Red Hat Ceph Storage 5

Administration Guide

Administration of Red Hat Ceph Storage

Last Updated: 2022-08-09

Red Hat Ceph Storage 5 Administration Guide

Administration of Red Hat Ceph Storage

Legal Notice

Copyright © 2022 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

This document describes how to manage 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 in a Red Hat Ceph Storage cluster. 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 on using systemd, see *Introduction to systemd* chapter, and the *Managing system services with systemctl* chapter in the *Configuring basic system settings* guide for Red Hat Enterprise Linux 8.

2.3. STARTING, STOPPING, AND RESTARTING ALL CEPH DAEMONS

You can start, stop, and restart all Ceph daemons as the root user from the host where you want to stop the Ceph daemons.

Prerequisites

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

Procedure

1. On the host where you want to start, stop, and restart the daemons, run the systemctl service to get the *SERVICE_ID* of the service.

   Example

   ```
   [root@host01 ~]# systemctl --type=service
   ceph-499829b4-832f-11eb-8d6d-001a4a000635@mon.host01.service
   ```

2. Starting all Ceph daemons:

   Syntax

   ```
   systemctl start SERVICE_ID
   ```

   Example

```
[root@host01 ~]# systemctl start ceph-499829b4-832f-11eb-8d6d-
001a4a000635@mon.host01.service
```

3. Stopping all Ceph daemons:

   Syntax

   ```
   systemctl stop SERVICE_ID
   ```

   Example

   ```
   [root@host01 ~]# systemctl stop ceph-499829b4-832f-11eb-8d6d-
   001a4a000635@mon.host01.service
   ```

4. Restarting all Ceph daemons:

   Syntax

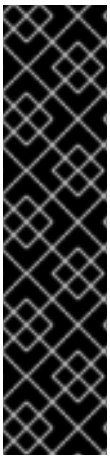
   ```
   systemctl restart SERVICE_ID
   ```

   Example

   ```
   [root@host01 ~]# systemctl restart ceph-499829b4-832f-11eb-8d6d-
   001a4a000635@mon.host01.service
   ```

2.4. STARTING, STOPPING, AND RESTARTING ALL CEPH SERVICES

Ceph services are logical groups of Ceph daemons of the same type, configured to run in the same Red Hat Ceph Storage cluster. The orchestration layer in Ceph allows the user to manage these services in a centralized way, making it easy to execute operations that affect all the Ceph daemons that belong to the same logical service. The Ceph daemons running in each host are managed through the Systemd service. You can start, stop, and restart all Ceph services from the host where you want to manage the Ceph services.



IMPORTANT

If you want to start,stop, or restart a specific Ceph daemon in a specific host, you need to use the SystemD service. To obtain a list of the SystemD services running in a specific host, connect to the host, and run the following command:

Example

```
[root@host01 ~]# systemctl list-units “ceph*”
```

The output will give you a list of the service names that you can use, to manage each Ceph daemon.

Prerequisites

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

Procedure

1. Log into the Cephadm shell:

   Example

   ```
   [root@host01 ~]# cephadm shell
   ```

2. Run the **ceph orch ls** command to get a list of Ceph services configured in the Red Hat Ceph Storage cluster and to get the specific service ID.

   Example

   ```
   [ceph: root@host01 /]# ceph orch ls
   NAME                       RUNNING  REFRESHED  AGE  PLACEMENT  IMAGE NAME
   IMAGE ID
   alertmanager               1/1  4m ago     4M   count:1    registry.redhat.io/openshift4/ose-
   prometheus-alertmanager:v4.5   b7bae610cd46
   crash                      3/3  4m ago     4M   *          registry.redhat.io/rhceph-alpha/rhceph-5-
   rhel8:latest           c88a5d60f510
   grafana                    1/1  4m ago     4M   count:1    registry.redhat.io/rhceph-alpha/rhceph-5-
   dashboard-rhel8:latest  bd3d7748747b
   mgr                        2/2  4m ago     4M   count:2    registry.redhat.io/rhceph-alpha/rhceph-5-
   rhel8:latest           c88a5d60f510
   mon                        2/2  4m ago     10w  count:2    registry.redhat.io/rhceph-alpha/rhceph-5-
   rhel8:latest           c88a5d60f510
   nfs.foo                    0/1  -          -    count:1    <unknown>
   <unknown>
   node-exporter              1/3  4m ago     4M   *          registry.redhat.io/openshift4/ose-
   prometheus-node-exporter:v4.5  mix
   osd.all-available-devices  5/5  4m ago     3M   *          registry.redhat.io/rhceph-
   alpha/rhceph-5-rhel8:latest            c88a5d60f510
   prometheus                 1/1  4m ago     4M   count:1    registry.redhat.io/openshift4/ose-
   prometheus:v4.6                bebb0ddef7f0
   rgw.test_realm.test_zone   2/2  4m ago     3M   count:2    registry.redhat.io/rhceph-
   alpha/rhceph-5-rhel8:latest            c88a5d60f510
   ```

3. To start a specific service, run the following command:

   Syntax

   ```
   ceph orch start SERVICE_ID
   ```

   Example

   ```
   [ceph: root@host01 /]# ceph orch start node-exporter
   ```

4. To stop a specific service, run the following command:

IMPORTANT

The **ceph orch stop SERVICE_ID** command results in the Red Hat Ceph Storage cluster being inaccessible, only for the MON and MGR service. It is recommended to use the **systemctl stop SERVICE_ID** command to stop a specific daemon in the host.

Syntax

```
ceph orch stop SERVICE_ID
```

Example

```
[ceph: root@host01 /]# ceph orch stop node-exporter
```

In the example the **ceph orch stop node-exporter** command removes all the daemons of the **node exporter** service.

5. To restart a specific service, run the following command:

Syntax

```
ceph orch restart SERVICE_ID
```

Example

```
[ceph: root@host01 /]# ceph orch restart node-exporter
```

2.5. VIEWING LOG FILES OF CEPH DAEMONS THAT RUN IN CONTAINERS

Use the **journald** daemon from the container host to view a log file 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 file, run a **journalctl** command as **root** composed in the following format:

Syntax

```
journalctl -u ceph SERVICE_ID
```

```
[root@host01 ~]# journalctl -u ceph-499829b4-832f-11eb-8d6d-
001a4a000635@osd.8.service
```

In the above example, you can view the entire log for the OSD with ID **osd.8**.

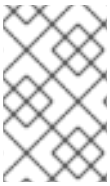
2. To show only the recent journal entries, use the **-f** option.

   Syntax

   ```
   journalctl -fu SERVICE_ID
   ```

   Example

   ```
   [root@host01 ~]# journalctl -fu ceph-499829b4-832f-11eb-8d6d-
   001a4a000635@osd.8.service
   ```



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** manual page.

2.6. 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 the host 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.

   Syntax

   ```
   ceph fs set FS_NAME max_mds 1
   ceph fs fail FS_NAME
   ceph status
   ceph fs set FS_NAME joinable false
   ```

Example

```
[ceph: root@host01 /]# ceph fs set cephfs max_mds 1
[ceph: root@host01 /]# ceph fs fail cephfs
[ceph: root@host01 /]# ceph status
[ceph: root@host01 /]# ceph fs set cephfs joinable false
```

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:

   Example

   ```
   [ceph: root@host01 /]# ceph osd set noout
   [ceph: root@host01 /]# ceph osd set norecover
   [ceph: root@host01 /]# ceph osd set norebalance
   [ceph: root@host01 /]# ceph osd set nobackfill
   [ceph: root@host01 /]# ceph osd set nodown
   [ceph: root@host01 /]# ceph osd set pause
   ```

5. If the MDS and RGW Ceph Object Gateway nodes are on their own dedicated nodes, power them off.

6. Shut down the OSD nodes one by one:

   Example

   ```
   [root@host01 ~]# systemctl stop ceph-499829b4-832f-11eb-8d6d-
   001a4a000635@osd.2.service
   ```

7. Shut down the monitor nodes one by one:

   Example

   ```
   [root@host01 ~]#  systemctl stop ceph-499829b4-832f-11eb-8d6d-
   001a4a000635@mon.host01.service
   ```

8. Shutdown the admin node.

Rebooting the Red Hat Ceph Storage cluster

1. If network equipment was involved, ensure it is powered on and stable prior to powering on any Ceph hosts or nodes.

2. Power on the administration node.

3. Power on the monitor nodes:

   Example

   ```
   [root@host01 ~]# systemctl start ceph-499829b4-832f-11eb-8d6d-
   001a4a000635@mon.host01.service
   ```

4. Power on the OSD nodes:

Example

```
[root@host01 ~]# systemctl start ceph-499829b4-832f-11eb-8d6d-
001a4a000635@osd.2.service
```

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

6. 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:

   Example

   ```
   [ceph: root@host01 /]# ceph osd unset noout
   [ceph: root@host01 /]# ceph osd unset norecover
   [ceph: root@host01 /]# ceph osd unset norebalance
   [ceph: root@host01 /]# ceph osd unset nobackfill
   [ceph: root@host01 /]# ceph osd unset nodown
   [ceph: root@host01 /]# ceph osd unset pause
   ```

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

   Syntax

   ```
   ceph fs set FS_NAME joinable true
   ```

   Example

   ```
   [ceph: root@host01 /]# ceph fs set cephfs joinable true
   ```

8. 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.

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.

   Syntax

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

   Replace

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

Example

```
[root@host01 ~]# podman exec -it ceph-499829b4-832f-11eb-8d6d-001a4a000635-mon.host01 /bin/bash
```

This example opens an interactive terminal session on **mon.host01**, 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. Log into the Cephadm shell:

   Example

   ```
   root@host01 ~]# cephadm shell
   ```

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

   Example

   ```
   [ceph: root@host01 /]# ceph health
   HEALTH_OK
   ```

3. You can check the status of the Ceph storage cluster by running **ceph status** command:

   Example

   ```
   [ceph: root@host01 /]# ceph status
   ```

   The output provides the following information:

   - 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 status of Ceph Managers.
   - The status of Object Gateways.

- 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.
  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. Log into the Cephadm shell:

   Example

   ```
   root@host01 ~]# cephadm shell
   ```

2. To watch the cluster's ongoing events, run the following command:

   Example

   ```
   [ceph: root@host01 /]# ceph -w
     cluster:
       id:     8c9b0072-67ca-11eb-af06-001a4a0002a0
       health: HEALTH_OK

     services:
       mon: 2 daemons, quorum Ceph5-2,Ceph5-adm (age 3d)
       mgr: Ceph5-1.nqikfh(active, since 3w), standbys: Ceph5-adm.meckej
       osd: 5 osds: 5 up (since 2d), 5 in (since 8w)
       rgw: 2 daemons active (test_realm.test_zone.Ceph5-2.bfdwcn,
   test_realm.test_zone.Ceph5-adm.acndrh)

     data:
       pools:   11 pools, 273 pgs
       objects: 459 objects, 32 KiB
       usage:   2.6 GiB used, 72 GiB / 75 GiB avail
       pgs:     273 active+clean

     io:
       client:   170 B/s rd, 730 KiB/s wr, 0 op/s rd, 729 op/s wr
   ```

```
2021-06-02 15:45:21.655871 osd.0 [INF] 17.71 deep-scrub ok
2021-06-02 15:45:47.880608 osd.1 [INF] 1.0 scrub ok
2021-06-02 15:45:48.865375 osd.1 [INF] 1.3 scrub ok
2021-06-02 15:45:50.866479 osd.1 [INF] 1.4 scrub ok
2021-06-02 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
2021-06-02 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
2021-06-02 15:45:53.997726 osd.1 [INF] 1.5 scrub ok
2021-06-02 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
2021-06-02 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
2021-06-02 15:46:06.836430 osd.0 [INF] 17.75 deep-scrub ok
2021-06-02 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
```

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

```
[ceph: root@host01 /]# 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

```
[ceph: root@host01 /]# 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 **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. It is **STORED** size * replication factor.
- **USED:** The notional amount of data stored in kilobytes, unless the number appends **M** for megabytes or **G** for gigabytes.
- **%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 MB without considering the replication factor, which is three by default.

  See the Red Hat Knowledgebase article titled *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.

Example

```
[ceph: root@host01 /]# 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 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. Log into the Cephadm shell:

   Example

   ```
   [root@host01 ~]# cephadm shell
   ```

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

   Example

   ```
   [ceph: root@host01 /]# ceph status
   ```

   Or

   Example

   ```
   [ceph: root@host01 /]# ceph -s
   ```

3. In interactive mode, type **ceph** and press **Enter**:

   Example

   ```
   [ceph: root@host01 /]# ceph
   ceph> status
     cluster:
       id:     499829b4-832f-11eb-8d6d-001a4a000635
       health: HEALTH_WARN
               1 stray daemon(s) not managed by cephadm
               1/3 mons down, quorum host03,host02
               too many PGs per OSD (261 > max 250)

     services:
       mon:     3 daemons, quorum host03,host02 (age 3d), out of quorum: host01
       mgr:     host01.hdhzwn(active, since 9d), standbys: host05.eobuuv, host06.wquwpj
       osd:     12 osds: 11 up (since 2w), 11 in (since 5w)
       rgw:     2 daemons active (test_realm.test_zone.host04.hgbvnq,
   test_realm.test_zone.host05.yqqilm)
       rgw-nfs: 1 daemon active (nfs.foo.host06-rgw)

     data:
       pools:   8 pools, 960 pgs
       objects: 414 objects, 1.0 MiB
       usage:   5.7 GiB used, 214 GiB / 220 GiB avail
   ```

```
    pgs:     960 active+clean

  io:
    client:   41 KiB/s rd, 0 B/s wr, 41 op/s rd, 27 op/s wr

ceph> health
HEALTH_WARN 1 stray daemon(s) not managed by cephadm; 1/3 mons down, quorum
host03,host02; too many PGs per OSD (261 > max 250)

ceph> mon stat
e3: 3 mons at {host01=[v2:10.74.255.0:3300/0,v1:10.74.255.0:6789/0],host02=
[v2:10.74.249.253:3300/0,v1:10.74.249.253:6789/0],host03=
[v2:10.74.251.164:3300/0,v1:10.74.251.164:6789/0]}, election epoch 6688, leader 1 host03,
quorum 1,2 host03,host02
```

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 you can 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 Ceph Monitors are running.

Check the 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 can prevent Ceph clients from reading and writing data.

Prerequisites

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

Procedure

1. Log into the Cephadm shell:

   Example

   ```
   [root@host01 ~]# cephadm shell
   ```

2. To display the Ceph Monitor map, execute the following:

   Example

   ```
   [ceph: root@host01 /]# ceph mon stat
   ```

   or

   Example

   ```
   [ceph: root@host01 /]# ceph mon dump
   ```

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

```
[ceph: root@host01 /]# ceph quorum_status -f json-pretty
```

Ceph returns the quorum status.

Example


```
{
    "election_epoch": 6686,
    "quorum": [
        0,
        1,
        2
    ],
    "quorum_names": [
        "host01",
        "host03",
        "host02"
    ],
    "quorum_leader_name": "host01",
    "quorum_age": 424884,
    "features": {
        "quorum_con": "4540138297136906239",
        "quorum_mon": [
            "kraken",
            "luminous",
            "mimic",
            "osdmap-prune",
            "nautilus",
            "octopus",
            "pacific",
            "elector-pinging"
        ]
    },
    "monmap": {
        "epoch": 3,
        "fsid": "499829b4-832f-11eb-8d6d-001a4a000635",
        "modified": "2021-03-15T04:51:38.621737Z",
        "created": "2021-03-12T12:35:16.911339Z",
        "min_mon_release": 16,
        "min_mon_release_name": "pacific",
        "election_strategy": 1,
        "disallowed_leaders: ": "",
        "stretch_mode": false,
        "features": {
            "persistent": [
                "kraken",
                "luminous",
                "mimic",
                "osdmap-prune",
                "nautilus",
                "octopus",
                "pacific",
                "elector-pinging"
            ],
            "optional": []
```

```
},
"mons": [
    {
        "rank": 0,
        "name": "host01",
        "public_addrs": {
            "addrvec": [
                {
                    "type": "v2",
                    "addr": "10.74.255.0:3300",
                    "nonce": 0
                },
                {
                    "type": "v1",
                    "addr": "10.74.255.0:6789",
                    "nonce": 0
                }
            ]
        },
        "addr": "10.74.255.0:6789/0",
        "public_addr": "10.74.255.0:6789/0",
        "priority": 0,
        "weight": 0,
        "crush_location": "{}"
    },
    {
        "rank": 1,
        "name": "host03",
        "public_addrs": {
            "addrvec": [
                {
                    "type": "v2",
                    "addr": "10.74.251.164:3300",
                    "nonce": 0
                },
                {
                    "type": "v1",
                    "addr": "10.74.251.164:6789",
                    "nonce": 0
                }
            ]
        },
        "addr": "10.74.251.164:6789/0",
        "public_addr": "10.74.251.164:6789/0",
        "priority": 0,
        "weight": 0,
        "crush_location": "{}"
    },
    {
        "rank": 2,
        "name": "host02",
        "public_addrs": {
            "addrvec": [
                {
                    "type": "v2",
                    "addr": "10.74.249.253:3300",
```

```
                    "nonce": 0
                },
                {
                    "type": "v1",
                    "addr": "10.74.249.253:6789",
                    "nonce": 0
                }
            ]
        },
        "addr": "10.74.249.253:6789/0",
        "public_addr": "10.74.249.253:6789/0",
        "priority": 0,
        "weight": 0,
        "crush_location": "{}"
    }
  ]
 }
}
```

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 Ceph Monitors or OSDs.

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

```
Error 111: Connection Refused
```

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.

Prerequisites

- A running Red Hat Ceph Storage cluster.

- Root-level access to the node.

Procedure

1. Log into the Cephadm shell:

   Example

   ```
   [root@host01 ~]# cephadm shell
   ```

2. To use the socket:

   Syntax

   ```
   ceph daemon MONITOR_ID COMMAND
   ```

   Replace:

   - ***MONITOR_ID*** of the daemon
   - ***COMMAND*** with the command to run. Use **help** to list the available commands for a given daemon.
     To view the status of a Ceph Monitor:

     Example

     ```
     [ceph: root@host01 /]# ceph daemon mon.host01 help
     {
         "add_bootstrap_peer_hint": "add peer address as potential bootstrap peer for cluster bringup",
         "add_bootstrap_peer_hintv": "add peer address vector as potential bootstrap peer for cluster bringup",
         "compact": "cause compaction of monitor's leveldb/rocksdb storage",
         "config diff": "dump diff of current config and default config",
         "config diff get": "dump diff get <field>: dump diff of current and default config setting <field>",
         "config get": "config get <field>: get the config value",
         "config help": "get config setting schema and descriptions",
         "config set": "config set <field> <val> [<val> ...]: set a config variable",
         "config show": "dump current config settings",
         "config unset": "config unset <field>: unset a config variable",
         "connection scores dump": "show the scores used in connectivity-based elections",
         "connection scores reset": "reset the scores used in connectivity-based elections",
         "dump_historic_ops": "dump_historic_ops",
         "dump_mempools": "get mempool stats",
         "get_command_descriptions": "list available commands",
         "git_version": "get git sha1",
         "heap": "show heap usage info (available only if compiled with tcmalloc)",
         "help": "list available commands",
         "injectargs": "inject configuration arguments into running daemon",
         "log dump": "dump recent log entries to log file",
         "log flush": "flush log entries to log file",
         "log reopen": "reopen log file",
         "mon_status": "report status of monitors",
         "ops": "show the ops currently in flight",
     ```

```
    "perf dump": "dump perfcounters value",
    "perf histogram dump": "dump perf histogram values",
    "perf histogram schema": "dump perf histogram schema",
    "perf reset": "perf reset <name>: perf reset all or one perfcounter name",
    "perf schema": "dump perfcounters schema",
    "quorum enter": "force monitor back into quorum",
    "quorum exit": "force monitor out of the quorum",
    "sessions": "list existing sessions",
    "smart": "Query health metrics for underlying device",
    "sync_force": "force sync of and clear monitor store",
    "version": "get ceph version"
}
```

Example

```
[ceph: root@host01 /]# ceph daemon mon.host01 mon_status

{
    "name": "host01",
    "rank": 0,
    "state": "leader",
    "election_epoch": 120,
    "quorum": [
        0,
        1,
        2
    ],
    "quorum_age": 206358,
    "features": {
        "required_con": "2449958747317026820",
        "required_mon": [
            "kraken",
            "luminous",
            "mimic",
            "osdmap-prune",
            "nautilus",
            "octopus",
            "pacific",
            "elector-pinging"
        ],
        "quorum_con": "4540138297136906239",
        "quorum_mon": [
            "kraken",
            "luminous",
            "mimic",
            "osdmap-prune",
            "nautilus",
            "octopus",
            "pacific",
            "elector-pinging"
        ]
    },
    "outside_quorum": [],
    "extra_probe_peers": [],
    "sync_provider": [],
    "monmap": {
```

```
"epoch": 3,
"fsid": "81a4597a-b711-11eb-8cb8-001a4a000740",
"modified": "2021-05-18T05:50:17.782128Z",
"created": "2021-05-17T13:13:13.383313Z",
"min_mon_release": 16,
"min_mon_release_name": "pacific",
"election_strategy": 1,
"disallowed_leaders: ": "",
"stretch_mode": false,
"features": {
    "persistent": [
        "kraken",
        "luminous",
        "mimic",
        "osdmap-prune",
        "nautilus",
        "octopus",
        "pacific",
        "elector-pinging"
    ],
    "optional": []
},
"mons": [
    {
        "rank": 0,
        "name": "host01",
        "public_addrs": {
            "addrvec": [
                {
                    "type": "v2",
                    "addr": "10.74.249.41:3300",
                    "nonce": 0
                },
                {
                    "type": "v1",
                    "addr": "10.74.249.41:6789",
                    "nonce": 0
                }
            ]
        },
        "addr": "10.74.249.41:6789/0",
        "public_addr": "10.74.249.41:6789/0",
        "priority": 0,
        "weight": 0,
        "crush_location": "{}"
    },
    {
        "rank": 1,
        "name": "host02",
        "public_addrs": {
            "addrvec": [
                {
                    "type": "v2",
                    "addr": "10.74.249.55:3300",
                    "nonce": 0
                },
```

```
                    {
                        "type": "v1",
                        "addr": "10.74.249.55:6789",
                        "nonce": 0
                    }
                ]
            },
            "addr": "10.74.249.55:6789/0",
            "public_addr": "10.74.249.55:6789/0",
            "priority": 0,
            "weight": 0,
            "crush_location": "{}"
        },
        {
            "rank": 2,
            "name": "host03",
            "public_addrs": {
                "addrvec": [
                    {
                        "type": "v2",
                        "addr": "10.74.249.49:3300",
                        "nonce": 0
                    },
                    {
                        "type": "v1",
                        "addr": "10.74.249.49:6789",
                        "nonce": 0
                    }
                ]
            },
            "addr": "10.74.249.49:6789/0",
            "public_addr": "10.74.249.49:6789/0",
            "priority": 0,
            "weight": 0,
            "crush_location": "{}"
        }
    ]
},
"feature_map": {
    "mon": [
        {
            "features": "0x3f01cfb9fffdffff",
            "release": "luminous",
            "num": 1
        }
    ],
    "osd": [
        {
            "features": "0x3f01cfb9fffdffff",
            "release": "luminous",
            "num": 3
        }
    ]
},
"stretch_mode": false
}
```

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

   Syntax

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

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

   Example

   ```
   [ceph: root@host01 /]# ceph daemon /var/run/ceph/ceph-osd.2.asok status
   ```

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

   Example

   ```
   [ceph: root@host01 /]# ls /var/run/ceph
   ```

Additional Resources

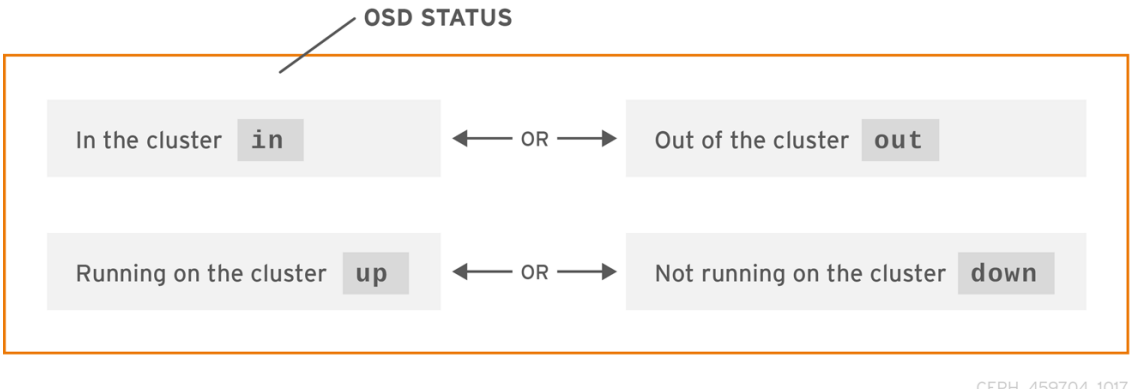
- See the *Red Hat Ceph Storage Troubleshooting Guide* for more information.

3.2.11. Understanding the Ceph OSD status

A Ceph OSD's status is either **in** the storage cluster, or **out** of the storage cluster. It is either **up** and running, or it is **down** and not running. If a Ceph OSD is **up**, it can 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 storage cluster and recently moved **out** of the storage cluster, Ceph starts migrating placement groups to other Ceph OSDs. If a Ceph OSD is **out** of the storage cluster, CRUSH will not assign placement groups to the Ceph OSD. If a Ceph OSD is **down**, it should also be **out**.

NOTE

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



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

- You have not started the storage cluster yet, and it is not responding.

- You have just started or restarted the storage cluster, and it is not ready yet, because the placement groups are getting created and the Ceph OSDs are in the process of peering.
- You just added or removed a Ceph OSD.
- You just modified the storage cluster map.

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

To see if all OSDs are running, execute:

Example

```
[ceph: root@host01 /]# ceph osd stat
```

or

Example

```
[ceph: root@host01 /]# 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 Ceph OSDs that are **in** the storage cluster are more than the number of Ceph OSDs that are **up**. Execute the following command to identify the **ceph-osd** daemons that are not running:

Example

```
[ceph: root@host01 /]# ceph osd tree

# 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 can help you troubleshoot the storage cluster by identifying the physical locations faster.

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

Syntax

```
systemctl start CEPH_OSD_SERVICE_ID
```

Example

```
[root@host01 ~]# systemctl start ceph-499829b4-832f-11eb-8d6d-001a4a000635@osd.6.service
```

Additional Resources

- See the *Red Hat Ceph Storage Dashboard Guide* for more details.

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 Ceph OSDs, it looks at the number of replicas for the pool and assigns the placement group to Ceph OSDs such that each replica of the placement group gets assigned to a different Ceph 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 Ceph OSDs in a large cluster. We refer to the set of Ceph 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, do not panic. Common examples include:

- You added or removed an OSD. Then, CRUSH reassigned the placement group to other Ceph OSDs, thereby changing the composition of the acting set and spawning the migration of data with a "backfill" process.
- A Ceph OSD was **down**, was restarted and is now **recovering**.
- A Ceph OSD in the acting set is **down** or unable to service requests, and another Ceph OSD has temporarily assumed its duties.

Ceph processes a client request using the **Up Set**, which is the set of Ceph OSDs that actually handle the requests. In most cases, the up set and the Acting Set are virtually identical. When they are not, it can indicate that Ceph is migrating data, an Ceph 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. Log into the Cephadm shell:

   Example

   ```
   [root@host01 ~]# cephadm shell
   ```

2. To retrieve a list of placement groups:

   Example

   ```
   [ceph: root@host01 /]# ceph pg dump
   ```

3. View which Ceph OSDs are in the **Acting Set** or in the **Up Set** for a given placement group:

   Syntax

   ```
   ceph pg map PG_NUM
   ```

   Example

   ```
   [ceph: root@host01 /]# ceph pg map 128
   ```

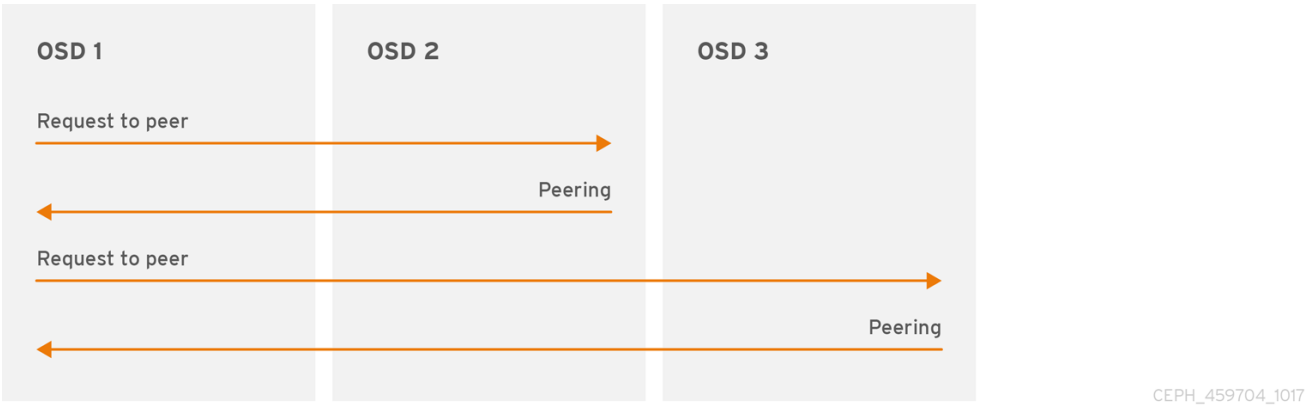
NOTE

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

Figure 3.1. Peering



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:

Example

```
[ceph: root@host01 /]# 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:

Syntax

```
POOL_NUM.PG_ID
```

Example output:

```
0.1f
```

- To retrieve a list of placement groups:

  #### Example

  ```
  [ceph: root@host01 /]# ceph pg dump
  ```

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

  #### Syntax

  ```
  ceph pg dump -o FILE_NAME --format=json
  ```

  #### Example

  ```
  [ceph: root@host01 /]# ceph pg dump -o test --format=json
  ```

- Query a particular placement group:

  #### Syntax

  ```
  ceph pg POOL_NUM.PG_ID query
  ```

  #### Example

  ```
  [ceph: root@host01 /]#  ceph pg 5.fe query
  {
      "snap_trimq": "[]",
      "snap_trimq_len": 0,
      "state": "active+clean",
      "epoch": 2449,
      "up": [
          3,
          8,
  ```

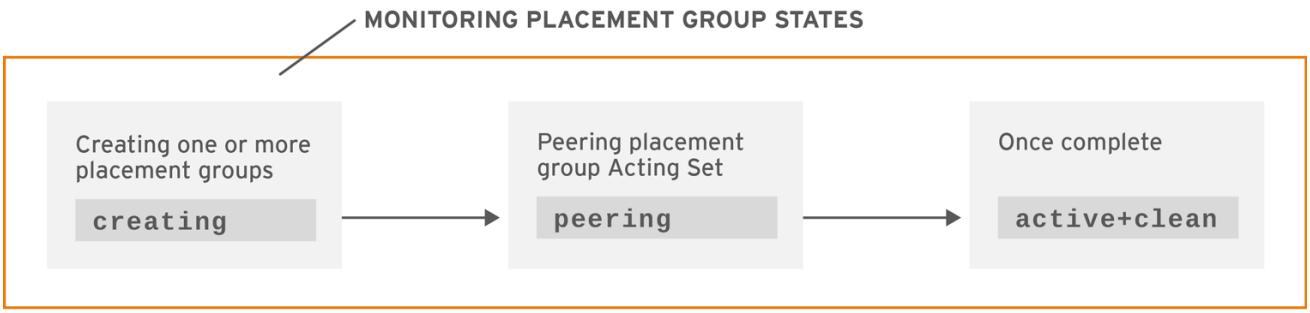
```
            10
        ],
        "acting": [
            3,
            8,
            10
        ],
        "acting_recovery_backfill": [
            "3",
            "8",
            "10"
        ],
        "info": {
            "pgid": "5.ff",
            "last_update": "0'0",
            "last_complete": "0'0",
            "log_tail": "0'0",
            "last_user_version": 0,
            "last_backfill": "MAX",
            "purged_snaps": [],
            "history": {
                "epoch_created": 114,
                "epoch_pool_created": 82,
                "last_epoch_started": 2402,
                "last_interval_started": 2401,
                "last_epoch_clean": 2402,
                "last_interval_clean": 2401,
                "last_epoch_split": 114,
                "last_epoch_marked_full": 0,
                "same_up_since": 2401,
                "same_interval_since": 2401,
                "same_primary_since": 2086,
                "last_scrub": "0'0",
                "last_scrub_stamp": "2021-06-17T01:32:03.763988+0000",
                "last_deep_scrub": "0'0",
                "last_deep_scrub_stamp": "2021-06-17T01:32:03.763988+0000",
                "last_clean_scrub_stamp": "2021-06-17T01:32:03.763988+0000",
                "prior_readable_until_ub": 0
            },
            "stats": {
                "version": "0'0",
                "reported_seq": "2989",
                "reported_epoch": "2449",
                "state": "active+clean",
                "last_fresh": "2021-06-18T05:16:59.401080+0000",
                "last_change": "2021-06-17T01:32:03.764162+0000",
                "last_active": "2021-06-18T05:16:59.401080+0000",
....
```


Additional Resources

- See the chapter *Object Storage Daemon (OSD) configuration options* in the *OSD Object storage daemon configuratopn options* section in *Red Hat Ceph Storage 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.



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 Ceph OSDs must peer again when the Ceph 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.

For example, if there are nine OSDs in a three way replica pool. If OSD number 9 goes down, the PGs assigned to OSD 9 goes into a degraded state. If OSD 9 does not recover, it goes out of the storage cluster and the storage 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 is not always trivial, because a hardware failure might cause a cascading failure of multiple Ceph 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 storage 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 Ceph OSDs can fail, restart and re-peer at staggered rates. The **osd recovery max active** setting limits the number of recovery requests a Ceph OSD works on simultaneously to prevent the Ceph 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 Ceph OSD joins the storage cluster, CRUSH will reassign placement groups from OSDs in the cluster to the newly added Ceph OSD. Forcing the new OSD to accept the reassigned placement groups immediately can put excessive load on the new Ceph 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 might 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 can be considered **incomplete**.

Ceph provides a number of settings to manage the load spike associated with reassigning placement groups to a Ceph OSD, especially a new Ceph OSD. By default, **osd_max_backfills** sets the maximum number of concurrent backfills to or from a Ceph OSD to 10. The **osd backfill full ratio** enables a Ceph 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. You can force a backfill rather than a recovery by setting the **osd_min_pg_log_entries** option to **1**, and setting the **osd_max_pg_log_entries** option to **2**. Contact your Red Hat Support account team for details on when this situation is appropriate for your workload.

3.3.12. 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.

3.3.13. 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.

3.3.14. 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.

3.3.15. 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.

3.3.16. Identifying stuck Placement Groups

A placement group is not necessarily problematic just because it is not in a **active+clean** state. Generally, Ceph's ability to self repair might 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:

Syntax

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

Example

```
[ceph: root@host01 /]# ceph pg dump_stuck stale
OK
```

3.3.17. 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:

   #### Syntax

   ```
   ceph osd map POOL_NAME OBJECT_NAME
   ```

   #### Example

   ```
   [ceph: root@host01 /]# ceph osd map mypool myobject
   ```

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:

   Syntax

   ```
   ceph osd set FLAG
   ```

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

   Example

   ```
   [ceph: root@host01 /]# ceph osd set noout
   ```

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

   Syntax

   ```
   ceph osd unset FLAG
   ```

   Example

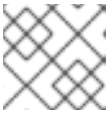
   ```
   [ceph: root@host01 /]# ceph osd unset noout
   ```

Flag	Description
noin	Prevents OSDs from being treated as **in** the cluster.

Flag	Description
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.
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**.

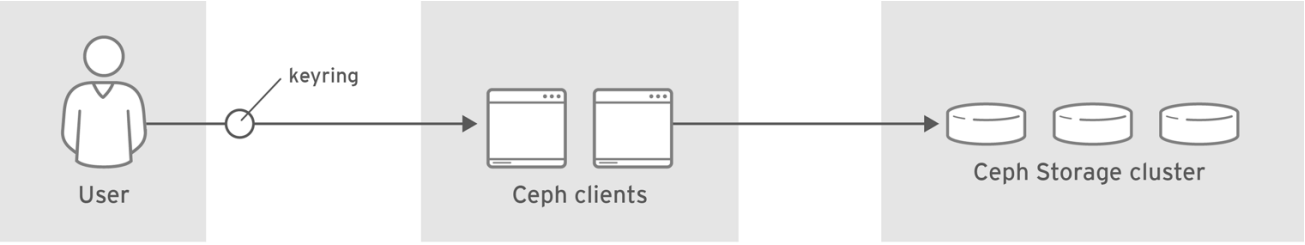
- **norecover**: If you need to replace an OSD disk and don't want the PGs to recover to another OSD while you are hotswapping disks, you can set **norecover** to prevent the other OSDs from copying a new set of PGs to other OSDs.

- **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, and access control to objects in the Red Hat Ceph Storage cluster.



5.1. PREREQUISITES

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

IMPORTANT

Cephadm manages the client keyrings for the Red Hat Ceph Storage cluster as long as the clients are within the scope of Cephadm. Users should not modify the keyrings that are managed by Cephadm, unless there is troubleshooting.

5.2. CEPH USER MANAGEMENT BACKGROUND

When Ceph runs with authentication and authorization enabled, you must specify a user name. If you do not specify a user name, Ceph will use the **client.admin** administrative user as the default user name.

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.

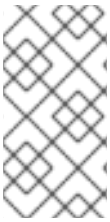
Storage Cluster Users

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. :

  #### Syntax

  ```
  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:

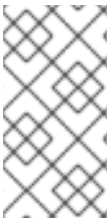
   Example

```
[ceph: root@host01 /]# ceph auth list
installed auth entries:

osd.10
 key: AQBW7U5gqOsEExAAg/CxSwZ/gSh8iOsDV3iQOA==
 caps: [mgr] allow profile osd
 caps: [mon] allow profile osd
 caps: [osd] allow *
osd.11
 key: AQBX7U5gtj/JIhAAPsLBNG+SfC2eMVEFkl3vfA==
 caps: [mgr] allow profile osd
 caps: [mon] allow profile osd
 caps: [osd] allow *
```

```
osd.9
 key: AQBV7U5g1XDULhAAKo2tw6ZhH1jki5aVui2v7g==
 caps: [mgr] allow profile osd
 caps: [mon] allow profile osd
 caps: [osd] allow *
client.admin
 key: AQADYEtgFfD3ExAAwH+C1qO7MSLE4TWRfD2g6g==
 caps: [mds] allow *
 caps: [mgr] allow *
 caps: [mon] allow *
 caps: [osd] allow *
client.bootstrap-mds
 key: AQAHYEtgpbkANBAANqoFlvzEXFwD8oB0w3TF4Q==
 caps: [mon] allow profile bootstrap-mds
client.bootstrap-mgr
 key: AQAHYEtg3dcANBAAVQf6brq3sxTSrCrPe0pKVQ==
 caps: [mon] allow profile bootstrap-mgr
client.bootstrap-osd
 key: AQAHYEtgD/QANBAATS9DuP3DbxEl86MTyKEmdw==
 caps: [mon] allow profile bootstrap-osd
client.bootstrap-rbd
 key: AQAHYEtgjxEBNBAANho25V9tWNNvIKnHknW59A==
 caps: [mon] allow profile bootstrap-rbd
client.bootstrap-rbd-mirror
 key: AQAHYEtgdE8BNBAAr6rLYxZci0b2hoIgH9GXYw==
 caps: [mon] allow profile bootstrap-rbd-mirror
client.bootstrap-rgw
 key: AQAHYEtgwGkBNBAAuRzI4WSrnowBhZxr2XtTFg==
 caps: [mon] allow profile bootstrap-rgw
client.crash.host04
 key: AQCQYEtgz8lGGhAAy5bJS8VH9fMdxuAZ3CqX5Q==
 caps: [mgr] profile crash
 caps: [mon] profile crash
client.crash.host02
 key: AQDuYUtgqgfdOhAAsyX+Mo35M+HFpURGad7nJA==
 caps: [mgr] profile crash
 caps: [mon] profile crash
client.crash.host03
 key: AQB98E5g5jHZAxAAklWSvmDsh2JaL5G7FvMrrA==
 caps: [mgr] profile crash
 caps: [mon] profile crash
client.nfs.foo.host03
 key: AQCgTk9gm+HvMxAAHbjG+XpdwL6prM/uMcdPdQ==
 caps: [mon] allow r
 caps: [osd] allow rw pool=nfs-ganesha namespace=foo
client.nfs.foo.host03-rgw
 key: AQCgTk9g8sJQNhAAPykcoYUuPc7IjubaFx09HQ==
 caps: [mon] allow r
 caps: [osd] allow rwx tag rgw *=*
client.rgw.test_realm.test_zone.host01.hgbvnq
 key: AQD5RE9gAQKdCRAAJzxDwD/dJObbInp9J95sXw==
 caps: [mgr] allow rw
 caps: [mon] allow *
 caps: [osd] allow rwx tag rgw *=*
client.rgw.test_realm.test_zone.host02.yqqilm
 key: AQD0RE9gkxA4ExAAFXp3pLJWdIhsyTe2ZR6Ilw==
```

```
 caps: [mgr] allow rw
 caps: [mon] allow *
 caps: [osd] allow rwx tag rgw *=*
mgr.host01.hdhzwn
 key: AQAEYEtg3lhIBxAAmHodoIpdvnxK0llWF80ltQ==
 caps: [mds] allow *
 caps: [mon] profile mgr
 caps: [osd] allow *
mgr.host02.eobuuv
 key: AQAn6U5gzUuiABAA2Fed+jPM1xwb4XDYtrQxaQ==
 caps: [mds] allow *
 caps: [mon] profile mgr
 caps: [osd] allow *
mgr.host03.wquwpj
 key: AQAd6U5gIzWsLBAAbOKUKZlUcAVe9kBLfajMKw==
 caps: [mds] allow *
 caps: [mon] profile mgr
 caps: [osd] allow *
```



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:

   Syntax

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

   Example

   ```
   [ceph: root@host01 /]# ceph auth export mgr.host02.eobuuv
   ```

2. You can also use the **-o** ***FILE_NAME*** option.

   Syntax

```
ceph auth export TYPE.ID -o FILE_NAME
```

Example

```
[ceph: root@host01 /]# ceph auth export osd.9 -o filename
export auth(key=AQBV7U5g1XDULhAAKo2tw6ZhH1jki5aVui2v7g==)
```

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**).

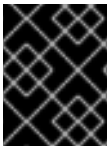
There are a few ways to add a user:

- **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. :

```
[ceph: root@host01 /]# ceph auth add client.john mon 'allow r' osd 'allow rw pool=mypool'
[ceph: root@host01 /]# ceph auth get-or-create client.paul mon 'allow r' osd 'allow rw pool=mypool'
[ceph: root@host01 /]# ceph auth get-or-create client.george mon 'allow r' osd 'allow rw pool=mypool'
-o george.keyring
[ceph: root@host01 /]# ceph auth get-or-create-key client.ringo mon 'allow r' osd 'allow rw
pool=mypool' -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.

Prerequisites

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

Procedure

1. To add capabilities, use the form:

   Syntax

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

   Example

   ```
   [ceph: root@host01 /]# ceph auth caps client.john mon 'allow r' osd 'allow rw pool=mypool'
   [ceph: root@host01 /]# ceph auth caps client.paul mon 'allow rw' osd 'allow rwx pool=mypool'
   [ceph: root@host01 /]# ceph auth caps client.brian-manager mon 'allow *' osd 'allow *'
   ```

2. To remove a capability, you may reset the capability. If you want the user to have no access to a particular daemon that was previously set, specify an empty string:

   Example

   ```
   [ceph: root@host01 /]# ceph auth caps client.ringo mon ' ' osd ' '
   ```

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**:

   Syntax

   ```
   ceph auth del TYPE.ID
   ```

-

Example

```
[ceph: root@host01 /]# ceph auth del osd.6
```

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:

   Syntax

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

   Example

   ```
   [ceph: root@host01 /]# ceph auth print-key osd.6

   AQBQ7U5gAry3JRAA3NoPrqBBThpFMcRL6Sr+5w==[ceph: root@host01 /]#
   ```

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

   Syntax

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

   Example

   ```
   [ceph: root@host01 /]#  mount -t ceph host02:/ceph -o name=client.user,secret=`ceph auth
   print-key client.user`
   ```

CHAPTER 6. THE CEPH-VOLUME UTILITY

As a storage administrator, you can prepare, list, create, activate, deactivate, batch, trigger, zap, and migrate 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 deploy 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.

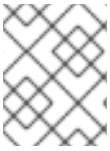
To use the LVM plugin, add **lvm** as a subcommand to the **ceph-volume** command within the cephadm shell:

```
[ceph: root@host01 /]# ceph-volume lvm
```

Following are the **lvm** subcommands:

- **prepare** - Format an LVM device and associate it with an OSD.
- **activate** - Discover and mount the LVM device associated with an OSD ID and start the Ceph OSD.
- **list** - List logical volumes and devices associated with Ceph.
- **batch** - Automatically size devices for multi-OSD provisioning with minimal interaction.
- **deactivate** - Deactivate OSDs.
- **create** - Create a new OSD from an LVM device.
- **trigger** - A systemd helper to activate an OSD.
- **zap** - Removes all data and filesystems from a logical volume or partition.
- **migrate** - Migrate BlueFS data from to another LVM device.

- **new-wal** - Allocate new WAL volume for the OSD at specified logical volume.
- **new-db** - Allocate new DB volume for the OSD at specified logical volume.



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 5, **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

   ```
   [ceph: root@host01 /]# 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

```
[ceph: root@host01 /]# 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

```
[ceph: root@host01 /]# 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. LISTING DEVICES USING CEPH-VOLUME

You can use the **ceph-volume lvm list** subcommand to list logical volumes and devices associated with a Ceph cluster, as long as they contain enough metadata to allow for that discovery. The output is grouped by the OSD ID associated with the devices. For logical volumes, the **devices key** is populated with the physical devices associated with the logical volume.

Prerequisites

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

Procedure

- List the devices in the Ceph cluster:

  Example

```
[ceph: root@host01 /]# ceph-volume lvm list


====== osd.6 =======

  [block]       /dev/ceph-83909f70-95e9-4273-880e-5851612cbe53/osd-block-7ce687d9-07e7-4f8f-a34e-d1b0efb89920

      block device              /dev/ceph-83909f70-95e9-4273-880e-5851612cbe53/osd-block-7ce687d9-07e7-4f8f-a34e-d1b0efb89920
      block uuid                4d7gzX-Nzxp-UUG0-bNxQ-Jacr-l0mP-IPD8cX
      cephx lockbox secret
      cluster fsid              1ca9f6a8-d036-11ec-8263-fa163ee967ad
      cluster name              ceph
      crush device class        None
      encrypted                 0
      osd fsid                  7ce687d9-07e7-4f8f-a34e-d1b0efb89920
      osd id                    6
      osdspec affinity          all-available-devices
      type                      block
      vdo                       0
      devices                   /dev/vdc
```

6.6. 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 OSD FSID from an OSD node:

   ```
   [ceph: root@host01 /]# ceph-volume lvm list
   ```

2. Activate the OSD:

   #### Syntax

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

   #### Example

   ```
   [ceph: root@host01 /]# ceph-volume lvm activate --bluestore 10 7ce687d9-07e7-4f8f-a34e-d1b0efb89920
   ```

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

Example

```
[ceph: root@host01 /]# ceph-volume lvm activate --all
```

3. Optionally, you can use the **trigger** subcommand. This command cannot be used directly, and it is used by **systemd** so that it proxies input to **ceph-volume lvm activate**. This parses the metadata coming from systemd and startup, detecting the UUID and ID associated with an OSD.

   Syntax

   ```
   ceph-volume lvm trigger SYSTEMD_DATA
   ```

   Here the *SYSTEMD_DATA* is in *OSD_ID-OSD_FSID* format.

   Example

   ```
   [ceph: root@host01 /]# ceph-volume lvm trigger 10 7ce687d9-07e7-4f8f-a34e-d1b0efb89920
   ```

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.

6.7. DEACTIVATING CEPH OSDS USING CEPH-VOLUME

You can deactivate the Ceph OSDs using the **ceph-volume lvm** subcommand. This subcommand removes the volume groups and the logical volume.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the Ceph OSD node.
- The Ceph OSDs are activated using the **ceph-volume** utility.

Procedure

1. Get the OSD ID from the OSD node:

   ```
   [ceph: root@host01 /]# ceph-volume lvm list
   ```

2. Deactivate the OSD:

   Syntax

```
ceph-volume lvm deactivate OSD_ID
```

Example

```
[ceph: root@host01 /]# ceph-volume lvm deactivate 16
```

Additional Resources

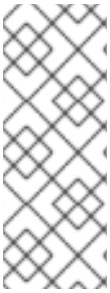
- See the *Activating Ceph OSDs using `ceph-volume`* section in the *Red Hat Ceph Storage Administration Guide* for more details.
- See the *Preparing Ceph OSDs using `ceph-volume`* section in the *Red Hat Ceph Storage Administration Guide* for more details.
- See the *Creating Ceph OSDs using `ceph-volume`*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

6.8. 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.

6.9. MIGRATING BLUEFS DATA

You can migrate the BlueStore file system (BlueFS) data, that is the RocksDB data, from the source volume to the target volume using the **migrate** LVM subcommand. The source volume, except the main one, is removed on success.

LVM volumes are primarily for the target only.

The new volumes are attached to the OSD, replacing one of the source drives.

Following are the placement rules for the LVM volumes:

- If source list has DB or WAL volume, then the target device replaces it.
- if source list has slow volume only, then explicit allocation using the **new-db** or **new-wal** command is needed.

The **new-db** and **new-wal** commands attaches the given logical volume to the given OSD as a DB or a WAL volume respectively.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to the Ceph OSD node.
- Ceph OSDs prepared by the **ceph-volume** utility.
- Volume groups and Logical volumes are created.

Procedure

1. Log in the **cephadm** shell:

   #### Example

   ```
   [root@host01 ~]# cephadm shell
   ```

2. Stop the OSD to which you have to add the DB or the WAL device:

   #### Example

   ```
   [ceph: root@host01 /]# ceph orch stop osd.1
   ```

3. Mount the new devices to the container:

   #### Example

   ```
   [root@host01 ~]# cephadm shell --mount /var/lib/ceph/72436d46-ca06-11ec-9809-ac1f6b5635ee/osd.1:/var/lib/ceph/osd/ceph-1
   ```

4. Attach the given logical volume to OSD as a DB/WAL device:

NOTE

This command fails if the OSD has an attached DB.

Syntax

```
ceph-volume lvm new-db --osd-id OSD_ID --osd-fsid OSD_FSID --target
VOLUME_GROUP_NAME/LOGICAL_VOLUME_NAME
```

Example

```
[ceph: root@host01 /]# ceph-volume lvm new-db --osd-id 1 --osd-fsid 7ce687d9-07e7-4f8f-
a34e-d1b0efb89921 --target vgname/new_db
[ceph: root@host01 /]# ceph-volume lvm new-wal --osd-id 1 --osd-fsid 7ce687d9-07e7-4f8f-
a34e-d1b0efb89921 --target vgname/new_wal
```

5. You can migrate BlueFS data in the following ways:

- Move BlueFS data from main device to LV that is already attached as DB:

  Syntax

  ```
  ceph-volume lvm migrate --osd-id OSD_ID --osd-fsid OSD_UUID --from data --target
  VOLUME_GROUP_NAME/LOGICAL_VOLUME_NAME
  ```

  Example

  ```
  [ceph: root@host01 /]# ceph-volume lvm migrate --osd-id 1 --osd-fsid 0263644D-0BF1-
  4D6D-BC34-28BD98AE3BC8 --from data --target vgname/db
  ```

- Move BlueFS data from shared main device to LV which shall be attached as a new DB:

  Syntax

  ```
  ceph-volume lvm migrate --osd-id OSD_ID --osd-fsid OSD_UUID --from data --target
  VOLUME_GROUP_NAME/LOGICAL_VOLUME_NAME
  ```

  Example

  ```
  [ceph: root@host01 /]# ceph-volume lvm migrate --osd-id 1 --osd-fsid 0263644D-0BF1-
  4D6D-BC34-28BD98AE3BC8 --from data --target vgname/new_db
  ```

- Move BlueFS data from DB device to new LV, and replace the DB device:

  Syntax

  ```
  ceph-volume lvm migrate --osd-id OSD_ID --osd-fsid OSD_UUID --from db --target
  VOLUME_GROUP_NAME/LOGICAL_VOLUME_NAME
  ```

Example

```
[ceph: root@host01 /]# ceph-volume lvm migrate --osd-id 1 --osd-fsid 0263644D-0BF1-4D6D-BC34-28BD98AE3BC8 --from db --target vgname/new_db
```

- Move BlueFS data from main and DB devices to new LV, and replace the DB device:

  Syntax

  ```
  ceph-volume lvm migrate --osd-id OSD_ID --osd-fsid OSD_UUID --from data db --target VOLUME_GROUP_NAME/LOGICAL_VOLUME_NAME
  ```

  Example

  ```
  [ceph: root@host01 /]# ceph-volume lvm migrate --osd-id 1 --osd-fsid 0263644D-0BF1-4D6D-BC34-28BD98AE3BC8 --from data db --target vgname/new_db
  ```

- Move BlueFS data from main, DB, and WAL devices to new LV, remove the WAL device, and replace the the DB device:

  Syntax

  ```
  ceph-volume lvm migrate --osd-id OSD_ID --osd-fsid OSD_UUID --from data db wal --target VOLUME_GROUP_NAME/LOGICAL_VOLUME_NAME
  ```

  Example

  ```
  [ceph: root@host01 /]# ceph-volume lvm migrate --osd-id 1 --osd-fsid 0263644D-0BF1-4D6D-BC34-28BD98AE3BC8 --from data db --target vgname/new_db
  ```

- Move BlueFS data from main, DB, and WAL devices to the main device, remove the WAL and DB devices:

  Syntax

  ```
  ceph-volume lvm migrate --osd-id OSD_ID --osd-fsid OSD_UUID --from db wal --target VOLUME_GROUP_NAME/LOGICAL_VOLUME_NAME
  ```

  Example

  ```
  [ceph: root@host01 /]# ceph-volume lvm migrate --osd-id 1 --osd-fsid 0263644D-0BF1-4D6D-BC34-28BD98AE3BC8 --from db wal --target vgname/data
  ```

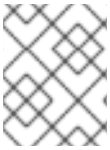
6.10. USING BATCH MODE WITH CEPH-VOLUME

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

   ```
   [ceph: root@host01 /]# 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.

6.11. ZAPPING DATA USING CEPH-VOLUME

The **zap** subcommand removes all data and filesystems from a logical volume or partition.

You can use the **zap** subcommand to zap logical volumes, partitions, or raw devices that are used by Ceph OSDs for reuse. Any filesystems present on the given logical volume or partition are removed and all data is purged.

Optionally, you can use the **--destroy** flag for complete removal of a logical volume, partition, or the physical device.

Prerequisites

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

Procedure

- Zap the logical volume:

Syntax

```
ceph-volume lvm zap VOLUME_GROUP_NAME/LOGICAL_VOLUME_NAME [--destroy]
```

Example

```
[ceph: root@host01 /]# ceph-volume lvm zap osd-vg/data-lv
```

- Zap the partition:

  Syntax

  ```
  ceph-volume lvm zap DEVICE_PATH_PARTITION [--destroy]
  ```

  Example

  ```
  [ceph: root@host01 /]# ceph-volume lvm zap /dev/sdc1
  ```

- Zap the raw device:

  Syntax

  ```
  ceph-volume lvm zap DEVICE_PATH --destroy
  ```

  Example

  ```
  [ceph: root@host01 /]# ceph-volume lvm zap /dev/sdc --destroy
  ```

- Purge multiple devices with the OSD ID:

  Syntax

  ```
  ceph-volume lvm zap --destroy --osd-id OSD_ID
  ```

  Example

  ```
  [ceph: root@host01 /]# ceph-volume lvm zap --destroy --osd-id 16
  ```

  NOTE

  All the relative devices are zapped.

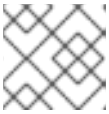
- Purge OSDs with the FSID:

  Syntax

  ```
  ceph-volume lvm zap --destroy --osd-fsid OSD_FSID
  ```

  Example

```
[ceph: root@host01 /]# ceph-volume lvm zap --destroy --osd-fsid 65d7b6b1-e41a-4a3c-b363-83ade63cb32b
```



NOTE

All the relative devices are zapped.

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

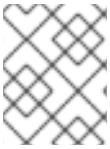
The OSD, including the journal, disks and the network throughput should each have a performance baseline to compare against. You can identify potential tuning opportunities by comparing the baseline performance data with the data from Ceph's native tools. Red Hat Enterprise Linux has many built-in tools, 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:

Example

```
[ceph: root@host01 /]# 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:

   Example

```
[ceph: root@host01 /]# ceph osd pool create testbench 100 100
```

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

Example

```
[ceph: root@host01 /]# rados bench -p testbench 10 write --no-cleanup

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:

Example

```
[ceph: root@host01 /]# rados bench -p testbench 10 seq

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:

   Example

```
[ceph: root@host01 /]# rados bench -p testbench 10 rand

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 might 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.

   Example

```
[ceph: root@host01 /]# rados bench -p testbench 10 write -t 4 --run-name client1

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:

   Example

   ```
   [ceph: root@host01 /]# 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:

   Example

   ```
   [root@host01 ~]# modprobe rbd
   ```

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

   Example

```
[root@host01 ~]# rbd create image01 --size 1024 --pool testbench
```

3. Map the image file to a device file:

   Example

```
[root@host01 ~]# rbd map image01 --pool testbench --name client.admin
```

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

   Example

```
[root@host01 ~]# mkfs.ext4 /dev/rbd/testbench/image01
```

5. Create a new directory:

   Example

```
[root@host01 ~]# mkdir /mnt/ceph-block-device
```

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

   Example

```
[root@host01 ~]# mount /dev/rbd/testbench/image01 /mnt/ceph-block-device
```

7. Execute the write performance test against the block device

   Example

```
[root@host01 ~]# rbd bench --io-type write image01 --pool=testbench

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 *Ceph block devices* chapter 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 *DAEMON_NAME* 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:

   Synatx

   ```
   ceph daemon DAEMON_NAME perf schema
   ```

   NOTE

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

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

   Example

   ```
   [ceph: root@host01 /]# ceph daemon mon.host01 perf schema
   ```

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

   Example

   ```
   [ceph: root@host01 /]# ceph daemon osd.11 perf schema
   ```

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* section in the *Red Hat Ceph Storage Administration Guide* 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:

   Syntax

   ```
   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: root@host01 /]# ceph daemon mon.host01 perf dump
   ```

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

   ```
   [ceph: root@host01 /]# ceph daemon osd.11 perf dump
   ```

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

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

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
	num_pg	2	Total number of placement groups

Collection Name	Metric Name	Bit Field Value	Short Description
	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
	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 8.4. 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
	refresh	10	Refreshes

Collection Name	Metric Name	Bit Field Value	Short Description
	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
	store_state_keys	6	Keys in transaction in stored state

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

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
	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
	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

Collection Name	Metric Name	Bit Field Value	Short Description
	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
	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

Collection Name	Metric Name	Bit Field Value	Short Description
	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

Collection Name	Metric Name	Bit Field Value	Short Description
	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
	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)

Collection Name	Metric Name	Bit Field Value	Short Description
	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
	loadavg	2	CPU load
	buffer_bytes	2	Total allocated buffer size
	numpg	2	Placement groups

Collection Name	Metric Name	Bit Field Value	Short Description
	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
	tier_evict	10	Tier evictions
	tier_whiteout	10	Tier whiteouts
	tier_dirty	10	Dirty tier flag set

Collection Name	Metric Name	Bit Field Value	Short Description
	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
	waitremotebackfillreserved_latency	5	Wait remote backfill reserved recovery state latency

Collection Name	Metric Name	Bit Field Value	Short Description
	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
	getlog_latency	5	Getlog recovery state latency

Collection Name	Metric Name	Bit Field Value	Short Description
	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

8.8. CEPH OBJECT GATEWAY METRICS

- *Ceph Object Gateway Client Table*
- *Objecter Metrics Table*
- *Ceph Object Gateway Throttle Metrics Table*

Table 8.13. Ceph Object Gateway 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

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

Collection Name	Metric Name	Bit Field Value	Short Description
	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
	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

Collection Name	Metric Name	Bit Field Value	Short Description
	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
	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

Collection Name	Metric Name	Bit Field Value	Short Description
	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

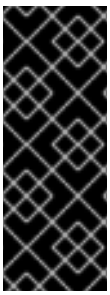
Table 8.15. Ceph Object Gateway Throttle Metrics Table

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

Collection Name	Metric Name	Bit Field Value	Short Description
	take_sum	10	Taken data
	put	10	Puts
	put_sum	10	Put data
	wait	5	Waiting latency

CHAPTER 9. BLUESTORE

BlueStore is the back-end object store for the OSD daemons and 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.

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.

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

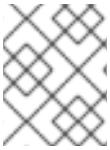
Please contact Red Hat support before enabling the **bluestore_default_buffered_write** option.

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.



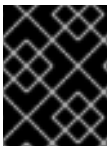
NOTE

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

9.5. TUNING CEPH BLUESTORE USING BLUESTORE_MIN_ALLOC_SIZE PARAMETER

In BlueStore, the raw partition is allocated and managed in chunks of **bluestore_min_alloc_size**. By default, **bluestore_min_alloc_size** is **4096**, equivalent to 4 KiB for HDDs and 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.



IMPORTANT

Changing the value of **bluestore_min_alloc_size** is not recommended. For any assistance, contact *Red Hat support*.

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.
- Ceph monitors and managers are deployed in the cluster.

- Servers or nodes that can be freshly provisioned as OSD nodes
- The admin keyring for the Ceph Monitor node, if you are redeploying an existing Ceph OSD node.

Procedure

1. On the bootstrapped node, change the value of **bluestore_min_alloc_size** parameter:

   Syntax

   ```
   ceph config set osd.OSD_ID bluestore_min_alloc_size_DEVICE_NAME_ VALUE
   ```

   Example

   ```
   [ceph: root@host01 /]# ceph config set osd.4 bluestore_min_alloc_size_hdd 6144
   ```

   You can see **bluestore_min_alloc_size** is set to 6144 bytes, which is equivalent to 6 KiB.

2. Restart the OSD's service.

   Syntax

   ```
   systemctl restart SERVICE_ID
   ```

   Example

   ```
   [ceph: root@host01 /]# systemctl restart ceph-499829b4-832f-11eb-8d6d-
   001a4a000635@osd.4.service
   ```

Verification

- Verify the setting using the **ceph daemon** command:

  Syntax

  ```
  ceph daemon osd.OSD_ID config get bluestore_min_alloc_size__DEVICE_
  ```

  Example

  ```
  [ceph: root@host01 /]# ceph daemon osd.4 config get bluestore_min_alloc_size_hdd

  ceph daemon osd.4 config get bluestore_min_alloc_size
  {
      "bluestore_min_alloc_size": "6144"
  }
  ```

9.6. RESHARDING THE ROCKSDB DATABASE USING THE BLUESTORE ADMIN TOOL

You can reshard the database with the BlueStore admin tool. It transforms BlueStore's RocksDB

database from one shape to another into several column families without redeploying the OSDs. Column families have the same features as the whole database, but allows users to operate on smaller data sets and apply different options. It leverages the different expected lifetime of keys stored. The keys are moved during the transformation without creating new keys or deleting existing keys.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- The object store configured as BlueStore.
- OSD nodes deployed on the hosts.
- Root level access to the all the hosts.
- The **ceph-common** and **cephadm** packages instaled on all the hosts.

Procedure

1. Log into the **cephadm** shell:

   Example

   ```
   [root@host01 ~]# cephadm shell
   ```

2. Fetch the *OSD_ID* and the host details from the administration node:

   Example

   ```
   [ceph: root@host01 /]# ceph orch ps
   ```

3. Log into the respective host as a **root** user and stop the OSD:

   Syntax

   ```
   cephadm unit --name OSD_ID stop
   ```

   Example

   ```
   [root@host02 ~]# cephadm unit --name osd.0 stop
   ```

4. Enter into the stopped OSD daemon container:

   Syntax

   ```
   cephadm shell --name OSD_ID
   ```

   Example

   ```
   [root@host02 ~]# cephadm shell --name osd.0
   ```

5. Log into the **cephadm shell** and check the file system consistency:

Syntax

```
ceph-bluestore-tool --path/var/lib/ceph/osd/ceph-OSD_ID/ fsck
```

Example

```
[ceph: root@host02 /]# ceph-bluestore-tool --path /var/lib/ceph/osd/ceph-0/ fsck

fsck success
```

6. Check the sharding status of the OSD node:

Syntax

```
ceph-bluestore-tool --path /var/lib/ceph/osd/ceph-OSD_ID/ show-sharding
```

Example

```
[ceph: root@host02 /]# ceph-bluestore-tool --path /var/lib/ceph/osd/ceph-6/ show-sharding

m(3) p(3,0-12) O(3,0-13) L P
```

7. Run the **ceph-bluestore-tool** command to reshard. Red Hat recommends to use the parameters as given in the command:

Syntax

```
ceph-bluestore-tool --log-level 10 -l log.txt --path /var/lib/ceph/osd/ceph-OSD_ID/ --
sharding="m(3) p(3,0-12) O(3,0-13)=block_cache={type=binned_lru} L P" reshard
```

Example

```
[ceph: root@host02 /]# ceph-bluestore-tool --path /var/lib/ceph/osd/ceph-6/ --sharding="m(3)
p(3,0-12) O(3,0-13)=block_cache={type=binned_lru} L P" reshard

reshard success
```

8. To check the sharding status of the OSD node, run the **show-sharding** command:

Syntax

```
ceph-bluestore-tool --path /var/lib/ceph/osd/ceph-OSD_ID/ show-sharding
```

Example

```
[ceph: root@host02 /]# ceph-bluestore-tool --path /var/lib/ceph/osd/ceph-6/ show-sharding

m(3) p(3,0-12) O(3,0-13)=block_cache={type=binned_lru} L P
```

9. Exit from the **cephadm** shell:

```
[ceph: root@host02 /]# exit
```

10. Log into the respective host as a **root** user and start the OSD:

    Syntax

    ```
    cephadm unit --name OSD_ID start
    ```

    Example

    ```
    [root@host02 ~]# cephadm unit --name osd.0 start
    ```

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 cluster.
- BlueStore OSDs.

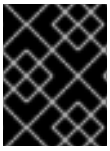
9.7.2. What is the BlueStore fragmentation tool?

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 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

      ```
      [ceph: root@host01 /]# ceph daemon osd.123 bluestore allocator score block
      ```

   b. A more detailed report:

      Syntax

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

      Example

      ```
      [ceph: root@host01 /]# ceph daemon osd.123 bluestore allocator dump block
      ```

Offline BlueStore fragmentation score

1. Follow the steps for resharding for checking the offline fragmentation score.
   Example

```
[root@host01 ~]# podman exec -it 7fbd6c6293c0 /bin/bash
```

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@7fbd6c6293c0 /]# 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
block:
{
    "fragmentation_rating": 0.018290238194701977
}
```

Example

```
[root@7fbd6c6293c0 /]# ceph-bluestore-tool --path /var/lib/ceph/osd/ceph-123 --allocator block free-dump
block:
{
    "capacity": 21470642176,
    "alloc_unit": 4096,
    "alloc_type": "hybrid",
    "alloc_name": "block",
    "extents": [
        {
            "offset": "0x370000",
            "length": "0x20000"
        },
        {
            "offset": "0x3a0000",
            "length": "0x10000"
        },
        {
            "offset": "0x3f0000",
            "length": "0x20000"
        },
        {
            "offset": "0x460000",
            "length": "0x10000"
        },
```

Additional Resources

- See the *BlueStore Fragmentation Tool* for details on the fragmentation score.
- See the *Resharding the RocskDB database using the BlueStore admin tool* for details on resharding.

9.8. CEPH BLUESTORE BLUEFS

BlueStore block database stores metadata as key-value pairs in a RocksDB database. The block database resides on a small BlueFS partition on the storage device. BlueFS is a minimal file system that is designed to hold the RocksDB files.

9.8.1. Viewing the bluefs_buffered_io setting

As a storage administrator, you can view the current setting for the **bluefs_buffered_io** parameter.

The option **bluefs_buffered_io** is set to **True** by default for Red Hat Ceph Storage. This option enable BlueFS to perform buffered reads in some cases, and enables the kernel page cache to act as a secondary cache for reads like RocksDB block reads.

IMPORTANT

Changing the value of **bluefs_buffered_io** is not recommended. Before changing the **bluefs_buffered_io** parameter, contact your Red Hat Support account team.

Prerequisites

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

Procedure

1. Log into the Cephadm shell:

   Example

   ```
   [root@host01 ~]# cephadm shell
   ```

2. You can view the current value of the **bluefs_buffered_io** parameter in three different ways:

Method 1

- View the value stored in the configuration database:

  Example

  ```
  [ceph: root@host01 /]# ceph config get osd bluefs_buffered_io
  ```

Method 2

- View the value stored in the configuration database for a specific OSD:

  Syntax

  ```
  ceph config get OSD_ID bluefs_buffered_io
  ```

  Example

```
[ceph: root@host01 /]# ceph config get osd.2 bluefs_buffered_io
```

Method 3

- View the running value for an OSD where the running value is different from the value stored in the configuration database:

  Syntax

  ```
  ceph config show OSD_ID bluefs_buffered_io
  ```

  Example

  ```
  [ceph: root@host01 /]# ceph config show osd.3 bluefs_buffered_io
  ```

CHAPTER 10. CEPHADM TROUBLESHOOTING

As a storage administrator, you can troubleshoot the Red Hat Ceph Storage cluster. Sometimes there is a need to investigate why a Cephadm command failed or why a specific service does not run properly.

10.1. PREREQUISITES

- A running Red Hat Ceph Storage cluster.

10.2. PAUSE OR DISABLE CEPHADM

If Cephadm does not behave as expected, you can pause most of the background activity with the following command:

Example

```
[ceph: root@host01 /]# ceph orch pause
```

This stops any changes, but Cephadm periodically checks hosts to refresh it's inventory of daemons and devices.

If you want to disable Cephadm completely, run the following command:

Example

```
[ceph: root@host01 /]# ceph orch backend
ceph mgr module disable cephadm
```

Note that previously deployed daemon containers continue to exist and start as they did before.

10.3. PER SERVICE AND PER DAEMON EVENT

Cephadm stores events per service and per daemon in order to aid in debugging failed daemon deployments. These events often contain relevant information:

Per service

Syntax

```
ceph orch ls --service_name SERVICE_NAME --format yaml
```

Example

```
[ceph: root@host01 /]# ceph orch ls --service_name alertmanager --format yaml
service_type: alertmanager
service_name: alertmanager
placement:
  hosts:
  - unknown_host
status:
  ...
  running: 1
```

```
  size: 1
events:
- 2021-02-01T08:58:02.741162 service:alertmanager [INFO] "service was created"
- '2021-02-01T12:09:25.264584 service:alertmanager [ERROR] "Failed to apply: Cannot
  place <AlertManagerSpec for service_name=alertmanager> on unknown_host: Unknown hosts"'
```

Per daemon

Syntax

```
ceph orch ps --service-name SERVICE_NAME --daemon-id DAEMON_ID --format yaml
```

Example

```
[ceph: root@host01 /]# ceph orch ps --service-name mds --daemon-id cephfs.hostname.ppdhsz --
format yaml
daemon_type: mds
daemon_id: cephfs.hostname.ppdhsz
hostname: hostname
status_desc: running
...
events:
- 2021-02-01T08:59:43.845866 daemon:mds.cephfs.hostname.ppdhsz [INFO] "Reconfigured
  mds.cephfs.hostname.ppdhsz on host 'hostname'"
```

10.4. CHECK CEPHADM LOGS

You can monitor the Cephadm log in real time with the following command:

Example

```
[ceph: root@host01 /]# ceph -W cephadm
```

You can see the last few messages with the following command:

Example

```
[ceph: root@host01 /]# ceph log last cephadm
```

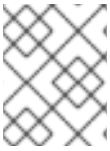
If you have enabled logging to files, you can see a Cephadm log file called **ceph.cephadm.log** on the monitor hosts.

10.5. GATHER LOG FILES

You can use the **journalctl** command, to gather the log files for all the daemons.

NOTE

You have to run all these commands outside the **cephadm** shell.



NOTE

By default, Cephadm stores logs in journald which means that daemon logs are no longer available in **/var/log/ceph**.

- To read the log file of a specific daemon, run the following command:

  Syntax

  ```
  cephadm logs --name DAEMON_NAME
  ```

  Example

  ```
  [root@host01 ~]# cephadm logs --name cephfs.hostname.ppdhsz
  ```

NOTE

This command works when run on the same hosts where the daemon is running.

- To read the log file of a specific daemon running on a different host, run the following command:

  Syntax

  ```
  cephadm logs --fsid FSID --name DAEMON_NAME
  ```

  Example

  ```
  [root@host01 ~]# cephadm logs --fsid 2d2fd136-6df1-11ea-ae74-002590e526e8 --name
  cephfs.hostname.ppdhsz
  ```

  where **fsid** is the cluster ID provided by the **ceph status** command.

- To fetch all log files of all the daemons on a given host, run the following command:

  Syntax

  ```
  for name in $(cephadm ls | python3 -c "import sys, json; [print(i['name']) for i in
  json.load(sys.stdin)]") ; do cephadm logs --fsid FSID_OF_CLUSTER --name "$name" >
  $name; done
  ```

  Example

  ```
  [root@host01 ~]# for name in $(cephadm ls | python3 -c "import sys, json; [print(i['name']) for
  i in json.load(sys.stdin)]") ; do cephadm logs --fsid 57bddb48-ee04-11eb-9962-
  001a4a000672 --name "$name" > $name; done
  ```

10.6. COLLECT SYSTEMD STATUS

- To print the state of a systemd unit, run the following command:

Example

```
[root@host01 ~]$ systemctl status ceph-a538d494-fb2a-48e4-82c8-
b91c37bb0684@mon.host01.service
```

10.7. LIST ALL DOWNLOADED CONTAINER IMAGES

To list all the container images that are downloaded on a host, run the following command:

Example

```
[ceph: root@host01 /]# podman ps -a --format json | jq '.[].Image'
"docker.io/library/rhel8"
"registry.redhat.io/rhceph-alpha/rhceph-5-
rhel8@sha256:9aaea414e2c263216f3cdcb7a096f57c3adf6125ec9f4b0f5f65fa8c43987155"
```

10.8. MANUALLY RUN CONTAINERS

Cephadm writes small wrappers that runs a container. Refer to **/var/lib/ceph/*CLUSTER_FSID*/*SERVICE_NAME*/unit** to run the container execution command.

Analysing SSH errors

If you get the following error:

Example

```
execnet.gateway_bootstrap.HostNotFound: -F /tmp/cephadm-conf-73z09u6g -i /tmp/cephadm-
identity-ky7ahp_5 root@10.10.1.2
...
raise OrchestratorError(msg) from e
orchestrator._interface.OrchestratorError: Failed to connect to 10.10.1.2 (10.10.1.2).
Please make sure that the host is reachable and accepts connections using the cephadm SSH key
```

Try the following options to troubleshoot the issue:

- To ensure Cephadm has a SSH identity key, run the following command:

  Example

  ```
  [ceph: root@host01 /]# ceph config-key get mgr/cephadm/ssh_identity_key >
  ~/cephadm_private_key
  INFO:cephadm:Inferring fsid f8edc08a-7f17-11ea-8707-000c2915dd98
  INFO:cephadm:Using recent ceph image docker.io/ceph/ceph:v15 obtained
  'mgr/cephadm/ssh_identity_key'
  [root@mon1 ~] # chmod 0600 ~/cephadm_private_key
  ```

  If the above command fails, Cephadm does not have a key. To generate a SSH key, run the following command:

  Example

  ```
  [ceph: root@host01 /]# chmod 0600 ~/cephadm_private_key
  ```

Or

Example

```
[ceph: root@host01 /]# cat ~/cephadm_private_key | ceph cephadm set-ssk-key -i-
```

- To ensure that the SSH configuration is correct, run the following command:

  Example

  ```
  [ceph: root@host01 /]# ceph cephadm get-ssh-config
  ```

- To verify the connection to the host, run the following command:

  Example

  ```
  [ceph: root@host01 /]# ssh -F config -i ~/cephadm_private_key root@host01
  ```

Verify public key is in authorized_keys.

To verify that the public key is in the **authorized_keys** file, run the following commands:

Example

```
[ceph: root@host01 /]# ceph cephadm get-pub-key
[ceph: root@host01 /]# grep "`cat ~/ceph.pub`" /root/.ssh/authorized_keys
```

10.9. CIDR NETWORK ERROR

Classless inter domain routing (CIDR) also known as supernetting, is a method of assigning Internet Protocol (IP) addresses,FThe Cephadm log entries shows the current state that improves the efficiency of address distribution and replaces the previous system based on Class A, Class B and Class C networks. If you see one of the following errors:

ERROR: Failed to infer CIDR network for mon ip *; pass --skip-mon-network to configure it later

Or

Must set public_network config option or specify a CIDR network, ceph addrvec, or plain IP

You need to run the following command:

Example

```
[ceph: root@host01 /]# ceph config set host public_network hostnetwork
```

10.10. ACCESS THE ADMIN SOCKET

Each Ceph daemon provides an admin socket that bypasses the MONs.

To access the admin socket, enter the daemon container on the host:

Example

```
[ceph: root@host01 /]# cephadm enter --name cephfs.hostname.ppdhsz
[ceph: root@mon1 /]# ceph --admin-daemon /var/run/ceph/ceph-cephfs.hostname.ppdhsz.asok
config show
```

10.11. MANUALLY DEPLOYING A MGR DAEMON

Cephadm requires a **mgr** daemon in order to manage the Red Hat Ceph Storage cluster. In case the last **mgr** daemon of a Red Hat Ceph Storage cluster was removed, you can manually deploy a **mgr** daemon, on a random host of the Red Hat Ceph Storage cluster.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Root-level access to all the nodes.
- Hosts are added to the cluster.

Procedure

1. Log into the Cephadm shell:

   Example

   ```
   [root@host01 ~]# cephadm shell
   ```

2. Disable the Cephadm scheduler to prevent Cephadm from removing the new MGR daemon, with the following command:

   Example

   ```
   [ceph: root@host01 /]# ceph config-key set mgr/cephadm/pause true
   ```

3. Get or create the **auth** entry for the new MGR daemon:

   Example

   ```
   [ceph: root@host01 /]# ceph auth get-or-create mgr.host01.smfvfd1 mon "profile mgr" osd
   "allow *" mds "allow *"
   [mgr.host01.smfvfd1]
   key = AQDhcORgW8toCRAAlMzlqWXnh3cGRjqYEa9ikw==
   ```

4. Open **ceph.conf** file:

   Example

   ```
   [ceph: root@host01 /]# ceph config generate-minimal-conf
   # minimal ceph.conf for 8c9b0072-67ca-11eb-af06-001a4a0002a0
   [global]
   fsid = 8c9b0072-67ca-11eb-af06-001a4a0002a0
   mon_host = [v2:10.10.200.10:3300/0,v1:10.10.200.10:6789/0]
   [v2:10.10.10.100:3300/0,v1:10.10.200.100:6789/0]
   ```

5. Get the container image:

   Example

   ```
   [ceph: root@host01 /]# ceph config get "mgr.host01.smfvfd1" container_image
   ```

6. Create a **config-json.json** file and add the following:

   NOTE

   Use the values from the output of the **ceph config generate-minimal-conf** command.

   Example

   ```
   {
    {
    "config": "# minimal ceph.conf for 8c9b0072-67ca-11eb-af06-001a4a0002a0\n[global]\n\tfsid
   = 8c9b0072-67ca-11eb-af06-001a4a0002a0\n\tmon_host =
   [v2:10.10.200.10:3300/0,v1:10.10.200.10:6789/0]
   [v2:10.10.10.100:3300/0,v1:10.10.200.100:6789/0]\n",
    "keyring": "[mgr.Ceph5-2.smfvfd1]\n\tkey =
   AQDhcORgW8toCRAAIMzlqWXnh3cGRjqYEa9ikw==\n"
   }
   }
   ```

7. Exit from the Cephadm shell:

   Example

   ```
   [ceph: root@host01 /]# exit
   ```

8. Deploy the MGR daemon:

   Example

   ```
   [root@host01 ~]# cephadm --image registry.redhat.io/rhceph-alpha/rhceph-5-rhel8:latest
   deploy --fsid  8c9b0072-67ca-11eb-af06-001a4a0002a0 --name mgr.host01.smfvfd1 --config-
   json config-json.json
   ```

Verification

In the Cephadm shell, run the following command:

Example

```
[ceph: root@host01 /]# ceph -s
```

You can see a new **mgr** daemon has been added.

CHAPTER 11. CEPHADM OPERATIONS

As a storage administrator, you can carry out Cephadm operations in the Red Hat Ceph Storage cluster.

11.1. PREREQUISITES

- A running Red Hat Ceph Storage cluster.

11.2. MONITOR CEPHADM LOG MESSAGES

Cephadm logs to the cephadm cluster log channel so you can monitor progress in real time.

- To monitor progress in realtime, run the following command:

  Example

  ```
  [ceph: root@host01 /]# ceph -W cephadm
  ```

  Example

  ```
  2022-06-10T17:51:36.335728+0000 mgr.Ceph5-1.nqikfh [INF] refreshing Ceph5-adm facts
  2022-06-10T17:51:37.170982+0000 mgr.Ceph5-1.nqikfh [INF] deploying 1 monitor(s) instead
  of 2 so monitors may achieve consensus
  2022-06-10T17:51:37.173487+0000 mgr.Ceph5-1.nqikfh [ERR] It is NOT safe to stop
  ['mon.Ceph5-adm']: not enough monitors would be available (Ceph5-2) after stopping mons
  [Ceph5-adm]
  2022-06-10T17:51:37.174415+0000 mgr.Ceph5-1.nqikfh [INF] Checking pool "nfs-ganesha"
  exists for service nfs.foo
  2022-06-10T17:51:37.176389+0000 mgr.Ceph5-1.nqikfh [ERR] Failed to apply nfs.foo spec
  NFSServiceSpec({'placement': PlacementSpec(count=1), 'service_type': 'nfs', 'service_id':
  'foo', 'unmanaged': False, 'preview_only': False, 'pool': 'nfs-ganesha', 'namespace': 'nfs-ns'}):
  Cannot find pool "nfs-ganesha" for service nfs.foo
  Traceback (most recent call last):
    File "/usr/share/ceph/mgr/cephadm/serve.py", line 408, in _apply_all_services
      if self._apply_service(spec):
    File "/usr/share/ceph/mgr/cephadm/serve.py", line 509, in _apply_service
      config_func(spec)
    File "/usr/share/ceph/mgr/cephadm/services/nfs.py", line 23, in config
      self.mgr._check_pool_exists(spec.pool, spec.service_name())
    File "/usr/share/ceph/mgr/cephadm/module.py", line 1840, in _check_pool_exists
      raise OrchestratorError(f'Cannot find pool "{pool}" for '
  orchestrator._interface.OrchestratorError: Cannot find pool "nfs-ganesha" for service nfs.foo
  2022-06-10T17:51:37.179658+0000 mgr.Ceph5-1.nqikfh [INF] Found osd claims -> {}
  2022-06-10T17:51:37.180116+0000 mgr.Ceph5-1.nqikfh [INF] Found osd claims for
  drivegroup all-available-devices -> {}
  2022-06-10T17:51:37.182138+0000 mgr.Ceph5-1.nqikfh [INF] Applying all-available-devices
  on host Ceph5-adm...
  2022-06-10T17:51:37.182987+0000 mgr.Ceph5-1.nqikfh [INF] Applying all-available-devices
  on host Ceph5-1...
  2022-06-10T17:51:37.183395+0000 mgr.Ceph5-1.nqikfh [INF] Applying all-available-devices
  on host Ceph5-2...
  2022-06-10T17:51:43.373570+0000 mgr.Ceph5-1.nqikfh [INF] Reconfiguring node-
  ```

```
exporter.Ceph5-1 (unknown last config time)...
2022-06-10T17:51:43.373840+0000 mgr.Ceph5-1.nqikfh [INF] Reconfiguring daemon node-
exporter.Ceph5-1 on Ceph5-1
```

- By default, the log displays info-level events and above. To see the debug-level messages, run the following commands:

  Example

```
[ceph: root@host01 /]# ceph config set mgr mgr/cephadm/log_to_cluster_level debug
[ceph: root@host01 /]# ceph -W cephadm --watch-debug
[ceph: root@host01 /]# ceph -W cephadm --verbose
```

- Return debugging level to default **info**:

  Example

```
[ceph: root@host01 /]# ceph config set mgr mgr/cephadm/log_to_cluster_level info
```

- To see the recent events, run the following command:

  Example

```
[ceph: root@host01 /]# ceph log last cephadm
```

Theses events are also logged to **ceph.cephadm.log** file on the monitor hosts and to the monitor daemon's **stderr**

11.3. CEPH DAEMON LOGS

You can view the Ceph daemon logs through **stderr** or files.

Logging to stdout

Traditionally, Ceph daemons have logged to **/var/log/ceph**. By default, Cephadm daemons log to **stderr** and the logs are captured by the container runtime environment. For most systems, by default, these logs are sent to **journald** and accessible through the **journalctl** command.

- For example, to view the logs for the daemon on **host01** for a storage cluster with ID **5c5a50ae-272a-455d-99e9-32c6a013e694**:

  Example

```
[ceph: root@host01 /]# journalctl -u ceph-5c5a50ae-272a-455d-99e9-
32c6a013e694@host01
```

This works well for normal Cephadm operations when logging levels are low.

- To disable logging to **stderr**, set the following values:

  Example

```
[ceph: root@host01 /]# ceph config set global log_to_stderr false
[ceph: root@host01 /]# ceph config set global mon_cluster_log_to_stderr false
```

Logging to files

You can also configure Ceph daemons to log to files instead of **stderr**. When logging to files, Ceph logs are located in **/var/log/ceph/*CLUSTER_FSID***.

- To enable logging to files, set the follwing values:

  Example

```
[ceph: root@host01 /]# ceph config set global log_to_file true
[ceph: root@host01 /]# ceph config set global mon_cluster_log_to_file true
```

NOTE

Red Hat recommends disabling logging to **stderr** to avoid double logs.

IMPORTANT

Currently log rotation to a non-default path is not supported.

By default, Cephadm sets up log rotation on each host to rotate these files. You can configure the logging retention schedule by modifying **/etc/logrotate.d/ceph.*CLUSTER_FSID***.

11.4. DATA LOCATION

Cephadm daemon data and logs are located in slightly different locations than the older versions of Ceph:

- **/var/log/ceph/*CLUSTER_FSID*** contains all the storage cluster logs. Note that by default Cephadm logs through **stderr** and the container runtime, so these logs are usually not present.
- **/var/lib/ceph/*CLUSTER_FSID*** contains all the cluster daemon data, besides logs.
- **var/lib/ceph/*CLUSTER_FSID*/*DAEMON_NAME*** contains all the data for an specific daemon.
- **/var/lib/ceph/*CLUSTER_FSID*/crash** contains the crash reports for the storage cluster.
- **/var/lib/ceph/*CLUSTER_FSID*/removed** contains old daemon data directories for the stateful daemons, for example monitor or Prometheus, that have been removed by Cephadm.

Disk usage

A few Ceph daemons may store a significant amount of data in **/var/lib/ceph**, notably the monitors and Prometheus daemon, hence Red Hat recommends moving this directory to its own disk, partition, or logical volume so that the root file system is not filled up.

11.5. CEPHADM HEALTH CHECKS

As a storage administrator, you can monitor the Red Hat Ceph Storage cluster with the additional healthchecks provided by the Cephadm module. This is supplementary to the default healthchecks provided by the storage cluster.

11.5.1. Prerequisites

- A running Red Hat Ceph Storage cluster.

11.5.2. Cephadm operations health checks

Healthchecks are executed when the Cephadm module is active. You can get the following health warnings:

CEPHADM_PAUSED

Cephadm background work is paused with the **ceph orch pause** command. Cephadm continues to perform passive monitoring activities such as checking the host and daemon status, but it does not make any changes like deploying or removing daemons. You can resume Cephadm work with the **ceph orch resume** command.

CEPHADM_STRAY_HOST

One or more hosts have running Ceph daemons but are not registered as hosts managed by the Cephadm module. This means that those services are not currently managed by Cephadm, for example, a restart and upgrade that is included in the **ceph orch ps** command. You can manage the host(s) with the **ceph orch host add *HOST_NAME*** command but ensure that SSH access to the remote hosts is configured. Alternatively, you can manually connect to the host and ensure that services on that host are removed or migrated to a host that is managed by Cephadm. You can also disable this warning with the setting **ceph config set mgr mgr/cephadm/warn_on_stray_hosts false**

CEPHADM_STRAY_DAEMON

One or more Ceph daemons are running but are not managed by the Cephadm module. This might be because they were deployed using a different tool, or because they were started manually. Those services are not currently managed by Cephadm, for example, a restart and upgrade that is included in the **ceph orch ps** command.

If the daemon is a stateful one that is a monitor or OSD daeon, these daemons should be adopted by Cephadm. For stateless daemons, you can provision a new daemon with the **ceph orch apply** command and then stop the unmanaged daemon.

You can disable this health warning with the setting **ceph config set mgr mgr/cephadm/warn_on_stray_daemons false**.

CEPHADM_HOST_CHECK_FAILED

One or more hosts have failed the basic Cephadm host check, which verifies that:name: value

- The host is reachable and you can execute Cephadm.
- The host meets the basic prerequisites, like a working container runtime that is Podman , and working time synchronization. If this test fails, Cephadm wont be able to manage the services on that host.

You can manually run this check with the **ceph cephadm check-host *HOST_NAME*** command. You can remove a broken host from management with the **ceph orch host rm *HOST_NAME*** command. You can disable this health warning with the setting **ceph config set mgr mgr/cephadm/warn_on_failed_host_check false**.

11.5.3. Cephadm configuration health checks

Cephadm periodically scans each of the hosts in the storage cluster, to understand the state of the OS, disks, and NICs . These facts are analyzed for consistency across the hosts in the storage cluster to identify any configuration anomalies. The configuration checks are an optional feature.

- You can enable this feature with the following command:

  Example

  ```
  [ceph: root@host01 /]# ceph config set mgr mgr/cephadm/config_checks_enabled true
  ```

The configuration checks are triggered after each host scan, which is for a duration of one minute.

- The **ceph -W cephadm** command shows log entries of the current state and outcome of the configuration checks as follows:
  Disabled state

  Example

  ```
  ALL cephadm checks are disabled, use 'ceph config set mgr
  mgr/cephadm/config_checks_enabled true' to enable
  ```

  Enabled state

  Example

  ```
  CEPHADM 8/8 checks enabled and executed (0 bypassed, 0 disabled). No issues detected
  ```

  The configuration checks themselves are managed through several **cephadm** subcommands.

- To determine whether the configuration checks are enabled, run the following command:

  Example

  ```
  [ceph: root@host01 /]# ceph cephadm config-check status
  ```

  This command returns the status of the configuration checker as either **Enabled** or **Disabled**.

- To list all the configuration checks and their current state, run the following command:

  Example

  ```
  [ceph: root@host01 /]# ceph cephadm config-check ls
  NAME             HEALTHCHECK                      STATUS   DESCRIPTION
  kernel_security  CEPHADM_CHECK_KERNEL_LSM         enabled  checks
  SELINUX/Apparmor profiles are consistent across cluster hosts
  os_subscription  CEPHADM_CHECK_SUBSCRIPTION       enabled  checks subscription
  states are consistent for all cluster hosts
  public_network   CEPHADM_CHECK_PUBLIC_MEMBERSHIP  enabled  check that all hosts
  have a NIC on the Ceph public_netork
  osd_mtu_size     CEPHADM_CHECK_MTU                enabled  check that OSD hosts share a
  common MTU setting
  osd_linkspeed    CEPHADM_CHECK_LINKSPEED          enabled  check that OSD hosts
  share a common linkspeed
  network_missing  CEPHADM_CHECK_NETWORK_MISSING    enabled  checks that the
  ```

```
cluster/public networks defined exist on the Ceph hosts
ceph_release    CEPHADM_CHECK_CEPH_RELEASE      enabled  check for Ceph version
consistency - ceph daemons should be on the same release (unless upgrade is active)
kernel_version  CEPHADM_CHECK_KERNEL_VERSION    enabled  checks that the
MAJ.MIN of the kernel on Ceph hosts is consistent
```

Each configuration check is described as follows:

CEPHADM_CHECK_KERNEL_LSM

Each host within the storage cluster is expected to operate within the same Linux Security Module (LSM) state. For example, if the majority of the hosts are running with SELINUX in **enforcing** mode, any host not running in this mode would be flagged as an anomaly and a healthcheck with a warning state is raised.

CEPHADM_CHECK_SUBSCRIPTION

This check relates to the status of the vendor subscription. This check is only performed for hosts using Red Hat Enterprise Linux, but helps to confirm that all the hosts are covered by an active subscription so that patches and updates are available.

CEPHADM_CHECK_PUBLIC_MEMBERSHIP

All members of the cluster should have NICs configured on at least one of the public network subnets. Hosts that are not on the public network will rely on routing which may affect performance.

CEPHADM_CHECK_MTU

The maximum transmission unit (MTU) of the NICs on OSDs can be a key factor in consistent performance. This check examines hosts that are running OSD services to ensure that the MTU is configured consistently within the cluster. This is determined by establishing the MTU setting that the majority of hosts are using, with any anomalies resulting in a Ceph healthcheck.

CEPHADM_CHECK_LINKSPEED

Similar to the MTU check, linkspeed consistency is also a factor in consistent cluster performance. This check determines the linkspeed shared by the majority of the OSD hosts, resulting in a healthcheck for any hosts that are set at a lower linkspeed rate.

CEPHADM_CHECK_NETWORK_MISSING

The **public_network** and **cluster_network** settings support subnet definitions for IPv4 and IPv6. If these settings are not found on any host in the storage cluster a healthcheck is raised.

CEPHADM_CHECK_CEPH_RELEASE

Under normal operations, the Ceph cluster should be running daemons under the same Ceph release, for example all Red Hat Ceph Storage cluster 5 releases. This check looks at the active release for each daemon, and reports any anomalies as a healthcheck. This check is bypassed if an upgrade process is active within the cluster.

CEPHADM_CHECK_KERNEL_VERSION

The OS kernel version is checked for consistency across the hosts. Once again, the majority of the hosts is used as the basis of identifying anomalies.

CHAPTER 12. MANAGING A RED HAT CEPH STORAGE CLUSTER USING CEPHADM-ANSIBLE MODULES

As a storage administrator, you can use **cephadm-ansible** modules in Ansible playbooks to administer your Red Hat Ceph Storage cluster. The **cephadm-ansible** package provides several modules that wrap **cephadm** calls to let you write your own unique Ansible playbooks to administer your cluster.

NOTE

At this time, **cephadm-ansible** modules only support the most important tasks. Any operation not covered by **cephadm-ansible** modules must be completed using either the **command** or **shell** Ansible modules in your playbooks.

12.1. THE CEPHADM-ANSIBLE MODULES

The **cephadm-ansible** modules are a collection of modules that simplify writing Ansible playbooks by providing a wrapper around **cephadm** and **ceph orch** commands. You can use the modules to write your own unique Ansible playbooks to administer your cluster using one or more of the modules.

The **cephadm-ansible** package includes the following modules:

- **cephadm_bootstrap**
- **ceph_orch_host**
- **ceph_config**
- **ceph_orch_apply**
- **ceph_orch_daemon**
- **cephadm_registry_login**

12.2. THE CEPHADM-ANSIBLE MODULES OPTIONS

The following tables list the available options for the **cephadm-ansible** modules. Options listed as required need to be set when using the modules in your Ansible playbooks. Options listed with a default value of **true** indicate that the option is automatically set when using the modules and you do not need to specify it in your playbook. For example, for the **cephadm_bootstrap** module, the Ceph Dashboard is installed unless you set **dashboard: false**.

Table 12.1. Available options for the cephadm_bootstrap module.

cephadm_bootstrap	Description	Required	Default
mon_ip	Ceph Monitor IP address.	true	
image	Ceph container image.	false	
docker	Use **docker** instead of **podman**.	false	

cephadm_bootstrap	Description	Required	Default
fsid	Define the Ceph FSID.	false	
pull	Pull the Ceph container image.	false	true
dashboard	Deploy the Ceph Dashboard.	false	true
dashboard_user	Specify a specific Ceph Dashboard user.	false	
dashboard_password	Ceph Dashboard password.	false	
monitoring	Deploy the monitoring stack.	false	true
firewalld	Manage firewall rules with firewalld.	false	true
allow_overwrite	Allow overwrite of existing --output-config, --output-keyring, or --output-pub-ssh-key files.	false	false
registry_url	URL for custom registry.	false	
registry_username	Username for custom registry.	false	
registry_password	Password for custom registry.	false	
registry_json	JSON file with custom registry login information.	false	
ssh_user	SSH user to use for **cephadm** ssh to hosts.	false	
ssh_config	SSH config file path for **cephadm** SSH client.	false	
allow_fqdn_hostname	Allow hostname that is a fully-qualified domain name (FQDN).	false	false

cephadm_bootstrap	Description	Required	Default
cluster_network	Subnet to use for cluster replication, recovery and heartbeats.	false	

Table 12.2. Available options for the **ceph_orch_host** module.

ceph_orch_host	Description	Required	Default
fsid	The FSID of the Ceph cluster to interact with.	false	
image	The Ceph container image to use.	false	
name	Name of the host to add, remove, or update.	true	
address	IP address of the host.	true when **state** is **present**.	
set_admin_label	Set the **_admin** label on the specified host.	false	false
labels	The list of labels to apply to the host.	false	[]
state	If set to **present**, it ensures the name specified in **name** is present. If set to **absent**, it removes the host specified in **name**. If set to **drain**, it schedules to remove all daemons from the host specified in **name**.	false	present

Table 12.3. Available options for the **ceph_config** module

ceph_config	Description	Required	Default
fsid	The FSID of the Ceph cluster to interact with.	false	
image	The Ceph container image to use.	false	

ceph_config	Description	Required	Default
action	Whether to **set** or **get** the parameter specified in **option**.	false	set
who	Which daemon to set the configuration to.	true	
option	Name of the parameter to **set** or **get**.	true	
value	Value of the parameter to set.	true if action is **set**	

Table 12.4. Available options for the **ceph_orch_apply** module.

ceph_orch_apply	Description	Required
fsid	The FSID of the Ceph cluster to interact with.	false
image	The Ceph container image to use.	false
spec	The service specification to apply.	true

Table 12.5. Available options for the **ceph_orch_daemon** module.

ceph_orch_daemon	Description	Required
fsid	The FSID of the Ceph cluster to interact with.	false
image	The Ceph container image to use.	false
state	The desired state of the service specified in **name**.	true If **started**, it ensures the service is started. If **stopped**, it ensures the service is stopped. If **restarted**, it will restart the service.
daemon_id	The ID of the service.	true

ceph_orch_daemon	Description	Required
daemon_type	The type of service.	true

Table 12.6. Available options for the**cephadm_registry_login** module

cephadm_registry_login	Description	Required	Default
state	Login or logout of a registry.	false	login
docker	Use **docker** instead of **podman**.	false	
registry_url	The URL for custom registry.	false	
registry_username	Username for custom registry.	**true** when **state** is **login**.	
registry_password	Password for custom registry.	**true** when **state** is **login**.	
registry_json	The path to a JSON file. This file must be present on remote hosts prior to running this task. This option is currently not supported.		

12.3. BOOTSTRAPPING A STORAGE CLUSTER USING THE CEPHADM_BOOTSTRAP AND CEPHADM_REGISTRY_LOGIN MODULES

As a storage administrator, you can bootstrap a storage cluster using Ansible by using the **cephadm_bootstrap** and **cephadm_registry_login** modules in your Ansible playbook.

Prerequisites

- An IP address for the first Ceph Monitor container, which is also the IP address for the first node in the storage cluster.
- Login access to **registry.redhat.io**.
- A minimum of 10 GB of free space for **/var/lib/containers/**.
- Red Hat Enterprise Linux 8.4 EUS or Red Hat Enterprise Linux 8.5.
- Installation of the **cephadm-ansible** package on the Ansible administration node.

- Passwordless SSH is set up on all hosts in the storage cluster.
- Hosts are registered with CDN.

Procedure

1. Log in to the Ansible administration node.
2. Navigate to the **/usr/share/cephadm-ansible** directory on the Ansible administration node:

   Example

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

3. Create the **hosts** file and add hosts, labels, and monitor IP address of the first host in the storage cluster:

   Syntax

   ```
   sudo vi INVENTORY_FILE

   HOST1 labels="['LABEL1', 'LABEL2']"
   HOST2 labels="['LABEL1', 'LABEL2']"
   HOST3 labels="['LABEL1']"

   [admin]
   ADMIN_HOST monitor_address=MONITOR_IP_ADDRESS labels="['ADMIN_LABEL',
   'LABEL1', 'LABEL2']"
   ```

   Example

   ```
   [ansible@admin cephadm-ansible]$ sudo vi hosts

   host02 labels="['mon', 'mgr']"
   host03 labels="['mon', 'mgr']"
   host04 labels="['osd']"
   host05 labels="['osd']"
   host06 labels="['osd']"

   [admin]
   host01 monitor_address=10.10.128.68 labels="['_admin', 'mon', 'mgr']"
   ```

4. Run the preflight playbook:

   Syntax

   ```
   ansible-playbook -i INVENTORY_FILE cephadm-preflight.yml --extra-vars "ceph_origin=rhcs"
   ```

   Example

   ```
   [ansible@admin cephadm-ansible]$ ansible-playbook -i hosts cephadm-preflight.yml --extra-
   vars "ceph_origin=rhcs"
   ```

5. Create a playbook to bootstrap your cluster:

Syntax

```
sudo vi PLAYBOOK_FILENAME.yml

---
- name: NAME_OF_PLAY
  hosts: BOOTSTRAP_HOST
  become: USE_ELEVATED_PRIVILEGES
  gather_facts: GATHER_FACTS_ABOUT_REMOTE_HOSTS
  tasks:
    -name: NAME_OF_TASK
     cephadm_registry_login:
       state: STATE
       registry_url: REGISTRY_URL
       registry_username: REGISTRY_USER_NAME
       registry_password: REGISTRY_PASSWORD

    - name: NAME_OF_TASK
      cephadm_bootstrap:
        mon_ip: "{{ monitor_address }}"
        dashboard_user: DASHBOARD_USER
        dashboard_password: DASHBOARD_PASSWORD
        allow_fqdn_hostname: ALLOW_FQDN_HOSTNAME
        cluster_network: NETWORK_CIDR
```

Example

```
[ansible@admin cephadm-ansible]$ sudo vi bootstrap.yml

---
- name: bootstrap the cluster
  hosts: host01
  become: true
  gather_facts: false
  tasks:
    - name: login to registry
      cephadm_registry_login:
        state: login
        registry_url: registry.redhat.io
        registry_username: user1
        registry_password: mypassword1

    - name: bootstrap initial cluster
      cephadm_bootstrap:
        mon_ip: "{{ monitor_address }}"
        dashboard_user: mydashboarduser
        dashboard_password: mydashboardpassword
        allow_fqdn_hostname: true
        cluster_network: 10.10.128.0/28
```

6. Run the playbook:

Syntax

```
ansible-playbook -i INVENTORY_FILE PLAYBOOK_FILENAME.yml -vvv
```

Example

```
ansible@admin cephadm-ansible]$ ansible-playbook -i hosts bootstrap.yml -vvv
```

Verification

- Review the Ansible output after running the playbook.

12.4. ADDING OR REMOVING HOSTS USING THE CEPH_ORCH_HOST MODULE

As a storage administrator, you can add and remove hosts in your storage cluster by using the **ceph_orch_host** module in your Ansible playbook.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Register the nodes to the CDN and attach subscriptions.
- Ansible user with sudo and passwordless SSH access to all nodes in the storage cluster.
- Installation of the **cephadm-ansible** package on the Ansible administration node.
- New hosts have the storage cluster's public SSH key. For more information about copying the storage cluster's public SSH keys to new hosts, see *Adding hosts* in the *Red Hat Ceph Storage Installation Guide*.

Procedure

1. Use the following procedure to add new hosts to the cluster:
   a. Log in to the Ansible administration node.
   b. Navigate to the **/usr/share/cephadm-ansible** directory on the Ansible administration node:

      Example

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

   c. Add the new hosts and labels to the Ansible inventory file.

      Syntax

      ```
      sudo vi INVENTORY_FILE

      NEW_HOST1 labels="['LABEL1', 'LABEL2']"
      NEW_HOST2 labels="['LABEL1', 'LABEL2']"
      NEW_HOST3 labels="['LABEL1']"
      ```

```
[admin]
ADMIN_HOST monitor_address=MONITOR_IP_ADDRESS labels="['ADMIN_LABEL',
'LABEL1', 'LABEL2']"
```

Example

```
[ansible@admin cephadm-ansible]$ sudo vi hosts

host02 labels="['mon', 'mgr']"
host03 labels="['mon', 'mgr']"
host04 labels="['osd']"
host05 labels="['osd']"
host06 labels="['osd']"

[admin]
host01 monitor_address= 10.10.128.68 labels="['_admin', 'mon', 'mgr']"
```

NOTE

If you have previously added the new hosts to the Ansible inventory file and ran the preflight playbook on the hosts, skip to step 3.

d. Run the preflight playbook with the **--limit** option:

Syntax

```
ansible-playbook -i INVENTORY_FILE cephadm-preflight.yml --extra-vars
"ceph_origin=rhcs" --limit NEWHOST
```

Example

```
[ansible@admin cephadm-ansible]$ ansible-playbook -i hosts cephadm-preflight.yml --
extra-vars "ceph_origin=rhcs" --limit host02
```

The preflight playbook installs **podman**, **lvm2**, **chronyd**, and **cephadm** on the new host. After installation is complete, **cephadm** resides in the **/usr/sbin/** directory.

e. Create a playbook to add the new hosts to the cluster:

Syntax

```
sudo vi PLAYBOOK_FILENAME.yml

---
- name: PLAY_NAME
  hosts: HOSTS_OR_HOST_GROUPS
  become: USE_ELEVATED_PRIVILEGES
  gather_facts: GATHER_FACTS_ABOUT_REMOTE_HOSTS
  tasks:
    - name: NAME_OF_TASK
      ceph_orch_host:
        name: "{{ ansible_facts['hostname'] }}"
        address: "{{ ansible_facts['default_ipv4']['address'] }}"
```

```
    labels: "{{ labels }}"
  delegate_to: HOST_TO_DELEGATE_TASK_TO

- name: NAME_OF_TASK
  when: inventory_hostname in groups['admin']
  ansible.builtin.shell:
    cmd: CEPH_COMMAND_TO_RUN
  register: REGISTER_NAME

- name: NAME_OF_TASK
  when: inventory_hostname in groups['admin']
  debug:
    msg: "{{ REGISTER_NAME.stdout }}"
```

NOTE

By default, Ansible executes all tasks on the host that matches the **hosts** line of your playbook. The **ceph orch** commands must run on the host that contains the admin keyring and the Ceph configuration file. Use the **delegate_to** keyword to specify the admin host in your cluster.

Example

```
[ansible@admin cephadm-ansible]$ sudo vi add-hosts.yml

---
- name: add additional hosts to the cluster
  hosts: all
  become: true
  gather_facts: true
  tasks:
    - name: add hosts to the cluster
      ceph_orch_host:
        name: "{{ ansible_facts['hostname'] }}"
        address: "{{ ansible_facts['default_ipv4']['address'] }}"
        labels: "{{ labels }}"
      delegate_to: host01

    - name: list hosts in the cluster
      when: inventory_hostname in groups['admin']
      ansible.builtin.shell:
        cmd: ceph orch host ls
      register: host_list

    - name: print current list of hosts
      when: inventory_hostname in groups['admin']
      debug:
        msg: "{{ host_list.stdout }}"
```

In this example, the playbook adds the new hosts to the cluster and displays a current list of hosts.

f. Run the playbook to add additional hosts to the cluster:

Syntax

```
ansible-playbook -i INVENTORY_FILE PLAYBOOK_FILENAME.yml
```

Example

```
[ansible@admin cephadm-ansible]$ ansible-playbook -i hosts add-hosts.yml
```

2. Use the following procedure to remove hosts from the cluster:

   a. Log in to the Ansible administration node.

   b. Navigate to the **/usr/share/cephadm-ansible** directory on the Ansible administration node:

   Example

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

   c. Create a playbook to remove a host or hosts from the cluster:

   Syntax

   ```
   sudo vi PLAYBOOK_FILENAME.yml

   ---
   - name: NAME_OF_PLAY
     hosts: ADMIN_HOST
     become: USE_ELEVATED_PRIVILEGES
     gather_facts: GATHER_FACTS_ABOUT_REMOTE_HOSTS
     tasks:
       - name: NAME_OF_TASK
         ceph_orch_host:
           name: HOST_TO_REMOVE
           state: STATE

       - name: NAME_OF_TASK
         ceph_orch_host:
           name: HOST_TO_REMOVE
           state: STATE
         retries: NUMBER_OF_RETRIES
         delay: DELAY
         until: CONTINUE_UNTIL
         register: REGISTER_NAME

       - name: NAME_OF_TASK
         ansible.builtin.shell:
           cmd: ceph orch host ls
         register: REGISTER_NAME

       - name: NAME_OF_TASK
           debug:
             msg: "{{ REGISTER_NAME.stdout }}"
   ```

   Example

```
[ansible@admin cephadm-ansible]$ sudo vi remove-hosts.yml

---
- name: remove host
  hosts: host01
  become: true
  gather_facts: true
  tasks:
    - name: drain host07
      ceph_orch_host:
        name: host07
        state: drain

    - name: remove host from the cluster
      ceph_orch_host:
        name: host07
        state: absent
      retries: 20
      delay: 1
      until: result is succeeded
      register: result

    - name: list hosts in the cluster
      ansible.builtin.shell:
        cmd: ceph orch host ls
      register: host_list

    - name: print current list of hosts
      debug:
        msg: "{{ host_list.stdout }}"
```

In this example, the playbook tasks drain all daemons on **host07**, removes the host from the cluster, and displays a current list of hosts.

d. Run the playbook to remove host from the cluster:

Syntax

```
ansible-playbook -i INVENTORY_FILE PLAYBOOK_FILENAME.yml
```

Example

```
[ansible@admin cephadm-ansible]$ ansible-playbook -i hosts remove-hosts.yml
```

Verification

- Review the Ansible task output displaying the current list of hosts in the cluster:

Example

```
TASK [print current hosts]
*****************************************************************************************************
Friday 24 June 2022  14:52:40 -0400 (0:00:03.365)       0:02:31.702 ***********
ok: [host01] =>
```

```
msg: |-
  HOST    ADDR          LABELS         STATUS
  host01  10.10.128.68  _admin mon mgr
  host02  10.10.128.69  mon mgr
  host03  10.10.128.70  mon mgr
  host04  10.10.128.71  osd
  host05  10.10.128.72  osd
  host06  10.10.128.73  osd
```

12.5. SETTING CONFIGURATION OPTIONS USING THE CEPH_CONFIG MODULE

As a storage administrator, you can set or get Red Hat Ceph Storage configuration options using the **ceph_config** module.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Ansible user with sudo and passwordless SSH access to all nodes in the storage cluster.
- Installation of the **cephadm-ansible** package on the Ansible administration node.
- The Ansible inventory file contains the cluster and admin hosts.

Procedure

1. Log in to the Ansible administration node.
2. Navigate to the **/usr/share/cephadm-ansible** directory on the Ansible administration node:

   #### Example

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

3. Create a playbook with configuration changes:

   #### Syntax

   ```
   sudo vi PLAYBOOK_FILENAME.yml

   ---
   - name: PLAY_NAME
     hosts: ADMIN_HOST
     become: USE_ELEVATED_PRIVILEGES
     gather_facts: GATHER_FACTS_ABOUT_REMOTE_HOSTS
     tasks:
       - name: NAME_OF_TASK
         ceph_config:
           action: GET_OR_SET
           who: DAEMON_TO_SET_CONFIGURATION_TO
           option: CEPH_CONFIGURATION_OPTION
           value: VALUE_OF_PARAMETER_TO_SET
   ```

```
- name: NAME_OF_TASK
  ceph_config:
    action: GET_OR_SET
    who: DAEMON_TO_SET_CONFIGURATION_TO
    option: CEPH_CONFIGURATION_OPTION
  register: REGISTER_NAME

- name: NAME_OF_TASK
  debug:
    msg: "MESSAGE_TO_DISPLAY {{ REGISTER_NAME.stdout }}"
```

Example

```
[ansible@admin cephadm-ansible]$ sudo vi change_configuration.yml

---
- name: set pool delete
  hosts: host01
  become: true
  gather_facts: false
  tasks:
    - name: set the allow pool delete option
      ceph_config:
        action: set
        who: mon
        option: mon_allow_pool_delete
        value: true

    - name: get the allow pool delete setting
      ceph_config:
        action: get
        who: mon
        option: mon_allow_pool_delete
      register: verify_mon_allow_pool_delete

    - name: print current mon_allow_pool_delete setting
      debug:
        msg: "the value of 'mon_allow_pool_delete' is {{ verify_mon_allow_pool_delete.stdout }}"
```

In this example, the playbook first sets the **mon_allow_pool_delete** option to **false**. The playbook then gets the current **mon_allow_pool_delete** setting and displays the value in the Ansible output.

4. Run the playbook:

Syntax

```
ansible-playbook -i INVENTORY_FILE _PLAYBOOK_FILENAME.yml
```

Example

```
[ansible@admin cephadm-ansible]$ ansible-playbook -i hosts change_configuration.yml
```

Verification

- Review the output from the playbook tasks.

  Example

```
TASK [print current mon_allow_pool_delete setting]
*************************************************************
Wednesday 29 June 2022  13:51:41 -0400 (0:00:05.523)       0:00:17.953 ********
ok: [host01] =>
  msg: the value of 'mon_allow_pool_delete' is true
```

Additional Resources

- See the *Red Hat Ceph Storage Configuration Guide* for more details on configuration options.

12.6. APPLYING A SERVICE SPECIFICATION USING THE CEPH_ORCH_APPLY MODULE

As a storage administrator, you can apply service specifications to your storage cluster using the **ceph_orch_apply** module in your Ansible playbooks. A service specification is a data structure to specify the service attributes and configuration settings that is used to deploy the Ceph service. You can use a service specification to deploy Ceph service types like **mon**, **crash**, **mds**, **mgr**, **osd**, **rdb**, or **rbd-mirror**.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Ansible user with sudo and passwordless SSH access to all nodes in the storage cluster.
- Installation of the **cephadm-ansible** package on the Ansible administration node.
- The Ansible inventory file contains the cluster and admin hosts.

Procedure

1. Log in to the Ansible administration node.
2. Navigate to the **/usr/share/cephadm-ansible** directory on the Ansible administration node:

   Example

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

3. Create a playbook with the service specifications:

   Syntax

```
sudo vi PLAYBOOK_FILENAME.yml

---
- name: PLAY_NAME
  hosts: HOSTS_OR_HOST_GROUPS
  become: USE_ELEVATED_PRIVILEGES
```

```
gather_facts: GATHER_FACTS_ABOUT_REMOTE_HOSTS
tasks:
  - name: NAME_OF_TASK
    ceph_orch_apply:
      spec: |
        service_type: SERVICE_TYPE
        service_id: UNIQUE_NAME_OF_SERVICE
        placement:
          host_pattern: 'HOST_PATTERN_TO_SELECT_HOSTS'
          label: LABEL
        spec:
          SPECIFICATION_OPTIONS:
```

Example

```
[ansible@admin cephadm-ansible]$ sudo vi deploy_osd_service.yml

---
- name: deploy osd service
  hosts: host01
  become: true
  gather_facts: true
  tasks:
    - name: apply osd spec
      ceph_orch_apply:
        spec: |
          service_type: osd
          service_id: osd
          placement:
            host_pattern: '*'
            label: osd
          spec:
            data_devices:
              all: true
```

In this example, the playbook deploys the Ceph OSD service on all hosts with the label **osd**.

4. Run the playbook:

Syntax

```
ansible-playbook -i INVENTORY_FILE _PLAYBOOK_FILENAME.yml
```

Example

```
[ansible@admin cephadm-ansible]$ ansible-playbook -i hosts deploy_osd_service.yml
```

Verification

- Review the output from the playbook tasks.

Additional Resources

- See the *Red Hat Ceph Storage Operations Guide* for more details on service specification options.

12.7. MANAGING CEPH DAEMON STATES USING THE CEPH_ORCH_DAEMON MODULE

As a storage administrator, you can start, stop, and restart Ceph daemons on hosts using the **ceph_orch_daemon** module in your Ansible playbooks.

Prerequisites

- A running Red Hat Ceph Storage cluster.
- Ansible user with sudo and passwordless SSH access to all nodes in the storage cluster.
- Installation of the **cephadm-ansible** package on the Ansible administration node.
- The Ansible inventory file contains the cluster and admin hosts.

Procedure

1. Log in to the Ansible administration node.
2. Navigate to the **/usr/share/cephadm-ansible** directory on the Ansible administration node:

   #### Example

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

3. Create a playbook with daemon state changes:

   #### Syntax

   ```
   sudo vi PLAYBOOK_FILENAME.yml

   ---
   - name: PLAY_NAME
     hosts: ADMIN_HOST
     become: USE_ELEVATED_PRIVILEGES
     gather_facts: GATHER_FACTS_ABOUT_REMOTE_HOSTS
     tasks:
       - name: NAME_OF_TASK
         ceph_orch_daemon:
           state: STATE_OF_SERVICE
           daemon_id: DAEMON_ID
           daemon_type: TYPE_OF_SERVICE
   ```

   #### Example

   ```
   [ansible@admin cephadm-ansible]$ sudo vi restart_services.yml

   ---
   - name: start and stop services
     hosts: host01
   ```

```
become: true
gather_facts: false
tasks:
  - name: start osd.0
    ceph_orch_daemon:
      state: started
      daemon_id: 0
      daemon_type: osd

  - name: stop mon.host02
    ceph_orch_daemon:
      state: stopped
      daemon_id: host02
      daemon_type: mon
```

In this example, the playbook starts the OSD with an ID of **0** and stops a Ceph Monitor with an id of **host02**.

4. Run the playbook:

Syntax

```
ansible-playbook -i INVENTORY_FILE _PLAYBOOK_FILENAME.yml
```

Example

```
[ansible@admin cephadm-ansible]$ ansible-playbook -i hosts restart_services.yml
```

Verification

- Review the output from the playbook tasks.