntax

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

   Example

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

3. To increase 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
   ```

4. To decrease the maximum size of a Ceph Block Device image for a specific pool:
   Syntax

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

   Example

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

Removing images

Remove a Ceph Block Device image with the **`rbd rm`** command.

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

- A running Red Hat Ceph Storage cluster.
- Root-level access to the client node.
- Remove a block device from the default `rbd` pool.

  ```
  rbd rm IMAGE_NAME
  ```

  For example,

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

- Remove a block device from a specific pool.

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

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

Moving images to trash

RADOS Block Device (RBD) images can be moved to the trash using the **`rbd trash`** command.

The **`rbd trash`** 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. Move an image to the trash:

   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. Run 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 run 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. Run 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.
   ```

Schedule automatic trash purge

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, run:

   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, run:

   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, run:

   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, run:

   Syntax

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

   Example

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

Enabling and disabling image features

The Ceph Block Device image features, 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
   ```

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

3. Disable a feature:

Syntax

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

- 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
  ```

Working with image metadata

Ceph supports adding custom image metadata as key-value pairs. The pairs do not have any strict format. By using metadata you can also set the RADOS Block Device (RBD) configuration parameters for particular images. Use the **`rbd image-meta`** command 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.

Reference

For a list of possible configuration options, see “General options” on page 84.

Moving images between pools

You can move RADOS Block Device (RBD) images between different pools within the same cluster. This process copies the source image 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 source 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_NAME/IMAGE_NAME` format.
   - `TARGET_IMAGE` with the name of the new image. Use the `POOL_NAME/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.

Migrating pools

You can migrate or copy RADOS Block Device (RBD) images. In migrating pools the source image is exported and then imported.

PREREQUISITE

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

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.
- 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.

- Migrate the volume.

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

For 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.

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

For 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
```

`rbdmap` service

The `systemd` unit file, `rbdmap.service`, is included with the `ceph-common` package. The `rbdmap.service` unit runs the `rbdmap` shell script. This script automates the mapping and unmapping of RADOS Block Devices (RBD) for one or more RBD images.

The `rbdmap` 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 POOL_NAME/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.

Reference

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

Configuring `rbdmap` service

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=1024'
   ```

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

   Example

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

Reference

For more information about the RBD system service, see “rbdmap service” on page 11.

Persistent write log cache

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

- 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

- 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

- 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.

Limitations

Understand the limitations when using Persistent Write Log (PWL) cache.

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.

Enabling

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. For more information about the `exclusive-lock` feature, see “Enabling and disabling image features” on page 7.

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:

   i. 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
```

ii. 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
```

iii. 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_DAX_ENABLED_FOLDER/WRITE_BACK_CACHE_FOLDER <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.

   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
   ```

Reference

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

Checking 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
  ```

Flushing

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
```

Reference

For more information, see “Discarding” on page 16.

Discarding

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
  ```

Managing Ceph Block Device namespaces

Ceph Block Device namespaces provide logical isolation for block device images within a pool. A namespace enables isolation of images for different applications, tenants, or workloads without requiring separate pools.

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

- Ensure that the target pool exists.
- Ensure that you have appropriate permissions to manage Ceph Block Device resources.
- Ensure that the Ceph cluster is accessible from the command-line interface.

Each namespace exists within a pool and is referenced as part of the image specification. You can create, list, remove, and use namespaces to organize Ceph Block Device images within a pool.

Using namespaces provides the following benefits:

- Logical separation of images within a pool.
- Improved organization of resources.
- Multi-tenant isolation within the same pool.
- Simplified management of large numbers of images.

An image-spec can be identified by using the following syntax:

```
[POOL_NAME/[NAMESPACE/]]IMAGE
```

Where,

- The default for *POOL_NAME* is `rbd`
- The default for *NAMESPACE* is "". When no namespace is specified, the image resides in the default namespace.

> **Note:** *POOL_NAME* is required if an image name contains a slash character (/).

Related information

Removing images

Creating a namespace

Create a new namespace within an existing pool to logically isolate block device images for specific applications or tenants.

- Create a namespace within a pool.

  ```
  rbd namespace create POOL_NAME/NAMESPACE
  ```

  For example,

  ```
  [root@rbd-client ~]# rbd namespace create mypool/dev
  ```

AFTER COMPLETING THE TASK

Verify that the namespace is created by listing namespaces in a pool. Check that the namespace is listed in the output.

```
rbd namespace list POOL_NAME
```

Removing a namespace

Delete a namespace from a pool when it is no longer needed. The namespace must be empty before removal.

1. Ensure that the namespace is empty.

   ```
   rbd ls -p POOL_NAME --namespace NAMESPACE
   ```

   The output should be empty. If any images are listed, the namespace is not empty.

   - If the output is empty, continue to step 3.
   - If the output contains remaining images, continue to step 2.

2. If needed, remove any images within the namespace.
   a. Remove the images.

      ```
      rbd rm -p POOL_NAME --namespace NAMESPACE IMAGE_NAME
      ```

   b. Verify that the namespace is empty.
   Repeat step 1 and 2a until no images are listed.

3. Remove the namespace.

   ```
   rbd namespace remove POOL_NAME/NAMESPACE
   ```

   For example,

   ```
   [root@rbd-client ~]# rbd namespace remove mypool/dev
   ```

AFTER COMPLETING THE TASK

Verify that the namespace is removed.

```
rbd namespace list POOL_NAME
```

The namespace will no longer be listed in the output.

Using namespaces in Ceph Block Device operations

Use a namespace to scope Ceph Block Device commands to a specific group of images within a pool, either by including the namespace in the image path or by using the **--namespace** option.

Both methods are functionally equivalent. When using these commands:

- Always specify both the pool and the namespace to avoid ambiguity.
- Use the **--namespace** option when scripting or when you want to separate the namespace from the image path.

- Choose one of the following methods to create an image in a namespace.
  - Specify the namespace in the image path.

    ```
    rbd create POOL_NAME/NAMESPACE/IMAGE --size SIZE
    ```

    For example,

    ```
    [root@rbd-client ~]# rbd create mypool/dev/myimage --size 10240
    ```

    This command creates the image `myimage` in the `dev` namespace within the `mypool` pool.
  - Use the **--namespace** option.

    ```
    rbd --namespace NAMESPACE -p POOL_NAME create IMAGE --size SIZE
    ```

    For example,

    ```
    [root@rbd-client ~]# rbd --namespace dev -p mypool create myimage --size 10240
    ```

    This command creates the image `myimage` in the `dev` namespace within the `mypool` pool.

Monitoring performance of Ceph Block Devices

Learn how to monitor performance of Ceph Block Devices using the command-line interface.

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.

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.

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 or 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.

Images with encryption support live migration.

> **Important:** Currently, the `krbd` kernel module does not support live migration.

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 or NBD export. 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:

1. 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.

   For migration preparation instructions, see "Preparing for default live migration" on page 23 or "Preparing import-only migration" on page 24, according to your needs..

2. Run 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.

   For migration instructions, see "Running the live migration" on page 25.

3. Finish migration.
   You can commit or stop (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. For committing instructions, see "Committing the live migration" on page 26.

   Stopping the migration remove the cross-links, and will remove the target image. For instructions to stop the migration, see "Stopping a live migration" on page 27.

Live migration formats

`Native`, `qcow`, and `raw` are supported live migration format types and layouts.

Use the following `source-spec` JSON example file for live migration.

```
{
    "type": "native",
    ["cluster_name": "CLUSTER_NAME",] (specify if image in another cluster, requires
`CLUSTER_NAME.conf` file)
    ["client_name": "CLIENT_NAME",] (for connecting to another cluster, default is
`client.admin`)
    "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",] (specify if image is in trash)
    "snap_name": "SNAP_NAME",
    ["snap_id": "SNAP_ID",] (optional, alternative to "SNAP_NAME" key)
}
```

`native` format

The `native` format does not include the stream object, since it utilizes native Ceph operations. For example, to import from the `rbd/ns1/image1@snap1` image, encode the `source-spec` as follows:

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

`qcow` format

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.

Link the `qcow` format data to any supported stream source, by using the following `source-spec` file:

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

`raw` format

Use the `raw` format to describe a thick-provisioned, raw block device export that is **`rbd export –export-format 1 SNAP_SPEC`**.

Link the `raw` format data to any supported stream source, by using the following `source-spec` file:

```
{
    "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.

Live migration streams

Use `file`, `HTTP`, `S3`, and `NBD` streams for live migration.

File stream

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

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

The following is a `source-spec` JSON file that imports a raw-format image from a file located at `/mnt/image.raw`.

```
{
    "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.

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

The following is a `source-spec` JSON file that imports a raw-format image from a file located at `http://download.ceph.com/image.raw`.

```
{
    "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.

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

The following is a `source-spec` JSON file that imports a raw-format image from a file located at `http://s3.ceph.com/bucket/image.raw`.

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

NBD stream

You can use the `NBD` stream to import from a remote NBD export.

`nbd-uri` parameter must follow the NBD URI format. The default NBD port is `10809`.

For more information, see The NBD Uniform Resource Indicator (URI) format.

```
{
            <format unique parameters>
            "stream": {
                "type": "nbd",
                "uri": "<nbd-uri>",
```

```
        }
    }
```

The following is a `source-spec` JSON file that imports a raw-format image from an NBD export located at `nbd://nbd.ceph.com/image.raw`.

```
{
            "type": "raw",
            "stream": {
                "type": "nbd",
                "uri": "nbd://nbd.ceph.com/image.raw",
            }
        }
```

Preparing for default live migration

Prepare the default live migration process for RADOS Block Device (RBD) images within the same Red Hat Ceph Storage cluster.

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

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

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 results in a failure.

Important: Cloned images are implicitly flattened during live importing (using the **`--import-only`** and these images are disassociated from any parent chain in the source cluster when migrated to another Ceph cluster.

1. Optional: If you are migrating the image from one Ceph cluster to another, copy the `ceph.conf` and `ceph.client.admin.keyring` of both the clusters to a common node. This ensures the client node has access to both clusters for migration.
   - Example of copying `ceph.conf` and `ceph.client.admin.keyring` of cluster `c1` to a common node.

     ```
     [root@rbd1-client /]# scp /etc/ceph/ceph.conf root@10.0.67.67:/etc/ceph/c1.conf
     root@10.0.67.67's password:
     ceph.conf
       100%  263     1.2MB/s   00:00

     [root@rbd1-client /]# scp /etc/ceph/ceph.client.admin.keyring  root@10.0.67.67:/
     etc/ceph/c1.keyring
     root@10.0.67.67's password:
     ceph.client.admin.keyring
     ```

   - Example of copying `ceph.conf` and `ceph.client.admin.keyring` of cluster `c2` to a common node.

```
[root@rbd2-client]# scp /etc/ceph/ceph.conf root@10.0.67.67:/etc/ceph/c2.conf
ceph.conf
  100%  261   864.5KB/s   00:00

[root@rbd2-client]# scp /etc/ceph/ceph.client.admin.keyring root@10.0.67.67:/etc/
ceph/c2.keyring
root@10.0.67.67's password:
ceph.client.admin.keyring
```

2. Prepare the live migration within the storage cluster or rename the source image.
   - Prepare the live migration.

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

     For example,

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

   - Rename the source image.

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

     In the following example, `newsourceimage1` is the renamed source image.

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

AFTER COMPLETING THE TASK

1. Check the current state of the live migration process.

   ```
   rbd status TARGET_POOL_NAME/SOURCE_IMAGE_NAME
   ```

   For 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.
   For example:

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

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

2. Run the live migration. For more information, see “Running the live migration” on page 25.

Preparing import-only migration

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

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

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

1. Create a JSON file.
   For 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.

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

   For example,

```
[ceph: root@rbd-client /]# rbd migration prepare --import-only --source-spec-path
"testspec.json" targetpool1/sourceimage1
```

AFTER COMPLETING THE TASK

1. Check the current state of the import-only live migration process.

```
rbd status TARGET_POOL_NAME/SOURCE_IMAGE_NAME
```

   For example,

```
[ceph: root@rbd-client /]# rbd status targetpool1/sourceimage1
Watchers: none
Migration:
source: {"stream":
{"access_key":"RLJOCP6345BGB38YQXI5","secret_key":"oahWRB2ote2rnLy4dojYjDrsvaBADriDDgt
Sfk6o","type":"s3","url":"http://10.74.253.18:80/testbucket1/image.raw"},"type":"raw"}
destination: targetpool1/sourceimage1 (b13865345e66)
state: prepared
```

   The following example shows migrating data from Ceph cluster `c1` to Ceph cluster `c2`:

```
[ceph: root@rbd-client /]# cat /tmp/native_spec
{
  "cluster_name": "c1",
  "type": "native",
  "pool_name": "pool1",
  "image_name": "image1",
  "snap_name": "snap1"
}
ceph: root@rbd-client /]# rbd migration prepare --import-only --source-spec-path /tmp/
native_spec c2pool1/c2image1 --cluster c2
ceph: root@rbd-client /]# rbd migration execute c2pool1/c2image1 --cluster c2
Image migration: 100% complete...done.
ceph: root@rbd-client /]# rbd migration commit c2pool1/c2image1 --cluster c2
Commit image migration: 100% complete...done.
```

2. Run the live migration. For more information, see “Running the live migration” on page 25.

Running the live migration

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

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

- A running Red Hat Ceph Storage cluster.
- Two block device pools.
- One block device image with migration, as prepared in either “Preparing for default live migration” on page 23 or “Preparing import-only migration” on page 24.

> **Note:** The sub-commands help users copy the image blocks. You are not required to take any further action other than the **execute** command.

- Run the live migration.

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

For example,

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

AFTER COMPLETING THE TASK

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

```
rbd status TARGET_POOL_NAME/SOURCE_IMAGE_NAME
```

For example,

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

2. Commit the live migration. For more information, see “Committing the live migration” on page 26.

Committing the live migration

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.

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

- A running Red Hat Ceph Storage cluster.
- Two block device pools.
- One block device image, that was used while running the live migration.
- A completed migration block deep-copy process, in the `executed` state.
- Commit the migration.

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

For example,

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

Committing the live migration removes the cross-links between the source and target images and removes the source image from the source pool.

AFTER COMPLETING THE TASK

To see which source images were removed from the source pool, use the **`trash list`** command.

```
rbd trash list --all SOURCE_POOL_NAME
```

For example,

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

Stopping a live migration

You can revert the live migration process.

PREREQUISITE

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

Stopping the live migration reverts the prepare and run steps.

- Stop the live migration process.

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

  For example,

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

AFTER COMPLETING THE TASK

After the live migration process is stopped, the target image is deleted and access to the original source image is restored in the source pool.

Verify that the source image is restored, by using the **`rbd ls`** command.

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

Image encryption

Learn how to set a secret key that is used to encrypt a specific RADOS Block Device (RBD) image. Image level encryption is handled internally by RBD clients.

> **Note:**
>
> - The `krbd` module does not support image level encryption.
> - You can use external tools such as `dm-crypt` or `QEMU` to encrypt an RBD image.

Before you begin, make sure that you have the following prerequisites in place:

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

Encryption format

Ceph Block Device 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:

- Currently you can only encrypt flat Ceph Block Device images. Clones of an encrypted RBD image are inherently encrypted using the same encryption profile and passphrase.
- Any data written to the RBD image before formatting might become unreadable, even though it might still occupy storage resources. Ceph Block Device images with the journal feature enabled cannot be encrypted.

Encryption load

By default, all RBD APIs treat encrypted Ceph Block Device images the same way as unencrypted Ceph Block Device 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. All I/Os for the opened RBD image are encrypted or decrypted, for a cloned RBD image, this includes I/Os 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 using the opened image context return the effective image size. The encryption is loaded automatically when mapping the Ceph Block Device 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.

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 will be stripped off.

```
rbd encryption format POOL_NAME/LUKS_IMAGE luks1|luks2 passphrase.txt
```

For example,

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

> **Note:** You can select either `luks1` or `luks2` 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 up to 136 MiB in LUKS, but it is usually upto 16 MiB, dependent on the version of `libcryptsetup` installed. For optimal performance, the encryption format will set the data offset to be aligned with the image object size. For example, expect a minimum overhead of 8 MiB if using an image configured with an 8 MiB 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, will 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 will further deteriorate performance. Avoid using 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:

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

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

Adding encryption format to images and clones

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.

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.

Prerequisites

- A running Red Hat Ceph Storage cluster with RADOS 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 luks2 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 luks2 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.

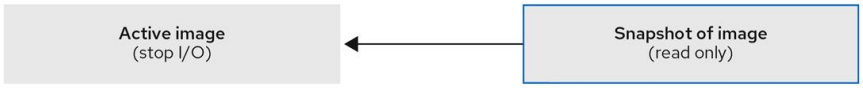
Managing snapshots

A snapshot is a read-only copy of the state of an image at a particular point in time. 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.

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. Stop any `I/O` usage 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: Ceph Block device snapshots



For more information see the `fsfreeze(8)` man page for more details.

Creating snapshots

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
     ```

Listing 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
```

Roll back snapshots

Learn how to 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 run 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
   ```

Deleting 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.

Reference

For more information, see "Flattening cloned images" on page 38.

Purging snapshots

Purge block device snapshots.

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

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

Note: Purging a snapshot means deleting all snapshots in the image, where deleting a snapshot refers to removing a specific snapshot from the image.

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

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

For example,

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

Renaming snapshots

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`.

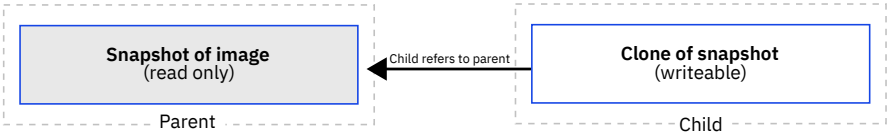
2. Run the `rbd help snap rename` command to display additional details on renaming snapshots.

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: 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.

Reference

- For more information on `flattening`, see "Flattening cloned images" on page 38.

Protecting snapshots

Learn how to protect a block device snapshot from being accidentally deleted when deleting the parent 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.

Cloning block device snapshot

Cloning block device snapshots provides an efficient way to create writable copies of existing block device snapshots without duplicating data. You can either clone a single block device snapshot or a group block device snapshot based on your requirement.

Cloning a single 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.

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

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

Note: Clone v2 does not require protection of snapshots.

- Clone a snapshot.
  To clone a snapshot, specify the parent pool, snapshot, child pool, and image name.

```
rbd clone --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
```

  For example,

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

Cloning block device group snapshot

You can clone new groups from group snapshots created with the **`rbd group snap create`** command with the latest **`--snap-id`** option for the **`rbd clone`** command.

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

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

Cloning from group snapshots is supported only with clone v2 `--rbd-default-clone-format 2`.

```
[root@rbd-client]# ] rbd clone --snap-id 4 pool1/image1 pool1/i1clone1 --rbd-default-clone-
format 2
```

1. Get the snap ID of the group snapshot.

```
rbd snap ls --all POOL_NAME/PARENT_IMAGE_NAME
```

   The following example has a group snapshot, indicated with `SNAPID` listed as **4** and the `NAMESPACE` as `group`.

```
[root@rbd-client] # rbd snap ls --all pool1/image1
SNAPID     NAME             SIZE     PROTECTED   TIMESTAMP
 NAMESPACE
3          snap1            10 GiB   yes         Thu Jul 25 06:21:33 2024     user
4          .group.2_39d     10 GiB               Wed Jul 31 02:28:49 2024     group
(pool1/group1@p1g1snap1)
```

2. Create a clone of the group snapshot using the **`--snap-id`** option.

```
rbd clone --snap-id <SNAP_ID> POOL_NAME/IMAGE_NAME POOL_NAME/CLONE_IMAGE_NAME --rbd-
default-clone-format 2
```

```
[root@rbd-client]# rbd clone --snap-id 4 pool1/image1 pool2/clone2 --rbd-default-
clone-format 2
```

AFTER COMPLETING THE TASK

Use the **`rbd ls`** command to verify the clone image of the group snapshot is created successfully.

```
[root@rbd-client]# rbd ls -p pool2
clone2
```

Unprotecting snapshots

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
   ```

Listing 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, run 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
   ```

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
   ```

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.

The `rbd-mirror` daemon can run either on a single Ceph storage cluster for one-way mirroring or two Ceph storage clusters for two-way mirroring.

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 writing those changes to the local, non-primary image. You can configure Ceph Block Device mirroring either by one-way or two-way mirroring. These mirroring configurations can be done either on pools or on individual images.

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

- A pool with the same name exists on both storage clusters.
- For journal-based mirroring, a pool contains journal-enabled images that 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. Also, the network must have sufficient bandwidth between the two data center sites to handle mirroring.

Mirroring is configured on a per-pool basis with mirror peering storage clusters. Red Hat Ceph Storage 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.

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

- You can modify images in the primary state.
- You can not modify images in the non-primary state.

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

- Implicitly by enabling mirroring in pool mode and only for journal-based images.
- Explicitly by enabling the mirroring of a specific image.

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

One-way replication

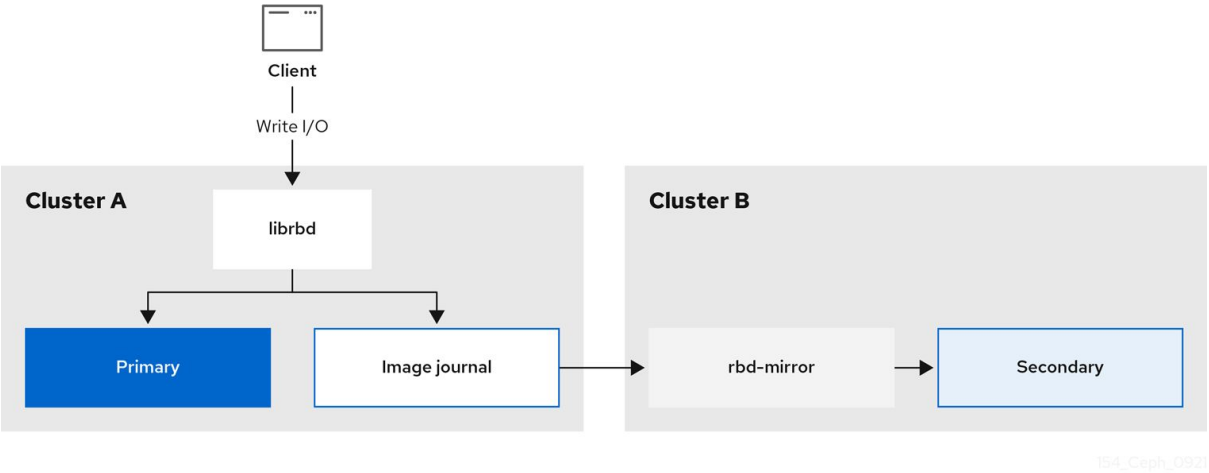
One-way mirroring implies that a primary image or pool of images in one storage cluster gets replicated to a secondary storage cluster. One-way mirroring also supports replicating to multiple secondary storage clusters.

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 Red Hat Ceph Storage storage clusters and you want to replicate images from a primary storage cluster to a secondary storage cluster.
- The secondary storage cluster contains the `rbd-mirror` daemon that can run on one of the cluster nodes. The `rbd-mirror` daemon connects to the primary storage cluster to sync images to the secondary storage cluster.

Figure: One-way mirroring



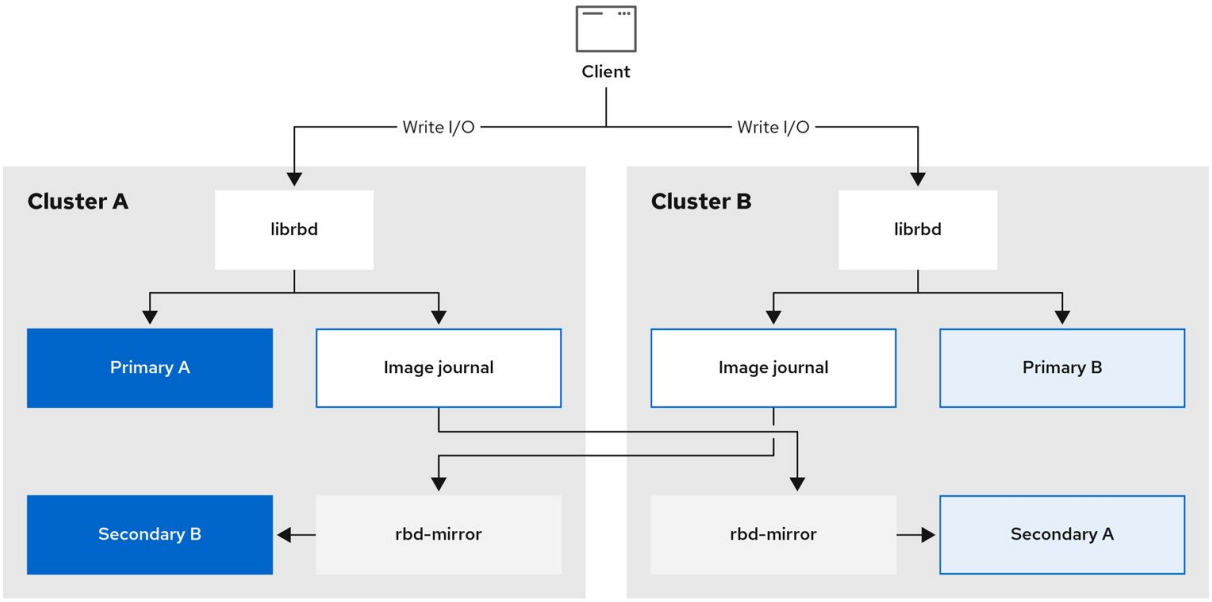
Two-way replication

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 `rbd-mirror` daemon running. The images that are primary on cluster1 are synced by the `rbd-mirror` daemon on the remote cluster and the images on the remote cluster are synced to the primary.

Figure: Two-way mirroring



Configuring Ceph Block Device mirroring

Configure one-way and two-way replication of a Ceph Block Device (`rbd`) pool to enable data synchronization between the primary storage cluster and a secondary cluster by using the command-line interface.

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

- A minimum of two 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.

Note:

- When using one-way replication, you can mirror to multiple secondary storage clusters.
- When using two-way replication you can only mirror between two storage clusters.

Examples in this section 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 that is used in these examples is called `data`.

Journal-based mirroring

Configure asynchronous replication for a block image using Ceph Block Device journaling to ensure near real-time, crash-consistent synchronization between clusters.

1. Log in to the `cephadm` shell on both the sites.
   For 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.

Note:

- For two-way replication, repeat this step on site-a.
- For one-way replication, continue to step "3" on page 42.

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

Where *NODENAME* is the host where you want to configure mirroring in the secondary cluster.

For example,

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

3. Enable journaling features on an image on `site-a`.
   a. Use the **`--image-feature`** option for new images.

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

      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`

      For example,

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

   b. Use the **`rbd feature enable`** command for existing images.
      Syntax,

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

      For example,

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

   c. Enable journaling on all new images by default.
      Set the configuration parameter by using the **`ceph config set`** command.
      For 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 on both the storage clusters.
   The mode can be either`pool`or`image`.

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

   Use one of the following options, enabling either with pool or image, based on your requirements.

   - The following example enables mirroring of the whole pool named `data`.

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

   - The following example enables image mode mirroring on the pool named `data`.

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

5. Verify that mirroring is successfully enabled at both the sites.

```
rbd mirror pool info POOL_NAME
```

For 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 the Ceph client node, bootstrap the storage cluster peers.
   a. Create Ceph user accounts, and register the storage cluster peer to the pool.

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

The following example bootstrap command creates the `client.rbd-mirror.site-a` and the `client.rbd-mirror-peer` Ceph users.

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

   b. Copy the bootstrap token file to the `site-b` storage cluster.
   c. Import the bootstrap token on the `site-b` storage cluster.

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

For 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
```

AFTER COMPLETING THE TASK

Continue with verifying the mirroring status, see "What to do next" on page 45.

Snapshot-based mirroring

Set up periodic replication for a block image by creating and syncing snapshots to provide point-in-time consistency across clusters.

1. Log in to the `cephadm` shell on both the sites.
   For 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.

> **Note:**
> - For two-way replication, repeat this step on site-a.
> - For one-way replication, continue to step "3" on page 44.

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

Where *NODENAME* is the host where you want to configure mirroring in the secondary cluster.

For example,

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

3. Enable image mode mirroring on the pool.

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

The following example enables image mode mirroring on the pool named `data`.

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

4. Verify that mirroring is successfully enabled at both the sites.

```
rbd mirror pool info POOL_NAME
```

For example,

```
[ceph: root@site-a /]# rbd mirror pool data
Mode: pool
Site Name: c13d8065-b33d-4cb5-b35f-127a02768e7f
Mirror UUID: be874515-c563-4bb0-a24a-962a2784c665
Remote Namesapace:

Peer Sites: none

[ceph: root@site-b /]# rbd mirror pool info data
Mode: pool
Site Name: a4c667e2-b635-47ad-b462-6faeeee78df7
Mirror UUID: be874515-c563-4bb0-a24a-962a2784c665

Peer Sites: none
```

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

> **Note:** For two-way mirroring, repeat these substeps in opposite order. First creating a bootstrap on `site-b` and then importing on `site-a`.

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

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

The following example bootstrap command creates the `client.rbd-mirror.site-a` and the `client.rbd-mirror-peer` Ceph users.

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

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

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

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

For 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
```

6. On the Primary site, create a new Ceph Block Device image in a specific pool.

```
rbd create POOL_NAME/IMAGE_NAME --size SIZE
```

For example,

```
[ceph: root@rbd-client-site-a /]# rbd create data/image1 --size 1024
```

7. Enable snapshot image mirroring.

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

For example,

```
[ceph: root@rbd-client-site-a /]# rbd mirror image enable data/image1 snapshot
```

8. Create a snapshot mirror schedule.

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

For more information, see “Scheduling mirror snapshots” on page 56.
For example,

```
[ceph: root@rbd-client-site-a /]# rbd --cluster site-a mirror snapshot schedule add --
pool data --image image1 6h
```

AFTER COMPLETING THE TASK

Continue with verifying the mirroring status, see “What to do next” on page 45.

What to do next

- Verify the mirroring status from a Ceph Monitor node on the primary and secondary sites.

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

In the following example, `up` means the `rbd-mirror` daemon is running, and `stopped` means that this image is not the target for replication from another storage cluster. The status is `stopped` because the image is primary on this storage cluster.

```
[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

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

Administering Ceph Block Device mirroring

As a storage administrator, you can perform a range of tasks to manage and maintain the Ceph Block Device (RBD) mirroring environment.

These tasks include:

- Viewing information about storage cluster peers.
- Add or remove a storage cluster peer.
- Getting mirroring status for a pool, image and group.
- Enabling mirroring on a pool, image and group.
- Disabling mirroring on a pool, image and group.

- Delaying block device replication.
- Promoting and demoting an image and group.

These tasks help ensure smooth replication operations and support failover and recovery scenarios in multi-site deployments.

Viewing peer information

Learn to view information about storage cluster peers using the command-line.

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

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

View information about storage cluster peers.

- 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
  ```

Managing mirroring on a pool

Learn to enable and disable mirroring on a Ceph Block Device pool by using the command-line interface. You can also get the mirroring status for a pool.

PREREQUISITE

Before you begin, make sure that you have root-level access to the node.

Enabling mirroring on a pool

- Enable mirroring on a pool, by using the **`mirror pool enable`** command.
  Be sure to run the command on both peer clusters.

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

  The following example enables `pool` mode mirroring of the whole pool named `data`.

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

Disabling mirroring on a pool

PREREQUISITE

Before disabling mirroring, remove any related peer clusters.
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.

- Disable mirroring on a pool, by using the **`mirror pool disable`** command.

```
rbd mirror pool disable POOL_NAME
```

The following example disables mirroring of a pool named `data`.

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

Getting mirroring status for a pool

Get the mirroring pool status summary, by using the **`mirror pool status`** command.

- Run the following command to get the pool status.
  Syntax

  ```
  rbd mirror pool status POOL_NAME
  ```

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

  Example

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

Managing mirroring on images

Learn to enable and disable mirroring on Ceph Block Device images by using the command-line interface. You can also get the mirroring status for an image. Also, learn to promote, demote, and resynchronize an image.

PREREQUISITE

Before you begin, make sure that you have root-level access to the node.

Enabling mirroring on an image

Enable mirroring on an image by using the command in the procedure.

- Enable mirroring for a specific image within the pool, by using the **`mirror image enable`** command.
  Syntax

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

  The following example enables mirroring for the `image2` image in the `data` pool.

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

Disabling mirroring on an image

Disable mirroring on an image by using the command in the procedure.

- Disable mirroring for a specific image, by using the **`mirror image disable`** command.
  Syntax

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

  The following example disables mirroring of the `image2` image in the `data` pool.

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

Getting mirroring status for an image

Get the mirroring status for an image by using the command in the procedure.

- Get the mirror status for an image, by running the **`mirror image status`** command.
  Syntax

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

The following example gets the status of the `image2` image in the `data` pool.

```
[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: 2025-05-01 09:37:58
```

Promoting an image

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

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

Promote an image by using the command in the procedure.

> **Warning:** Do not force promote non-primary images that are still syncing as the images are not valid after the promotion.

- Promote an image to primary, by using the **`mirror image promote`** command.
  Syntax

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

  The following example promotes `image2` in the `data` pool.

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

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

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

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

Demoting an image

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

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

Demote an image by using the command in the procedure.

> **Warning:** Do not force promote non-primary images that are still syncing as the images are not valid after the promotion.

- Demote an image to non-primary, by using the **`mirror image demote`** command.
  Syntax

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

The following example demotes the `image2` image in the `data` pool.

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

Resynchronizing an image

You can resynchronize an image if an inconsistency occurs between the two peer clusters. When an image enters an inconsistent state, the `rbd-mirror` daemon skips mirroring for that image until the issue is resolved.

- Resynchronize to the primary image, by using the **`mirror image resync`** command.
  Syntax

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

  The following example requests resynchronization of `image2` in the `data` pool.

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

Managing mirroring on a consistency group

Learn to enable and disable mirroring on Ceph Block Device consistency group by using the command-line interface. You can also get the mirroring status for the group. Also, learn to promote, demote, and resynchronize the group. The commands in the following section supports the optional **`namespace`** parameter and you can use this parameter in the command as **`POOL_NAME/NAMESPACE/GROUP_NAME`**.

PREREQUISITE

> **Note:** You can mirror a maximum of 100 images in total per cluster and a maximum of 50 images per group.

- Make sure that you have root-level access to the node.
- Pool mirroring is enabled in the `image` mode. For more information, see step 5 in Configuring two-way Ceph device block mirroring.

Enabling mirroring on a group

> **Note:**
> - Group mirroring works with `image` mode only.
> - Consistency group mirroring only supports `snapshot` mirroring mode and `journal` mode is not supported.

Enable mirroring for a group within the pool, by using the **`mirror group enable`** command.

- Run the following command to enable mirroring on a group.
  Syntax

  ```
  rbd mirror group enable POOL_NAME/GROUP_NAME
  ```

  The following example enables mirroring for the `test_group` group in the `test_pool` pool.

  ```
  [root@rbd-client ~]# rbd mirror group enable test_pool/test_group
  ```

Note:

- After you enable mirroring for a group, you cannot add or remove new images from that group. If you want to add images, disable group mirroring and add the images. You can then re-enable mirroring.
- Images in the group cannot be enabled for mirroring.

Disabling mirroring on a group

Disable mirroring for a group, by using the **`mirror group disable`** command.

– Run the following command to disable mirroring on a group.
Syntax

```
rbd mirror group disable POOL_NAME/GROUP_NAME
```

The following example disables mirroring of the `test_group` group in the `test_pool` pool.

```
[root@rbd-client ~]# rbd mirror group disable test_pool/test_group
```

Getting mirroring status for a group

Get the mirror status for the group, by running the **`mirror group status`** command.

– Run the following command to get the group mirroring status.
Syntax

```
rbd mirror group status POOL_NAME/GROUP_NAME
```

The following example gets the status of the `test_group` group in the `test_pool` pool.

```
[root@site-a ~]# rbd mirror group status test_pool/test_group
test_group:
  global_id:   8206677f-00e5-4b28-a6cc-121be028cb84
  state:       up+stopped
  description: local group is primary
  service:     admin on dhcp53-181.lab.eng.abc.pqr.com
  last_update: 2025-06-13 17:59:16
  images:
    image:       3/9c6c0e9b-abbb-4219-891a-b451396487a4
    state:       up+stopped
    description: local image is primary
  peer_sites:
    name: site-b
    state: up+replaying
    description: replaying,
{"last_snapshot_bytes":0,"last_snapshot_complete_seconds":10,"local_snapshot_timestamp"
:1749817730,"remote_snapshot_timestamp":1749817730}
    last_update: 2025-06-13 17:59:16
    images:
      image:       3/9c6c0e9b-abbb-4219-891a-b451396487a4
      state:       up+replaying
      description: replaying,
{"bytes_per_second":0.0,"bytes_per_snapshot":0.0,"last_snapshot_bytes":0,"last_snapshot
_sync_seconds":0,"local_snapshot_timestamp":1749817730,"remote_snapshot_timestamp":1749
817730,"replay_state":"idle"}
  snapshots:
    .mirror.primary.8206677f-00e5-4b28-a6cc-121be028cb84.104dda662be0
```

Promoting a group

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

– A running Red Hat Ceph Storage cluster with snapshot-based group mirroring is configured.

– Root-level access to the node.

Promote a group by using the command in the procedure.

- Promote a group to primary, by using the **`mirror group promote`** command.
  Syntax

  ```
  rbd mirror group promote POOL_NAME/GROUP_NAME
  ```

  The following example promotes the group `test_group` in the `test_pool` pool to primary.

  ```
  [root@rbd-client ~]# rbd mirror group promote test_pool/test_group
  ```

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

  Important:
  Before executing the **`mirror group promote`** command with **--force** flag, ensure that the secondary mirror daemon is stopped. This step is a temporary workaround for known issues and may not be necessary once those issues are resolved.

  - Run the command to stop and kill the secondary mirror daemon.

    ```
    ceph orch stop rbd-mirror

    kill -SIGKILL PID_OF_SECONDARY_RBD_MIRROR
    ```

  - Execute the **`mirror group promote`** command with the **--force** flag and restart the mirror daemon after the **`mirror group promote`** command completes execution.

    ```
    rbd mirror group promote --force POOL_NAME/GROUP_NAME

    ceph orch start rbd-mirror
    ```

  Example

  ```
  [root@rbd-client ~]# rbd mirror group promote --force test_pool/test_group
  ```

Demoting a group

Demote a group by using the command in the procedure.

- Demote a group to non-primary, by using the **`mirror group demote`** command.
  Syntax

  ```
  rbd mirror group demote POOL_NAME/GROUP_NAME
  ```

  The following example demotes the group `test_group` image in the `test_pool` pool to non-primary.

  ```
  [root@rbd-client ~]# rbd mirror group demote test_pool/test_group
  ```

Resynchronizing a group

You can resynchronize a group if an inconsistency occurs between the two peer clusters. When a group enters an inconsistent state, the `rbd-mirror` daemon skips mirroring for that group until the issue is resolved.

Resynchronize a group by using the command in the procedure.

- Resynchronize to the primary group, by using the **`mirror group resync`** command.
  Syntax

  ```
  rbd mirror group resync POOL_NAME/GROUP_NAME
  ```

  The following example requests resynchronization of `test_group` in the `test_pool` pool.

  ```
  [root@rbd-client ~]# rbd mirror group resync test_pool/test_group
  ```

Managing mirroring on a namespace

Enable and disable Ceph Block Device mirroring on namespaces by using the command-line interface.

PREREQUISITE

Before you begin, make sure that you have root-level access to the node.
Before configuring a namespace to mirror with a different remote namespace, mirroring must be disabled for both the local and remote namespaces on both the clusters.

Enabling mirroring on a namespace

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

- 2 running Red Hat Ceph Storage clusters.
- `rbd-mirror` daemon service enabled on both clusters.
- Mirroring is in enabled for the pool in which the namespace is added.

> **Note:**

You can mirror a namespace to a namespace with a different name in the remote pool by using the **`--remote-namespace`** option. By default, the namespace is mirrored with the same name on the remote pool.

- Enable mirroring on a namespace, by using the **`mirror pool enable`** command. Be sure to run the command on both peer clusters.
  Syntax

```
rbd mirror pool enable POOL_NAME/LOCAL_NAMESPACE_NAMEMODE --remote-
namespace REMOTE_NAMESPACE_NAME
```

  The mirroring mode options are `image` or `pool`. The default mirroring mode is `pool`.

  `image`

  When configured in `image` mode, mirroring must be explicitly enabled on each image.

  `pool`

  When configured in `pool` mode, all images in the namespace with the journaling feature enabled are mirrored.

  The namespace and remote namespace on the first cluster must match the remote namespace and namespace respectively on the remote cluster.

> **Note:** If the **`--remote-namespace`** option is not provided, the namespace is mirrored to a namespace with the same name in the remote pool.

  The following example enables image mode mirroring between `image-pool/namespace-a` on the first cluster and `image-pool/namespace-b` on the second, remote, cluster.

```
[root@rbd-client ~]#  rbd mirror pool enable image-pool/namespace-a image --remote-
namespace namespace-b

[root@rbd-client ~]#  rbd mirror pool enable image-pool/namespace-b image --remote-
namespace namespace-a
```

Disabling mirroring on a namespace

PREREQUISITE

Before disabling mirroring, any consistency group mirror enabled images in the namespace must be explicitly disabled before disabling mirroring on the namespace if configured in `image` mode.

Disable mirroring on a namespace, by using the **`mirror pool disable`** command.

- Run the following command to disable mirroring on a namespace.
  Syntax

```
rbd mirror pool disable POOL_NAME/NAMESPACE_NAME
```

  Example

```
[root@rbd-client ~]# rbd mirror pool disable image-pool/namespace-a
[root@rbd-client ~]# rbd mirror pool disable image-pool/namespace-b
```

Adding a storage cluster peer

Learn to add a storage cluster peer by using the command-line.

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

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

Add a storage cluster peer to enable the `rbd-mirror` daemon to discover the remote cluster.

For example, to add the `site-a` storage cluster as a peer to `site-b`, perform the following steps from a client node in the `site-b` cluster.

- 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
```

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

Removing a storage cluster peer

Learn to remove a storage cluster peer by using the command-line.

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

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

Specify the peer UUID to remove a storage cluster peer.

- Specify the pool name and the peer Universally Unique Identifier (UUID).
  Syntax:

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

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

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

Delaying block device replication

Learn to delay block device replication by using the command-line.

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

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

Whether using one-way or two-way replication, you can configure delayed replication for RADOS Block Device (RBD) mirrored images. Delayed replication provides a buffer period, allowing you to revert unintended changes on the primary image before they are propagated 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.

- Utilize delayed replication for a specific image, on the primary image by running the following **`rbd`** command.
  Syntax,

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

  The following example sets a 10 minute minimum replication delay on image `vm-1` in the `vms` pool.

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

Converting journal-based mirroring to snapshot-based mirroring

Learn to convert journal-based mirroring to snapshot-based mirroring by using the command-line.

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

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

Convert journal-based mirroring to snapshot-based mirroring by disabling mirroring and enabling snapshots.

1. Log in to 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
   ```

   The following example enables snapshot-based mirroring for the `mirror_image` image in the `mirror_pool` pool.

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

Creating an image mirror-snapshot

Learn to create an image mirror-snapshot by using the command-line.

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

- A minimum of two 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.

Create a mirror snapshot to replicate changes in a Ceph Block Device image when using snapshot-based mirroring.

> **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.

- Create an image-mirror snapshot.
  Syntax

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

  Example

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

Creating consistency group mirror snapshot

Learn to create and get the status consistency group mirror snapshots by using the command-line interface.

PREREQUISITE

Before you begin, make sure that you have the following prerequisites in place:

- A minimum of two 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.

Create a mirror group snapshot to replicate changes in a Ceph Block Device when using snapshot-based mirroring.

- Create a mirror group mirroring snapshot.

  ```
  rbd --cluster CLUSTER_NAME mirror group snapshot POOL_NAME/GROUP_NAME
  ```

  For example,

  ```
  [root@site-a ~]# rbd --cluster site-a mirror group snapshot test_pool/test_group
  ```

AFTER COMPLETING THE TASK

View the created mirror group snapshot by running the **`snap list`** command.

```
rbd group snap list --group GROUP_NAME --pool POOL_NAME
```

Verify that the snapshot is in the `copied` state. Check the snapshot `copied` or `not copied` state from the output in the secondary cluster.

`copied`

The data is completely synced to the secondary cluster

`not copied`

The data is still syncing to the secondary cluster. Continue monitoring the snapshot until it reaches the `copied` state.

Example output for primary cluster

```
[ceph: root@site-a]# rbd group snap list --group group_1 --pool pool_1 --debug_rbd 0
ID            NAME               STATE   NAMESPACE
3a08a33beae1 .mirror.primary.123 created mirror (primary peer_uuids:[0b44e1fd-d4e9-4c65-
b5e6-489ad0a87762])
[ceph: root@site-a]#
```

Example output for secondary cluster

```
[ceph: root@site-b]# rbd group snap list --group group_1 --pool pool_1 --debug_rbd 0

ID            NAME                    STATE   NAMESPACE
3a08a33beae1 .mirror.non-primary.123 created mirror (non-primary peer_uuids:[]
75aeb329-4891-495e-9f4d-f938fbf33fb1:3a08a33beae1 **copied**)
[ceph: root@site-b]#
```

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.

PREREQUISITE

- A minimum of two 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.

Creating a schedule

Create a mirror-snapshot schedule, by using the **`snapshot schedule`** command.

- Run the following command to create a schedule for mirror snapshots.
  Syntax

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

  - Only use **`--cluster`** when the cluster name is different from the default name, `ceph`.
  - Specify the interval in days, hours, or minutes using `d`, `h`, or `m` suffix, respectively.
  - The optional *START_TIME* must be specified using the ISO 8601 (hh:mm:ss+|–hh:mm) time format.

  The following example creates a snapshot schedule with the default `ceph` cluster, every 6 hours, and does not specify a start time.

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

The following example creates a snapshot schedule with the default ceph cluster, every 24 hours, with a start time of 2:00 PM at -5 GMT.

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

Listing snapshot schedules by level

List all snapshot schedules for a specific global, pool or image level, with an optional pool or image name, by using the **snapshot schedule ls** command.

– Run the following command to view the schedules by level.
Syntax

```
rbd --cluster CLUSTER_NAME mirror snapshot schedule ls --pool POOL_NAME --recursive
```

Use the **--recursive** option to list all schedules at the specified level

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
```

Removing a schedule

Remove a mirror-snapshot schedule, by using the **snapshot schedule** command.

– Run the following command to remove the mirror snapshot schedule.
Syntax

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

- Only use **--cluster** when the cluster name is different from the default name, ceph.
- Specify the interval in days, hours, or minutes using d, h, or m suffix, respectively.
- The optional *START_TIME* must be specified using the ISO 8601 (hh:mm:ss+|–hh:mm) time format.

The following example removes a snapshot schedule with the default ceph cluster, every 6 hours, and does not specify a start time.

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

The following example removes a snapshot schedule with the default ceph cluster, every 24 hours, with a start time of 2:00 PM at -5 GMT.

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

Viewing upcoming snapshot schedule

View the status for the next snapshots to be created for snapshot-based mirroring Ceph Block Device images.

– View the status for the next snapshots to be created.
Syntax

```
rbd --cluster CLUSTER_NAME mirror snapshot schedule status --pool POOL_NAME --
image IMAGE_NAME
```

Example

```
[root@rbd-client ~]# rbd --cluster site-a mirror snapshot schedule status --ppol data
--group group1
SCHEDULE    TIME       IMAGE
2025-05-01 18:00:00 data/image1
```

Scheduling consistency group mirror snapshots

Learn to create, remove, list, and check the status of group mirror snapshot schedule by using the command-line interface.

PREREQUISITE

- A minimum of two 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 creation of group mirror-snapshots need to be scheduled.

Creating a schedule

Create a group snapshot schedule, by using the **`mirror group snapshot schedule add`** command.

- Run the following command to create mirror snapshot schedule for group.
  Syntax

  ```
  rbd --cluster CLUSTER_NAME mirror group snapshot schedule add --pool POOL_NAME --
  group GROUP_NAME INTERVAL [START_TIME]
  ```

  - Only use **`--cluster`** when the cluster name is different from the default name, `ceph`.
  - Specify the interval in days, hours, or minutes using `d`, `h`, or `m` suffix, respectively.
  - The optional *START_TIME* must be specified using the ISO 8601 (hh:mm:ss+|–hh:mm) time format.

  The following example creates a mirror group snapshot schedule with the default `ceph` cluster, every 6 hours, and does not specify a start time.

  ```
  [root@site-a ~]# rbd mirror group snapshot schedule add --pool test_pool --group
  test_group 6h
  ```

  The following example creates a mirror group snapshot schedule with the default `ceph` cluster, every 24 hours, with a start time of 2:00 PM at -5 GMT.

  ```
  root@site-a ~]# rbd mirror group snapshot schedule add --pool test_pool --group
  test_group 24h 14:00:00-05:00
  ```

Listing snapshot schedules by level

List all snapshot schedules at the global, pool, namespace or group level by using the **`mirror group snapshot schedule ls`** command.

- List all snapshot schedules at the global, pool, namespace or group level by running the following command.
  Syntax

  ```
  rbd --cluster CLUSTER_NAME mirror group snapshot schedule ls --pool POOL_NAME --
  namespace NAMESPACE --group GROUP_NAME --recursive
  ```

  Use the **`--recursive`** option to list all schedules at the specified level.

  Example

  ```
  [root@rbd-client ~]# # rbd --cluster site-a mirror group snapshot schedule ls --pool
  test_pool --recursive
  POOL NAMESPACE GROUP SCHEDULE
  test_pool test_group every 30m
  ```

- To list the mirror group snapshot schedule, run the following command.
  Syntax

  ```
  rbd --cluster CLUSTER_NAME mirror group snapshot schedule ls --pool POOL_NAME --
  group GROUP_NAME --recursive
  ```

Example

```
[root@rbd-client ~]# rbd --cluster site-a mirror group snapshot schedule ls --pool
test_pool --group test_group --recursive

POOL NAMESPACE GROUP SCHEDULE
test_pool test_group every 30m
```

Removing a schedule

Remove a group snapshot schedule, by using the **`mirror group snapshot schedule remove`** command.

– Run the following command to remove the snapshot schedule for a group.
Syntax

```
rbd --cluster CLUSTER_NAME mirror group snapshot schedule remove --pool POOL_NAME --
group GROUP_NAME INTERVAL START_TIME
```

- Only use **`--cluster`** when the cluster name is different from the default name, `ceph`.
- Specify the interval in days, hours, or minutes using `d`, `h`, or `m` suffix, respectively.
- The optional *START_TIME* must be specified using the ISO 8601 (hh:mm:ss+|–hh:mm) time format.

The following example removes a snapshot schedule with the default `ceph` cluster, every 6 hours, and does not specify a start time.

```
[root@site-a ~]# rbd mirror group snapshot schedule remove --pool test_pool --group
test_group 6h
```

The following example removes a snapshot schedule with the default `ceph` cluster, every 24 hours, with a start time of 2:00 PM at -5 GMT.

```
[root@site-a ~]# rbd mirror group snapshot schedule remove --pool test_pool --group
test_group 24h 14:00:00-05:00
```

Viewing upcoming snapshot schedule

View the status for the next snapshots to be created for snapshot-based mirroring Ceph Block Device consistency group.

– View the status for the next snapshots to be created.
Syntax

```
rbd --cluster CLUSTER_NAME mirror group snapshot schedule status --pool POOL_NAME --
namespace NAMESPACE_NAME --group GROUP_NAME
```

Example

```
[root@rbd-client ~]# rbd --cluster site-a mirror group snapshot schedule status --pool
test_pool --group test_group
SCHEDULE    TIME       GROUP
2025-05-01 18:00:00 test_pool/test_group
```

Configure graceful shutdown for `rbd-mirror`

Configure `rbd-mirror` for immediate shutdown to interrupt Ceph Block Device replication tasks and enable fast failover during maintenance or orchestration updates.

When you manage `rbd-mirror` services, you might need to control how quickly the daemon shuts down during orchestration updates or failover events. Use the **`termination_grace_period_seconds`** option in the service specification to set the time, in seconds, that a daemon can terminate gracefully before it is forcefully stopped.

A long grace period can delay `rbd-mirror` shutdown and affect automated failover processes.

To ensure fast failover, set **`termination_grace_period_seconds`** to `0` in the rbd-mirror service specification.

```
service_type: rbd-mirror
placement:
  count: 1
spec:
  termination_grace_period_seconds: 0
```

Important: External tools rely on quick termination to avoid hung states. When using external tools, configure **`termination_grace_period_seconds`** to `0` so the daemon exits immediately when required.

Note:

- Other services, such as OSDs, can retain defaults as high as 30 seconds for a clean shutdown.
- This setting is propagated into systemd unit files and affects daemon termination during orchestration updates or failover events.

Setting up mirroring between the default and non-default namespaces

Learn to set up RBD mirroring between the default namespace on one cluster and a non-default namespace on the other cluster.

PREREQUISITE

- All nodes of both clusters must be running Red Hat Ceph Storage 8.1.
- A pool with same name exists on both the clusters and is initialized for RBD.
- A namespace is created in the pool on the cluster.

To enable setting up mirroring between the default and non-default namespaces, a new mode `init-only` is introduced in Red Hat Ceph Storage 8.1.

When a pool is configured in `init-only` mode, images in the default namespace are not mirrored. However, mirroring can still be configured for other namespaces.

This configuration is useful when you want to mirror images from a different namespace into the default namespace of a remote pool. While this is the primary use case, `init-only` mode is beneficial in scenarios where selective mirroring is required.

Note: `init-only` mode does not prevent configuration of mirroring rules for non-default namespaces. It simply disables automatic mirroring of images in the default namespace.

The procedure for setting up RBD mirroring from a non-default namespace to the default namespace is effectively the same as configuring mirroring from the default namespace to a non-default namespace, with the clusters reversed.

The steps in the procedure set up RBD mirroring from the default namespace in the `test-pool` pool on Cluster 1 to the non-default `test-pool/test-ns` namespace on Cluster 2.

Warning:

- All nodes in both clusters must run Red Hat Ceph Storage 8.1 or later. Mixed-version clusters are not supported.
- If a namespace is disabled on one cluster, you must disable the corresponding paired namespace on the other cluster before performing any operations on it.

1. On cluster 1, enable mirroring on the default namespace for the pool, with the remote namespace and mode set to **image**.
   Syntax

   ```
   rbd mirror pool enable POOL_NAME image --site-name SITE_NAME --remote-
   namespace REMOTE_NAMESPACE_NAME
   ```

   In this example, the pool name is `test-pool` and the remote namespace name is `test-ns`.

   ```
   [root@test-server1]# rbd mirror pool enable test-pool image --site-name site-a --
   remote-namespace test-ns
   ```

2. Check mirror information for the pool.
   Syntax

   ```
   rbd mirror pool info POOL_NAME
   ```

   ```
   [root@test-server1]# rbd mirror pool info test-pool
   Mode: image
   Site Name: site-a
   Mirror UUID: e2bb7e8a-3385-48f3-a01c-cb9b546d4a21
   Remote Namespace: test-ns
   Peer Sites: none
   ```

3. On cluster 2, enable the pool for mirroring using the **init-only** mode. This allows the pool to be configured for mirroring without enabling mirroring on the default namespace.
   Syntax

   ```
   rbd mirror pool enable --site-name SITE_NAME POOL_NAME init-only
   ```

   In this example, mirroring is configured for the pool but is not enabled on the default namespace with mode **init-only**.

   ```
   [root@test-server2]# rbd mirror pool enable --site-name site-b test-pool init-only
   ```

4. On the 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
      ```

      The following example bootstrap command creates the `client.rbd-mirror.site-a` and the `client.rbd-mirror-peer` Ceph users.

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

   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 POOL_NAME PATH_TO_BOOTSTRAP_TOKEN
      ```

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

      Example

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

AFTER COMPLETING THE TASK

1. Check that the mirroring is set up as expected. Run the **`mirror pool info`** command on both the clusters.
   a. Cluster 1:

```
[root@test-server1]# rbd mirror pool info test-pool
Mode: image
Site Name: site-a
Mirror UUID: e2bb7e8a-3385-48f3-a01c-cb9b546d4a21
Remote Namespace: test-ns

Peer Sites:

UUID: 7ca5c1a2-4831-4625-8191-fa0c9064dc48
Name: site-b
Mirror UUID: 1e5e1b21-441b-4111-a475-5eb7fd19ab1b
Direction: rx-tx
Client: client.rbd-mirror-peer
```

   b. Cluster 2:

```
[root@test-server2]# rbd mirror pool info test-pool
Mode: init-only
Site Name: site-b
Mirror UUID: 1e5e1b21-441b-4111-a475-5eb7fd19ab1b
Remote Namespace:

Peer Sites:

UUID: e35fa86d-a37f-4f39-9696-c0ec2a976dac
Name: site-a
Mirror UUID: e2bb7e8a-3385-48f3-a01c-cb9b546d4a21
Direction: rx-tx
Client: client.rbd-mirror-peer
```

2. Configure mirroring on `pool/remote_namespace` to mirror to the default namespace on cluster 1. If the `Remote Namespace` field is not empty, the default namespace is not configured correctly.

```
[root@test-server2]# rbd  mirror pool enable test-pool/test-ns --remote-namespace ""
image
[root@test-server2]# rbd mirror pool info test-pool/test-ns
Mode: image
Mirror UUID: 1e5e1b21-441b-4111-a475-5eb7fd19ab1b
Remote Namespace:
```

> **Note:** Make sure the value passed to the remote-namespace is a properly quoted empty space.

3. **Optional:** To check whether the mirroring is working correctly, create and mirror enable an image in the default namespace on cluster 1 and verify that the image is mirrored to the `test-ns` namespace in cluster 2.
   a. Cluster 1:

```
[root@test-server1]# rbd create -s 12M test-pool/img-1
[root@test-server1]# rbd mirror image enable test-pool/img-1 snapshot
Mirroring enabled
[root@test-server1]# rbd ls test-pool
img-1
```

   b. Cluster 2:

```
[root@test-server2]# rbd ls test-pool   <-- the image is not mirrored to the
default ns on cluster2
[root@test-server2]# rbd ls test-pool/test-ns  <-- the image is mirrored to test-
pool/test-ns on cluster2
img-1
```

c. Compare the global IDs of the images on both clusters.

```
[root@test-server1]# rbd info test-pool/img-1 | grep "global id"
    mirroring global id: 6923936e-1e5d-4844-a4af-6b20258edf67
[root@test-server2]# rbd info test-pool/test-ns/img-1 | grep "global id"
    mirroring global id: 6923936e-1e5d-4844-a4af-6b20258edf67
```

4. To disable mirroring on a namespace with `rbd`, specify the **`mirror pool disable`** command and the namespace.

```
rbd mirror pool disable POOL_NAME/NAMESPACE
```

When configured in `image` mode, any mirror enabled images in the namespace must be explicitly disabled before disabling mirroring on the namespace.

To disable mirroring on the default namespace, run the **`mirror pool enable`** command with the **`init-only`** mode.

Example

```
[root@test-server1]# rbd --cluster site-a mirror pool enable test-pool init-only
[root@test-server2]# rbd --cluster site-b mirror pool disable test-pool/test-ns
```

Disaster recovery

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`.

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 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.
- For more information, see Mirroring Ceph Block Devices.
- For more information about data transmission over the wire in an encrypted state, see Encryption in transit.

Recover with one-way mirroring

Learn how to 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.

Recover 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.

Failover after an orderly shutdown

Learn how to perform a failover to a 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.

Reference

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

Failover after a non-orderly shutdown

Learn how to perform a failover to a 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 successful promotion of the site.

   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
   ```

Preparing 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, run 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
    ```

Reference

For more information, see Ceph user management

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
   ```

Removing 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
   ```

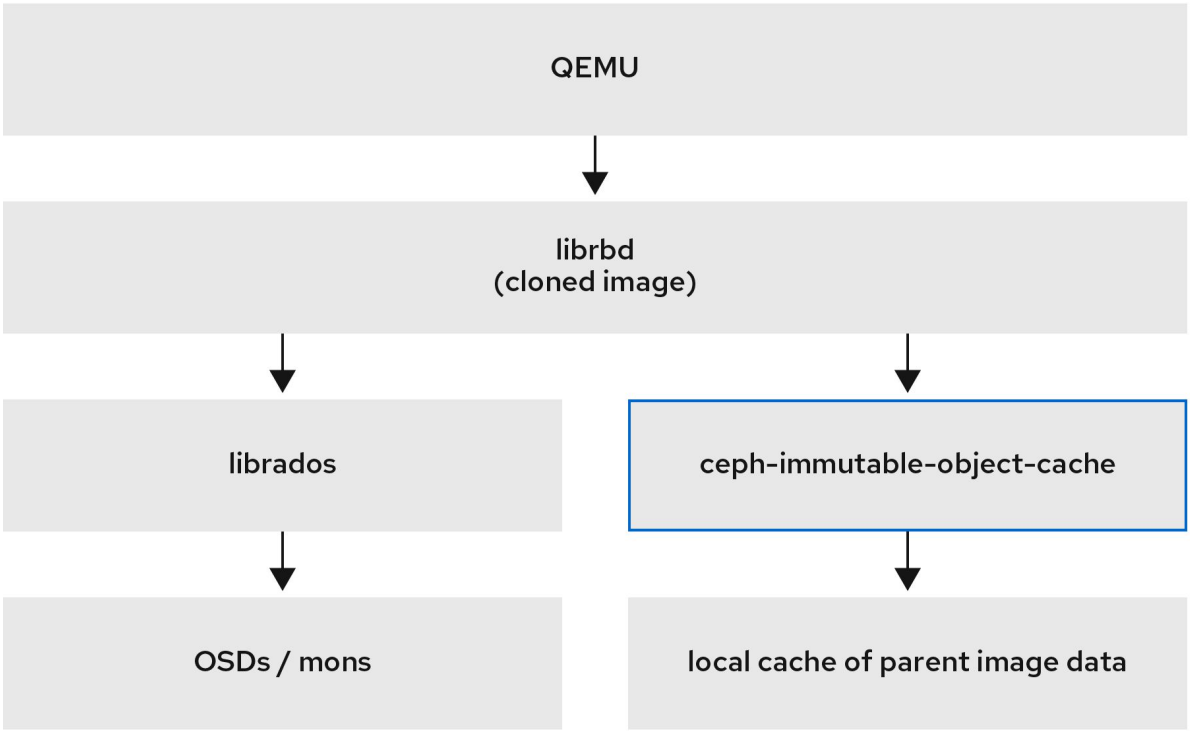
   Example

   ```
   [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
   ```

Managing `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: Ceph immutable cache daemon



Overview

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 boot-up, 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

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.

Configuration

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

PREREQUISITE

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

> **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.

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
```

2. Restart the cluster.
3. Install the `ceph-immutable-object-cache` package:
   Example

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

4. 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.
5. 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
   ```

6. 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.
   ```

7. 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
   ```

8. 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>
```

Generic settings

Understand the generic options for `ceph-immutable-object-cache` Ceph Block Device daemons.

"Ceph Block Device ceph-immutable-object-cache generic settings" on page 73 lists the different generic settings for `ceph-immutable-object-cache` Ceph Block Device daemons.

Ceph Block Device `ceph-immutable-object-cache` generic settings

Option	Description	Type	Default
immutable_object_cache_max_size	The maximum size for immutable cache.	Size	1G
immutable_object_cache_path	The immutable object cache data directory.	String	`/tmp/ceph_immutable_object_cache`
immutable_object_cache_sock	The path to the domain socket used for communication between librbd clients and the ceph-immutable-object-cache daemon.	String	`/var/run/ceph/immutable_object_cache_sock`
immutable_object_cache_watermark	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.	Float	0.9

QOS settings

The `ceph-immutable-object-cache` daemons support throttling. Use this information to know the supported settings.

immutable_object_cache_qos_schedule_tick_min

Description

Minimum schedule tick for immutable object cache.

Type

Milliseconds

Default

`50`

immutable_object_cache_qos_bps_burst_seconds

Description

The desired burst limit of read operations.

Type

Seconds

Default

1

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_limit

Description

User-defined immutable object cache IO bytes limit per second.

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_iops_burst

Description

User-defined burst limit of immutable object cache IO operations.

Type

Integer

Default

0

immutable_object_cache_qos_iops_limit

Description

User-defined immutable object cache IO operations limit per second.

Type

Integer

Default

0

`rbd` kernel module

Access Ceph Block Devices through the `rbd` kernel module. Through the module, you can map and unmap a Block Device and display the mappings. A list of images can also be retrieved 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 addresses 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.

Creating a Ceph Block Device and using it from a Linux kernel module client

Create a Ceph Block Device for a Linux kernel module client on 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.

Creating a Ceph Block device for a Linux kernel module client requires:

- A running Red Hat Ceph Storage cluster.
- A Red Hat Enterprise Linux client.

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.

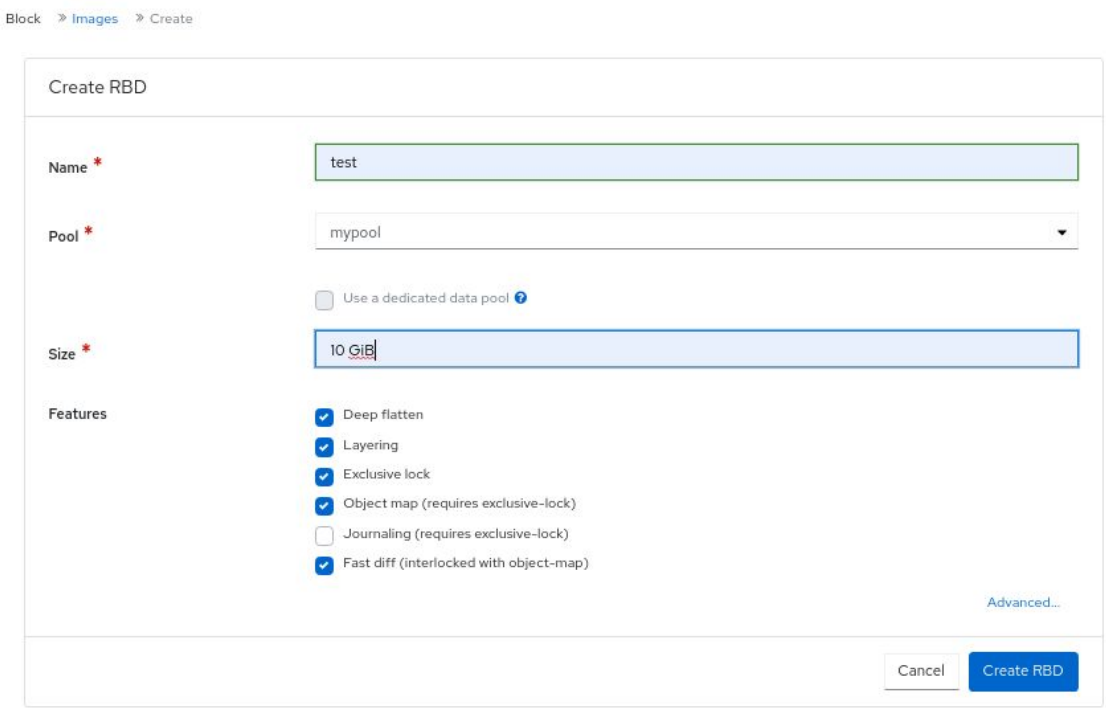
Prerequisites

- 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.

   Figure: Create RBD window



4. Click **Create RBD**.

Verification

- You get a notification that the image is created successfully.

Map and mount a Ceph Block Device on Linux using the command-line

Map and mount a Ceph Block Device to enable writing files to it by using the command-line interface.

PREREQUISITE

- A running Red Hat Ceph Storage cluster.
- A Ceph Block Device for a Linux kernel module client that uses the dashboard is created.
- A Red Hat Enterprise Linux client.

1. On the Red Hat Enterprise Linux client node, enable the Red Hat Ceph Storage tools repository.

```
subscription-manager repos --enable=rhceph-9-tools-for-rhel-9-x86_64-rpms
```

   For example,

```
[root@rbd-client ~]# subscription-manager repos --enable=rhceph-9-tools-for-rhel-9-x86_64-rpms
```

2. Install the `ceph-common` RPM package.

```
dnf install ceph-common
```

3. Copy the Ceph configuration file from a Monitor node to the Client node.

```
scp root@MONITOR_NODE:/etc/ceph/ceph.conf /etc/ceph/ceph.conf
```

   For 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
```

4. Copy the key file from a Monitor node to the Client node.

```
scp root@MONITOR_NODE:/etc/ceph/ceph.client.admin.keyring /etc/ceph/
ceph.client.admin.keyring
```

For 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
```

5. Map the image.

```
rbd map --pool POOL_NAME IMAGE_NAME --id admin
```

For example,

```
[root@rbd-client ~]# rbd map --pool block-device-pool image1 --id admin
/dev/rbd0
```

6. Create a partition table on the block device.

```
parted /dev/MAPPED_BLOCK_DEVICE mklabel msdos
```

For 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.

```
parted /dev/MAPPED_BLOCK_DEVICE mkpart primary xfs 0% 100%
```

For example,

```
[root@rbd-client ~]# parted /dev/rbd0 mkpart primary xfs 0% 100%
Information: You may need to update /etc/fstab.
```

8. Format the partition.

```
mkfs.xfs /dev/MAPPED_BLOCK_DEVICE_WITH_PARTITION_NUMBER
```

For 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.

```
mkdir PATH_TO_DIRECTORY
```

For example,

```
[root@rbd-client ~]# mkdir /mnt/ceph
```

10. Mount the file system.

```
mount /dev/MAPPED_BLOCK_DEVICE_WITH_PARTITION_NUMBER PATH_TO_DIRECTORY
```

For example,

```
[root@rbd-client ~]# mount /dev/rbd0p1 /mnt/ceph/
```

11. Verify that the file system is mounted and showing the correct size.

```
df -h PATH_TO_DIRECTORY
```

For example,

```
[root@rbd-client ~]# df -h /mnt/ceph/
Filesystem      Size  Used Avail Use% Mounted on
/dev/rbd0p1      10G  105M  9.9G   2% /mnt/ceph
```

Mapping a block device

Use the **rbd** command 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
```

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

- Display the mapped block devices:

```
[root@rbd-client ~]# rbd device list
```

Unmapping a block device

You can unmap a block device image with the **rbd** command, by using theunmapoption 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
```

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)
```

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.

Default options

Ceph creates images with the 2 format and with no striping.

“Ceph Block Device default options” on page 81 lists the different default options for Ceph Block Device configuration.

> **Note:** The default settings can be overridden.

Ceph Block Device default options

Option	Description	Type	Default	Notes
`rbd_default_format`	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.	Integer	`2`	
`rbd_default_order`	The default order if no other order is specified.	Integer	`22`	
`rbd_default_stripe_count`	The default stripe count if no other stripe count is specified. Changing the default value requires striping v2 feature.	64-bit Unsigned Integer	`0`	
`rbd_default_stripe_unit`	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.	64-bit Unsigned Integer	`0`	

Option	Description	Type	Default	Notes
`rbd_default_features`	The default features enabled when creating a block device image. The settings only apply to format two images. The settings are as follows: **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**	Integer	`61`	Layering, exclusive-lock, object-map, fast-diff, and deep-flatten are enabled. **Important:** The current default setting is not compatible with the RADOS Block Device kernel driver or older RADOS Block Device clients.

Option	Description	Type	Default	Notes
	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**			

Option	Description	Type	Default	Notes
	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.			
`rbd_default_map_options`	Most of the options are useful mainly for debugging and benchmarking. For more information, see `man rbd` under “Map Options”.	String	`""`	

General options

Understand the general options for Ceph Block Devices.

“Ceph Block Device general options” on page 85 lists the different default options for Ceph Block Device configuration.

Ceph Block Device general options			
Option	**Description**	**Type**	**Default**
`rbd_clone_copy_on_read`	When set to `true`, copy-on-read cloning is enabled.	Boolean	`false`
`rbd_concurrent_management_ops`	The maximum number of concurrent management operations in flight (for example, deleting or resizing an image).	Integer	`10`
`rbd_enable_alloc_hint`	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.	Boolean	`true`
`rbd_non_blocking_aio`	If `true`, Ceph will process block device asynchronous I/O operations from a worker thread to prevent blocking.	Boolean	`true`
`rbd_op_thread_timeout`	The timeout (in seconds) for block device operation threads.	Integer	`60`
`rbd_op_threads`	The number of block device operation threads. **Warning:** Do not change the default value of `rbd_op_threads` because setting it to a number higher than `1` might cause data corruption.	Integer	`1`
`rbd_request_timed_out_seconds`	The number of seconds before a maintenance request times out.	Integer	`30`
`rbd_skip_partial_discard`	If `true`, the block device will skip zeroing a range when trying to discard a range inside an object.	Boolean	`true`
`rbd_tracing`	Set this option to `true` to enable the Linux Trace Toolkit Next Generation User Space Tracer (LTTng-UST) tracepoints. For more information, see Tracing RADOS Block Device (RBD) Workloads with the RBD Replay Feature within the Red Hat Customer Portal.	Boolean	`false`

Option	Description	Type	Default
`rbd_validate_names`	Set this option to `true` to validate image specifications.	Boolean	`true`
`rbd_validate_pool`	Set this option to `true` to validate empty pools for RBD compatibility.	Boolean	`true`

Caching options

The user space implementation of the Ceph Block Device, `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 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 a 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 can 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`.

"Red Hat Ceph Storage cache option settings" on page 86 lists the available cache settings.

Red Hat Ceph Storage cache option settings

Option	Description	Type	Required	Constraint	Default
`rbd_cache`	Enable caching for RADOS Block Device (RBD).	Boolean	No	N/A	`true`
`rbd_cache_size`	The RADOS Block Devices (RBD) cache size in bytes.	64-bit integer	No	N/A	`32 MiB`
`rbd_cache_max_dirty`	The `dirty` limit in bytes at which the cache triggers a write-back. If `0`, uses write-through caching.	64-bit integer	No	Must be less than `rbd cache size`.	`24 MiB`

Option	Description	Type	Required	Constraint	Default
`rbd_cache_target_dirty`	The `dirty target` before the cache begins writing data to the data storage. Does not block writes to the cache.	64-bit integer	No	Must be less than `rbd cache max dirty`.	`16 MiB`
`rbd_cache_max_dirty_age`	The number of seconds dirty data is in the cache before writeback starts.	Float	No	N/A	`1.0`
`rbd_cache_max_dirty_object`	The dirty limit for objects. Set to `0` to auto calculate from `rbd_cache_size`.	Integer	N/A	N/A	`0`
`rbd_cache_block_writes_upfront`	If `true`, it will block writes to the cache before the `aio_write` call completes. If `false`, it blocks before the `aio_completion` is called.	Boolean	N/A	N/A	`false`
`rbd_cache_writethrough_until_flush`	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.	Boolean	No	N/A	`true`

Parent and child read options

Understand the parent and child read options for Ceph Block Devices.

"Red Hat Ceph Storage cache option settings" on page 86 lists the parent and child read options for Ceph Block Device configuration.

Parent and child read options			
Option	**Description**	**Type**	**Default**
`rbd_balance_snap_reads`	Ceph typically reads objects from the primary OSD. Since reads are immutable, you can enable this feature to balance snap reads between the primary OSD and the replicas. This option randomizes the replica for reading a snapshot.	Boolean	`false`
`rbd_localize_snap_reads`	The block device looks to the CRUSH map to find the closest or local OSD for reading the snapshot.	Boolean	`false`
`rbd_balance_parent_reads`	Ceph typically reads objects from the primary OSD. Since reads are immutable, you can enable this feature to balance parent reads between the primary OSD and the replicas. This option randomizes the replica for reading a parent.	Boolean	`false`
`rbd_localize_parent_reads`	The block device looks to the CRUSH map to find the closest or local OSD for reading the parent.	Boolean	`true`

Read ahead options

Ceph Block Devices support read-ahead and prefetching to optimize small, sequential reads. This is generally handled by the guest OS in the case of a virtual machine (VM), but boot loaders potentially do not issue efficient reads. Read-ahead is automatically disabled if caching is disabled.

"Read-ahead configuration options" on page 88 lists the read ahead configuration options for Ceph Block Devices.

Read-ahead configuration options				
Option	**Description**	**Type**	**Required**	**Default**
`rbd_readahead_trigger_requests`	Number of sequential read requests necessary to trigger read-ahead.	Integer	No	`10`
`rbd_readahead_max_bytes`	Maximum size of a read-ahead request. If zero, read-ahead is disabled.	64-bit integer	No	`512 KiB`

Option	Description	Type	Required	Default
`rbd_readahead_disable_after_bytes`	After this many bytes are 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 after it is started. If zero, read-ahead stays enabled.	64-bit integer	No	`50 MiB`

Block-list options

Understand the block-list options for Ceph Block Devices.

"Block-list configuration options" on page 89 lists the block-list configuration options for Ceph Block Devices.

Block-list configuration options

Option	Description	Type	Default
`rbd_blocklist_on_break_lock`	Whether to block-list clients whose lock was broken.	Boolean	`true`
`rbd_blocklistexpire_seconds`	The number of seconds to block-list - set to `0` for OSD default.	Integer	`0`

Journal options

Understand the journal options for Ceph Block Devices.

"Journal configuration options" on page 89 lists the journal configuration options for Ceph Block Devices.

Journal configuration options

Option	Description	Type	Default
`rbd_journal_order`	The number of bits to shift to compute the journal object maximum size. The value is between `12` and `64`.	32-bit unsigned integer	`24`
`rbd_journal_splay_width`	The number of active journal objects.	32-bit unsigned integer	`4`
`rbd_journal_commit_age`	The commit time interval in seconds.	Double Precision Floating Point Number	`5`
`rbd_journal_object_flush_interval`	The maximum number of pending commits per a journal object.	Integer	`0`
`rbd_journal_object_flush_bytes`	The maximum number of pending bytes per a journal object.	Integer	`0`
`rbd_journal_object_flush_age`	The maximum time interval in seconds for pending commits.	Double Precision Floating Point Number	`0`
`rbd_journal_pool`	Specifies a pool for journal objects.	String	`""`

Configuration override options

Use these configuration override options for Ceph Block Devices on global and pool levels.

Global level

"Global level override key options" on page 90 lists the override key options on a global level.

Global level override key options

Key	Description	Type	Default
`rbd_qos_bps_burst`	The desired burst limit of I/O bytes.	Integer	0
`rbd_qos_bps_limit`	The desired limit of I/O bytes per second.	Integer	0
`rbd_qos_iops_burst`	The desired burst limit of I/O operations.	Integer	0
`rbd_qos_iops_limit`	The desired limit of I/O operations per second.	Integer	0
`rbd_qos_read_bps_burst`	The desired burst limit of read bytes.	Integer	0
`rbd_qos_read_bps_limit`	The desired limit of read bytes per second.	Integer	0
`rbd_qos_read_iops_burst`	The desired burst limit of read operations.	Integer	0
`rbd_qos_read_iops_limit`	The desired limit of read operations per second.	Integer	0
`rbd_qos_write_bps_burst`	The desired burst limit of write bytes.	Integer	0
`rbd_qos_write_bps_limit`	The desired limit of write bytes per second.	Integer	0
`rbd_qos_write_iops_burst`	The desired burst limit of write operations.	Integer	0
`rbd_qos_write_iops_limit`	The desired burst limit of write operations per second.	Integer	0

"Global override command usage" on page 90 describes how to use the global override options listed in "Global level override key options" on page 90.

- Replace *`CONFIG_ENTITY`* with the global client or client ID.
- Replace *`KEY`* with the configuration key.
- Replace *`VALUE`* with the configuration value.

Global override command usage

Command syntax	Description
`rbd config global set CONFIG_ENTITY KEY VALUE`	Set a global level configuration override.
`rbd config global get CONFIG_ENTITY KEY`	Get a global level configuration override.
`rbd config global list CONFIG_ENTITY`	List the global level configuration overrides.

Command syntax	Description
`rbd config global remove CONFIG_ENTITY KEY`	Remove a global level configuration override.

Pool level

"Pool level override command usage" on page 91 describes how to override pool options.

- Replace *CONFIG_ENTITY* with the global client or client ID.
- Replace *KEY* with the configuration key.
- Replace *VALUE* with the configuration value.
- Replace *POOL_NAME* with the name of the pool.

Pool level override command usage

Command syntax	Description
`rbd config pool set POOL_NAME KEY VALUE`	Set a pool level configuration override.
`rbd config pool get POOL_NAME KEY`	Get a pool level configuration override.
`rbd config pool list POOL_NAME`	List the pool level configuration overrides.
`rbd config pool remove POOL_NAME KEY`	Remove a pool level configuration override.

Input and output options

Understand the input and output (I/O) options for Ceph Block Devices.

"I/O configuration options" on page 92 lists the block-list configuration options for Ceph Block Devices.

I/O configuration options					
Option	**Description**	**Type**	**Required**	**Values**	**Default**
`rbd_compression_hint`	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 does not attempt to compress data.	Enum	No	– `none` – `compressible` – `incompressible`	`none`

Option	Description	Type	Required	Values	Default
`rbd_read_from_replica_policy`	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 are sent to a randomly selected OSD within the replica set. If set to `localize`, read operations are sent to the closest OSD as determined by the CRUSH map and the `crush_location` configuration option, where the `crush_location` is denoted by 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.	Enum	No	– `default` – `balance` – `localize`	`default`