Red Hat Ceph Storage 4

Administration Guide

Administration of Red Hat Ceph Storage

Last Updated: 2022-11-04

Red Hat Ceph Storage 4 Administration Guide

Administration of Red Hat Ceph Storage

Legal Notice

Abstract

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

Table of Contents

CHAPTER 1. CEPH ADMINISTRATION

A Red Hat Ceph Storage cluster is the foundation for all Ceph deployments. After deploying a Red Hat Ceph Storage cluster, there are administrative operations for keeping a Red Hat Ceph Storage cluster healthy and performing optimally.

The Red Hat Ceph Storage Administration Guide helps storage administrators to perform such tasks as:

- How do I check the health of my Red Hat Ceph Storage cluster?
- How do I start and stop the Red Hat Ceph Storage cluster services?
- How do I add or remove an OSD from a running Red Hat Ceph Storage cluster?
- How do I manage user authentication and access controls to the objects stored in a Red Hat Ceph Storage cluster?
- I want to understand how to use overrides with a Red Hat Ceph Storage cluster.
- I want to monitor the performance of the Red Hat Ceph Storage cluster.

A basic Ceph storage cluster consist of two types of daemons:

- A Ceph Object Storage Device (OSD) stores data as objects within placement groups assigned to the OSD
- A Ceph Monitor maintains a master copy of the cluster map

A production system will have three or more Ceph Monitors for high availability and typically a minimum of 50 OSDs for acceptable load balancing, data re-balancing and data recovery.

Additional Resources

- Red Hat Ceph Storage Installation Guide

CHAPTER 2. UNDERSTANDING PROCESS MANAGEMENT FOR CEPH

As a storage administrator, you can manipulate the various Ceph daemons by type or instance, on bare-metal or in containers. Manipulating these daemons allows you to start, stop and restart all of the Ceph services as needed.

2.1. PREREQUISITES

- Installation of the Red Hat Ceph Storage software.

2.2. CEPH PROCESS MANAGEMENT

In Red Hat Ceph Storage, all process management is done through the Systemd service. Each time you want to **start**, **restart**, and **stop** the Ceph daemons, you must specify the daemon type or the daemon instance.

Additional Resources

- For more information about using Systemd, see the chapter *Managing services with systemd* in the Red Hat Enterprise Linux *System Administrator's Guide*.

2.3. STARTING, STOPPING, AND RESTARTING ALL CEPH DAEMONS

Start, stop, and restart all Ceph daemons as an **admin** from the node.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Having **root** access to the node.

Procedure

1. Starting all Ceph daemons:

```
[root@admin ~]# systemctl start ceph.target
```

2. Stopping all Ceph daemons:

```
[root@admin ~]# systemctl stop ceph.target
```

3. Restarting all Ceph daemons:

```
[root@admin ~]# systemctl restart ceph.target
```

2.4. STARTING, STOPPING, AND RESTARTING THE CEPH DAEMONS BY TYPE

To start, stop, or restart all Ceph daemons of a particular type, follow these procedures on the node running the Ceph daemons.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Having **root** access to the node.

Procedure

- On **Ceph Monitor** nodes:

  Starting:

  ```
  [root@mon ~]# systemctl start ceph-mon.target
  ```

  Stopping:

  ```
  [root@mon ~]# systemctl stop ceph-mon.target
  ```

  Restarting:

  ```
  [root@mon ~]# systemctl restart ceph-mon.target
  ```

- On **Ceph Manager** nodes:

  Starting:

  ```
  [root@mgr ~]# systemctl start ceph-mgr.target
  ```

  Stopping:

  ```
  [root@mgr ~]# systemctl stop ceph-mgr.target
  ```

  Restarting:

  ```
  [root@mgr ~]# systemctl restart ceph-mgr.target
  ```

- On **Ceph OSD** nodes:

  Starting:

  ```
  [root@osd ~]# systemctl start ceph-osd.target
  ```

  Stopping:

  ```
  [root@osd ~]# systemctl stop ceph-osd.target
  ```

  Restarting:

  ```
  [root@osd ~]# systemctl restart ceph-osd.target
  ```

- On **Ceph Object Gateway** nodes:

Starting:

```
[root@rgw ~]# systemctl start ceph-radosgw.target
```

Stopping:

```
[root@rgw ~]# systemctl stop ceph-radosgw.target
```

Restarting:

```
[root@rgw ~]# systemctl restart ceph-radosgw.target
```

2.5. STARTING, STOPPING, AND RESTARTING THE CEPH DAEMONS BY INSTANCE

To start, stop, or restart a Ceph daemon by instance, follow these procedures on the node running the Ceph daemons.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Having **root** access to the node.

Procedure

- On a **Ceph Monitor** node:

  Starting:

  ```
  [root@mon ~]# systemctl start ceph-mon@MONITOR_HOST_NAME
  ```

  Stopping:

  ```
  [root@mon ~]# systemctl stop ceph-mon@MONITOR_HOST_NAME
  ```

  Restarting:

  ```
  [root@mon ~]# systemctl restart ceph-mon@MONITOR_HOST_NAME
  ```

  Replace

  - ***MONITOR_HOST_NAME*** with the name of the Ceph Monitor node.

- On a **Ceph Manager** node:

  Starting:

  ```
  [root@mgr ~]# systemctl start ceph-mgr@MANAGER_HOST_NAME
  ```

  Stopping:

  ```
  [root@mgr ~]# systemctl stop ceph-mgr@MANAGER_HOST_NAME
  ```

  Restarting:

  ```
  [root@mgr ~]# systemctl restart ceph-mgr@MANAGER_HOST_NAME
  ```

  Replace

  - ***MANAGER_HOST_NAME*** with the name of the Ceph Manager node.

- On a **Ceph OSD** node:

  Starting:

  ```
  [root@osd ~]# systemctl start ceph-osd@OSD_NUMBER
  ```

  Stopping:

  ```
  [root@osd ~]# systemctl stop ceph-osd@OSD_NUMBER
  ```

  Restarting:

  ```
  [root@osd ~]# systemctl restart ceph-osd@OSD_NUMBER
  ```

  Replace

  - ***OSD_NUMBER*** with the **ID** number of the Ceph OSD.
    For example, when looking at the **ceph osd tree** command output, **osd.0** has an **ID** of **0**.

- On a **Ceph Object Gateway** node:

  Starting:

  ```
  [root@rgw ~]# systemctl start ceph-radosgw@rgw.OBJ_GATEWAY_HOST_NAME
  ```

  Stopping:

  ```
  [root@rgw ~]# systemctl stop ceph-radosgw@rgw.OBJ_GATEWAY_HOST_NAME
  ```

  Restarting:

  ```
  [root@rgw ~]# systemctl restart ceph-radosgw@rgw.OBJ_GATEWAY_HOST_NAME
  ```

  Replace

  - ***OBJ_GATEWAY_HOST_NAME*** with the name of the Ceph Object Gateway node.

2.6. STARTING, STOPPING, AND RESTARTING CEPH DAEMONS THAT RUN IN CONTAINERS

Use the **systemctl** command start, stop, or restart Ceph daemons that run in containers.

Prerequisites

- Installation of the Red Hat Ceph Storage software.
- Root-level access to the node.

Procedure

1. To start, stop, or restart a Ceph daemon running in a container, run a **systemctl** command as **root** composed in the following format:

   ```
   systemctl ACTION ceph-DAEMON@ID
   ```

   Replace

   - *ACTION* is the action to perform; **start**, **stop**, or **restart**.
   - *DAEMON* is the daemon; **osd**, **mon**, **mds**, or **rgw**.
   - *ID* is either:
     - The short host name where the **ceph-mon**, **ceph-mds**, or **ceph-rgw** daemons are running.
     - The ID of the **ceph-osd** daemon if it was deployed.

   For example, to restart a **ceph-osd** daemon with the ID **osd01**:

   ```
   [root@osd ~]# systemctl restart ceph-osd@osd01
   ```

   To start a **ceph-mon** demon that runs on the **ceph-monitor01** host:

   ```
   [root@mon ~]# systemctl start ceph-mon@ceph-monitor01
   ```

   To stop a **ceph-rgw** daemon that runs on the **ceph-rgw01** host:

   ```
   [root@rgw ~]# systemctl stop ceph-radosgw@ceph-rgw01
   ```

2. Verify that the action was completed successfully.

   ```
   systemctl status ceph-DAEMON@ID
   ```

   For example:

   ```
   [root@mon ~]# systemctl status ceph-mon@ceph-monitor01
   ```

Additional Resources

- See the Understanding process management for Ceph chapter in the *Red Hat Ceph Storage Administration Guide* for more information.

2.7. VIEWING THE LOGS OF CEPH DAEMONS THAT RUN IN CONTAINERS

Use the **journald** daemon from the container host to view the logs of a Ceph daemon from a container.

Prerequisites

- Installation of the Red Hat Ceph Storage software.
- Root-level access to the node.

Procedure

1. To view the entire Ceph log, run a **journalctl** command as **root** composed in the following format:

   ```
   journalctl -u ceph-DAEMON@ID
   ```

   Replace

   - *DAEMON* is the Ceph daemon; **osd**, **mon**, or **rgw**.
   - *ID* is either:
     - The short host name where the **ceph-mon**, **ceph-mds**, or **ceph-rgw** daemons are running.
     - The ID of the **ceph-osd** daemon if it was deployed.

     For example, to view the entire log for the **ceph-osd** daemon with the ID **osd01**:

     ```
     [root@osd ~]# journalctl -u ceph-osd@osd01
     ```

2. To show only the recent journal entries, use the **-f** option.

   ```
   journalctl -fu ceph-DAEMON@ID
   ```

   For example, to view only recent journal entries for the **ceph-mon** daemon that runs on the **ceph-monitor01** host:

   ```
   [root@mon ~]# journalctl -fu ceph-mon@ceph-monitor01
   ```

> **NOTE**
>
> You can also use the **sosreport** utility to view the **journald** logs. For more details about SOS reports, see the What is an sosreport and how to create one in Red Hat Enterprise Linux? solution on the Red Hat Customer Portal.

Additional Resources

- The **journalctl(1)** manual page.

2.8. ENABLING LOGGING TO A FILE FOR CONTAINERIZED CEPH DAEMONS

By default, containerized Ceph daemons do not log to files. You can use centralized configuration management to enable containerized Ceph daemons to log to files.

Prerequisites

- Installation of the Red Hat Ceph Storage software.
- Root-level access to the node where the containerized daemon runs.

Procedure

1. Navigate to the **var/log/ceph** directory:

   Example

   ```
   [root@host01 ~]# cd /var/log/ceph
   ```

2. Note any existing log files.

   Syntax

   ```
   ls -l /var/log/ceph/
   ```

   Example

   ```
   [root@host01 ceph]# ls -l /var/log/ceph/
   total 396
   -rw-r--r--. 1 ceph ceph 107230 Feb  5 14:42 ceph-osd.0.log
   -rw-r--r--. 1 ceph ceph 107230 Feb  5 14:42 ceph-osd.3.log
   -rw-r--r--. 1 root root 181641 Feb  5 14:42 ceph-volume.log
   ```

   In the example, logging to files for OSD.0 and OSD.3 are already enabled.

3. Fetch the container name of the daemon for which you want to enable logging:

   Red Hat Enterprise Linux 7

   ```
   [root@host01 ceph]# docker ps -a
   ```

   Red Hat Enterprise Linux 8

   ```
   [root@host01 ceph]# podman ps -a
   ```

4. Use centralized configuration management to enable logging to a file for a Ceph daemon.

   Red Hat Enterprise Linux 7

```
docker exec CONTAINER_NAME ceph config set DAEMON_NAME log_to_file true
```

Red Hat Enterprise Linux 8

```
podman exec CONTAINER_NAME ceph config set DAEMON_NAME log_to_file true
```

The *DAEMON_NAME* is derived from the *CONTAINER_NAME*. Remove **ceph-** and replace the hyphen between the daemon and daemon ID with a period.

Red Hat Enterprise Linux 7

```
[root@host01 ceph]# docker exec ceph-mon-host01 ceph config set mon.host01 log_to_file
true
```

Red Hat Enterprise Linux 8

```
[root@host01 ceph]# podman exec ceph-mon-host01 ceph config set mon.host01 log_to_file
true
```

5. Optional: To enable logging to a file for the cluster log, use the **mon_cluster_log_to_file** option:

   Red Hat Enterprise Linux 7

   ```
   docker exec CONTAINER_NAME ceph config set DAEMON_NAME mon_cluster_log_to_file
   true
   ```

   Red Hat Enterprise Linux 8

   ```
   podman exec CONTAINER_NAME ceph config set DAEMON_NAME
   mon_cluster_log_to_file true
   ```

   Red Hat Enterprise Linux 7

   ```
   [root@host01 ceph]# docker exec ceph-mon-host01 ceph config set mon.host01
   mon_cluster_log_to_file true
   ```

   Red Hat Enterprise Linux 8

   ```
   [root@host01 ceph]# podman exec ceph-mon-host01 ceph config set mon.host01
   mon_cluster_log_to_file true
   ```

6. Validate the updated configuration:

   Red Hat Enterprise Linux 7

   ```
   docker exec CONTAINER_NAME ceph config show-with-defaults DAEMON_NAME | grep
   log_to_file
   ```

   Red Hat Enterprise Linux 8

```
podman exec CONTAINER_NAME ceph config show-with-defaults DAEMON_NAME | grep
log_to_file
```

Example

```
[root@host01 ceph]# podman exec ceph-mon-host01 ceph config show-with-defaults
mon.host01 | grep log_to_file
log_to_file                                    true
mon      default[false]
mon_cluster_log_to_file                               true
mon      default[false]
```

7. Optional: Restart the Ceph daemon:

Syntax

```
systemctl restart ceph-DAEMON@DAEMON_ID
```

Example

```
[root@host01 ceph]# systemctl restart ceph-mon@host01
```

8. Validate that the new log files exist:

Syntax

```
ls -l /var/log/ceph/
```

Example

```
[root@host01 ceph]# ls -l /var/log/ceph/
total 408
-rw-------. 1 ceph ceph    202 Feb  5 16:06 ceph.audit.log
-rw-------. 1 ceph ceph   3182 Feb  5 16:06 ceph.log
-rw-r--r--. 1 ceph ceph   2049 Feb  5 16:06 ceph-mon.host01.log
-rw-r--r--. 1 ceph ceph 107230 Feb  5 14:42 ceph-osd.0.log
-rw-r--r--. 1 ceph ceph 107230 Feb  5 14:42 ceph-osd.3.log
-rw-r--r--. 1 root root 181641 Feb  5 14:42 ceph-volume.log
```

Two new files were created, **ceph-mon.host01.log** for a Monitor daemon and **ceph.log** for the cluster log.

Additional Resources

- For more information, see Configuring logging section in the *Red Hat Ceph Storage Troubleshooting Guide*.

2.9. GATHERING LOG FILES OF CEPH DAEMONS

To gather log files of Ceph daemons, run the **gather-ceph-logs.yml** Ansible playbook. Currently, Red Hat Ceph Storage supports gathering logs for non-containerized deployments only.

Prerequisites

- A running Red Hat Ceph Storage cluster deployed.
- Admin-level access to the Ansible node.

Procedure

1. Navigate to the **/usr/share/ceph-ansible** directory:

   ```
   [ansible@admin ~]# cd /usr/share/ceph-ansible
   ```

2. Run the playbook:

   ```
   [ansible@admin ~]# ansible-playbook infrastructure-playbooks/gather-ceph-logs.yml -i hosts
   ```

3. Wait for the logs to be collected on the Ansible administration node.

Additional Resources

- See the *Viewing log files of Ceph daemons that run in containers* section in the *Red Hat Ceph Storage Administration Guide* for more details.

2.10. POWERING DOWN AND REBOOTING RED HAT CEPH STORAGE CLUSTER

Follow the below procedure for powering down and rebooting the Ceph cluster.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Having **root** access.

Procedure

Powering down the Red Hat Ceph Storage cluster

1. Stop the clients from using the RBD images and RADOS Gateway on this cluster and any other clients.

2. The cluster must be in healthy state (**Health_OK** and all PGs **active+clean**) before proceeding. Run **ceph status** on a node with the client keyrings, for example, the Ceph Monitor or OpenStack controller nodes, to ensure the cluster is healthy.

3. If you use the Ceph File System (**CephFS**), the **CephFS** cluster must be brought down. Taking a **CephFS** cluster down is done by reducing the number of ranks to **1**, setting the **cluster_down** flag, and then failing the last rank.

   Example:

   ```
   [root@osd ~]# ceph fs set FS_NAME max_mds 1
   [root@osd ~]# ceph mds deactivate FS_NAME:1 # rank 2 of 2
   [root@osd ~]# ceph status # wait for rank 1 to finish stopping
   ```

```
[root@osd ~]# ceph fs set FS_NAME cluster_down true
[root@osd ~]# ceph mds fail FS_NAME:0
```

Setting the **cluster_down** flag prevents standbys from taking over the failed rank.

4. Set the **noout**, **norecover**, **norebalance**, **nobackfill**, **nodown** and **pause** flags. Run the following on a node with the client keyrings. For example, the Ceph Monitor or OpenStack controller node:

   ```
   [root@mon ~]# ceph osd set noout
   [root@mon ~]# ceph osd set norecover
   [root@mon ~]# ceph osd set norebalance
   [root@mon ~]# ceph osd set nobackfill
   [root@mon ~]# ceph osd set nodown
   [root@mon ~]# ceph osd set pause
   ```

5. Shut down the OSD nodes one by one:

   ```
   [root@osd ~]# systemctl stop ceph-osd.target
   ```

6. Shut down the monitor nodes one by one:

   ```
   [root@mon ~]# systemctl stop ceph-mon.target
   ```

Rebooting the Red Hat Ceph Storage cluster

1. Power on the administration node.

2. Power on the monitor nodes:

   ```
   [root@mon ~]# systemctl start ceph-mon.target
   ```

3. Power on the OSD nodes:

   ```
   [root@osd ~]# systemctl start ceph-osd.target
   ```

4. Wait for all the nodes to come up. Verify all the services are up and the connectivity is fine between the nodes.

5. Unset the **noout**, **norecover**, **norebalance**, **nobackfill**, **nodown** and **pause** flags. Run the following on a node with the client keyrings. For example, the Ceph Monitor or OpenStack controller node:

   ```
   [root@mon ~]# ceph osd unset noout
   [root@mon ~]# ceph osd unset norecover
   [root@mon ~]# ceph osd unset norebalance
   [root@mon ~]# ceph osd unset nobackfill
   [root@mon ~]# ceph osd unset nodown
   [root@mon ~]# ceph osd unset pause
   ```

6. If you use the Ceph File System (**CephFS**), the **CephFS** cluster must be brought back up by setting the **cluster_down** flag to **false**:

```
[root@admin~]# ceph fs set FS_NAME cluster_down false
```

7. Verify the cluster is in healthy state (**Health_OK** and all PGs **active+clean**). Run **ceph status** on a node with the client keyrings. For example, the Ceph Monitor or OpenStack controller nodes, to ensure the cluster is healthy.

2.11. ADDITIONAL RESOURCES

- For more information on installing Ceph see the *Red Hat Ceph Storage Installation Guide*

CHAPTER 3. MONITORING A CEPH STORAGE CLUSTER

As a storage administrator, you can monitor the overall health of the Red Hat Ceph Storage cluster, along with monitoring the health of the individual components of Ceph.

Once you have a running Red Hat Ceph Storage cluster, you might begin monitoring the storage cluster to ensure that the Ceph Monitor and Ceph OSD daemons are running, at a high-level. Ceph storage cluster clients connect to a Ceph Monitor and receive the latest version of the storage cluster map before they can read and write data to the Ceph pools within the storage cluster. So the monitor cluster must have agreement on the state of the cluster before Ceph clients can read and write data.

Ceph OSDs must peer the placement groups on the primary OSD with the copies of the placement groups on secondary OSDs. If faults arise, peering will reflect something other than the **active + clean** state.

3.1. PREREQUISITES

- A running Red Hat Ceph Storage cluster.

3.2. HIGH-LEVEL MONITORING OF A CEPH STORAGE CLUSTER

As a storage administrator, you can monitor the health of the Ceph daemons to ensure that they are up and running. High level monitoring also involves checking the storage cluster capacity to ensure that the storage cluster does not exceed its **full ratio**. The Red Hat Ceph Storage Dashboard is the most common way to conduct high-level monitoring. However, you can also use the command-line interface, the Ceph admin socket or the Ceph API to monitor the storage cluster.

3.2.1. Prerequisites

- A running Red Hat Ceph Storage cluster.

3.2.2. Using the Ceph command interface interactively

You can interactively interface with the Ceph storage cluster by using the **ceph** command-line utility.

Prerequisites

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

Procedure

1. To run the **ceph** utility in interactive mode.

   a. **Bare-metal** deployments:

      Example

```
[root@mon ~]# ceph
ceph> health
ceph> status
ceph> quorum_status
ceph> mon_status
```

b. **Container** deployments:

Red Hat Enterprise Linux 7

```
docker exec -it ceph-mon-MONITOR_NAME /bin/bash
```

Red Hat Enterprise Linux 8

```
podman exec -it ceph-mon-MONITOR_NAME /bin/bash
```

Replace

- *MONITOR_NAME* with the name of the Ceph Monitor container, found by running the **docker ps** or **podman ps** command respectively.

  Example

  ```
  [root@container-host ~]# podman exec -it ceph-mon-mon01 /bin/bash
  ```

  This example opens an interactive terminal session on **mon01**, where you can start the Ceph interactive shell.

3.2.3. Checking the storage cluster health

After you start the Ceph storage cluster, and before you start reading or writing data, check the storage cluster's health first.

Prerequisites

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

Procedure

1. You can check on the health of the Ceph storage cluster with the following:

```
[root@mon ~]# ceph health
```

2. If you specified non-default locations for the configuration or keyring, you can specify their locations:

```
[root@mon ~]# ceph -c /path/to/conf -k /path/to/keyring health
```

Upon starting the Ceph cluster, you will likely encounter a health warning such as **HEALTH_WARN XXX num placement groups stale**. Wait a few moments and check it again. When the storage cluster is ready, **ceph health** should return a message such as **HEALTH_OK**. At that point, it is okay to begin using the cluster.

3.2.4. Watching storage cluster events

You can watch events that are happening with the Ceph storage cluster using the command-line interface.

Prerequisites

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

Procedure

1. To watch the cluster's ongoing events on the command line, open a new terminal, and then enter:

   ```
   [root@mon ~]# ceph -w
   ```

   Ceph will print each event. For example, a tiny Ceph cluster consisting of one monitor and two OSDs may print the following:

   ```
   cluster b370a29d-9287-4ca3-ab57-3d824f65e339
    health HEALTH_OK
    monmap e1: 1 mons at {ceph1=10.0.0.8:6789/0}, election epoch 2, quorum 0 ceph1
    osdmap e63: 2 osds: 2 up, 2 in
     pgmap v41338: 952 pgs, 20 pools, 17130 MB data, 2199 objects
           115 GB used, 167 GB / 297 GB avail
                952 active+clean

   2014-06-02 15:45:21.655871 osd.0 [INF] 17.71 deep-scrub ok
   2014-06-02 15:45:47.880608 osd.1 [INF] 1.0 scrub ok
   2014-06-02 15:45:48.865375 osd.1 [INF] 1.3 scrub ok
   2014-06-02 15:45:50.866479 osd.1 [INF] 1.4 scrub ok
   2014-06-02 15:45:01.345821 mon.0 [INF] pgmap v41339: 952 pgs: 952 active+clean; 17130
   MB data, 115 GB used, 167 GB / 297 GB avail
   2014-06-02 15:45:05.718640 mon.0 [INF] pgmap v41340: 952 pgs: 1
   active+clean+scrubbing+deep, 951 active+clean; 17130 MB data, 115 GB used, 167 GB /
   297 GB avail
   2014-06-02 15:45:53.997726 osd.1 [INF] 1.5 scrub ok
   2014-06-02 15:45:06.734270 mon.0 [INF] pgmap v41341: 952 pgs: 1
   active+clean+scrubbing+deep, 951 active+clean; 17130 MB data, 115 GB used, 167 GB /
   297 GB avail
   2014-06-02 15:45:15.722456 mon.0 [INF] pgmap v41342: 952 pgs: 952 active+clean; 17130
   MB data, 115 GB used, 167 GB / 297 GB avail
   2014-06-02 15:46:06.836430 osd.0 [INF] 17.75 deep-scrub ok
   2014-06-02 15:45:55.720929 mon.0 [INF] pgmap v41343: 952 pgs: 1
   active+clean+scrubbing+deep, 951 active+clean; 17130 MB data, 115 GB used, 167 GB /
   297 GB avail
   ```

   The output provides:

   - Cluster ID
   - Cluster health status
   - The monitor map epoch and the status of the monitor quorum

- The OSD map epoch and the status of OSDs
- The placement group map version
- The number of placement groups and pools
- The *notional* amount of data stored and the number of objects stored
- The total amount of data stored

3.2.5. How Ceph calculates data usage

The **used** value reflects the *actual* amount of raw storage used. The **xxx GB** / **xxx GB** value means the amount available, the lesser of the two numbers, of the overall storage capacity of the cluster. The notional number reflects the size of the stored data before it is replicated, cloned or snapshotted. Therefore, the amount of data actually stored typically exceeds the notional amount stored, because Ceph creates replicas of the data and may also use storage capacity for cloning and snapshotting.

3.2.6. Understanding the storage clusters usage stats

To check a cluster's data usage and data distribution among pools, use the **df** option. It is similar to the Linux **df** command. You can run either the **ceph df** command or **ceph df detail** command.

Example

```
[root@mon ~]# ceph df
RAW STORAGE:
    CLASS     SIZE       AVAIL      USED        RAW USED     %RAW USED
    hdd       90 GiB     84 GiB     100 MiB     6.1 GiB          6.78
    TOTAL     90 GiB     84 GiB     100 MiB     6.1 GiB          6.78

POOLS:
    POOL                        ID     STORED      OBJECTS     USED        %USED     MAX AVAIL
    .rgw.root                    1     1.3 KiB           4     768 KiB         0        26 GiB
    default.rgw.control          2         0 B           8         0 B         0        26 GiB
    default.rgw.meta             3     2.5 KiB          12     2.1 MiB         0        26 GiB
    default.rgw.log              4     3.5 KiB         208     6.2 MiB         0        26 GiB
    default.rgw.buckets.index    5     2.4 KiB          33     2.4 KiB         0        26 GiB
    default.rgw.buckets.data     6     9.6 KiB          15     1.7 MiB         0        26 GiB
    testpool                    10       231 B           5     384 KiB         0        40 GiB
```

The **ceph df detail** command gives more details about other pool statistics such as quota objects, quota bytes, used compression, and under compression.

Example

```
[root@mon ~]# ceph df detail
RAW STORAGE:
    CLASS     SIZE       AVAIL      USED        RAW USED     %RAW USED
    hdd       90 GiB     84 GiB     100 MiB     6.1 GiB          6.78
    TOTAL     90 GiB     84 GiB     100 MiB     6.1 GiB          6.78

POOLS:
    POOL                        ID     STORED      OBJECTS     USED        %USED     MAX AVAIL
QUOTA OBJECTS     QUOTA BYTES     DIRTY     USED COMPR     UNDER COMPR
```

```
    .rgw.root                  1     1.3 KiB         4     768 KiB       0       26 GiB     N/A             N/A
4           0 B           0 B
    default.rgw.control          2        0 B          8        0 B       0       26 GiB     N/A             N/A
8           0 B           0 B
    default.rgw.meta             3     2.5 KiB         12     2.1 MiB       0       26 GiB     N/A             N/A
12           0 B           0 B
    default.rgw.log             4     3.5 KiB        208     6.2 MiB       0       26 GiB     N/A             N/A
208           0 B           0 B
    default.rgw.buckets.index     5     2.4 KiB         33     2.4 KiB       0       26 GiB     N/A             N/A
33           0 B           0 B
    default.rgw.buckets.data      6     9.6 KiB         15     1.7 MiB       0       26 GiB     N/A             N/A
15           0 B           0 B
    testpool                   10      231 B          5     384 KiB       0       40 GiB     N/A             N/A
5           0 B           0 B
```

The **RAW STORAGE** section of the output provides an overview of the amount of storage the storage cluster uses for data.

- **CLASS:** The type of devices used.
- **SIZE:** The overall storage capacity managed by the storage cluster.
  In the above example, if the **SIZE** is 90 GiB, it is the total size without the replication factor, which is three by default. The total available capacity with the replication factor is 90 GiB/3 = 30 GiB. Based on the full ratio, which is 0.85% by default, the maximum available space is 30 GiB * 0.85 = 25.5 GiB
- **AVAIL:** The amount of free space available in the storage cluster.
  In the above example, if the **SIZE** is 90 GiB and the **USED** space is 6 GiB, then the **AVAIL** space is 84 GiB. The total available space with the replication factor, which is three by default, is 84 GiB/3 = 28 GiB
- **USED:** The amount of used space in the storage cluster consumed by user data, internal overhead, or reserved capacity.
  In the above example, 100 MiB is the total space available after considering the replication factor. The actual available size is 33 MiB.
- **RAW USED:** The sum of **USED** space and the space allocated the **db** and **wal** BlueStore partitions.
- **% RAW USED:** The percentage of of **RAW USED**. Use this number in conjunction with the **full ratio** and **near full ratio** to ensure that you are not reaching the storage cluster's capacity.

The **POOLS** section of the output provides a list of pools and the notional usage of each pool. The output from this section **DOES NOT** reflect replicas, clones or snapshots. For example, if you store an object with 1 MB of data, the notional usage will be 1 MB, but the actual usage may be 3 MB or more depending on the number of replicas for example, **size = 3**, clones and snapshots.

- **POOL:** The name of the pool.
- **ID:** The pool ID.
- **STORED:** The actual amount of data stored by the user in the pool.
- **OBJECTS:** The notional number of objects stored per pool.
- **USED:** The notional amount of data stored in kilobytes, unless the number appends **M** for megabytes or **G** for gigabytes. It is **STORED** size * replication factor.

- **%USED:** The notional percentage of storage used per pool.
- **MAX AVAIL:** An estimate of the notional amount of data that can be written to this pool. It is the amount of data that can be used before the first OSD becomes full. It considers the projected distribution of data across disks from the CRUSH map and uses the first OSD to fill up as the target.
  In the above example, **MAX AVAIL** is 153.85 without considering the replication factor, which is three by default.

  See the KnowledgeBase article *ceph df MAX AVAIL is incorrect for simple replicated pool* to calculate the value of **MAX AVAIL**.
- **QUOTA OBJECTS:** The number of quota objects.
- **QUOTA BYTES:** The number of bytes in the quota objects.
- **USED COMPR:** The amount of space allocated for compressed data including his includes compressed data, allocation, replication and erasure coding overhead.
- **UNDER COMPR:** The amount of data passed through compression and beneficial enough to be stored in a compressed form.

NOTE

The numbers in the **POOLS** section are notional. They are not inclusive of the number of replicas, snapshots or clones. As a result, the sum of the **USED** and **%USED** amounts will not add up to the **RAW USED** and **%RAW USED** amounts in the **GLOBAL** section of the output.

NOTE

The MAX AVAIL value is a complicated function of the replication or erasure code used, the CRUSH rule that maps storage to devices, the utilization of those devices, and the configured **mon_osd_full_ratio**.

Additional Resources

- See How Ceph calculates data usage for details.
- See Understanding the OSD usage stats for details.

3.2.7. Understanding the OSD usage stats

Use the **ceph osd df** command to view OSD utilization stats.

```
[root@mon]# ceph osd df
ID CLASS WEIGHT  REWEIGHT SIZE    USE     DATA    OMAP    META    AVAIL   %USE VAR
PGS
 3   hdd 0.90959  1.00000  931GiB 70.1GiB 69.1GiB      0B    1GiB  861GiB 7.53 2.93  66
 4   hdd 0.90959  1.00000  931GiB 1.30GiB  308MiB      0B    1GiB  930GiB 0.14 0.05  59
 0   hdd 0.90959  1.00000  931GiB 18.1GiB 17.1GiB      0B    1GiB  913GiB 1.94 0.76  57
MIN/MAX VAR: 0.02/2.98  STDDEV: 2.91
```

- **ID:** The name of the OSD.

- **CLASS:** The type of devices the OSD uses.
- **WEIGHT:** The weight of the OSD in the CRUSH map.
- **REWEIGHT:** The default reweight value.
- **SIZE:** The overall storage capacity of the OSD.
- **USE:** The OSD capacity.
- **DATA:** The amount of OSD capacity that is used by user data.
- **OMAP:** An estimate value of the **bluefs** storage that is being used to store object map (**omap**) data (key value pairs stored in **rocksdb**).
- **META:** The **bluefs** space allocated, or the value set in the **bluestore_bluefs_min** parameter, whichever is larger, for internal metadata which is calculated as the total space allocated in **bluefs** minus the estimated **omap** data size.
- **AVAIL:** The amount of free space available on the OSD.
- **%USE:** The notional percentage of storage used by the OSD
- **VAR:** The variation above or below average utilization.
- **PGS:** The number of placement groups in the OSD.
- **MIN/MAX VAR:** The minimum and maximum variation across all OSDs.

Additional Resources

- See *How Ceph calculates data usage* for details.
- See *Understanding the OSD usage stats* for details.
- See *CRUSH Weights* in *Red Hat Ceph Storage Storage Strategies Guide* for details.

3.2.8. Checking the Red Hat Ceph Storage cluster status

You can check the status of the Red Hat Ceph Storage cluster from the command-line interface. The **status** sub command or the **-s** argument will display the current status of the storage cluster.

Prerequisites

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

Procedure

1. To check a storage cluster's status, execute the following:

   ```
   [root@mon ~]# ceph status
   ```

   Or:

```
[root@mon ~]# ceph -s
```

2. In interactive mode, type **status** and press **Enter**:

```
[root@mon ~]# ceph> status
```

For example, a tiny Ceph cluster consisting of one monitor, and two OSDs can print the following:

```
cluster b370a29d-9287-4ca3-ab57-3d824f65e339
 health HEALTH_OK
 monmap e1: 1 mons at {ceph1=10.0.0.8:6789/0}, election epoch 2, quorum 0 ceph1
 osdmap e63: 2 osds: 2 up, 2 in
  pgmap v41332: 952 pgs, 20 pools, 17130 MB data, 2199 objects
        115 GB used, 167 GB / 297 GB avail
               1 active+clean+scrubbing+deep
             951 active+clean
```

3.2.9. Checking the Ceph Monitor status

If the storage cluster has multiple Ceph Monitors, which is a requirement for a production Red Hat Ceph Storage cluster, then check the Ceph Monitor quorum status after starting the storage cluster, and before doing any reading or writing of data.

A quorum must be present when multiple monitors are running.

Check Ceph Monitor status periodically to ensure that they are running. If there is a problem with the Ceph Monitor, that prevents an agreement on the state of the storage cluster, the fault may prevent Ceph clients from reading and writing data.

Prerequisites

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

Procedure

1. To display the monitor map, execute the following:

```
[root@mon ~]# ceph mon stat
```

or

```
[root@mon ~]# ceph mon dump
```

2. To check the quorum status for the storage cluster, execute the following:

```
[root@mon ~]# ceph quorum_status -f json-pretty
```

Ceph will return the quorum status. A Red Hat Ceph Storage cluster consisting of three monitors may return the following:

Example

```
{ "election_epoch": 10,
  "quorum": [
        0,
        1,
        2],
  "monmap": { "epoch": 1,
      "fsid": "444b489c-4f16-4b75-83f0-cb8097468898",
      "modified": "2011-12-12 13:28:27.505520",
      "created": "2011-12-12 13:28:27.505520",
      "mons": [
            { "rank": 0,
              "name": "a",
              "addr": "127.0.0.1:6789\/0"},
            { "rank": 1,
              "name": "b",
              "addr": "127.0.0.1:6790\/0"},
            { "rank": 2,
              "name": "c",
              "addr": "127.0.0.1:6791\/0"}
           ]
    }
}
```

3.2.10. Using the Ceph administration socket

Use the administration socket to interact with a given daemon directly by using a UNIX socket file. For example, the socket enables you to:

- List the Ceph configuration at runtime
- Set configuration values at runtime directly without relying on Monitors. This is useful when Monitors are **down**.
- Dump historic operations
- Dump the operation priority queue state
- Dump operations without rebooting
- Dump performance counters

In addition, using the socket is helpful when troubleshooting problems related to Monitors or OSDs.

> **IMPORTANT**
>
> The administration socket is only available while a daemon is running. When you shut down the daemon properly, the administration socket is removed. However, if the daemon terminates unexpectedly, the administration socket might persist.

Regardless, if the daemon is not running, a following error is returned when attempting to use the administration socket:

```
Error 111: Connection Refused
```

Prerequisites

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

Procedure

1. To use the socket:

   Syntax

   ```
   [root@mon ~]# ceph daemon TYPE.ID COMMAND
   ```

   Replace:

   - ***TYPE*** with the type of the Ceph daemon (**mon**, **osd**, **mds**).
   - ***ID*** with the daemon ID
   - ***COMMAND*** with the command to run. Use **help** to list the available commands for a given daemon.

     Example

     To view a Monitor status of a Ceph Monitor named **mon.0**:

     ```
     [root@mon ~]# ceph daemon mon.0 mon_status
     ```

2. Alternatively, specify the Ceph daemon by using its socket file:

   ```
   ceph daemon /var/run/ceph/SOCKET_FILE COMMAND
   ```

3. To view the status of an Ceph OSD named **osd.2**:

   ```
   [root@mon ~]# ceph daemon /var/run/ceph/ceph-osd.2.asok status
   ```

4. To list all socket files for the Ceph processes:

   ```
   [root@mon ~]# ls /var/run/ceph
   ```

Additional Resources

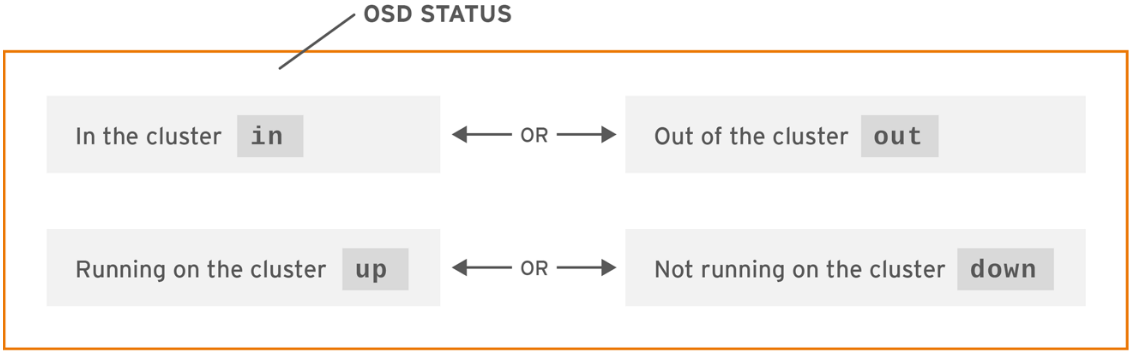
- See the *Red Hat Ceph Storage Troubleshooting Guide* for more information.

3.2.11. Understanding the Ceph OSD status

An OSD's status is either in the cluster, **in**, or out of the cluster, **out**. It is either up and running, **up**, or it is down and not running, or **down**. If an OSD is **up**, it may be either **in** the storage cluster, where data can be read and written, or it is **out** of the storage cluster. If it was **in** the cluster and recently moved **out** of the cluster, Ceph will migrate placement groups to other OSDs. If an OSD is **out** of the cluster, CRUSH will not assign placement groups to the OSD. If an OSD is **down**, it should also be **out**.

NOTE

If an OSD is **down** and **in**, there is a problem and the cluster will not be in a healthy state.



CEPH_459704_1017

If you execute a command such as **ceph health**, **ceph -s** or **ceph -w**, you may notice that the cluster does not always echo back **HEALTH OK**. Don't panic. With respect to OSDs, you should expect that the cluster will **NOT** echo **HEALTH OK** in a few expected circumstances:

- You haven't started the cluster yet, it won't respond.
- You have just started or restarted the cluster and it's not ready yet, because the placement groups are getting created and the OSDs are in the process of peering.
- You just added or removed an OSD.
- You just have modified the cluster map.

An important aspect of monitoring OSDs is to ensure that when the cluster is up and running that all OSDs that are **in** the cluster are **up** and running, too.

To see if all OSDs are running, execute:

```
[root@mon ~]# ceph osd stat
```

or

```
[root@mon ~]# ceph osd dump
```

The result should tell you the map epoch, **eNNNN**, the total number of OSDs, **x**, how many, **y**, are **up**, and how many, **z**, are **in**:

```
eNNNN: x osds: y up, z in
```

If the number of OSDs that are **in** the cluster is more than the number of OSDs that are **up**. Execute the following command to identify the **ceph-osd** daemons that aren't running:

```
[root@mon ~]# ceph osd tree
```

Example

```
# id    weight  type name   up/down reweight
-1  3   pool default
-3  3       rack mainrack
-2  3           host osd-host
0   1               osd.0   up  1
1   1               osd.1   up  1
2   1               osd.2   up  1
```

TIP

The ability to search through a well-designed CRUSH hierarchy may help you troubleshoot the storage cluster by identifying the physical locations faster.

If an OSD is **down**, connect to the node and start it. You can use Red Hat Storage Console to restart the OSD node, or you can use the command line.

Example

```
[root@mon ~]# systemctl start ceph-osd@OSD_ID
```

3.2.12. Additional Resources

- *Red Hat Ceph Storage Dashboard Guide*.

3.3. LOW-LEVEL MONITORING OF A CEPH STORAGE CLUSTER

As a storage administrator, you can monitor the health of a Red Hat Ceph Storage cluster from a low-level perspective. Low-level monitoring typically involves ensuring that Ceph OSDs are peering properly. When peering faults occur, placement groups operate in a degraded state. This degraded state can be the result of many different things, such as hardware failure, a hung or crashed Ceph daemon, network latency, or a complete site outage.

3.3.1. Prerequisites

- A running Red Hat Ceph Storage cluster.

3.3.2. Monitoring Placement Group Sets

When CRUSH assigns placement groups to OSDs, it looks at the number of replicas for the pool and assigns the placement group to OSDs such that each replica of the placement group gets assigned to a different OSD. For example, if the pool requires three replicas of a placement group, CRUSH may assign them to **osd.1**, **osd.2** and **osd.3** respectively. CRUSH actually seeks a pseudo-random placement that will take into account failure domains you set in the CRUSH map, so you will rarely see placement groups assigned to nearest neighbor OSDs in a large cluster. We refer to the set of OSDs that should contain the replicas of a particular placement group as the **Acting Set**. In some cases, an OSD in the Acting Set is **down** or otherwise not able to service requests for objects in the placement group. When these situations arise, don’t panic. Common examples include:

- You added or removed an OSD. Then, CRUSH reassigned the placement group to other OSDs—thereby changing the composition of the Acting Set and spawning the migration of data with a "backfill" process.

- An OSD was **down**, was restarted and is now **recovering**.

- An OSD in the Acting Set is **down** or unable to service requests, and another OSD has temporarily assumed its duties.

Ceph processes a client request using the **Up Set**, which is the set of OSDs that will actually handle the requests. In most cases, the Up Set and the Acting Set are virtually identical. When they are not, it may indicate that Ceph is migrating data, an OSD is recovering, or that there is a problem, that is, Ceph usually echoes a **HEALTH WARN** state with a "stuck stale" message in such scenarios.

Prerequisites

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

Procedure

1. To retrieve a list of placement groups:

```
[root@mon ~]# ceph pg dump
```

2. To view which OSDs are in the Acting Set or in the Up Set for a given placement group:

```
[root@mon ~]# ceph pg map PG_NUM
```

The result should tell you the osdmap epoch, **eNNN**, the placement group number, ***PG_NUM***, the OSDs in the Up Set **up[]**, and the OSDs in the acting set, **acting[]**:

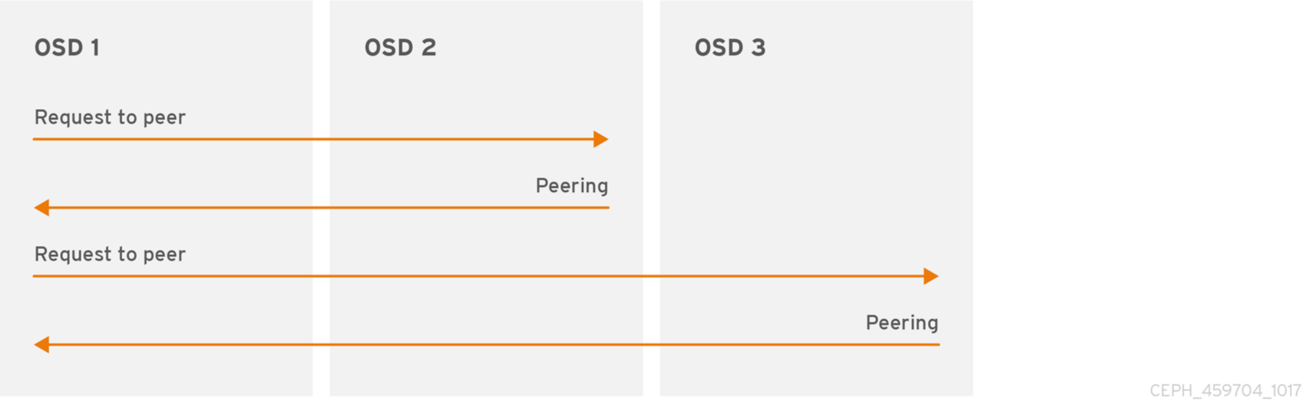
```
[root@mon ~]# ceph osdmap eNNN pg PG_NUM-> up [0,1,2] acting [0,1,2]
```

NOTE

If the Up Set and Acting Set do not match, this may be an indicator that the cluster rebalancing itself or of a potential problem with the cluster.

3.3.3. Ceph OSD peering

Before you can write data to a placement group, it must be in an **active** state, and it **should** be in a **clean** state. For Ceph to determine the current state of a placement group, the primary OSD of the placement group that is, the first OSD in the acting set, peers with the secondary and tertiary OSDs to establish agreement on the current state of the placement group. Assuming a pool with 3 replicas of the PG.



3.3.4. Placement Group States

If you execute a command such as **ceph health**, **ceph -s** or **ceph -w**, you may notice that the cluster does not always echo back **HEALTH OK**. After you check to see if the OSDs are running, you should also check placement group states. You should expect that the cluster will **NOT** echo **HEALTH OK** in a number of placement group peering-related circumstances:

- You have just created a pool and placement groups haven't peered yet.
- The placement groups are recovering.
- You have just added an OSD to or removed an OSD from the cluster.
- You have just modified the CRUSH map and the placement groups are migrating.
- There is inconsistent data in different replicas of a placement group.
- Ceph is scrubbing a placement group's replicas.
- Ceph doesn't have enough storage capacity to complete backfilling operations.

If one of the foregoing circumstances causes Ceph to echo **HEALTH WARN**, don't panic. In many cases, the cluster will recover on its own. In some cases, you may need to take action. An important aspect of monitoring placement groups is to ensure that when the cluster is up and running that all placement groups are **active**, and preferably in the **clean** state.

To see the status of all placement groups, execute:

```
[root@mon ~]# ceph pg stat
```

The result should tell you the placement group map version, **vNNNNNN**, the total number of placement groups, **x**, and how many placement groups, **y**, are in a particular state such as **active+clean**:

```
vNNNNNN: x pgs: y active+clean; z bytes data, aa MB used, bb GB / cc GB avail
```

NOTE

It is common for Ceph to report multiple states for placement groups.

Snapshot Trimming PG States

When snapshots exist, two additional PG states will be reported.

- **snaptrim** : The PGs are currently being trimmed
- **snaptrim_wait** : The PGs are waiting to be trimmed

Example Output:

```
244 active+clean+snaptrim_wait
 32 active+clean+snaptrim
```

In addition to the placement group states, Ceph will also echo back the amount of data used, **aa**, the amount of storage capacity remaining, **bb**, and the total storage capacity for the placement group. These numbers can be important in a few cases:

- You are reaching the **near full ratio** or **full ratio**.
- Your data isn't getting distributed across the cluster due to an error in the CRUSH configuration.

Placement Group IDs

Placement group IDs consist of the pool number, and not the pool name, followed by a period (.) and the placement group ID—a hexadecimal number. You can view pool numbers and their names from the output of **ceph osd lspools**. The default pool names **data**, **metadata** and **rbd** correspond to pool numbers **0**, **1** and **2** respectively. A fully qualified placement group ID has the following form:

```
POOL_NUM.PG_ID
```

Example output:

```
0.1f
```

- To retrieve a list of placement groups:

  ```
  [root@mon ~]# ceph pg dump
  ```

- To format the output in JSON format and save it to a file:

  ```
  [root@mon ~]# ceph pg dump -o FILE_NAME --format=json
  ```

- To query a particular placement group:

  ```
  [root@mon ~]# ceph pg POOL_NUM.PG_ID query
  ```

  Example output in JSON format:

  ```
  {
    "state": "active+clean",
    "up": [
      1,
      0
    ],
    "acting": [
      1,
      0
    ],
    "info": {
      "pgid": "1.e",
      "last_update": "4'1",
      "last_complete": "4'1",
      "log_tail": "0'0",
      "last_backfill": "MAX",
      "purged_snaps": "[]",
      "history": {
        "epoch_created": 1,
        "last_epoch_started": 537,
        "last_epoch_clean": 537,
        "last_epoch_split": 534,
  ```

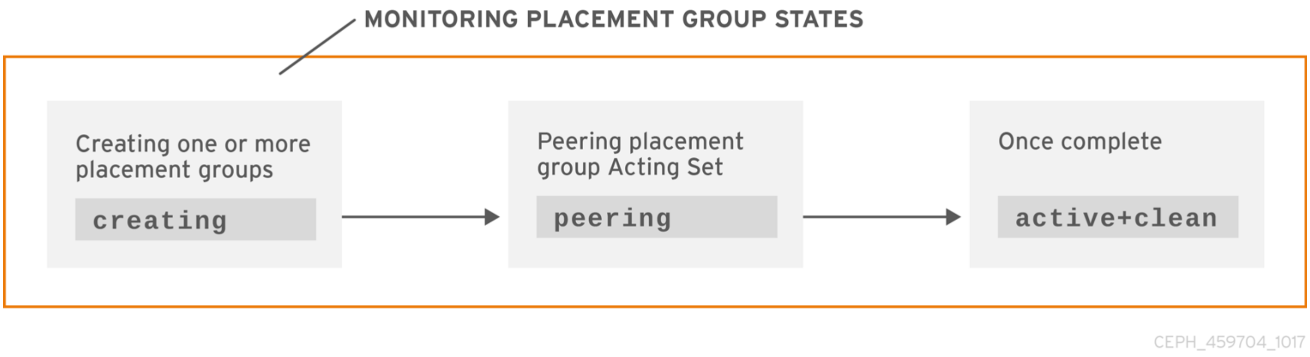
```
      "same_up_since": 536,
      "same_interval_since": 536,
      "same_primary_since": 536,
      "last_scrub": "4'1",
      "last_scrub_stamp": "2013-01-25 10:12:23.828174"
    },
    "stats": {
      "version": "4'1",
      "reported": "536'782",
      "state": "active+clean",
      "last_fresh": "2013-01-25 10:12:23.828271",
      "last_change": "2013-01-25 10:12:23.828271",
      "last_active": "2013-01-25 10:12:23.828271",
      "last_clean": "2013-01-25 10:12:23.828271",
      "last_unstale": "2013-01-25 10:12:23.828271",
      "mapping_epoch": 535,
      "log_start": "0'0",
      "ondisk_log_start": "0'0",
      "created": 1,
      "last_epoch_clean": 1,
      "parent": "0.0",
      "parent_split_bits": 0,
      "last_scrub": "4'1",
      "last_scrub_stamp": "2013-01-25 10:12:23.828174",
      "log_size": 128,
      "ondisk_log_size": 128,
      "stat_sum": {
        "num_bytes": 205,
        "num_objects": 1,
        "num_object_clones": 0,
        "num_object_copies": 0,
        "num_objects_missing_on_primary": 0,
        "num_objects_degraded": 0,
        "num_objects_unfound": 0,
        "num_read": 1,
        "num_read_kb": 0,
        "num_write": 3,
        "num_write_kb": 1
      },
      "stat_cat_sum": {

      },
      "up": [
        1,
        0
      ],
      "acting": [
        1,
        0
      ]
    },
    "empty": 0,
    "dne": 0,
    "incomplete": 0
  },
  "recovery_state": [
    {
      "name": "Started\/Primary\/Active",
      "enter_time": "2013-01-23 09:35:37.594691",
      "might_have_unfound": [

      ],
      "scrub": {
        "scrub_epoch_start": "536",
        "scrub_active": 0,
        "scrub_block_writes": 0,
        "finalizing_scrub": 0,
        "scrub_waiting_on": 0,
        "scrub_waiting_on_whom": [

        ]
      }
    },
    {
      "name": "Started",
      "enter_time": "2013-01-23 09:35:31.581160"
    }
  ]
}
```

Additional Resources

- See the chapter *Object Storage Daemon (OSD) configuration options* in the Red Hat Ceph Storage 4 *Configuration Guide* for more details on the snapshot trimming settings.

3.3.5. Placement Group creating state

When you create a pool, it will create the number of placement groups you specified. Ceph will echo **creating** when it is creating one or more placement groups. Once they are created, the OSDs that are part of a placement group's Acting Set will peer. Once peering is complete, the placement group status should be **active+clean**, which means a Ceph client can begin writing to the placement group.


MONITORING PLACEMENT GROUP STATES
Creating one or more placement groups
creating
Peering placement group Acting Set
peering
Once complete
active+clean
CEPH_459704_1017

3.3.6. Placement group peering state

When Ceph is Peering a placement group, Ceph is bringing the OSDs that store the replicas of the placement group into **agreement about the state** of the objects and metadata in the placement group. When Ceph completes peering, this means that the OSDs that store the placement group agree about the current state of the placement group. However, completion of the peering process does **NOT** mean that each replica has the latest contents.

Authoritative History

Ceph will **NOT** acknowledge a write operation to a client, until all OSDs of the acting set persist the write operation. This practice ensures that at least one member of the acting set will have a record of every acknowledged write operation since the last successful peering operation.

With an accurate record of each acknowledged write operation, Ceph can construct and disseminate a new authoritative history of the placement group. A complete, and fully ordered set of operations that, if performed, would bring an OSD’s copy of a placement group up to date.

3.3.7. Placement group active state

Once Ceph completes the peering process, a placement group may become **active**. The **active** state means that the data in the placement group is generally available in the primary placement group and the replicas for read and write operations.

3.3.8. Placement Group clean state

When a placement group is in the **clean** state, the primary OSD and the replica OSDs have successfully peered and there are no stray replicas for the placement group. Ceph replicated all objects in the placement group the correct number of times.

3.3.9. Placement Group degraded state

When a client writes an object to the primary OSD, the primary OSD is responsible for writing the replicas to the replica OSDs. After the primary OSD writes the object to storage, the placement group will remain in a **degraded** state until the primary OSD has received an acknowledgement from the replica OSDs that Ceph created the replica objects successfully.

The reason a placement group can be **active+degraded** is that an OSD may be **active** even though it doesn’t hold all of the objects yet. If an OSD goes **down**, Ceph marks each placement group assigned to the OSD as **degraded**. The OSDs must peer again when the OSD comes back online. However, a client can still write a new object to a **degraded** placement group if it is **active**.

If an OSD is **down** and the **degraded** condition persists, Ceph may mark the **down** OSD as **out** of the cluster and remap the data from the **down** OSD to another OSD. The time between being marked **down** and being marked **out** is controlled by **mon_osd_down_out_interval**, which is set to **600** seconds by default.

A placement group can also be **degraded**, because Ceph cannot find one or more objects that Ceph thinks should be in the placement group. While you cannot read or write to unfound objects, you can still access all of the other objects in the **degraded** placement group.

Let’s say there are 9 OSDs in a three way replica pool. If OSD number 9 goes down, the PGs assigned to OSD 9 go in a degraded state. If OSD 9 doesn’t recover, it goes out of the cluster and the cluster rebalances. In that scenario, the PGs are degraded and then recover to an active state.

3.3.10. Placement Group recovering state

Ceph was designed for fault-tolerance at a scale where hardware and software problems are ongoing. When an OSD goes **down**, its contents may fall behind the current state of other replicas in the placement groups. When the OSD is back **up**, the contents of the placement groups must be updated to reflect the current state. During that time period, the OSD may reflect a **recovering** state.

Recovery isn’t always trivial, because a hardware failure might cause a cascading failure of multiple

OSDs. For example, a network switch for a rack or cabinet may fail, which can cause the OSDs of a number of host machines to fall behind the current state of the cluster. Each one of the OSDs must recover once the fault is resolved.

Ceph provides a number of settings to balance the resource contention between new service requests and the need to recover data objects and restore the placement groups to the current state. The **osd recovery delay start** setting allows an OSD to restart, re-peer and even process some replay requests before starting the recovery process. The **osd recovery threads** setting limits the number of threads for the recovery process, by default one thread. The **osd recovery thread timeout** sets a thread timeout, because multiple OSDs may fail, restart and re-peer at staggered rates. The **osd recovery max active** setting limits the number of recovery requests an OSD will entertain simultaneously to prevent the OSD from failing to serve . The **osd recovery max chunk** setting limits the size of the recovered data chunks to prevent network congestion.

3.3.11. Back fill state

When a new OSD joins the cluster, CRUSH will reassign placement groups from OSDs in the cluster to the newly added OSD. Forcing the new OSD to accept the reassigned placement groups immediately can put excessive load on the new OSD. Backfilling the OSD with the placement groups allows this process to begin in the background. Once backfilling is complete, the new OSD will begin serving requests when it is ready.

During the backfill operations, you may see one of several states: * **backfill_wait** indicates that a backfill operation is pending, but isn't underway yet * **backfill** indicates that a backfill operation is underway * **backfill_too_full** indicates that a backfill operation was requested, but couldn't be completed due to insufficient storage capacity.

When a placement group cannot be backfilled, it may be considered **incomplete**.

Ceph provides a number of settings to manage the load spike associated with reassigning placement groups to an OSD, especially a new OSD. By default, **osd_max_backfills** sets the maximum number of concurrent backfills to or from an OSD to 10. The **osd backfill full ratio** enables an OSD to refuse a backfill request if the OSD is approaching its full ratio, by default 85%. If an OSD refuses a backfill request, the **osd backfill retry interval** enables an OSD to retry the request, by default after 10 seconds. OSDs can also set **osd backfill scan min** and **osd backfill scan max** to manage scan intervals, by default 64 and 512.

For some workloads, it is beneficial to avoid regular recovery entirely and use backfill instead. Since backfilling occurs in the background, this allows I/O to proceed on the objects in the OSD. To force backfill rather than recovery, set **osd_min_pg_log_entries** to **1**, and set **osd_max_pg_log_entries** to **2**. Contact your Red Hat Support account team for details on when this situation is appropriate for your workload.

3.3.12. Changing the priority of recovery or backfill operations

You might encounter a situation where some placement groups (PGs) require recovery and/or backfill, and some of those placement groups contain more important data than do others. Use the **pg force-recovery** or **pg force-backfill** command to ensure that the PGs with the higher-priority data undergo recovery or backfill first.

Prerequisites

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

Procedure

1. Issue the **pg force-recovery** or **pg force-backfill** command and specify the order of priority for the PGs with the higher-priority data:

   Syntax

   ```
   ceph pg force-recovery PG1 [PG2] [PG3 ...]
   ceph pg force-backfill PG1 [PG2] [PG3 ...]
   ```

   Example

   ```
   [root@node]# ceph pg force-recovery group1 group2
   [root@node]# ceph pg force-backfill group1 group2
   ```

   This command causes Red Hat Ceph Storage to perform recovery or backfill on specified placement groups (PGs) first, before processing other placement groups. Issuing the command does not interrupt backfill or recovery operations that are currently executing. After the currently running operations have finished, recovery or backfill takes place as soon as possible for the specified PGs.

3.3.13. Changing or canceling a recovery or backfill operation on specified placement groups

If you cancel a high-priority **force-recovery** or **force-backfill** operation on certain placement groups (PGs) in a storage cluster, operations for those PGs revert to the default recovery or backfill settings.

Prerequisites

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

Procedure

1. To change or cancel a recovery or backfill operation on specified placement groups:

   Syntax

   ```
   ceph pg cancel-force-recovery PG1 [PG2] [PG3 ...]
   ceph pg cancel-force-backfill PG1 [PG2] [PG3 ...]
   ```

   Example

   ```
   [root@node]# ceph pg cancel-force-recovery group1 group2
   [root@node]# ceph pg cancel-force-backfill group1 group2
   ```

   This cancels the **force** flag and processes the PGs in the default order.

   After recovery or backfill operations for the specified PGs have completed, processing order reverts to the default.

Additional Resources

- For more information about the order of priority of recovery and backfill operations in RADOS, see Priority of placement group recovery and backfill in RADOS .

3.3.14. Forcing high-priority recovery or backfill operations for pools

If all of the placement groups in a pool require high-priority recovery or backfill, use the **force-recovery** or **force-backfill** options to initiate the operation.

Prerequisites

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

Procedure

1. To force the high-priority recovery or backfill on all placement groups in a specified pool:

   Syntax

   ```
   ceph osd pool force-recovery POOL_NAME
   ceph osd pool force-backfill POOL_NAME
   ```

   Example

   ```
   [root@node]# ceph osd pool force-recovery pool1
   [root@node]# ceph osd pool force-backfill pool1
   ```

   NOTE

   Use the **force-recovery** and **force-backfill** commands with caution. Changing the priority of these operations might break the ordering of Ceph's internal priority computations.

3.3.15. Canceling high-priority recovery or backfill operations for pools

If you cancel a high-priority **force-recovery** or **force-backfill** operation on all placement groups in a pool, operations for the PGs in that pool revert to the default recovery or backfill settings.

Prerequisites

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

Procedure

1. To cancel a high-priority recovery or backfill operation on all placement groups in a specified pool:

   Syntax

```
ceph osd pool cancel-force-recovery POOL_NAME
ceph osd pool cancel-force-backfill POOL_NAME
```

Example

```
[root@node]# ceph osd pool cancel-force-recovery pool1
[root@node]# ceph osd pool cancel-force-backfill pool1
```

3.3.16. Rearranging the priority of recovery or backfill operations for pools

If you have multiple pools that currently use the same underlying OSDs and some of the pools contain high-priority data, you can rearrange the order in which the operations execute. Use the **recovery_priority** option to assign a higher priority value to the pools with the higher-priority data. Those pools will execute before pools with lower priority values, or pools that are set to default priority.

Prerequisites

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

Procedure

1. To rearrange the recovery/backfill priority for the pools:

   Syntax

   ```
   ceph osd pool set POOL_NAME recovery_priority VALUE
   ```

   Example

   ```
   ceph osd pool set pool1 recovery_priority 10
   ```

   VALUE sets the order of priority. For example, if you have 10 pools, the pool with a priority value of 10 gets processed first, followed by the pool with priority 9, and so on. If only some pools have high priority, you can set priority values for just those pools. The pools without set priority values are processed in the default order.

3.3.17. Priority of placement group recovery in RADOS

This section describes the relative priority values for the recovery and backfilling of placement groups (PGs) in RADOS. Higher values are processed first. Inactive PGs receive higher priority values than active or degraded PGs.

Operation	Value	Description
OSD_RECOVERY_PRIORITY_MIN	0	Minimum recovery value
OSD_BACKFILL_PRIORITY_BASE	100	Base backfill priority for MBackfillReserve

Operation	Value	Description
OSD_BACKFILL_DEGRADED_PRIORITY_BASE	140	Base backfill priority for MBackfillReserve (degraded PG)
OSD_RECOVERY_PRIORITY_BASE	180	Base recovery priority for MBackfillReserve
OSD_BACKFILL_INACTIVE_PRIORITY_BASE	220	Base backfill priority for MBackfillReserve (inactive PG)
OSD_RECOVERY_INACTIVE_PRIORITY_BASE	220	Base recovery priority for MRecoveryReserve (inactive PG)
OSD_RECOVERY_PRIORITY_MAX	253	Max manually/automatically set recovery priority for MBackfillReserve
OSD_BACKFILL_PRIORITY_FORCED	254	Backfill priority for MBackfillReserve, when forced manually
OSD_RECOVERY_PRIORITY_FORCED	255	Recovery priority for MRecoveryReserve, when forced manually
OSD_DELETE_PRIORITY_NORMAL	179	Priority for PG deletion when the OSD is not fullish
OSD_DELETE_PRIORITY_FULLISH	219	Priority for PG deletion when the OSD is approaching full
OSD_DELETE_PRIORITY_FULL	255	Priority for deletion when the OSD is full

3.3.18. Placement Group remapped state

When the Acting Set that services a placement group changes, the data migrates from the old acting set to the new acting set. It may take some time for a new primary OSD to service requests. So it may ask the old primary to continue to service requests until the placement group migration is complete. Once data migration completes, the mapping uses the primary OSD of the new acting set.

3.3.19. Placement Group stale state

While Ceph uses heartbeats to ensure that hosts and daemons are running, the **ceph-osd** daemons may also get into a **stuck** state where they aren't reporting statistics in a timely manner. For example, a temporary network fault. By default, OSD daemons report their placement group, up thru, boot and failure statistics every half second, that is, **0.5**, which is more frequent than the heartbeat thresholds. If the **Primary OSD** of a placement group's acting set fails to report to the monitor or if other OSDs have reported the primary OSD **down**, the monitors will mark the placement group **stale**.

When you start the storage cluster, it is common to see the **stale** state until the peering process completes. After the storage cluster has been running for awhile, seeing placement groups in the **stale** state indicates that the primary OSD for those placement groups is **down** or not reporting placement group statistics to the monitor.

3.3.20. Placement Group misplaced state

There are some temporary backfilling scenarios where a PG gets mapped temporarily to an OSD. When that **temporary** situation should no longer be the case, the PGs might still reside in the temporary location and not in the proper location. In which case, they are said to be **misplaced**. That's because the correct number of extra copies actually exist, but one or more copies is in the wrong place.

For example, there are 3 OSDs: 0,1,2 and all PGs map to some permutation of those three. If you add another OSD (OSD 3), some PGs will now map to OSD 3 instead of one of the others. However, until OSD 3 is backfilled, the PG will have a temporary mapping allowing it to continue to serve I/O from the old mapping. During that time, the PG is **misplaced**, because it has a temporary mapping, but not **degraded**, since there are 3 copies.

Example

```
pg 1.5: up=acting: [0,1,2]
ADD_OSD_3
pg 1.5: up: [0,3,1] acting: [0,1,2]
```

[0,1,2] is a temporary mapping, so the **up** set is not equal to the **acting** set and the PG is **misplaced** but not **degraded** since [0,1,2] is still three copies.

Example

```
pg 1.5: up=acting: [0,3,1]
```

OSD 3 is now backfilled and the temporary mapping is removed, not degraded and not misplaced.

3.3.21. Placement Group incomplete state

A PG goes into a **incomplete** state when there is incomplete content and peering fails, that is, when there are no complete OSDs which are current enough to perform recovery.

Lets say OSD 1, 2, and 3 are the acting OSD set and it switches to OSD 1, 4, and 3, then **osd.1** will request a temporary acting set of OSD 1, 2, and 3 while backfilling 4. During this time, if OSD 1, 2, and 3 all go down, **osd.4** will be the only one left which might not have fully backfilled all the data. At this time, the PG will go **incomplete** indicating that there are no complete OSDs which are current enough to perform recovery.

Alternately, if **osd.4** is not involved and the acting set is simply OSD 1, 2, and 3 when OSD 1, 2, and 3 go down, the PG would likely go **stale** indicating that the mons have not heard anything on that PG since the acting set changed. The reason being there are no OSDs left to notify the new OSDs.

3.3.22. Identifying stuck Placement Groups

As previously noted, a placement group isn't necessarily problematic just because its state isn't **active+clean**. Generally, Ceph's ability to self repair may not be working when placement groups get stuck. The stuck states include:

- **Unclean**: Placement groups contain objects that are not replicated the desired number of times. They should be recovering.
- **Inactive**: Placement groups cannot process reads or writes because they are waiting for an OSD with the most up-to-date data to come back **up**.
- **Stale**: Placement groups are in an unknown state, because the OSDs that host them have not reported to the monitor cluster in a while, and can be configured with the **mon osd report timeout** setting.

Prerequisites

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

Procedure

1. To identify stuck placement groups, execute the following:

```
ceph pg dump_stuck {inactive|unclean|stale|undersized|degraded
[inactive|unclean|stale|undersized|degraded...]} {<int>}
```

3.3.23. Finding an object's location

The Ceph client retrieves the latest cluster map and the CRUSH algorithm calculates how to map the object to a placement group, and then calculates how to assign the placement group to an OSD dynamically.

Prerequisites

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

Procedure

1. To find the object location, all you need is the object name and the pool name:

```
ceph osd map POOL_NAME OBJECT_NAME
```

CHAPTER 4. OVERRIDE CEPH BEHAVIOR

As a storage administrator, you need to understand how to use overrides for the Red Hat Ceph Storage cluster to change Ceph options during runtime.

4.1. PREREQUISITES

- A running Red Hat Ceph Storage cluster.

4.2. SETTING AND UNSETTING CEPH OVERRIDE OPTIONS

You can set and unset Ceph options to override Ceph's default behavior.

Prerequisites

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

Procedure

1. To override Ceph's default behavior, use the **ceph osd set** command and the behavior you wish to override:

   ```
   ceph osd set FLAG
   ```

   Once you set the behavior, **ceph health** will reflect the override(s) that you have set for the cluster.

2. To cease overriding Ceph's default behavior, use the **ceph osd unset** command and the override you wish to cease.

   ```
   ceph osd unset FLAG
   ```

Flag	Description
noin	Prevents OSDs from being treated as **in** the cluster.
noout	Prevents OSDs from being treated as **out** of the cluster.
noup	Prevents OSDs from being treated as **up** and running.
nodown	Prevents OSDs from being treated as **down**.
full	Makes a cluster appear to have reached its **full_ratio**, and thereby prevents write operations.
pause	Ceph will stop processing read and write operations, but will not affect OSD **in**, **out**, **up** or **down** statuses.

Flag	Description
nobackfill	Ceph will prevent new backfill operations.
norebalance	Ceph will prevent new rebalancing operations.
norecover	Ceph will prevent new recovery operations.
noscrub	Ceph will prevent new scrubbing operations.
nodeep-scrub	Ceph will prevent new deep scrubbing operations.
notieragent	Ceph will disable the process that is looking for cold/dirty objects to flush and evict.

4.3. CEPH OVERRIDE USE CASES

- **noin**: Commonly used with **noout** to address flapping OSDs.
- **noout**: If the **mon osd report timeout** is exceeded and an OSD has not reported to the monitor, the OSD will get marked **out**. If this happens erroneously, you can set **noout** to prevent the OSD(s) from getting marked **out** while you troubleshoot the issue.
- **noup**: Commonly used with **nodown** to address flapping OSDs.
- **nodown**: Networking issues may interrupt Ceph 'heartbeat' processes, and an OSD may be **up** but still get marked down. You can set **nodown** to prevent OSDs from getting marked down while troubleshooting the issue.
- **full**: If a cluster is reaching its **full_ratio**, you can pre-emptively set the cluster to **full** and expand capacity.

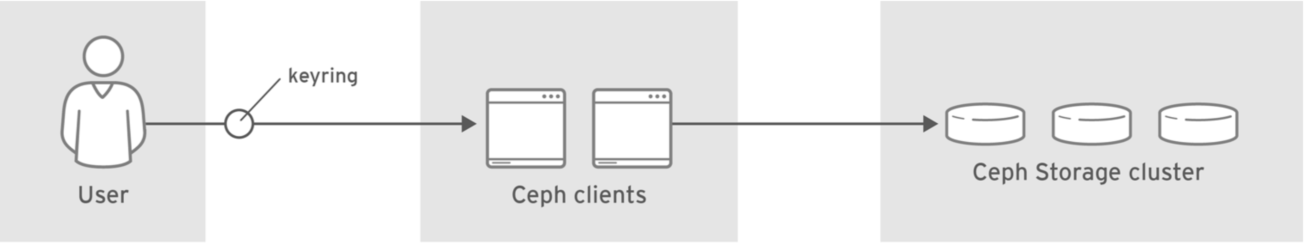
  NOTE

  Setting the cluster to **full** will prevent write operations.

- **pause**: If you need to troubleshoot a running Ceph cluster without clients reading and writing data, you can set the cluster to **pause** to prevent client operations.
- **nobackfill**: If you need to take an OSD or node **down** temporarily, for example, upgrading daemons, you can set **nobackfill** so that Ceph will not backfill while the OSDs is **down**.
- **norecover**: If you need to replace an OSD disk and don’t want the PGs to recover to another OSD while you are hotswapping disks, you can set **norecover** to prevent the other OSDs from copying a new set of PGs to other OSDs.
- **noscrub** and **nodeep-scrubb**: If you want to prevent scrubbing for example, to reduce overhead during high loads, recovery, backfilling, and rebalancing you can set **noscrub** and/or **nodeep-scrub** to prevent the cluster from scrubbing OSDs.
- **notieragent**: If you want to stop the tier agent process from finding cold objects to flush to the backing storage tier, you may set **notieragent**.

CHAPTER 5. CEPH USER MANAGEMENT

As a storage administrator, you can manage the Ceph user base by providing authentication, keyring management and access control to objects in the Red Hat Ceph Storage cluster.



CEPH_459704_1017

5.1. PREREQUISITES

- A running Red Hat Ceph Storage cluster.
- Access to a Ceph Monitor or Ceph client node.

5.2. CEPH USER MANAGEMENT BACKGROUND

When Ceph runs with authentication and authorization enabled, you must specify a user name and a keyring containing the secret key of the specified user. If you do not specify a user name, Ceph will use the **client.admin** administrative user as the default user name. If you do not specify a keyring, Ceph will look for a keyring by using the **keyring** setting in the Ceph configuration. For example, if you execute the **ceph health** command without specifying a user or keyring:

```
# ceph health
```

Ceph interprets the command like this:

```
# ceph -n client.admin --keyring=/etc/ceph/ceph.client.admin.keyring health
```

Alternatively, you may use the **CEPH_ARGS** environment variable to avoid re-entry of the user name and secret.

Irrespective of the type of Ceph client, for example, block device, object store, file system, native API, or the Ceph command line, Ceph stores all data as objects within pools. Ceph users must have access to pools in order to read and write data. Additionally, administrative Ceph users must have permissions to execute Ceph's administrative commands.

The following concepts can help you understand Ceph user management.

Storage Cluster Users

A user of the Red Hat Ceph Storage cluster is either an individual or as an application. Creating users allows you to control who can access the storage cluster, its pools, and the data within those pools.

Ceph has the notion of a **type** of user. For the purposes of user management, the type will always be **client**. Ceph identifies users in period (.) delimited form consisting of the user type and the user ID. For example, **TYPE.ID**, **client.admin**, or **client.user1**. The reason for user typing is that Ceph Monitors, and

OSDs also use the Cephx protocol, but they are not clients. Distinguishing the user type helps to distinguish between client users and other users—streamlining access control, user monitoring and traceability.

Sometimes Ceph's user type may seem confusing, because the Ceph command line allows you to specify a user with or without the type, depending upon the command line usage. If you specify **--user** or **--id**, you can omit the type. So **client.user1** can be entered simply as **user1**. If you specify **--name** or **-n**, you must specify the type and name, such as **client.user1**. Red Hat recommends using the type and name as a best practice wherever possible.

> NOTE
>
> A Red Hat Ceph Storage cluster user is not the same as a Ceph Object Gateway user. The object gateway uses a Red Hat Ceph Storage cluster user to communicate between the gateway daemon and the storage cluster, but the gateway has its own user management functionality for its end users.

Authorization capabilities

Ceph uses the term "capabilities" (caps) to describe authorizing an authenticated user to exercise the functionality of the Ceph Monitors and OSDs. Capabilities can also restrict access to data within a pool or a namespace within a pool. A Ceph administrative user sets a user's capabilities when creating or updating a user. Capability syntax follows the form:

Syntax

```
DAEMON_TYPE 'allow CAPABILITY' [DAEMON_TYPE 'allow CAPABILITY']
```

- **Monitor Caps:** Monitor capabilities include **r**, **w**, **x**, **allow profile *CAP***, and **profile rbd**.

  #### Example

  ```
  mon 'allow rwx`
  mon 'allow profile osd'
  ```

- **OSD Caps:** OSD capabilities include **r**, **w**, **x**, **class-read**, **class-write**, **profile osd**, **profile rbd**, and **profile rbd-read-only**. Additionally, OSD capabilities also allow for pool and namespace settings. :

  ```
  osd 'allow CAPABILITY' [pool=POOL_NAME] [namespace=NAMESPACE_NAME]
  ```

> NOTE
>
> The Ceph Object Gateway daemon (**radosgw**) is a client of the Ceph storage cluster, so it isn't represented as a Ceph storage cluster daemon type.

The following entries describe each capability.

allow	Precedes access settings for a daemon.
r	Gives the user read access. Required with monitors to retrieve the CRUSH map.

w	Gives the user write access to objects.
x	Gives the user the capability to call class methods (that is, both read and write) and to conduct **auth** operations on monitors.
class-read	Gives the user the capability to call class read methods. Subset of **x**.
class-write	Gives the user the capability to call class write methods. Subset of **x**.
*	Gives the user read, write and execute permissions for a particular daemon or pool, and the ability to execute admin commands.
profile osd	Gives a user permissions to connect as an OSD to other OSDs or monitors. Conferred on OSDs to enable OSDs to handle replication heartbeat traffic and status reporting.
profile bootstrap-osd	Gives a user permissions to bootstrap an OSD, so that they have permissions to add keys when bootstrapping an OSD.
profile rbd	Gives a user read-write access to the Ceph Block Devices.
profile rbd-read-only	Gives a user read-only access to the Ceph Block Devices.

Pool

A pool defines a storage strategy for Ceph clients, and acts as a logical partition for that strategy.

In Ceph deployments, it is common to create a pool to support different types of use cases. For example, cloud volumes or images, object storage, hot storage, cold storage, and so on. When deploying Ceph as a back end for OpenStack, a typical deployment would have pools for volumes, images, backups and virtual machines, and users such as **client.glance**, **client.cinder**, and so on.

Namespace

Objects within a pool can be associated to a namespace—a logical group of objects within the pool. A user's access to a pool can be associated with a namespace such that reads and writes by the user take place only within the namespace. Objects written to a namespace within the pool can only be accessed by users who have access to the namespace.

NOTE

Currently, namespaces are only useful for applications written on top of **librados**. Ceph clients such as block device and object storage do not currently support this feature.

The rationale for namespaces is that pools can be a computationally expensive method of segregating data by use case, because each pool creates a set of placement groups that get mapped to OSDs. If multiple pools use the same CRUSH hierarchy and ruleset, OSD performance may degrade as load increases.

For example, a pool should have approximately 100 placement groups per OSD. So an exemplary cluster with 1000 OSDs would have 100,000 placement groups for one pool. Each pool mapped to the same

CRUSH hierarchy and ruleset would create another 100,000 placement groups in the exemplary cluster. By contrast, writing an object to a namespace simply associates the namespace to the object name with out the computational overhead of a separate pool. Rather than creating a separate pool for a user or set of users, you may use a namespace.

NOTE

Only available using **librados** at this time.

Additional Resources

- See the *Red Hat Ceph Storage Configuration Guide* for details on configuring the use of authentication.

5.3. MANAGING CEPH USERS

As a storage administrator, you can manage Ceph users by creating, modifying, deleting, and importing users. A Ceph client user can be either individuals or applications, which use Ceph clients to interact with the Red Hat Ceph Storage cluster daemons.

5.3.1. Prerequisites

- A running Red Hat Ceph Storage cluster.
- Access to a Ceph Monitor or Ceph client node.

5.3.2. Listing Ceph users

You can list the users in the storage cluster using the command-line interface.

Prerequisites

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

Procedure

1. To list the users in the storage cluster, execute the following:

```
[root@mon ~]# ceph auth list
```

Ceph will list out all users in the storage cluster. For example, in a two-node exemplary storage cluster, **ceph auth list** will output something that looks like this:

Example

```
installed auth entries:

osd.0
    key: AQCvCbtToC6MDhAATtuT70Sl+DymPCfDSsyV4w==
    caps: [mon] allow profile osd
    caps: [osd] allow *
```

```
osd.1
   key: AQC4CbtTCFJBChAAVq5spj0ff4eHZICxIOVZeA==
   caps: [mon] allow profile osd
   caps: [osd] allow *
client.admin
   key: AQBHCbtT6APDHhAA5W00cBchwkQjh3dkKsyPjw==
   caps: [mds] allow
   caps: [mon] allow *
   caps: [osd] allow *
client.bootstrap-mds
   key: AQBICbtTOK9uGBAAdbe5zcIGHZL3T/u2g6EBww==
   caps: [mon] allow profile bootstrap-mds
client.bootstrap-osd
   key: AQBHCbtT4GxqORAADE5u7RkpCN/oo4e5W0uBtw==
   caps: [mon] allow profile bootstrap-osd
```

NOTE

The **TYPE.ID** notation for users applies such that **osd.0** is a user of type **osd** and its ID is **0**, **client.admin** is a user of type **client** and its ID is **admin**, that is, the default **client.admin** user. Note also that each entry has a **key: *VALUE*** entry, and one or more **caps:** entries.

You may use the **-o *FILE_NAME*** option with **ceph auth list** to save the output to a file.

5.3.3. Display Ceph user information

You can display a Ceph's user information using the command-line interface.

Prerequisites

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

Procedure

1. To retrieve a specific user, key and capabilities, execute the following:

   ```
   ceph auth export TYPE.ID
   ```

   Example

   ```
   [root@mon ~]# ceph auth get client.admin
   ```

2. You can also use the **-o *FILE_NAME*** option with **ceph auth get** to save the output to a file. Developers can also execute the following:

   ```
   ceph auth export TYPE.ID
   ```

   Example

```
[root@mon ~]# ceph auth export client.admin
```

The **auth export** command is identical to **auth get**, but also prints out the internal **auid**, which isn’t relevant to end users.

5.3.4. Add a new Ceph user

Adding a user creates a username, that is, **TYPE.ID**, a secret key and any capabilities included in the command you use to create the user.

A user’s key enables the user to authenticate with the Ceph storage cluster. The user’s capabilities authorize the user to read, write, or execute on Ceph monitors (**mon**), Ceph OSDs (**osd**) or Ceph Metadata Servers (**mds**).

There are a few ways to add a user:

- **ceph auth add**: This command is the canonical way to add a user. It will create the user, generate a key and add any specified capabilities.
- **ceph auth get-or-create**: This command is often the most convenient way to create a user, because it returns a keyfile format with the user name (in brackets) and the key. If the user already exists, this command simply returns the user name and key in the keyfile format. You may use the **-o *FILE_NAME*** option to save the output to a file.
- **ceph auth get-or-create-key**: This command is a convenient way to create a user and return the user’s key only. This is useful for clients that need the key only, for example, **libvirt**. If the user already exists, this command simply returns the key. You may use the **-o *FILE_NAME*** option to save the output to a file.

When creating client users, you may create a user with no capabilities. A user with no capabilities is useless beyond mere authentication, because the client cannot retrieve the cluster map from the monitor. However, you can create a user with no capabilities if you wish to defer adding capabilities later using the **ceph auth caps** command.

A typical user has at least read capabilities on the Ceph monitor and read and write capability on Ceph OSDs. Additionally, a user’s OSD permissions are often restricted to accessing a particular pool. :

```
[root@mon ~]# ceph auth add client.john mon 'allow r' osd 'allow rw pool=liverpool'
[root@mon ~]# ceph auth get-or-create client.paul mon 'allow r' osd 'allow rw pool=liverpool'
[root@mon ~]# ceph auth get-or-create client.george mon 'allow r' osd 'allow rw pool=liverpool' -o
george.keyring
[root@mon ~]# ceph auth get-or-create-key client.ringo mon 'allow r' osd 'allow rw pool=liverpool' -o
ringo.key
```

IMPORTANT

If you provide a user with capabilities to OSDs, but you DO NOT restrict access to particular pools, the user will have access to ALL pools in the cluster!

5.3.5. Modifying a Ceph User

The **ceph auth caps** command allows you to specify a user and change the user’s capabilities. Setting new capabilities will overwrite current capabilities. Therefore, view the current capabilities first and include them when you add new capabilities.

Prerequisites

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

Procedure

1. View current capabilities:

   ```
   ceph auth get USERTYPE.USERID
   ```

   #### Example

   ```
   [root@mon ~]# ceph auth get client.john
   exported keyring for client.john
   [client.john]
    key = AQAHjy1gkxhIMBAAxsaoFNuxlUhr/zKsmnAZOA==
    caps mon = "allow r"
    caps osd = "allow rw pool=liverpool"
   ```

2. To add capabilities, use the form:

   ```
   ceph auth caps USERTYPE.USERID DAEMON 'allow [r|w|x|*|...] [pool=POOL_NAME]
   [namespace=NAMESPACE_NAME]'
   ```

   #### Example

   ```
   [root@mon ~]# ceph auth caps client.john mon 'allow r' osd 'allow rwx pool=liverpool'
   ```

   In the example, execute capabilities on the OSDs have been added.

3. Verify the added capabilities:

   ```
   ceph auth get _USERTYPE_._USERID_
   ```

   #### Example

   ```
   [root@mon ~]# ceph auth get client.john
   exported keyring for client.john
   [client.john]
    key = AQAHjy1gkxhIMBAAxsaoFNuxlUhr/zKsmnAZOA==
    caps mon = "allow r"
    caps osd = "allow rwx pool=liverpool"
   ```

   In the example, execute capabilities on the OSDs can be seen.

4. To remove a capability, set all the current capabilities except the ones you want to remove.

   ```
   ceph auth caps USERTYPE.USERID DAEMON 'allow [r|w|x|*|...] [pool=POOL_NAME]
   [namespace=NAMESPACE_NAME]'
   ```

   #### Example

```
[root@mon ~]# ceph auth caps client.john mon 'allow r' osd 'allow rw pool=liverpool'
```

In the example, execute capabilities on the OSDs were not included and thus will be removed.

5. Verify the removed capabilities:

```
ceph auth get _USERTYPE_._USERID_
```

Example

```
[root@mon ~]# ceph auth get client.john
exported keyring for client.john
[client.john]
 key = AQAHjy1gkxhIMBAAxsaoFNuxlUhr/zKsmnAZOA==
 caps mon = "allow r"
 caps osd = "allow rw pool=liverpool"
```

In the example, execute capabilities on the OSDs are no longer listed.

Additional Resources

- See *Authorization capabilities* for additional details on capabilities.

5.3.6. Deleting a Ceph user

You can delete a user from the Ceph storage cluster using the command-line interface.

Prerequisites

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

Procedure

1. To delete a user, use **ceph auth del**:

```
[root@mon ~]# ceph auth del TYPE.ID
```

Where ***TYPE*** is one of **client**, **osd**, **mon**, or **mds**, and ***ID*** is the user name or ID of the daemon.

5.3.7. Print a Ceph user key

You can display a Ceph user's key information using the command-line interface.

Prerequisites

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

Procedure

1. To print a user's authentication key to standard output, execute the following:

   ```
   ceph auth print-key TYPE.ID
   ```

   Where ***TYPE*** is one of **client**, **osd**, **mon**, or **mds**, and ***ID*** is the user name or ID of the daemon.

2. Printing a user's key is useful when you need to populate client software with a user's key, for example, **libvirt**.

   ```
   mount -t ceph HOSTNAME:/MOUNT_POINT -o name=client.user,secret=ceph auth print-key
   client.user
   ```

5.3.8. Import Ceph user

You can import a Ceph user using the command-line interface.

Prerequisites

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

Procedure

1. To import one or more users, use **ceph auth import** and specify a keyring:

   ```
   ceph auth import -i /PATH/TO/KEYRING
   ```

   #### Example

   ```
   [root@mon ~]# ceph auth import -i /etc/ceph/ceph.keyring
   ```

NOTE

The Ceph storage cluster will add new users, their keys and their capabilities and will update existing users, their keys and their capabilities.

5.4. MANAGING CEPH KEYRINGS

As a storage administrator, managing Ceph user keys is important for accessing the Red Hat Ceph Storage cluster. You can create keyrings, add users to keyrings, and modifying users with keyrings.

5.4.1. Prerequisites

- A running Red Hat Ceph Storage cluster.
- Access to a Ceph Monitor or Ceph client node.

5.4.2. Creating a keyring

You need to provide user keys to the Ceph clients so that the Ceph client can retrieve the key for the specified user and authenticate with the Ceph Storage Cluster. Ceph Clients access keyrings to lookup a user name and retrieve the user's key.

The **ceph-authtool** utility allows you to create a keyring.

Prerequisites

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

Procedure

1. To create an empty keyring, use **--create-keyring** or **-C**.

   #### Example

   ```
   [root@mon ~]# ceph-authtool --create-keyring /path/to/keyring
   ```

   When creating a keyring with multiple users, we recommend using the cluster name. For example, ***CLUSTER_NAME*.keyring`** for the keyring file name and saving it in the **/etc/ceph/** directory so that the **keyring** configuration default setting will pick up the filename without requiring you to specify it in the local copy of the Ceph configuration file.

2. Create **ceph.keyring** by executing the following:

   ```
   [root@mon ~]# ceph-authtool -C /etc/ceph/ceph.keyring
   ```

When creating a keyring with a single user, we recommend using the cluster name, the user type and the user name and saving it in the **/etc/ceph/** directory. For example, **ceph.client.admin.keyring** for the **client.admin** user.

To create a keyring in **/etc/ceph/**, you must do so as **root**. This means the file will have **rw** permissions for the **root** user only, which is appropriate when the keyring contains administrator keys. However, if you intend to use the keyring for a particular user or group of users, ensure that you execute **chown** or **chmod** to establish appropriate keyring ownership and access.

5.4.3. Adding a user to the keyring

When you add a user to the Ceph storage cluster, you can use the **get** procedure to retrieve a user, key and capabilities, then save the user to a keyring file. When you only want to use one user per keyring, the *Display Ceph user information* procedure with the **-o** option will save the output in the keyring file format.

Prerequisites

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

Procedure

1. To create a keyring for the **client.admin** user, execute the following:

```
[root@mon ~]# ceph auth get client.admin -o /etc/ceph/ceph.client.admin.keyring
```

Notice that we use the recommended file format for an individual user.

2. When you want to import users to a keyring, you can use **ceph-authtool** to specify the destination keyring and the source keyring.

```
[root@mon ~]# ceph-authtool /etc/ceph/ceph.keyring --import-keyring
/etc/ceph/ceph.client.admin.keyring
```

5.4.4. Creating a Ceph user with a keyring

Ceph provides the ability to create a user directly in the Red Hat Ceph Storage cluster. However, you can also create a user, keys and capabilities directly on a Ceph client keyring.

Prerequisites

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

Procedure

1. Import a user into the keyring:

   Example

```
[root@mon ~]# ceph-authtool -n client.ringo --cap osd 'allow rwx' --cap mon 'allow rwx'
/etc/ceph/ceph.keyring
```

2. Create a keyring and add a new user to the keyring simultaneously:

   Example:

```
[root@mon ~]# ceph-authtool -C /etc/ceph/ceph.keyring -n client.ringo --cap osd 'allow rwx' -
-cap mon 'allow rwx' --gen-key
```

   In the foregoing scenarios, the new user **client.ringo** is only in the keyring.

3. To add the new user to the Ceph storage cluster:

```
[root@mon ~]# ceph auth add client.ringo -i /etc/ceph/ceph.keyring
```

Additional Resources

- See Ceph user management background for additional details on capabilities.

5.4.5. Modifying a Ceph user with a keyring

You can modify a Ceph user and their keyring using the command-line interface.

Prerequisites

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

Procedure

1. To modify the capabilities of a user record in a keyring, specify the keyring, and the user followed by the capabilities, for example:

```
[root@mon ~]# ceph-authtool /etc/ceph/ceph.keyring -n client.ringo --cap osd 'allow rwx' --cap mon 'allow rwx'
```

1. To update the user to the Red Hat Ceph Storage cluster, you must update the user in the keyring to the user entry in the Red Hat Ceph Storage cluster:

```
[root@mon ~]# ceph auth import -i /etc/ceph/ceph.keyring
```

You may also modify user capabilities directly in the storage cluster, store the results to a keyring file; then, import the keyring into the main **ceph.keyring** file.

Additional Resources

- See *Import user* for details on updating a Red Hat Ceph Storage cluster user from a keyring.

5.4.6. Command Line usage for Ceph users

Ceph supports the following usage for user name and secret:

--id | **--user**

Description

Ceph identifies users with a type and an ID. For example, **TYPE.ID** or **client.admin**, **client.user1**. The **id**, **name** and **-n** options enable you to specify the ID portion of the user name. For example, **admin**, **user1**, or **foo**. You can specify the user with the **--id** and omit the type. For example, to specify user **client.foo** enter the following:

```
[root@mon ~]# ceph --id foo --keyring /path/to/keyring health
[root@mon ~]# ceph --user foo --keyring /path/to/keyring health
```

--name | **-n**

Description

Ceph identifies users with a type and an ID. For example, **TYPE.ID** or **client.admin**, **client.user1**. The **--name** and **-n** options enables you to specify the fully qualified user name. You must specify the user type (typically **client**) with the user ID. For example:

```
[root@mon ~]# ceph --name client.foo --keyring /path/to/keyring health
[root@mon ~]# ceph -n client.foo --keyring /path/to/keyring health
```

--keyring

Description

The path to the keyring containing one or more user name and secret. The **--secret** option provides the same functionality, but it does not work with Ceph RADOS Gateway, which uses **--secret** for another purpose. You may retrieve a keyring with **ceph auth get-or-create** and store it locally. This is a preferred approach, because you can switch user names without switching the keyring path. For example:

```
[root@mon ~]# rbd map foo --pool rbd myimage --id client.foo --keyring /path/to/keyring
```

5.4.7. Ceph user management limitations

The **cephx** protocol authenticates Ceph clients and servers to each other. It is not intended to handle authentication of human users or application programs run on their behalf. If that effect is required to handle the access control needs, you must have another mechanism, which is likely to be specific to the front end used to access the Ceph object store. This other mechanism has the role of ensuring that only acceptable users and programs are able to run on the machine that Ceph will permit to access its object store.

The keys used to authenticate Ceph clients and servers are typically stored in a plain text file with appropriate permissions in a trusted host.

IMPORTANT

Storing keys in plaintext files has security shortcomings, but they are difficult to avoid, given the basic authentication methods Ceph uses in the background. Those setting up Ceph systems should be aware of these shortcomings.

In particular, arbitrary user machines, especially portable machines, should not be configured to interact directly with Ceph, since that mode of use would require the storage of a plaintext authentication key on an insecure machine. Anyone who stole that machine or obtained surreptitious access to it could obtain the key that will allow them to authenticate their own machines to Ceph.

Rather than permitting potentially insecure machines to access a Ceph object store directly, users should be required to sign in to a trusted machine in the environment using a method that provides sufficient security for the purposes. That trusted machine will store the plaintext Ceph keys for the human users. A future version of Ceph may address these particular authentication issues more fully.

At the moment, none of the Ceph authentication protocols provide secrecy for messages in transit. Thus, an eavesdropper on the wire can hear and understand all data sent between clients and servers in Ceph, even if he cannot create or alter them. Those storing sensitive data in Ceph should consider encrypting their data before providing it to the Ceph system.

For example, Ceph Object Gateway provides S3 API Server-side Encryption, which encrypts unencrypted data received from a Ceph Object Gateway client before storing it in the Ceph Storage cluster and similarly decrypts data retrieved from the Ceph Storage cluster before sending it back to the client. To ensure encryption in transit between the client and the Ceph Object Gateway, the Ceph Object Gateway should be configured to use SSL.

CHAPTER 6. THE CEPH-VOLUME UTILITY

As a storage administrator, you can prepare, create, and activate Ceph OSDs using the **ceph-volume** utility. The **ceph-volume** utility is a single purpose command-line tool to deploy logical volumes as OSDs. It uses a plugin-type framework to deploying OSDs with different device technologies. The **ceph-volume** utility follows a similar workflow of the **ceph-disk** utility for deploying OSDs, with a predictable, and robust way of preparing, activating, and starting OSDs. Currently, the **ceph-volume** utility only supports the **lvm** plugin, with the plan to support others technologies in the future.

IMPORTANT

The **ceph-disk** command is deprecated.

6.1. PREREQUISITES

- A running Red Hat Ceph Storage cluster.

6.2. CEPH VOLUME LVM PLUGIN

By making use of LVM tags, the **lvm** sub-command is able to store and re-discover by querying devices associated with OSDs so they can be activated. This includes support for lvm-based technologies like **dm-cache** as well.

When using **ceph-volume**, the use of **dm-cache** is transparent, and treats **dm-cache** like a logical volume. The performance gains and losses when using **dm-cache** will depend on the specific workload. Generally, random and sequential reads will see an increase in performance at smaller block sizes. While random and sequential writes will see a decrease in performance at larger block sizes.

To use the LVM plugin, add **lvm** as a subcommand to the **ceph-volume** command:

```
[root@osd ~]# ceph-volume lvm
```

There are three subcommands to the **lvm** subcommand, as follows:

- **prepare**
- **activate**
- **create**
- **batch**

NOTE

Using the **create** subcommand combines the **prepare** and **activate** subcommands into one subcommand.

Additional Resources

- See the **create** subcommand section for more details.

6.3. WHY DOES CEPH-VOLUME REPLACE CEPH-DISK?

Previous versions of Red Hat Ceph Storage used the **ceph-disk** utility to prepare, activate, and create OSDs. Starting with Red Hat Ceph Storage 4, **ceph-disk** is replaced by the **ceph-volume** utility that aims to be a single purpose command-line tool to deploy logical volumes as OSDs, while maintaining a similar API to **ceph-disk** when preparing, activating, and creating OSDs.

How does ceph-volume work?

The **ceph-volume** is a modular tool that currently supports two ways of provisioning hardware devices, legacy **ceph-disk** devices and LVM (Logical Volume Manager) devices. The **ceph-volume lvm** command uses the LVM tags to store information about devices specific to Ceph and its relationship with OSDs. It uses these tags to later re-discover and query devices associated with OSDS so that it can activate them. It supports technologies based on LVM and **dm-cache** as well.

The **ceph-volume** utility uses **dm-cache** transparently and treats it as a logical volume. You might consider the performance gains and losses when using **dm-cache**, depending on the specific workload you are handling. Generally, the performance of random and sequential read operations increases at smaller block sizes; while the performance of random and sequential write operations decreases at larger block sizes. Using **ceph-volume** does not introduce any significant performance penalties.

IMPORTANT

The **ceph-disk** utility is deprecated.

NOTE

The **ceph-volume simple** command can handle legacy **ceph-disk** devices, if these devices are still in use.

How does ceph-disk work?

The **ceph-disk** utility was required to support many different types of init systems, such as **upstart** or **sysvinit**, while being able to discover devices. For this reason, **ceph-disk** concentrates only on GUID Partition Table (GPT) partitions. Specifically on GPT GUIDs that label devices in a unique way to answer questions like:

- Is this device a **journal**?
- Is this device an encrypted data partition?
- Was the device left partially prepared?

To solve these questions, **ceph-disk** uses UDEV rules to match the GUIDs.

What are disadvantages of using ceph-disk?

Using the UDEV rules to call **ceph-disk** can lead to a back-and-forth between the **ceph-disk systemd** unit and the **ceph-disk** executable. The process is very unreliable and time consuming and can cause OSDs to not come up at all during the boot process of a node. Moreover, it is hard to debug, or even replicate these problems given the asynchronous behavior of UDEV.

Because **ceph-disk** works with GPT partitions exclusively, it cannot support other technologies, such as Logical Volume Manager (LVM) volumes, or similar device mapper devices.

To ensure the GPT partitions work correctly with the device discovery workflow, **ceph-disk** requires a large number of special flags to be used. In addition, these partitions require devices to be exclusively owned by Ceph.

6.4. PREPARING CEPH OSDS USING CEPH-VOLUME

The **prepare** subcommand prepares an OSD back-end object store and consumes logical volumes (LV) for both the OSD data and journal. It does not modify the logical volumes, except for adding some extra metadata tags using LVM. These tags make volumes easier to discover, and they also identify the volumes as part of the Ceph Storage Cluster and the roles of those volumes in the storage cluster.

The BlueStore OSD backend supports the following configurations:

- A block device, a **block.wal** device, and a **block.db** device
- A block device and a **block.wal** device
- A block device and a **block.db** device
- A single block device

The **prepare** subcommand accepts a whole device or partition, or a logical volume for **block**.

Prerequisites

- Root-level access to the OSD nodes.
- Optionally, create logical volumes. If you provide a path to a physical device, the subcommand turns the device into a logical volume. This approach is simpler, but you cannot configure or change the way the logical volume is created.

Procedure

1. Prepare the LVM volumes:

   Syntax

   ```
   ceph-volume lvm prepare --bluestore --data VOLUME_GROUP/LOGICAL_VOLUME
   ```

   Example

   ```
   [root@osd ~]# ceph-volume lvm prepare --bluestore --data example_vg/data_lv
   ```

   a. Optionally, if you want to use a separate device for RocksDB, specify the **--block.db** and **--block.wal** options:

   Syntax

   ```
   ceph-volume lvm prepare --bluestore --block.db --block.wal --data
   VOLUME_GROUP/LOGICAL_VOLUME
   ```

   Example

   ```
   [root@osd ~]# ceph-volume lvm prepare --bluestore --block.db --block.wal --data
   example_vg/data_lv
   ```

   b. Optionally, to encrypt data, use the **--dmcrypt** flag:

Syntax

```
ceph-volume lvm prepare --bluestore --dmcrypt --data
VOLUME_GROUP/LOGICAL_VOLUME
```

Example

```
[root@osd ~]# ceph-volume lvm prepare --bluestore --dmcrypt --data
example_vg/data_lv
```

Additional Resources

- See the *Activating Ceph OSDs using `ceph-volume`* section in the *Red Hat Ceph Storage Administration Guide* for more details.
- See the *Creating Ceph OSDs using `ceph-volume`* section in the *Red Hat Ceph Storage Administration Guide* for more details.

6.5. ACTIVATING CEPH OSDS USING CEPH-VOLUME

The activation process enables a **systemd** unit at boot time, which allows the correct OSD identifier and its UUID to be enabled and mounted.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the Ceph OSD node.
- Ceph OSDs prepared by the **ceph-volume** utility.

Procedure

1. Get the OSD ID and UUID from an OSD node:

   ```
   [root@osd ~]# ceph-volume lvm list
   ```

2. Activate the OSD:

   ### Syntax

   ```
   ceph-volume lvm activate --bluestore OSD_ID OSD_UUID
   ```

   ### Example

   ```
   [root@osd ~]# ceph-volume lvm activate --bluestore 0 0263644D-0BF1-4D6D-BC34-
   28BD98AE3BC8
   ```

   To activate all OSDs that are prepared for activation, use the **--all** option:

   ### Example

```
[root@osd ~]# ceph-volume lvm activate --all
```

Additional Resources

- See the *Preparing Ceph OSDs using `ceph-volume`* section in the *Red Hat Ceph Storage Administration Guide* for more details.
- See the *Creating Ceph OSDs using `ceph-volume`* section in the *Red Hat Ceph Storage Administration Guide* for more details.

6.6. CREATING CEPH OSDS USING CEPH-VOLUME

The **create** subcommand calls the **prepare** subcommand, and then calls the **activate** subcommand.

Prerequisites

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

> **NOTE**
>
> If you prefer to have more control over the creation process, you can use the **prepare** and **activate** subcommands separately to create the OSD, instead of using **create**. You can use the two subcommands to gradually introduce new OSDs into a storage cluster, while avoiding having to rebalance large amounts of data. Both approaches work the same way, except that using the **create** subcommand causes the OSD to become *up* and *in* immediately after completion.

Procedure

1. To create a new OSD:

   Syntax

   ```
   ceph-volume lvm create --bluestore --data VOLUME_GROUP/LOGICAL_VOLUME
   ```

   Example

   ```
   [root@osd ~]# ceph-volume lvm create --bluestore --data example_vg/data_lv
   ```

Additional Resources

- See the *Preparing Ceph OSDs using `ceph-volume`* section in the *Red Hat Ceph Storage Administration Guide* for more details.
- See the *Activating Ceph OSDs using `ceph-volume`* section in the *Red Hat Ceph Storage Administration Guide* for more details.

6.7. USING BATCH MODE WITH CEPH-VOLUME

The **batch** subcommand automates the creation of multiple OSDs when single devices are provided.

The **ceph-volume** command decides the best method to use to create the OSDs, based on drive type. Ceph OSD optimization depends on the available devices:

- If all devices are traditional hard drives, **batch** creates one OSD per device.
- If all devices are solid state drives, **batch** creates two OSDs per device.
- If there is a mix of traditional hard drives and solid state drives, **batch** uses the traditional hard drives for data, and creates the largest possible journal (**block.db**) on the solid state drive.

NOTE

The **batch** subcommand does not support the creation of a separate logical volume for the write-ahead-log (**block.wal**) device.

Prerequisites

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

Procedure

1. To create OSDs on several drives:

   #### Syntax

   ```
   ceph-volume lvm batch --bluestore PATH_TO_DEVICE [PATH_TO_DEVICE]
   ```

   #### Example

   ```
   [root@osd ~]# ceph-volume lvm batch --bluestore /dev/sda /dev/sdb /dev/nvme0n1
   ```

Additional Resources

- See the *Creating Ceph OSDs using `ceph-volume`* section in the *Red Hat Ceph Storage Administration Guide* for more details.

CHAPTER 7. CEPH PERFORMANCE BENCHMARK

As a storage administrator, you can benchmark performance of the Red Hat Ceph Storage cluster. The purpose of this section is to give Ceph administrators a basic understanding of Ceph's native benchmarking tools. These tools will provide some insight into how the Ceph storage cluster is performing. This is not the definitive guide to Ceph performance benchmarking, nor is it a guide on how to tune Ceph accordingly.

7.1. PREREQUISITES

- A running Red Hat Ceph Storage cluster.

7.2. PERFORMANCE BASELINE

The OSD, including the journal, disks and the network throughput should each have a performance baseline to compare against. You can identify potential tuning opportunities by comparing the baseline performance data with the data from Ceph's native tools. Red Hat Enterprise Linux has many built-in tools, along with a plethora of open source community tools, available to help accomplish these tasks.

Additional Resources

- For more details about some of the available tools, see this Knowledgebase article.

7.3. BENCHMARKING CEPH PERFORMANCE

Ceph includes the **rados bench** command to do performance benchmarking on a RADOS storage cluster. The command will execute a write test and two types of read tests. The **--no-cleanup** option is important to use when testing both read and write performance. By default the **rados bench** command will delete the objects it has written to the storage pool. Leaving behind these objects allows the two read tests to measure sequential and random read performance.

NOTE

Before running these performance tests, drop all the file system caches by running the following:

```
[root@mon~ ]# echo 3 | sudo tee /proc/sys/vm/drop_caches && sudo sync
```

Prerequisites

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

Procedure

1. Create a new storage pool:

```
[root@osd~ ]# ceph osd pool create testbench 100 100
```

2. Execute a write test for 10 seconds to the newly created storage pool:

```
[root@osd~ ]# rados bench -p testbench 10 write --no-cleanup
```

Example Output

```
Maintaining 16 concurrent writes of 4194304 bytes for up to 10 seconds or 0 objects
 Object prefix: benchmark_data_cephn1.home.network_10510
   sec Cur ops   started  finished  avg MB/s  cur MB/s  last lat   avg lat
     0       0         0         0         0         0         -         0
     1      16        16         0         0         0         -         0
     2      16        16         0         0         0         -         0
     3      16        16         0         0         0         -         0
     4      16        17         1  0.998879         1   3.19824   3.19824
     5      16        18         2   1.59849         4   4.56163   3.87993
     6      16        18         2   1.33222         0         -   3.87993
     7      16        19         3   1.71239         2   6.90712     4.889
     8      16        25         9   4.49551        24   7.75362   6.71216
     9      16        25         9   3.99636         0         -   6.71216
    10      16        27        11   4.39632         4   9.65085   7.18999
    11      16        27        11   3.99685         0         -   7.18999
    12      16        27        11   3.66397         0         -   7.18999
    13      16        28        12   3.68975   1.33333   12.8124   7.65853
    14      16        28        12   3.42617         0         -   7.65853
    15      16        28        12   3.19785         0         -   7.65853
    16      11        28        17   4.24726   6.66667   12.5302   9.27548
    17      11        28        17   3.99751         0         -   9.27548
    18      11        28        17   3.77546         0         -   9.27548
    19      11        28        17   3.57683         0         -   9.27548
 Total time run:         19.505620
Total writes made:      28
Write size:             4194304
Bandwidth (MB/sec):     5.742

Stddev Bandwidth:       5.4617
Max bandwidth (MB/sec): 24
Min bandwidth (MB/sec): 0
Average Latency:        10.4064
Stddev Latency:         3.80038
Max latency:            19.503
Min latency:            3.19824
```

3. Execute a sequential read test for 10 seconds to the storage pool:

```
[root@osd~ ]## rados bench -p testbench 10 seq
```

Example Output

```
sec Cur ops   started  finished  avg MB/s  cur MB/s  last lat   avg lat
  0       0         0         0         0         0         -         0
Total time run:        0.804869
Total reads made:      28
Read size:             4194304
Bandwidth (MB/sec):    139.153
```

```
Average Latency:       0.420841
Max latency:           0.706133
Min latency:           0.0816332
```

4. Execute a random read test for 10 seconds to the storage pool:

```
[root@osd ~]# rados bench -p testbench 10 rand
```

Example Output

```
sec Cur ops   started  finished  avg MB/s  cur MB/s  last lat   avg lat
  0       0         0         0         0         0         -         0
  1      16        46        30   119.801       120  0.440184  0.388125
  2      16        81        65   129.408       140  0.577359  0.417461
  3      16       120       104   138.175       156  0.597435  0.409318
  4      15       157       142   141.485       152  0.683111  0.419964
  5      16       206       190   151.553       192  0.310578  0.408343
  6      16       253       237   157.608       188 0.0745175  0.387207
  7      16       287       271   154.412       136  0.792774   0.39043
  8      16       325       309   154.044       152  0.314254   0.39876
  9      16       362       346   153.245       148  0.355576  0.406032
 10      16       405       389   155.092       172   0.64734  0.398372
Total time run:       10.302229
Total reads made:     405
Read size:            4194304
Bandwidth (MB/sec):   157.248

Average Latency:       0.405976
Max latency:           1.00869
Min latency:           0.0378431
```

5. To increase the number of concurrent reads and writes, use the **-t** option, which the default is 16 threads. Also, the **-b** parameter can adjust the size of the object being written. The default object size is 4 MB. A safe maximum object size is 16 MB. Red Hat recommends running multiple copies of these benchmark tests to different pools. Doing this shows the changes in performance from multiple clients.
   Add the **--run-name <label>** option to control the names of the objects that get written during the benchmark test. Multiple **rados bench** commands may be ran simultaneously by changing the **--run-name** label for each running command instance. This prevents potential I/O errors that can occur when multiple clients are trying to access the same object and allows for different clients to access different objects. The **--run-name** option is also useful when trying to simulate a real world workload. For example:

```
[root@osd ~]# rados bench -p testbench 10 write -t 4 --run-name client1
```

Example Output

```
Maintaining 4 concurrent writes of 4194304 bytes for up to 10 seconds or 0 objects
 Object prefix: benchmark_data_node1_12631
   sec Cur ops   started  finished  avg MB/s  cur MB/s  last lat   avg lat
     0       0         0         0         0         0         -         0
     1       4         4         0         0         0         -         0
     2       4         6         2   3.99099         4   1.94755   1.93361
     3       4         8         4   5.32498         8     2.978   2.44034
```

```
    4       4         8         4   3.99504         0           -     2.44034
    5       4        10         6   4.79504         4     2.92419      2.4629
    6       3        10         7   4.64471         4     3.02498      2.5432
    7       4        12         8   4.55287         4     3.12204     2.61555
    8       4        14        10    4.9821         8     2.55901     2.68396
    9       4        16        12   5.31621         8     2.68769     2.68081
   10       4        17        13   5.18488         4     2.11937     2.63763
   11       4        17        13   4.71431         0           -     2.63763
   12       4        18        14   4.65486         2      2.4836     2.62662
   13       4        18        14   4.29757         0           -     2.62662
Total time run:         13.123548
Total writes made:      18
Write size:             4194304
Bandwidth (MB/sec):     5.486

Stddev Bandwidth:       3.0991
Max bandwidth (MB/sec): 8
Min bandwidth (MB/sec): 0
Average Latency:        2.91578
Stddev Latency:         0.956993
Max latency:            5.72685
Min latency:            1.91967
```

6. Remove the data created by the **rados bench** command:

```
[root@osd ~]# rados -p testbench cleanup
```

7.4. BENCHMARKING CEPH BLOCK PERFORMANCE

Ceph includes the **rbd bench-write** command to test sequential writes to the block device measuring throughput and latency. The default byte size is 4096, the default number of I/O threads is 16, and the default total number of bytes to write is 1 GB. These defaults can be modified by the **--io-size**, **--io-threads** and **--io-total** options respectively.

Prerequisites

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

Procedure

1. Load the **rbd** kernel module, if not already loaded:

```
[root@mon ~]# modprobe rbd
```

2. Create a 1 GB **rbd** image file in the **testbench** pool:

```
[root@mon ~]# rbd create image01 --size 1024 --pool testbench
```

3. Map the image file to a device file:

```
[root@mon ~]# rbd map image01 --pool testbench --name client.admin
```

4. Create an **ext4** file system on the block device:

```
[root@mon ~]# mkfs.ext4 /dev/rbd/testbench/image01
```

5. Create a new directory:

```
[root@mon ~]# mkdir /mnt/ceph-block-device
```

6. Mount the block device under **/mnt/ceph-block-device/**:

```
[root@mon ~]# mount /dev/rbd/testbench/image01 /mnt/ceph-block-device
```

7. Execute the write performance test against the block device

```
[root@mon ~]# rbd bench --io-type write image01 --pool=testbench
```

Example

```
bench-write  io_size 4096 io_threads 16 bytes 1073741824 pattern seq
  SEC       OPS   OPS/SEC   BYTES/SEC
    2     11127   5479.59  22444382.79
    3     11692   3901.91  15982220.33
    4     12372   2953.34  12096895.42
    5     12580   2300.05  9421008.60
    6     13141   2101.80  8608975.15
    7     13195    356.07  1458459.94
    8     13820    390.35  1598876.60
    9     14124    325.46  1333066.62
    ..
```

Additional Resources

- See the *Block Device Commands* section in the *Red Hat Ceph Storage Block Device Guide* for more information on the **rbd** command.

CHAPTER 8. CEPH PERFORMANCE COUNTERS

As a storage administrator, you can gather performance metrics of the Red Hat Ceph Storage cluster. The Ceph performance counters are a collection of internal infrastructure metrics. The collection, aggregation, and graphing of this metric data can be done by an assortment of tools and can be useful for performance analytics.

8.1. PREREQUISITES

- A running Red Hat Ceph Storage cluster.

8.2. ACCESS TO CEPH PERFORMANCE COUNTERS

The performance counters are available through a socket interface for the Ceph Monitors and the OSDs. The socket file for each respective daemon is located under **/var/run/ceph**, by default. The performance counters are grouped together into collection names. These collections names represent a subsystem or an instance of a subsystem.

Here is the full list of the Monitor and the OSD collection name categories with a brief description for each :

Monitor Collection Name Categories

- Cluster Metrics - Displays information about the storage cluster: Monitors, OSDs, Pools, and PGs
- Level Database Metrics - Displays information about the back-end **KeyValueStore** database
- Monitor Metrics - Displays general monitor information
- Paxos Metrics - Displays information on cluster quorum management
- Throttle Metrics - Displays the statistics on how the monitor is throttling

OSD Collection Name Categories

- Write Back Throttle Metrics - Displays the statistics on how the write back throttle is tracking unflushed IO
- Level Database Metrics - Displays information about the back-end **KeyValueStore** database
- Objecter Metrics - Displays information on various object-based operations
- Read and Write Operations Metrics - Displays information on various read and write operations
- Recovery State Metrics - Displays - Displays latencies on various recovery states
- OSD Throttle Metrics - Display the statistics on how the OSD is throttling

RADOS Gateway Collection Name Categories

- Object Gateway Client Metrics - Displays statistics on GET and PUT requests
- Objecter Metrics - Displays information on various object-based operations

- Object Gateway Throttle Metrics - Display the statistics on how the OSD is throttling

8.3. DISPLAY THE CEPH PERFORMANCE COUNTERS

The **ceph daemon .. perf schema** command outputs the available metrics. Each metric has an associated bit field value type.

Prerequisites

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

Procedure

1. To view the metric's schema:

```
ceph daemon DAEMON_NAME perf schema
```

NOTE

You must run the **ceph daemon** command from the node running the daemon.

2. Executing **ceph daemon .. perf schema** command from the Monitor node:

```
[root@mon ~]# ceph daemon mon.`hostname -s` perf schema
```

Example

```
{
    "cluster": {
        "num_mon": {
            "type": 2
        },
        "num_mon_quorum": {
            "type": 2
        },
        "num_osd": {
            "type": 2
        },
        "num_osd_up": {
            "type": 2
        },
        "num_osd_in": {
            "type": 2
        },
...
```

3. Executing the **ceph daemon .. perf schema** command from the OSD node:

```
[root@mon ~]# ceph daemon osd.0 perf schema
```

Example

```
...
"filestore": {
        "journal_queue_max_ops": {
            "type": 2
        },
        "journal_queue_ops": {
            "type": 2
        },
        "journal_ops": {
            "type": 10
        },
        "journal_queue_max_bytes": {
            "type": 2
        },
        "journal_queue_bytes": {
            "type": 2
        },
        "journal_bytes": {
            "type": 10
        },
        "journal_latency": {
            "type": 5
        },
...
```

Table 8.1. The bit field value definitions

Bit	Meaning
1	Floating point value
2	Unsigned 64-bit integer value
4	Average (Sum + Count)
8	Counter

Each value will have bit 1 or 2 set to indicate the type, either a floating point or an integer value. When bit 4 is set, there will be two values to read, a sum and a count. When bit 8 is set, the average for the previous interval would be the sum delta, since the previous read, divided by the count delta. Alternatively, dividing the values outright would provide the lifetime average value. Typically these are used to measure latencies, the number of requests and a sum of request latencies. Some bit values are combined, for example 5, 6 and 10. A bit value of 5 is a combination of bit 1 and bit 4. This means the average will be a floating point value. A bit value of 6 is a combination of bit 2 and bit 4. This means the average value will be an integer. A bit value of 10 is a combination of bit 2 and bit 8. This means the counter value will be an integer value.

Additional Resources

- See *Average count and sum* for more details.

8.4. DUMP THE CEPH PERFORMANCE COUNTERS

The **ceph daemon .. perf dump** command outputs the current values and groups the metrics under the collection name for each subsystem.

Prerequisites

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

Procedure

1. To view the current metric data:

```
# ceph daemon DAEMON_NAME perf dump
```

NOTE

You must run the **ceph daemon** command from the node running the daemon.

2. Executing **ceph daemon .. perf dump** command from the Monitor node:

```
# ceph daemon mon.`hostname -s` perf dump
```

Example

```
{
    "cluster": {
        "num_mon": 1,
        "num_mon_quorum": 1,
        "num_osd": 2,
        "num_osd_up": 2,
        "num_osd_in": 2,
...
```

3. Executing the **ceph daemon .. perf dump** command from the OSD node:

```
# ceph daemon osd.0 perf dump
```

Example

```
...
"filestore": {
        "journal_queue_max_ops": 300,
        "journal_queue_ops": 0,
        "journal_ops": 992,
        "journal_queue_max_bytes": 33554432,
        "journal_queue_bytes": 0,
        "journal_bytes": 934537,
        "journal_latency": {
            "avgcount": 992,
            "sum": 254.975925772
        },
...
```

Additional Resources

- To view a short description of each Monitor metric available, please see the *Ceph monitor metrics table*.

8.5. AVERAGE COUNT AND SUM

All latency numbers have a bit field value of 5. This field contains floating point values for the average count and sum. The **avgcount** is the number of operations within this range and the **sum** is the total latency in seconds. When dividing the **sum** by the **avgcount** this will provide you with an idea of the latency per operation.

Additional Resources

- To view a short description of each OSD metric available, please see the *Ceph OSD table*.

8.6. CEPH MONITOR METRICS

Table 8.2. Cluster Metrics Table

Collection Name	Metric Name	Bit Field Value	Short Description
cluster	**num_mon**	2	Number of monitors
	num_mon_quorum	2	Number of monitors in quorum
	num_osd	2	Total number of OSD
	num_osd_up	2	Number of OSDs that are up
	num_osd_in	2	Number of OSDs that are in cluster
	osd_epoch	2	Current epoch of OSD map
	osd_bytes	2	Total capacity of cluster in bytes
	osd_bytes_used	2	Number of used bytes on cluster
	osd_bytes_avail	2	Number of available bytes on cluster
	num_pool	2	Number of pools

Collection Name	Metric Name	Bit Field Value	Short Description
	num_pg	2	Total number of placement groups
	num_pg_active_clean	2	Number of placement groups in active+clean state
	num_pg_active	2	Number of placement groups in active state
	num_pg_peering	2	Number of placement groups in peering state
	num_object	2	Total number of objects on cluster
	num_object_degraded	2	Number of degraded (missing replicas) objects
	num_object_misplaced	2	Number of misplaced (wrong location in the cluster) objects
	num_object_unfound	2	Number of unfound objects
	num_bytes	2	Total number of bytes of all objects
	num_mds_up	2	Number of MDSs that are up
	num_mds_in	2	Number of MDS that are in cluster
	num_mds_failed	2	Number of failed MDS
	mds_epoch	2	Current epoch of MDS map

Table 8.3. Level Database Metrics Table

Collection Name	Metric Name	Bit Field Value	Short Description
leveldb	**leveldb_get**	10	Gets
	leveldb_transaction	10	Transactions
	leveldb_compact	10	Compactions

Collection Name	Metric Name	Bit Field Value	Short Description
	leveldb_compact_range	10	Compactions by range
	leveldb_compact_queue_merge	10	Mergings of ranges in compaction queue
	leveldb_compact_queue_len	2	Length of compaction queue

Table 8.4. General Monitor Metrics Table

Collection Name	Metric Name	Bit Field Value	Short Description
mon	**num_sessions**	2	Current number of opened monitor sessions
	session_add	10	Number of created monitor sessions
	session_rm	10	Number of remove_session calls in monitor
	session_trim	10	Number of trimed monitor sessions
	num_elections	10	Number of elections monitor took part in
	election_call	10	Number of elections started by monitor
	election_win	10	Number of elections won by monitor
	election_lose	10	Number of elections lost by monitor

Table 8.5. Paxos Metrics Table

Collection Name	Metric Name	Bit Field Value	Short Description
paxos	**start_leader**	10	Starts in leader role
	start_peon	10	Starts in peon role
	restart	10	Restarts

Collection Name	Metric Name	Bit Field Value	Short Description
	refresh	10	Refreshes
	refresh_latency	5	Refresh latency
	begin	10	Started and handled begins
	begin_keys	6	Keys in transaction on begin
	begin_bytes	6	Data in transaction on begin
	begin_latency	5	Latency of begin operation
	commit	10	Commits
	commit_keys	6	Keys in transaction on commit
	commit_bytes	6	Data in transaction on commit
	commit_latency	5	Commit latency
	collect	10	Peon collects
	collect_keys	6	Keys in transaction on peon collect
	collect_bytes	6	Data in transaction on peon collect
	collect_latency	5	Peon collect latency
	collect_uncommitted	10	Uncommitted values in started and handled collects
	collect_timeout	10	Collect timeouts
	accept_timeout	10	Accept timeouts
	lease_ack_timeout	10	Lease acknowledgement timeouts
	lease_timeout	10	Lease timeouts
	store_state	10	Store a shared state on disk

Collection Name	Metric Name	Bit Field Value	Short Description
	store_state_keys	6	Keys in transaction in stored state
	store_state_bytes	6	Data in transaction in stored state
	store_state_latency	5	Storing state latency
	share_state	10	Sharings of state
	share_state_keys	6	Keys in shared state
	share_state_bytes	6	Data in shared state
	new_pn	10	New proposal number queries
	new_pn_latency	5	New proposal number getting latency

Table 8.6. Throttle Metrics Table

Collection Name	Metric Name	Bit Field Value	Short Description
throttle-*	**val**	10	Currently available throttle
	max	10	Max value for throttle
	get	10	Gets
	get_sum	10	Got data
	get_or_fail_fail	10	Get blocked during get_or_fail
	get_or_fail_success	10	Successful get during get_or_fail
	take	10	Takes
	take_sum	10	Taken data
	put	10	Puts
	put_sum	10	Put data

Collection Name	Metric Name	Bit Field Value	Short Description
	wait	5	Waiting latency

Additional Resources

- *Cluster Metrics Table*
- *Level Database Metrics Table*
- *General Monitor Metrics Table*
- *Paxos Metrics Table*
- *Throttle Metrics Table*

8.7. CEPH OSD METRICS

Table 8.7. Write Back Throttle Metrics Table

Collection Name	Metric Name	Bit Field Value	Short Description
WBThrottle	**bytes_dirtied**	2	Dirty data
	bytes_wb	2	Written data
	ios_dirtied	2	Dirty operations
	ios_wb	2	Written operations
	inodes_dirtied	2	Entries waiting for write
	inodes_wb	2	Written entries

Table 8.8. Level Database Metrics Table

Collection Name	Metric Name	Bit Field Value	Short Description
leveldb	**leveldb_get**	10	Gets
	leveldb_transaction	10	Transactions
	leveldb_compact	10	Compactions
	leveldb_compact_range	10	Compactions by range

Collection Name	Metric Name	Bit Field Value	Short Description
	leveldb_compact_queue_merge	10	Mergings of ranges in compaction queue
	leveldb_compact_queue_len	2	Length of compaction queue

Table 8.9. Objecter Metrics Table

Collection Name	Metric Name	Bit Field Value	Short Description
objecter	**op_active**	2	Active operations
	op_laggy	2	Laggy operations
	op_send	10	Sent operations
	op_send_bytes	10	Sent data
	op_resend	10	Resent operations
	op_ack	10	Commit callbacks
	op_commit	10	Operation commits
	op	10	Operation
	op_r	10	Read operations
	op_w	10	Write operations
	op_rmw	10	Read-modify-write operations
	op_pg	10	PG operation
	osdop_stat	10	Stat operations
	osdop_create	10	Create object operations
	osdop_read	10	Read operations
	osdop_write	10	Write operations
	osdop_writefull	10	Write full object operations
	osdop_append	10	Append operation

Collection Name	Metric Name	Bit Field Value	Short Description
	osdop_zero	10	Set object to zero operations
	osdop_truncate	10	Truncate object operations
	osdop_delete	10	Delete object operations
	osdop_mapext	10	Map extent operations
	osdop_sparse_read	10	Sparse read operations
	osdop_clonerange	10	Clone range operations
	osdop_getxattr	10	Get xattr operations
	osdop_setxattr	10	Set xattr operations
	osdop_cmpxattr	10	Xattr comparison operations
	osdop_rmxattr	10	Remove xattr operations
	osdop_resetxattrs	10	Reset xattr operations
	osdop_tmap_up	10	TMAP update operations
	osdop_tmap_put	10	TMAP put operations
	osdop_tmap_get	10	TMAP get operations
	osdop_call	10	Call (execute) operations
	osdop_watch	10	Watch by object operations
	osdop_notify	10	Notify about object operations
	osdop_src_cmpxattr	10	Extended attribute comparison in multi operations
	osdop_other	10	Other operations
	linger_active	2	Active lingering operations
	linger_send	10	Sent lingering operations
	linger_resend	10	Resent lingering operations

Collection Name	Metric Name	Bit Field Value	Short Description
	linger_ping	10	Sent pings to lingering operations
	poolop_active	2	Active pool operations
	poolop_send	10	Sent pool operations
	poolop_resend	10	Resent pool operations
	poolstat_active	2	Active get pool stat operations
	poolstat_send	10	Pool stat operations sent
	poolstat_resend	10	Resent pool stats
	statfs_active	2	Statfs operations
	statfs_send	10	Sent FS stats
	statfs_resend	10	Resent FS stats
	command_active	2	Active commands
	command_send	10	Sent commands
	command_resend	10	Resent commands
	map_epoch	2	OSD map epoch
	map_full	10	Full OSD maps received
	map_inc	10	Incremental OSD maps received
	osd_sessions	2	Open sessions
	osd_session_open	10	Sessions opened
	osd_session_close	10	Sessions closed
	osd_laggy	2	Laggy OSD sessions

Table 8.10. Read and Write Operations Metrics Table

Collection Name	Metric Name	Bit Field Value	Short Description
osd	**op_wip**	2	Replication operations currently being processed (primary)
	op_in_bytes	10	Client operations total write size
	op_out_bytes	10	Client operations total read size
	op_latency	5	Latency of client operations (including queue time)
	op_process_latency	5	Latency of client operations (excluding queue time)
	op_r	10	Client read operations
	op_r_out_bytes	10	Client data read
	op_r_latency	5	Latency of read operation (including queue time)
	op_r_process_latency	5	Latency of read operation (excluding queue time)
	op_w	10	Client write operations
	op_w_in_bytes	10	Client data written
	op_w_rlat	5	Client write operation readable/applied latency
	op_w_latency	5	Latency of write operation (including queue time)
	op_w_process_latency	5	Latency of write operation (excluding queue time)
	op_rw	10	Client read-modify-write operations
	op_rw_in_bytes	10	Client read-modify-write operations write in
	op_rw_out_bytes	10	Client read-modify-write operations read out

Collection Name	Metric Name	Bit Field Value	Short Description
	op_rw_rlat	5	Client read-modify-write operation readable/applied latency
	op_rw_latency	5	Latency of read-modify-write operation (including queue time)
	op_rw_process_latency	5	Latency of read-modify-write operation (excluding queue time)
	subop	10	Suboperations
	subop_in_bytes	10	Suboperations total size
	subop_latency	5	Suboperations latency
	subop_w	10	Replicated writes
	subop_w_in_bytes	10	Replicated written data size
	subop_w_latency	5	Replicated writes latency
	subop_pull	10	Suboperations pull requests
	subop_pull_latency	5	Suboperations pull latency
	subop_push	10	Suboperations push messages
	subop_push_in_bytes	10	Suboperations pushed size
	subop_push_latency	5	Suboperations push latency
	pull	10	Pull requests sent
	push	10	Push messages sent
	push_out_bytes	10	Pushed size
	push_in	10	Inbound push messages
	push_in_bytes	10	Inbound pushed size
	recovery_ops	10	Started recovery operations

Collection Name	Metric Name	Bit Field Value	Short Description
	loadavg	2	CPU load
	buffer_bytes	2	Total allocated buffer size
	numpg	2	Placement groups
	numpg_primary	2	Placement groups for which this osd is primary
	numpg_replica	2	Placement groups for which this osd is replica
	numpg_stray	2	Placement groups ready to be deleted from this osd
	heartbeat_to_peers	2	Heartbeat (ping) peers we send to
	heartbeat_from_peers	2	Heartbeat (ping) peers we recv from
	map_messages	10	OSD map messages
	map_message_epochs	10	OSD map epochs
	map_message_epoch_dups	10	OSD map duplicates
	stat_bytes	2	OSD size
	stat_bytes_used	2	Used space
	stat_bytes_avail	2	Available space
	copyfrom	10	Rados 'copy-from' operations
	tier_promote	10	Tier promotions
	tier_flush	10	Tier flushes
	tier_flush_fail	10	Failed tier flushes
	tier_try_flush	10	Tier flush attempts
	tier_try_flush_fail	10	Failed tier flush attempts

Collection Name	Metric Name	Bit Field Value	Short Description
	tier_evict	10	Tier evictions
	tier_whiteout	10	Tier whiteouts
	tier_dirty	10	Dirty tier flag set
	tier_clean	10	Dirty tier flag cleaned
	tier_delay	10	Tier delays (agent waiting)
	tier_proxy_read	10	Tier proxy reads
	agent_wake	10	Tiering agent wake up
	agent_skip	10	Objects skipped by agent
	agent_flush	10	Tiering agent flushes
	agent_evict	10	Tiering agent evictions
	object_ctx_cache_hit	10	Object context cache hits
	object_ctx_cache_total	10	Object context cache lookups

Table 8.11. Recovery State Metrics Table

Collection Name	Metric Name	Bit Field Value	Short Description
recoverystate_perf	**initial_latency**	5	Initial recovery state latency
	started_latency	5	Started recovery state latency
	reset_latency	5	Reset recovery state latency
	start_latency	5	Start recovery state latency
	primary_latency	5	Primary recovery state latency
	peering_latency	5	Peering recovery state latency
	backfilling_latency	5	Backfilling recovery state latency

Collection Name	Metric Name	Bit Field Value	Short Description
	waitremotebackfillreserved_latency	5	Wait remote backfill reserved recovery state latency
	waitlocalbackfillreserved_latency	5	Wait local backfill reserved recovery state latency
	notbackfilling_latency	5	Notbackfilling recovery state latency
	repnotrecovering_latency	5	Repnotrecovering recovery state latency
	repwaitrecoveryreserved_latency	5	Rep wait recovery reserved recovery state latency
	repwaitbackfillreserved_latency	5	Rep wait backfill reserved recovery state latency
	RepRecovering_latency	5	RepRecovering recovery state latency
	activating_latency	5	Activating recovery state latency
	waitlocalrecoveryreserved_latency	5	Wait local recovery reserved recovery state latency
	waitremoterecoveryreserved_latency	5	Wait remote recovery reserved recovery state latency
	recovering_latency	5	Recovering recovery state latency
	recovered_latency	5	Recovered recovery state latency
	clean_latency	5	Clean recovery state latency
	active_latency	5	Active recovery state latency
	replicaactive_latency	5	Replicaactive recovery state latency
	stray_latency	5	Stray recovery state latency
	getinfo_latency	5	Getinfo recovery state latency

Collection Name	Metric Name	Bit Field Value	Short Description
	getlog_latency	5	Getlog recovery state latency
	waitactingchange_latency	5	Waitactingchange recovery state latency
	incomplete_latency	5	Incomplete recovery state latency
	getmissing_latency	5	Getmissing recovery state latency
	waitupthru_latency	5	Waitupthru recovery state latency

Table 8.12. OSD Throttle Metrics Table

Collection Name	Metric Name	Bit Field Value	Short Description
throttle-*	**val**	10	Currently available throttle
	max	10	Max value for throttle
	get	10	Gets
	get_sum	10	Got data
	get_or_fail_fail	10	Get blocked during get_or_fail
	get_or_fail_success	10	Successful get during get_or_fail
	take	10	Takes
	take_sum	10	Taken data
	put	10	Puts
	put_sum	10	Put data
	wait	5	Waiting latency

Additional Resources

- *Write Back Throttle Metrics Table*

- *Level Database Metrics Table*
- *Objecter Metrics Table*
- *Read and Write Operations Metrics Table*
- *Recovery State Metrics Table*
- *OSD Throttle Metrics Table*

8.8. CEPH OBJECT GATEWAY METRICS

Table 8.13. RADOS Client Metrics Table

Collection Name	Metric Name	Bit Field Value	Short Description
client.rgw.<rgw_node_name>	**req**	10	Requests
	failed_req	10	Aborted requests
	get	10	Gets
	get_b	10	Size of gets
	get_initial_lat	5	Get latency
	put	10	Puts
	put_b	10	Size of puts
	put_initial_lat	5	Put latency
	qlen	2	Queue length
	qactive	2	Active requests queue
	cache_hit	10	Cache hits
	cache_miss	10	Cache miss
	keystone_token_cache_hit	10	Keystone token cache hits
	keystone_token_cache_miss	10	Keystone token cache miss

Collection Name	Metric Name	Bit Field Value	Short Description
	gc_retire_object	10	Count of objects retired since the last restart of the Ceph Object Gateway

Table 8.14. Objecter Metrics Table

Collection Name	Metric Name	Bit Field Value	Short Description
objecter	**op_active**	2	Active operations
	op_laggy	2	Laggy operations
	op_send	10	Sent operations
	op_send_bytes	10	Sent data
	op_resend	10	Resent operations
	op_ack	10	Commit callbacks
	op_commit	10	Operation commits
	op	10	Operation
	op_r	10	Read operations
	op_w	10	Write operations
	op_rmw	10	Read-modify-write operations
	op_pg	10	PG operation
	osdop_stat	10	Stat operations
	osdop_create	10	Create object operations
	osdop_read	10	Read operations
	osdop_write	10	Write operations
	osdop_writefull	10	Write full object operations
	osdop_append	10	Append operation
	osdop_zero	10	Set object to zero operations

Collection Name	Metric Name	Bit Field Value	Short Description
	osdop_truncate	10	Truncate object operations
	osdop_delete	10	Delete object operations
	osdop_mapext	10	Map extent operations
	osdop_sparse_read	10	Sparse read operations
	osdop_clonerange	10	Clone range operations
	osdop_getxattr	10	Get xattr operations
	osdop_setxattr	10	Set xattr operations
	osdop_cmpxattr	10	Xattr comparison operations
	osdop_rmxattr	10	Remove xattr operations
	osdop_resetxattrs	10	Reset xattr operations
	osdop_tmap_up	10	TMAP update operations
	osdop_tmap_put	10	TMAP put operations
	osdop_tmap_get	10	TMAP get operations
	osdop_call	10	Call (execute) operations
	osdop_watch	10	Watch by object operations
	osdop_notify	10	Notify about object operations
	osdop_src_cmpxattr	10	Extended attribute comparison in multi operations
	osdop_other	10	Other operations
	linger_active	2	Active lingering operations
	linger_send	10	Sent lingering operations
	linger_resend	10	Resent lingering operations
	linger_ping	10	Sent pings to lingering operations

Collection Name	Metric Name	Bit Field Value	Short Description
	poolop_active	2	Active pool operations
	poolop_send	10	Sent pool operations
	poolop_resend	10	Resent pool operations
	poolstat_active	2	Active get pool stat operations
	poolstat_send	10	Pool stat operations sent
	poolstat_resend	10	Resent pool stats
	statfs_active	2	Statfs operations
	statfs_send	10	Sent FS stats
	statfs_resend	10	Resent FS stats
	command_active	2	Active commands
	command_send	10	Sent commands
	command_resend	10	Resent commands
	map_epoch	2	OSD map epoch
	map_full	10	Full OSD maps received
	map_inc	10	Incremental OSD maps received
	osd_sessions	2	Open sessions
	osd_session_open	10	Sessions opened
	osd_session_close	10	Sessions closed
	osd_laggy	2	Laggy OSD sessions

Table 8.15. RADOS Gateway Throttle Metrics Table

Collection Name	Metric Name	Bit Field Value	Short Description
throttle-*	**val**	10	Currently available throttle

Collection Name	Metric Name	Bit Field Value	Short Description
	max	10	Max value for throttle
	get	10	Gets
	get_sum	10	Got data
	get_or_fail_fail	10	Get blocked during get_or_fail
	get_or_fail_success	10	Successful get during get_or_fail
	take	10	Takes
	take_sum	10	Taken data
	put	10	Puts
	put_sum	10	Put data
	wait	5	Waiting latency

Additional Resources

- RADOS Gateway Client Table
- Objecter Metrics Table
- RADOS Gateway Throttle Metrics Table

CHAPTER 9. BLUESTORE

Starting with Red Hat Ceph Storage 4, BlueStore is the default object store for the OSD daemons. The earlier object store, FileStore, requires a file system on top of raw block devices. Objects are then written to the file system. BlueStore does not require an initial file system, because BlueStore puts objects directly on the block device.

> IMPORTANT
>
> BlueStore provides a high-performance backend for OSD daemons in a production environment. By default, BlueStore is configured to be self-tuning. If you determine that your environment performs better with BlueStore tuned manually, please contact Red Hat support and share the details of your configuration to help us improve the auto-tuning capability. Red Hat looks forward to your feedback and appreciates your recommendations.

9.1. CEPH BLUESTORE

The following are some of the main features of using BlueStore:

Direct management of storage devices

BlueStore consumes raw block devices or partitions. This avoids any intervening layers of abstraction, such as local file systems like XFS, that might limit performance or add complexity.

Metadata management with RocksDB

BlueStore uses the RocksDB' key-value database to manage internal metadata, such as the mapping from object names to block locations on a disk.

Full data and metadata checksumming

By default all data and metadata written to BlueStore is protected by one or more checksums. No data or metadata are read from disk or returned to the user without verification.

Efficient copy-on-write

The Ceph Block Device and Ceph File System snapshots rely on a copy-on-write clone mechanism that is implemented efficiently in BlueStore. This results in efficient I/O both for regular snapshots and for erasure coded pools which rely on cloning to implement efficient two-phase commits.

No large double-writes

BlueStore first writes any new data to unallocated space on a block device, and then commits a RocksDB transaction that updates the object metadata to reference the new region of the disk. Only when the write operation is below a configurable size threshold, it falls back to a write-ahead journaling scheme, similar to how FileStore operates.

Multi-device support

BlueStore can use multiple block devices for storing different data. For example: Hard Disk Drive (HDD) for the data, Solid-state Drive (SSD) for metadata, Non-volatile Memory (NVM) or Non-volatile random-access memory (NVRAM) or persistent memory for the RocksDB write-ahead log (WAL). See *Ceph BlueStore devices* for details.

> NOTE
>
> The **ceph-disk** utility does not yet provision multiple devices. To use multiple devices, OSDs must be set up manually.

Efficient block device usage

Because BlueStore does not use any file system, it minimizes the need to clear the storage device cache.

9.2. CEPH BLUESTORE DEVICES

This section explains what block devices the BlueStore back end uses.

BlueStore manages either one, two, or three storage devices.

- Primary
- WAL
- DB

In the simplest case, BlueStore consumes a single (primary) storage device. The storage device is partitioned into two parts that contain:

- **OSD metadata**: A small partition formatted with XFS that contains basic metadata for the OSD. This data directory includes information about the OSD, such as its identifier, which cluster it belongs to, and its private keyring.
- **Data**: A large partition occupying the rest of the device that is managed directly by BlueStore and that contains all of the OSD data. This primary device is identified by a block symbolic link in the data directory.

You can also use two additional devices:

- **A WAL (write-ahead-log) device**: A device that stores BlueStore internal journal or write-ahead log. It is identified by the **block.wal** symbolic link in the data directory. Consider using a WAL device only if the device is faster than the primary device. For example, when the WAL device uses an SSD disk and the primary devices uses an HDD disk.
- **A DB device**: A device that stores BlueStore internal metadata. The embedded RocksDB database puts as much metadata as it can on the DB device instead of on the primary device to improve performance. If the DB device is full, it starts adding metadata to the primary device. Consider using a DB device only if the device is faster than the primary device.

WARNING

If you have only a less than a gigabyte storage available on fast devices. Red Hat recommends using it as a WAL device. If you have more fast devices available, consider using it as a DB device. The BlueStore journal is always placed on the fastest device, so using a DB device provides the same benefit that the WAL device while also allows for storing additional metadata.

9.3. CEPH BLUESTORE CACHING

The BlueStore cache is a collection of buffers that, depending on configuration, can be populated with data as the OSD daemon does reading from or writing to the disk. By default in Red Hat Ceph Storage, BlueStore will cache on reads, but not writes. This is because the **bluestore_default_buffered_write** option is set to **false** to avoid potential overhead associated with cache eviction.

If the **bluestore_default_buffered_write** option is set to **true**, data is written to the buffer first, and then committed to disk. Afterwards, a write acknowledgement is sent to the client, allowing subsequent reads faster access to the data already in cache, until that data is evicted.

Read-heavy workloads will not see an immediate benefit from BlueStore caching. As more reading is done, the cache will grow over time and subsequent reads will see an improvement in performance. How fast the cache populates depends on the BlueStore block and database disk type, and the client's workload requirements.

IMPORTANT

Please contact Red Hat support before enabling the **bluestore_default_buffered_write** option.

9.4. SIZING CONSIDERATIONS FOR CEPH BLUESTORE

When mixing traditional and solid state drives using BlueStore OSDs, it is important to size the RocksDB logical volume (**block.db**) appropriately. Red Hat recommends that the RocksDB logical volume be no less than 4% of the block size with object, file and mixed workloads. Red Hat supports 1% of the BlueStore block size with RocksDB and OpenStack block workloads. For example, if the block size is 1 TB for an object workload, then at a minimum, create a 40 GB RocksDB logical volume.

When not mixing drive types, there is no requirement to have a separate RocksDB logical volume. BlueStore will automatically manage the sizing of RocksDB.

BlueStore's cache memory is used for the key-value pair metadata for RocksDB, BlueStore metadata and object data.

NOTE

The BlueStore cache memory values are in addition to the memory footprint already being consumed by the OSD.

9.5. ADDING CEPH BLUESTORE OSDS

This section describes how to install a new Ceph OSD node with the BlueStore back end object store.

Prerequisites

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

Procedure

1. Add a new OSD node to the **[osds]** section in Ansible inventory file, by default located at **/etc/ansible/hosts**.

```
[osds]
node1
node2
```

```
node3
HOST_NAME
```

Replace:

- ***HOST_NAME*** with the name of the OSD node

Example

```
[osds]
node1
node2
node3
node4
```

2. Navigate to the **/usr/share/ceph-ansible/** directory.

```
[user@admin ~]$ cd /usr/share/ceph-ansible
```

3. Create the **host_vars** directory.

```
[root@admin ceph-ansible] mkdir host_vars
```

4. Create the configuration file for the newly added OSD in **host_vars**.

```
[root@admin ceph-ansible] touch host_vars/HOST_NAME.yml
```

Replace:

- ***HOST_NAME*** with the host name of the newly added OSD

Example

```
[root@admin ceph-ansible] touch host_vars/node4.yml
```

5. Add the following setting to the newly created file:

```
osd_objectstore: bluestore
```

NOTE

To use BlueStore for all OSDs, add **osd_objectstore:bluestore** to the **group_vars/all.yml** file.

6. Configure the BlueStore OSDs, in **host_vars/*HOST_NAME*.yml**:

Syntax

```
lvm_volumes:
  - data: DATALV
    data_vg: DATAVG
```

Replace:

- ***DATALV*** with the data logical volume name
- ***DATAVG*** with the data logical volume group name

Example

```
lvm_volumes:
  - data: data-lv1
    data_vg: vg1
```

7. Optional. If you want to store the **block.wal** and **block.db** on dedicated logical volumes, edit the **host_vars/*HOST_NAME*.yml** file as follows:

```
lvm_volumes:
  - data: DATALV
    wal: WALLV
    wal_vg: VG
    db: DBLV
    db_vg: VG
```

Replace:

- *DATALV* with the logical volume where the data should be contained
- *WALLV* with the logical volume where the write-ahead-log should be contained
- *VG* with the volume group the WAL and/or DB device LVs are on
- *DBLV* with the logical volume the BlueStore internal metadata should be contained

Example

```
lvm_volumes:
  - data: data-lv3
    wal: wal-lv1
    wal_vg: vg3
    db: db-lv3
    db_vg: vg3
```

NOTE

When using **lvm_volumes:** with **osd_objectstore: bluestore** the **lvm_volumes** YAML dictionary must contain at least **data**. When defining **wal** or **db**, it must have both the LV name and VG name (**db** and **wal** are not required). This allows for four combinations: just data, data and wal, data and wal and db, or data and db. Data can be a raw device, lv or partition. The **wal** and **db** can be a lv or partition. When specifying a raw device or partition **ceph-volume** will put logical volumes on top of them.

NOTE

Currently, **ceph-ansible** does not create the volume groups or the logical volumes. This must be done before running the Anisble playbook.

8. Optional: You can override the **block.db** default size in the **group_vars/all.yml** file:

   Syntax

   ```
   ceph_conf_overrides:
     osd:
       bluestore_block_db_size: VALUE
   ```

   Example

   ```
   ceph_conf_overrides:
     osd:
       bluestore_block_db_size: 24336000000
   ```

   NOTE

   The value of **bluestore_block_db_size** should be greater than 2 GB.

9. Open and edit the **group_vars/all.yml** file, and uncomment the **osd_memory_target** option. Adjust the value on how much memory you want the OSD to consume.

   NOTE

   The default value for the **osd_memory_target** option is **4000000000**, which is 4 GB. This option pins the BlueStore cache in memory.

   IMPORTANT

   The **osd_memory_target** option only applies to BlueStore-backed OSDs.

10. Run the following Ansible playbook:

    ```
    [user@admin ceph-ansible]$ ansible-playbook site.yml
    ```

11. From a Ceph Monitor node, verify that the new OSD has been successfully added:

    ```
    [root@mon ~]# ceph osd tree
    ```

9.6. TUNING CEPH BLUESTORE FOR SMALL WRITES

In BlueStore, the raw partition is allocated and managed in chunks of **bluestore_min_alloc_size**. By default, **bluestore_min_alloc_size** is 64 KB for HDDs, and 4 KB for SSDs. The unwritten area in each chunk is filled with zeroes when it is written to the raw partition. This can lead to wasted unused space when not properly sized for your workload, for example when writing small objects.

It is best practice to set **bluestore_min_alloc_size** to match the smallest write so this can write amplification penalty can be avoided.

For example, if your client writes 4 KB objects frequently, use **ceph-ansible** to configure the following setting on OSD nodes:

bluestore_min_alloc_size = 4096

NOTE

The settings **bluestore_min_alloc_size_ssd** and **bluestore_min_alloc_size_hdd** are specific to SSDs and HDDs, respectively, but setting them is not necessary because setting **bluestore_min_alloc_size** overrides them.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- New servers that can be freshly provisioned as OSD nodes, or:
- OSD nodes that can be redeployed.
- The admin keyring for the Ceph Monitor node, if you are redeploying an existing Ceph OSD node.

Procedure

1. Optional: If redeploying an existing OSD node, use the **shrink-osd.yml** Ansible playbook to remove the OSD from the cluster.

   ```
   ansible-playbook -v infrastructure-playbooks/shrink-osd.yml -e osd_to_kill=OSD_ID
   ```

   Example

   ```
   [admin@admin ceph-ansible]$ ansible-playbook -v infrastructure-playbooks/shrink-osd.yml -e osd_to_kill=1
   ```

2. If redeploying an existing OSD node, wipe the OSD drives and reinstall the OS.

3. Prepare the node for OSD provisioning using Ansible. Examples of preparation tasks include enabling Red Hat Ceph Storage repositories, adding an Ansible user, and enabling password-less SSH login.

4. Add the **bluestore_min_alloc_size** to the **ceph_conf_overrides** section of the **group_vars/all.yml** Ansible playbook:

   ```
   ceph_conf_overrides:
     osd:
       bluestore_min_alloc_size: 4096
   ```

5. If deploying a new node, add it to the Ansible inventory file, normally **/etc/ansible/hosts**:

   ```
   [osds]
   OSD_NODE_NAME
   ```

Example

```
[osds]
osd1 devices="[ '/dev/sdb' ]"
```

6. If redeploying an existing OSD, copy the admin keyring file in the Ceph Monitor node to the node where you want to deploy the OSD.

7. Provision the OSD node using Ansible:

```
ansible-playbook -v site.yml -l OSD_NODE_NAME
```

Example

```
[admin@admin ceph-ansible]$ ansible-playbook -v site.yml -l osd1
```

8. After the playbook finishes, verify the setting using the **ceph daemon** command:

```
ceph daemon OSD.ID config get bluestore_min_alloc_size
```

Example

```
[root@osd1 ~]# ceph daemon osd.1 config get bluestore_min_alloc_size
{
    "bluestore_min_alloc_size": "4096"
}
```

You can see **bluestore_min_alloc_size** is set to 4096 bytes, which is equivalent to 4 KiB.

Additional Resources

- See the `*Red Hat Ceph Storage Installation Guide* for more information.

9.7. THE BLUESTORE FRAGMENTATION TOOL

As a storage administrator, you will want to periodically check the fragmentation level of your BlueStore OSDs. You can check fragmentation levels with one simple command for offline or online OSDs.

9.7.1. Prerequisites

- A running Red Hat Ceph Storage 3.3 or higher storage cluster.
- BlueStore OSDs.

9.7.2. What is the BlueStore fragmentation tool?

For BlueStore OSDs, the free space gets fragmented over time on the underlying storage device. Some fragmentation is normal, but when there is excessive fragmentation this causes poor performance.

The BlueStore fragmentation tool generates a score on the fragmentation level of the BlueStore OSD. This fragmentation score is given as a range, 0 through 1. A score of 0 means no fragmentation, and a score of 1 means severe fragmentation.

Table 9.1. Fragmentation scores' meaning

Score	Fragmentation Amount
0.0 - 0.4	None to tiny fragmentation.
0.4 - 0.7	Small and acceptable fragmentation.
0.7 - 0.9	Considerable, but safe fragmentation.
0.9 - 1.0	Severe fragmentation and that causes performance issues.

IMPORTANT

If you have severe fragmentation, and need some help in resolving the issue, contact Red Hat Support.

9.7.3. Checking for fragmentation

Checking the fragmentation level of BlueStore OSDs can be done either online or offline.

Prerequisites

- A running Red Hat Ceph Storage 3.3 or higher storage cluster.
- BlueStore OSDs.

Online BlueStore fragmentation score

1. Inspect a running BlueStore OSD process:
   a. Simple report:

      Syntax

      ```
      ceph daemon OSD_ID bluestore allocator score block
      ```

      Example

      ```
      [root@osd ~]# ceph daemon osd.123 bluestore allocator score block
      ```

   b. A more detailed report:

      Syntax

      ```
      ceph daemon OSD_ID bluestore allocator dump block
      ```

      Example

      ```
      [root@osd ~]# ceph daemon osd.123 bluestore allocator dump block
      ```

–

Offline BlueStore fragmentation score

1. Inspect a non-running BlueStore OSD process:

   a. Simple report:

      Syntax

      ```
      ceph-bluestore-tool --path PATH_TO_OSD_DATA_DIRECTORY --allocator block free-score
      ```

      Example

      ```
      [root@osd ~]# ceph-bluestore-tool --path /var/lib/ceph/osd/ceph-123 --allocator block free-score
      ```

   b. A more detailed report:

      Syntax

      ```
      ceph-bluestore-tool --path PATH_TO_OSD_DATA_DIRECTORY --allocator block free-dump
      ```

      Example

      ```
      [root@osd ~]# ceph-bluestore-tool --path /var/lib/ceph/osd/ceph-123 --allocator block free-dump
      ```

Additional Resources

- See the Red Hat Ceph Storage 4.1 *BlueStore Fragmentation Tool* for details on the fragmentation score.

9.8. HOW TO MIGRATE THE OBJECT STORE FROM FILESTORE TO BLUESTORE

As a storage administrator, you can migrate from the traditional object store, FileStore, to the new object store, BlueStore.

9.8.1. Prerequisites

- A healthy and running Red Hat Ceph Storage cluster.

9.8.2. Migrating from FileStore to BlueStore

BlueStore improves performance and robustness, compared to the traditional FileStore. A single Red Hat Ceph Storage cluster can contain a mix of both FileStore and BlueStore devices.

Converting an individual OSD cannot be done in place, or in isolation. The conversion process will rely either on the storage cluster's normal replication and healing process or tools and strategies that copy OSD content from an old (FileStore) device to a new (BlueStore) device. There are two approach to migrate from FileStore to BlueStore.

First Approach

The first approach is to mark out each device in turn, wait for the data to replicate across the storage cluster, reprovision the OSD, and mark it back "in" again. Here are the advantages and disadvantage to this approach:

Advantages

- Simple.
- Can be done on a device-by-device basis.
- No spare devices or nodes are required.

Disadvantages

- Copying data over the network happens twice.

> **NOTE**
>
> One copy to some other OSD in the storage cluster, allowing you to maintain the desired number of replicas, and then another copy back to the reprovisioned BlueStore OSD.

Second Approach

The second approach is doing a whole node replacement. You need to have an empty node that has no data.

There are two ways to do this: * Starting with a new, empty node that is not part of the storage cluster. * By offloading data from an existing node in the storage cluster.

Advantages

- Data is copied over the network only once.
- Converts an entire node's OSDs at once.
- Can parallelize to converting multiple nodes at a time.
- No spare devices are required on each node.

Disadvantages

- A spare node is required.
- An entire node's worth of OSDs will be migrating data at a time. This is like likely to impact overall cluster performance.
- All migrated data still makes one full hop over the network.

9.8.3. Migrating from FileStore to BlueStore using Ansible

Migrating from FileStore to BlueStore using Ansible will shrinks and redeploys all OSDs on the node. The Ansible playbook does a capacity check before starting the migration. The **ceph-volume** utility then redeploys the OSDs.

Prerequisites

- A healthy and running Red Hat Ceph Storage 4 cluster.
- The **ansible** user account for use with the Ansible application.

Procedure

1. Log in as the **ansible** user on the Ansible administration node.
2. Edit the **group_vars/osd.yml** file, add and set the following options:

```
nb_retry_wait_osd_up: 50
delay_wait_osd_up: 30
```

3. Run the following Ansible playbook:

Syntax

```
ansible-playbook infrastructure-playbooks/filestore-to-bluestore.yml --limit
OSD_NODE_TO_MIGRATE
```

Example

```
[ansible@admin ~]$ ansible-playbook infrastructure-playbooks/filestore-to-bluestore.yml --
limit osd1
```

> **WARNING**
>
> If you explicitly set **osd_crush_update_on_start = False** in your Ceph configuration file, the conversion fails. It creates a new OSD with a different ID and misplaces it in the CRUSH rule. Additionally, it does not clear the old OSD data directory.

4. Wait for the migration to complete before starting on the next OSD node in the storage cluster.

9.8.4. Migrating from FileStore to BlueStore using the mark out and replace approach

The simplest approach to migrate from FileStore to BlueStore is to mark out each device in turn, wait for the data to replicate across the storage cluster, reprovision the OSD, and mark it back "in" again.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- **root** access to the node.

Procedure

Replace the variable ***OSD_ID*** below with the ODS identification number.

1. Find a FileStore OSD to replace.

   a. Get the OSD identification number:

   ```
   [root@ceph-client ~]# ceph osd tree
   ```

   b. Identify whether an OSD is using FileStore or BlueStore:

   #### Syntax

   ```
   ceph osd metadata OSD_ID | grep osd_objectstore
   ```

   #### Example

   ```
   [root@ceph-client ~]# ceph osd metadata 0 | grep osd_objectstore
       "osd_objectstore": "filestore",
   ```

   c. To view the current count of FileStore devices versus BlueStore devices:

   ```
   [root@ceph-client ~]# ceph osd count-metadata osd_objectstore
   ```

2. Mark the FileStore OSD out:

   ```
   ceph osd out OSD_ID
   ```

3. Wait for the data to migrate off the OSD:

   ```
   while ! ceph osd safe-to-destroy OSD_ID ; do sleep 60 ; done
   ```

4. Stop the OSD:

   ```
   systemctl stop ceph-osd@OSD_ID
   ```

5. Capture which device this OSD is using:

   ```
   mount | grep /var/lib/ceph/osd/ceph-OSD_ID
   ```

6. Unmount the OSD:

   ```
   umount /var/lib/ceph/osd/ceph-OSD_ID
   ```

7. Destroy the OSD data, using the value from step 5 as ***DEVICE***:

   ```
   ceph-volume lvm zap DEVICE
   ```

IMPORTANT

Be **EXTREMELY CAREFUL** as this will destroy the contents of the device. Be certain the data on the device is not needed, that is the storage cluster is healthy, before proceeding.``

NOTE

If the OSD is encrypted, then the unmount the **osd-lockbox** and remove the encryption before zapping the OSD using **dmsetup remove**.

NOTE

If the OSD contains logical volumes, then use the **--destroy** option on the **ceph-volume lvm zap** command.

8. Make the storage cluster aware that the OSD has been destroyed:

```
[root@ceph-client ~]# ceph osd destroy OSD_ID --yes-i-really-mean-it
```

9. Reprovision the OSD as a BlueStore OSD, using ***DEVICE*** from step 5, and the same ***OSD_ID***:

```
[root@ceph-client ~]# ceph-volume lvm create --bluestore --data DEVICE --osd-id OSD_ID
```

10. Repeat this procedure.

NOTE

The refilling of the new BlueStore OSD can happen concurrently with the draining of the next FileStore OSD, as long as you ensure the storage cluster is **HEALTH_OK** before destroying any OSDs. Failure to do so will reduce the redundancy of your data and increase the risk of, or the potentially of data loss.

9.8.5. Migrating from FileStore to BlueStore using the whole node replacement approach

Migrating from FileStore to BlueStore can be done on a node-by-node basis by transferring each stored copy of the data only once. This migration can be done with a spare node in the storage cluster, or having the sufficient free space to evacuate an entire node from the storage cluster in order to use it as a spare. Ideally, the node must have roughly the same capacity as the other nodes you will be migrating.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- **root** access to the node.
- An empty node that has no data.

Procedure

- Replace the variable ***NEWNODE*** below with the new node name.

- Replace the variable ***EXISTING_NODE_TO_CONVERT*** below with the node name already existing in the storage cluster.
- Replace the variable ***OSD_ID*** below with the OSD identification number.

1. Using a new node that is not in the storage cluster. For using an existing node already in the storage cluster, skip to step 3.

   a. Add the node to the CRUSH hierarchy:

   ```
   [root@mon ~]# ceph osd crush add-bucket NEWNODE node
   ```

   IMPORTANT

   Do not attach it to the root.

   b. Install the Ceph software packages:

   ```
   [root@mon ~]# yum install ceph-osd
   ```

   NOTE

   Copy the Ceph configuration file, by default **/etc/ceph/ceph.conf**, and keyrings to the new node.

2. Skip to step 5.

3. If you are using an existing node already in the storage cluster, use the following command:

   ```
   [root@mon ~]# ceph osd crush unlink EXISTING_NODE_TO_CONVERT default
   ```

   NOTE

   Where **default** is the immediate ancestor in the CRUSH map.

4. Skip to step 8.

5. Provision new BlueStore OSDs for all devices:

   ```
   [root@mon ~]# ceph-volume lvm create --bluestore --data /dev/DEVICE
   ```

6. Verify that OSDs joined the cluster:

   ```
   [root@mon ~]# ceph osd tree
   ```

   You should see the new node name with all of the OSDs underneath the node name, but the node must **not** be nested underneath any other node in hierarchy.

   Example

   ```
   [root@mon ~]# ceph osd tree
   ID CLASS WEIGHT  TYPE NAME     STATUS REWEIGHT PRI-AFF
   ```

```
-5           0 node newnode
10   ssd 1.00000     osd.10       up  1.00000 1.00000
11   ssd 1.00000     osd.11       up  1.00000 1.00000
12   ssd 1.00000     osd.12       up  1.00000 1.00000
-1       3.00000 root default
-2       3.00000     node oldnode1
0   ssd 1.00000        osd.0     up  1.00000 1.00000
1   ssd 1.00000        osd.1     up  1.00000 1.00000
2   ssd 1.00000        osd.2     up  1.00000 1.00000
```

7. Swap the new node into the old node’s position in the cluster:

```
[root@mon ~]# ceph osd crush swap-bucket NEWNODE
EXISTING_NODE_TO_CONVERT
```

At this point, all data on the ***EXISTING_NODE_TO_CONVERT*** will start migrating to OSDs on the ***NEWNODE***.

NOTE

If there is a difference in the total capacity of the old and new nodes you might also see some data migrate to or from other nodes in the storage cluster, but as long as the nodes are similarly sized this will be a relatively small amount of data.

8. Wait for data migration to complete:

```
while ! ceph osd safe-to-destroy $(ceph osd ls-tree EXISTING_NODE_TO_CONVERT);
do sleep 60 ; done
```

9. Log into the ***EXISTING_NODE_TO_CONVERT***, stop and unmount all old OSDs on the now-empty ***EXISTING_NODE_TO_CONVERT***:

```
[root@mon ~]# systemctl stop ceph-osd@OSD_ID
[root@mon ~]# umount /var/lib/ceph/osd/ceph-OSD_ID
```

10. Destroy and purge the old OSDs:

```
for osd in ceph osd ls-tree EXISTING_NODE_TO_CONVERT; do ceph osd purge $osd -
-yes-i-really-mean-it ; done
```

11. Wipe the old OSD devices. This requires you do identify which devices are to be wiped manually. Do the following command for each device:

```
[root@mon ~]# ceph-volume lvm zap DEVICE
```

IMPORTANT

Be **EXTREMELY CAREFUL** as this will destroy the contents of the device. Be certain the data on the device is not needed, that is the storage cluster is healthy, before proceeding.

NOTE

If the OSD is encrypted, then the unmount the **osd-lockbox** and remove the encryption before zapping the OSD using **dmsetup remove**.

NOTE

If the OSD contains logical volumes, then use the **--destroy** option on the **ceph-volume lvm zap** command.

12. Use the now-empty old node as the new node, and repeat the process.