



Red Hat Ceph Storage

1.2.3

Ceph Block Device

Red Hat Ceph Storage Block Device

Red Hat Customer Content
Services

Red Hat Ceph Storage 1.2.3 Ceph Block Device

Red Hat Ceph Storage Block Device

Legal Notice

Copyright © 2015 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, MetaMatrix, Fedora, the Infinity Logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This document describes how to manage create, configure and use Red Hat Ceph Storage block devices.

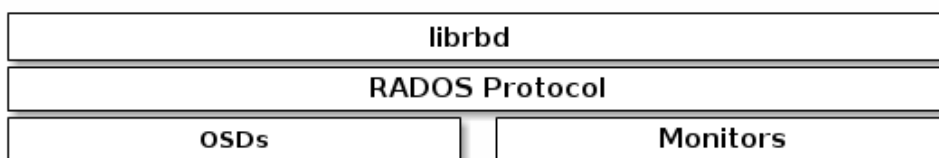
Table of Contents

PREFACE	3
PART I. BLOCK DEVICE COMMANDS	4
CHAPTER 1. PREREQUISITES	5
1.1. EXECUTE PRE-INSTALLATION REQUIREMENTS	5
1.2. INSTALL CEPH COMMON PACKAGE	5
1.3. ENABLE ACCESS TO THE CEPH CLUSTER	5
CHAPTER 2. CREATING A BLOCK DEVICE IMAGE	6
CHAPTER 3. LISTING BLOCK DEVICE IMAGES	7
CHAPTER 4. RETRIEVING IMAGE INFORMATION	8
CHAPTER 5. RESIZING A BLOCK DEVICE IMAGE	9
CHAPTER 6. REMOVING A BLOCK DEVICE IMAGE	10
PART II. SNAPSHOTS	11
CHAPTER 7. CEPHX NOTES	12
CHAPTER 8. SNAPSHOT BASICS	13
8.1. CREATE SNAPSHOT	13
8.2. LIST SNAPSHOTS	13
8.3. ROLLBACK SNAPSHOT	13
8.4. DELETE A SNAPSHOT	14
8.5. PURGE SNAPSHOTS	14
CHAPTER 9. LAYERING	15
9.1. GETTING STARTED WITH LAYERING	15
9.2. PROTECTING A SNAPSHOT	16
9.3. CLONING A SNAPSHOT	17
9.4. UNPROTECTING A SNAPSHOT	17
9.5. LISTING CHILDREN OF A SNAPSHOT	17
9.6. FLATTENING A CLONED IMAGE	18
PART III. LIBRBD SETTINGS	19
CHAPTER 10. CACHE SETTINGS	20
CHAPTER 11. READ-AHEAD SETTINGS	23
PART IV. LIBRBD (PYTHON)	25
CHAPTER 12. EXAMPLE: CREATING AND WRITING TO AN IMAGE	26
PART V. KERNEL MODULE OPERATIONS	28
CHAPTER 13. GET A LIST OF IMAGES	29
CHAPTER 14. MAP A BLOCK DEVICE	30
CHAPTER 15. SHOW MAPPED BLOCK DEVICES	31
CHAPTER 16. UNMAPPING A BLOCK DEVICE	32

PREFACE

A block is a sequence of bytes (for example, a 512-byte block of data). Block-based storage interfaces are the most common way to store data with rotating media such as hard disks, CDs, floppy disks, and even traditional 9-track tape. The ubiquity of block device interfaces makes a virtual block device an ideal candidate to interact with a mass data storage system like Ceph.

Ceph block devices are thin-provisioned, resizable and store data striped over multiple OSDs in a Ceph cluster. Ceph block devices leverage RADOS (Reliable Autonomic Distributed Object Store) capabilities such as snapshotting, replication and consistency. Ceph's RADOS (Reliable Autonomic Distributed Object Store) Block Devices (RBD) interact with OSDs using the **librbd** library.



Note

Ceph's block devices deliver high performance with infinite scalability to KVMs (kernel virtual machines) such as Qemu, and cloud-based computing systems like OpenStack and CloudStack that rely on libvirt and Qemu to integrate with Ceph block devices. You can use the same cluster to operate the Ceph Object Gateway and Ceph block devices simultaneously.



Important

To use Ceph Block Devices, you must have access to a running Ceph cluster.

PART I. BLOCK DEVICE COMMANDS

The **rbd** command enables you to create, list, introspect and remove block device images. You can also use it to clone images, create snapshots, rollback an image to a snapshot, view a snapshot, etc.



Important

To use Ceph Block Device commands, you must have access to a running Ceph cluster.

CHAPTER 1. PREREQUISITES

To use Ceph Block Device commands on your client node, you will have to meet some prerequisites.

1.1. EXECUTE PRE-INSTALLATION REQUIREMENTS

For streamlined installation and deployment, you should execute the pre-installation procedures mentioned in [Pre-Installation Requirements](#) on your Ceph client node. Specifically, you should disable **requiretty**, set SELinux to **Permissive** and set up a Ceph Deploy user with password-less **sudo**.

1.2. INSTALL CEPH COMMON PACKAGE

The **ceph-common** package provides the **rbd** command. To install **ceph-common** package you will have to enable **RH-Common** repository. Execute the following steps on client node:

1. Enable RH-Common repository.

For RHEL 6, execute:

```
sudo subscription-manager repos --enable=rhel-6-server-rh-common-rpms
sudo yum update
```

For RHEL 7, execute:

```
sudo subscription-manager repos --enable=rhel-7-server-rh-common-rpms
sudo yum update
```

2. Install Ceph Common package.

Execute:

```
sudo yum install ceph-common
```

1.3. ENABLE ACCESS TO THE CEPH CLUSTER

The client node where you will add a block device, also needs to have access to the Ceph Storage Cluster. To be able to access the cluster, the client node will need the cluster configuration file.

Execute from the **admin node** of the cluster:

```
ceph-deploy --overwrite-conf config push <client-node>
```

CHAPTER 2. CREATING A BLOCK DEVICE IMAGE

Before you can add a block device to a node, you must create an image for it in the Ceph Storage Cluster first. To create a block device image, execute the following:

```
rbd create {image-name} --size {megabytes} --pool {pool-name}
```

For example, to create a 1GB image named **foo** that stores information in a pool named **swimmingpool**, execute the following:

```
rbd create foo --size 1024
rbd create bar --size 1024 --pool swimmingpool
```



Note

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

CHAPTER 3. LISTING BLOCK DEVICE IMAGES

To list block devices in the **rbd** pool, execute the following (i.e., **rbd** is the default pool name):

```
| rbd ls
```

To list block devices in a particular pool, execute the following, but replace **{poolname}** with the name of the pool:

```
| rbd ls {poolname}
```

For example:

```
| rbd ls swimmingpool
```

CHAPTER 4. RETRIEVING IMAGE INFORMATION

To retrieve information from a particular image, execute the following, but replace **{image-name}** with the name for the image:

```
rbd --image {image-name} info
```

For example:

```
rbd --image foo info
```

To retrieve information from an image within a pool, execute the following, but replace **{image-name}** with the name of the image and replace **{pool-name}** with the name of the pool:

```
rbd --image {image-name} -p {pool-name} info
```

For example:

```
rbd --image bar -p swimmingpool info
```

CHAPTER 5. RESIZING A BLOCK DEVICE IMAGE

Ceph Block Device images are thin provisioned. They don't 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. If you want to increase (or decrease) the maximum size of a Ceph Block Device image, execute the following:

```
rbd resize --image foo --size 2048
```

CHAPTER 6. REMOVING A BLOCK DEVICE IMAGE

To remove a block device, execute the following, but replace **{image-name}** with the name of the image you want to remove:

```
┃ rbd rm {image-name}
```

For example:

```
┃ rbd rm foo
```

To remove a block device from a pool, execute the following, but replace **{image-name}** with the name of the image to remove and replace **{pool-name}** with the name of the pool:

```
┃ rbd rm {image-name} -p {pool-name}
```

For example:

```
┃ rbd rm bar -p swimmingpool
```

PART II. SNAPSHOTS

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



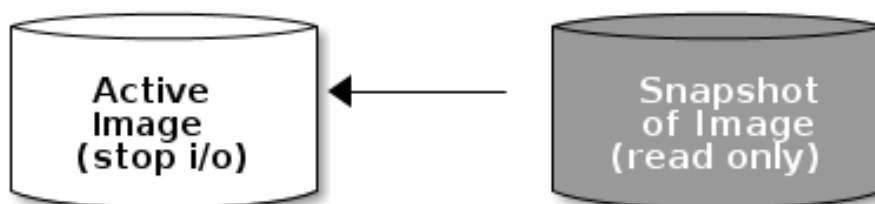
Important

To use RBD snapshots, you must have a running Ceph cluster.



Note

STOP I/O BEFORE taking a snapshot of an image. If the image contains a filesystem, the filesystem must be in a consistent state **BEFORE** taking a snapshot.



CHAPTER 7. CEPHX NOTES

When **cephx** is enabled (it is by default), you must specify a user name or ID and a path to the keyring containing the corresponding key for the user. You may also add the **CEPH_ARGS** environment variable to avoid re-entry of the following parameters:

```
rbd --id {user-ID} --keyring=/path/to/secret [commands]
rbd --name {username} --keyring=/path/to/secret [commands]
```

For example:

```
rbd --id admin --keyring=/etc/ceph/ceph.keyring [commands]
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 don't need to enter them each time.

CHAPTER 8. SNAPSHOT BASICS

The following procedures demonstrate how to create, list, and remove snapshots using the **rbd** command on the command line.

8.1. CREATE SNAPSHOT

To create a snapshot with **rbd**, specify the **snap create** option, the pool name and the image name:

```
rbd --pool {pool-name} snap create --snap {snap-name} {image-name}
rbd snap create {pool-name}/{image-name}@{snap-name}
```

For example:

```
rbd --pool rbd snap create --snap snapname foo
rbd snap create rbd/foo@snapname
```

8.2. LIST SNAPSHOTS

To list snapshots of an image, specify the pool name and the image name:

```
rbd --pool {pool-name} snap ls {image-name}
rbd snap ls {pool-name}/{image-name}
```

For example:

```
rbd --pool rbd snap ls foo
rbd snap ls rbd/foo
```

8.3. ROLLBACK SNAPSHOT

To rollback to a snapshot with **rbd**, specify the **snap rollback** option, the pool name, the image name and the snap name:

```
rbd --pool {pool-name} snap rollback --snap {snap-name} {image-name}
rbd snap rollback {pool-name}/{image-name}@{snap-name}
```

For example:

```
rbd --pool rbd snap rollback --snap snapname foo
rbd snap rollback rbd/foo@snapname
```

**Note**

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

8.4. DELETE A SNAPSHOT

To delete a snapshot with **rbd**, specify the **snap rm** option, the pool name, the image name and the username:

```
rbd --pool {pool-name} snap rm --snap {snap-name} {image-name}
rbd snap rm {pool-name}/{image-name}@{snap-name}
```

For example:

```
rbd --pool rbd snap rm --snap snapname foo
rbd snap rm rbd/foo@snapname
```

**Note**

Ceph OSDs delete data asynchronously, so deleting a snapshot doesn't free up the disk space immediately.

8.5. PURGE SNAPSHOTS

To delete all snapshots for an image with **rbd**, specify the **snap purge** option and the image name:

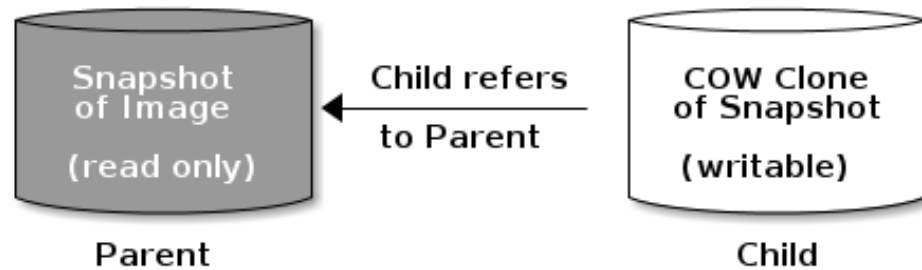
```
rbd --pool {pool-name} snap purge {image-name}
rbd snap purge {pool-name}/{image-name}
```

For example:

```
rbd --pool rbd snap purge foo
rbd snap purge rbd/foo
```

CHAPTER 9. LAYERING

Ceph supports the ability to create many copy-on-write (COW) 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 copy-on-write clones as you like. A snapshot is read-only, so cloning a snapshot simplifies semantics—making it possible to create clones rapidly.



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 (child) stores a reference to its parent image, which enables the cloned image to open the parent snapshot and read it.

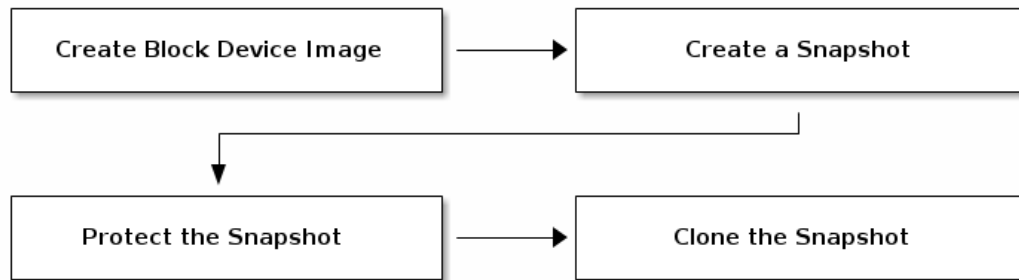
A COW clone of a snapshot behaves exactly like any other Ceph block device image. You can read to, write from, clone, and resize cloned images. There are no special restrictions with cloned images. However, the copy-on-write clone of a snapshot refers to the snapshot, so you **MUST** protect the snapshot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

Note

Ceph only supports cloning for **format 2** images (i.e., created with `rbd create --image-format 2`), and is not yet supported by the kernel `rbd` module. So you **MUST** use QEMU/KVM or `librbd` directly to access clones in the current release.

9.1. GETTING STARTED WITH LAYERING

Ceph block device layering is a simple process. You must have an image. You must create a snapshot of the image. You must protect the snapshot. Once you have performed these steps, you can begin cloning the snapshot.



The cloned image has a reference to the parent snapshot, and includes the pool ID, image ID and snapshot ID. The inclusion of the pool ID means that you may clone snapshots from one pool to images in another pool.

1. **Image Template:** A common use case for block device layering is to create a master image and a snapshot that serves as a template for clones. For example, a user may create an image for a RHEL7 distribution and create a snapshot for it. Periodically, the user may update the image and create a new snapshot (e.g. `sudo yum update`, `sudo yum upgrade`, followed by `rbd snap create`). As the image matures, the user can clone any one of the snapshots.
2. **Extended Template:** A more advanced use case includes extending a template image that provides more information than a base image. For example, a user may clone an image (e.g., a VM template) and install other software (e.g., a database, a content management system, an analytics system, etc.) and then snapshot the extended image, which itself may be updated just like the base image.
3. **Template Pool:** One way to use block device layering is to create a pool that contains master images that act as templates, and snapshots of those templates. You may then extend read-only privileges to users so that they may clone the snapshots without the ability to write or execute within the pool.
4. **Image Migration/Recovery:** One way to use block device layering is to migrate or recover data from one pool into another pool.

9.2. PROTECTING A SNAPSHOT

Clones access the parent snapshots. All clones would break if a user inadvertently deleted the parent snapshot. To prevent data loss, you **MUST** protect the snapshot before you can clone it. To do so, run the following commands:

```
rbd --pool {pool-name} snap protect --image {image-name} --snap
{snapshot-name}
rbd snap protect {pool-name}/{image-name}@{snapshot-name}
```

For example:

```
rbd --pool rbd snap protect --image my-image --snap my-snapshot
rbd snap protect rbd/my-image@my-snapshot
```

**Note**

You cannot delete a protected snapshot.

9.3. CLONING A SNAPSHOT

To clone a snapshot, you need to specify the parent pool, image and snapshot; and the child pool and image name. You must protect the snapshot before you can clone it. To do so, run the following commands:

```

rdp --pool {pool-name} --image {parent-image} --snap {snap-name} --
dest-pool {pool-name} --dest {child-image}
rdp clone {pool-name}/{parent-image}@{snap-name} {pool-name}/{child-
image-name}

```

For example:

```

rdp clone rbd/my-image@my-snapshot rbd/new-image

```

**Note**

You may clone a snapshot from one pool to an image in another pool. For example, you may maintain read-only images and snapshots as templates in one pool, and writeable clones in another pool.

9.4. UNPROTECTING A SNAPSHOT

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

```

rdp --pool {pool-name} snap unprotect --image {image-name} --snap
{snapshot-name}
rdp snap unprotect {pool-name}/{image-name}@{snapshot-name}

```

For example:

```

rdp --pool rbd snap unprotect --image my-image --snap my-snapshot
rdp snap unprotect rbd/my-image@my-snapshot

```

9.5. LISTING CHILDREN OF A SNAPSHOT

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

```

rdp --pool {pool-name} children --image {image-name} --snap {snap-name}
rdp children {pool-name}/{image-name}@{snapshot-name}

```

For example:

■

```
rdp --pool rbd children --image my-image --snap my-snapshot  
rdp children rbd/my-image@my-snapshot
```

9.6. FLATTENING A CLONED IMAGE

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. To delete a snapshot, you must flatten the child images first:

```
rdp --pool {pool-name} flatten --image {image-name}  
rdp flatten {pool-name}/{image-name}
```

For example:

```
rdp --pool rbd flatten --image my-image  
rdp flatten rbd/my-image
```



Note

Since a flattened image contains all the information from the snapshot, a flattened image will take up more storage space than a layered clone.

PART III. LIBRBD SETTINGS

CHAPTER 10. CACHE SETTINGS

The user space implementation of the Ceph block device (i.e, **librbd**) cannot take advantage of the Linux page cache, so it includes its own in-memory caching, called **RBD caching**. RBD caching behaves just like well-behaved hard disk caching. When the OS sends a barrier or a flush request, all dirty data is written to the OSDs. This means that using write-back caching is just as safe as using a well-behaved physical hard disk with a VM that properly sends flushes (i.e. Linux kernel \geq 2.6.32). The cache uses a Least Recently Used (LRU) algorithm, and in write-back mode it can coalesce contiguous requests for better throughput.

Ceph supports write-back caching for RBD. To enable it, add **rbid cache = true** to the **[client]** section of your **ceph.conf** 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 **rbid cache max dirty** unflushed bytes. In this case, the write triggers writeback and blocks until enough bytes are flushed.

Ceph supports write-through caching for RBD. 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 **rbid cache max dirty** to 0. This means writes return only when the data is on disk on all replicas, but reads may come from the cache. The cache is in memory on the client, and each RBD image has its own. Since the cache is local to the client, there's no coherency if there are others accessing the image. Running GFS or OCFS on top of RBD will not work with caching enabled.

The **ceph.conf** file settings for RBD should be set in the **[client]** section of your configuration file. The settings include:

rbid cache

Description

Enable caching for RADOS Block Device (RBD).

Type

Boolean

Required

No

Default

true

rbid cache size

Description

The RBD cache size in bytes.

Type

64-bit Integer

Required

No

Default

32 MiB

rbd cache max dirty

Description

The **dirty** limit in bytes at which the cache triggers write-back. If **0**, uses write-through caching.

Type

64-bit Integer

Required

No

Constraint

Must be less than **rbd cache size**.

Default

24 MiB

rbd cache target dirty

Description

The **dirty target** before the cache begins writing data to the data storage. Does not block writes to the cache.

Type

64-bit Integer

Required

No

Constraint

Must be less than **rbd cache max dirty**.

Default

16 MiB

rbd cache max dirty age

Description

The number of seconds dirty data is in the cache before writeback starts.

Type

Float

Required

No

Default**1.0****rdp cache writethrough until flush****Description**

Start out in write-through mode, and switch to write-back after the first flush request is received. Enabling this is a conservative but safe setting in case VMs running on rbd are too old to send flushes, like the virtio driver in Linux before 2.6.32.

Type

Boolean

Required

No

Default**true**

CHAPTER 11. READ-AHEAD SETTINGS

RBD supports read-ahead/prefetching to optimize small, sequential reads. This should normally be handled by the guest OS in the case of a VM, but boot loaders may not issue efficient reads. Read-ahead is automatically disabled if caching is disabled.

rbid readahead trigger requests

Description

Number of sequential read requests necessary to trigger read-ahead.

Type

Integer

Required

No

Default

10

rbid readahead max bytes

Description

Maximum size of a read-ahead request. If zero, read-ahead is disabled.

Type

64-bit Integer

Required

No

Default

512 KiB

rbid readahead disable after bytes

Description

After this many bytes have been read from an RBD image, read-ahead is disabled for that image until it is closed. This allows the guest OS to take over read-ahead once it is booted. If zero, read-ahead stays enabled.

Type

64-bit Integer

Required

No

Default

50 MiB

PART IV. LIBRBD (PYTHON)

The rbd python module provides file-like access to RBD images.

CHAPTER 12. EXAMPLE: CREATING AND WRITING TO AN IMAGE

To use `rd`, you must first connect to RADOS and open an IO context:

```
cluster = rados.Rados(conffile='my_ceph.conf')
cluster.connect()
ioctx = cluster.open_ioctx('mypool')
```

Then you instantiate an `:class:rd.RBD` object, which you use to create the image:

```
rd_inst = rd.RBD()
size = 4 * 1024**3 # 4 GiB
rd_inst.create(ioctx, 'myimage', size)
```

To perform I/O on the image, you instantiate an `:class:rd.Image` object:

```
image = rd.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` - `Librd` does not know how to deal with characters wider than a `:type:char`.

In the end, you'll close the image, the IO context and the connection to RADOS:

```
image.close()
ioctx.close()
cluster.shutdown()
```

To be safe, each of these calls would need to be in a separate `:finally` block:

```
cluster = rados.Rados(conffile='my_ceph_conf')
try:
    ioctx = cluster.open_ioctx('my_pool')
    try:
        rd_inst = rd.RBD()
        size = 4 * 1024**3 # 4 GiB
        rd_inst.create(ioctx, 'myimage', size)
        image = rd.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/shutdown 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)
```

PART V. KERNEL MODULE OPERATIONS



Note

To use kernel module operations, you must have a running Ceph cluster.



Important

This feature has been added in for use with **RHEL 7.1** and **Ubuntu 14.04**. This feature **IS NOT SUPPORTED** for use with **RHEL 6** or **RHEL 7.0**.

CHAPTER 13. GET A LIST OF IMAGES

To mount a block device image, first return a list of the images.

To do so, execute the following:

```
| rbd list
```

CHAPTER 14. MAP A BLOCK DEVICE

Use **rbd** to map an image name to a kernel module. You must specify the image name, the pool name and the user name. **rbd** will load RBD kernel module on your behalf if it's not already loaded.

To do so, execute the following:

```
sudo rbd map {image-name} --pool {pool-name} --id {user-name}
```

For example:

```
sudo rbd map --pool rbd myimage --id admin
```

If you use **cephx** authentication, you must also specify a secret. It may come from a keyring or a file containing the secret.

To do so, execute the following:

```
sudo rbd map --pool rbd myimage --id admin --keyring /path/to/keyring
sudo rbd map --pool rbd myimage --id admin --keyfile /path/to/file
```

CHAPTER 15. SHOW MAPPED BLOCK DEVICES

To show block device images mapped to kernel modules with the **rbd** command, specify the **showmapped** option.

To do so, execute the following:

```
| rbd showmapped
```

CHAPTER 16. UNMAPPING A BLOCK DEVICE

To unmap a block device image with the **rbd** command, specify the **unmap** option and the device name (i.e. by convention the same as the block device image name).

To do so, execute the following:

```
sudo rbd unmap /dev/rbd/{poolname}/{imagename}
```

For example:

```
sudo rbd unmap /dev/rbd/rbd/foo
```