Red Hat

Red Hat Ceph Storage 3

Administration Guide

Administration of Red Hat Ceph Storage

Last Updated: 2020-06-18

Red Hat Ceph Storage 3 Administration Guide

Administration of Red Hat Ceph Storage

Abstract

This document describes how to manage processes, monitor cluster states, manage users, and add and remove daemons for Red Hat Ceph Storage.

Table of Contents

CHAPTER 1. CEPH ADMINISTRATION OVERVIEW

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 for:
  - Red Hat Enterprise Linux
  - Ubuntu

CHAPTER 2. UNDERSTANDING PROCESS MANAGEMENT FOR CEPH

As a storage administrator, you can manipulate the Ceph daemons in various ways. Manipulating these daemons allows you to start, stop and restart all of the Ceph services as needed.

- Starting, Stopping, and Restarting All Ceph Daemons
- Starting, Stopping, Restarting a Ceph Daemon by Type
- Starting, Stopping, Restarting a Ceph Daemon by Instance

2.1. PREREQUISITES

- A running Red Hat Ceph Storage cluster.

2.2. AN OVERVIEW OF PROCESS MANAGEMENT FOR CEPH

In Red Hat Ceph Storage 3, all process management is done through the Systemd service. Each time you want to **start**, **restart**, and **stop** the Ceph daemons, you must specify the daemon type or the daemon instance.

Additional Resources

- For more information about using Systemd, see *Chapter 9* in the Red Hat Enterprise Linux System Administrator's Guide.

2.3. STARTING, STOPPING, AND RESTARTING ALL THE CEPH DAEMONS

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

Prerequisites

- Having **root** access to the node.

Procedure

- Starting all Ceph daemons:

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

- Stopping all Ceph daemons:

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

- Restarting all Ceph daemons:

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

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

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

Prerequisites

- Having **root** access to the node.

Procedure

- On **Ceph Monitor** nodes:

  Starting

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

  Stopping

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

  Restarting

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

- On **Ceph Manager** nodes:

  Starting

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

  Stopping

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

  Restarting

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

- On **Ceph OSD** nodes:

  Starting

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

  Stopping

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

  Restarting

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

- On **Ceph Object Gateway** nodes:

  Starting

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

  Stopping

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

  Restarting

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

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

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

Prerequisites

- Having **root** access to the node.

Procedure

- On a **Ceph Monitor** node:

  Starting

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

  Stopping

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

  Restarting

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

  Replace

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

- On a **Ceph Manager** node:

  Starting

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

Stopping

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

Restarting

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

Replace

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

- On a **Ceph OSD** node:

Starting

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

Stopping

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

Restarting

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

Replace

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

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

Starting

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

Stopping

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

Restarting

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

Replace

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

2.6. POWERING DOWN AND REBOOTING A RED HAT CEPH STORAGE CLUSTER

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

Prerequisites

- Having **root** access.

Procedure

Powering down the Red Hat Ceph Storage cluster

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

   - On the NFS-Ganesha Gateway node:

     ```
     # systemctl stop nfs-ganesha.service
     ```

   - On the RADOS Gateway node:

     ```
     # systemctl stop ceph-radosgw.target
     ```

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

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

   ```
   #ceph fs set <fs_name> max_mds 1
   #ceph mds deactivate <fs_name>:1 # rank 2 of 2
   #ceph status # wait for rank 1 to finish stopping
   #ceph fs set <fs_name> cluster_down true
   #ceph mds fail <fs_name>:0
   ```

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

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

   ```
   #ceph osd set noout
   #ceph osd set norecover
   #ceph osd set norebalance
   #ceph osd set nobackfill
   #ceph osd set nodown
   #ceph osd set pause
   ```

5. Shut down the OSD nodes one by one:

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

6. Shut down the monitor nodes one by one:

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

Rebooting the Red Hat Ceph Storage cluster

1. Power on the monitor nodes:

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

2. Power on the OSD nodes:

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

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

4. 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:

   ```
   #ceph osd unset noout
   #ceph osd unset norecover
   #ceph osd unset norebalance
   #ceph osd unset nobackfill
   #ceph osd unset nodown
   #ceph osd unset pause
   ```

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

   ```
   [root@admin~]# ceph fs set <fs_name> cluster_down false
   ```

6. Start the RADOS Gateway and NFS-Ganesha Gateway.

   - On the RADOS Gateway node:

     ```
     # systemctl start ceph-radosgw.target
     ```

   - On the NFS-Ganesha Gateway node:

     ```
     # systemctl start nfs-ganesha.service
     ```

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

2.7. ADDITIONAL RESOURCES

- For more information about installing Red Hat Ceph Storage, see:
  - Installation Guide for Red Hat Enterprise Linux

- Installation Guide for Ubuntu

CHAPTER 3. MONITORING

Once you have a running cluster, you may begin monitoring the storage cluster to ensure that the Ceph Monitor and OSD daemons are running, at a high-level. Ceph storage cluster clients must connect to a Ceph monitor and receive the latest version of the Ceph cluster map before they can read and write data to the Ceph pools of 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. HIGH-LEVEL MONITORING

High level monitoring of a storage cluster typically involves checking the status of Ceph OSD and Monitor daemons to ensure that they are up and running. High level monitoring also involves checking the storage cluster capacity to ensure that the cluster doesn't exceed its **full ratio**. The Calamari instance on the Ansible Tower or Red Hat Storage Console node is the most common way to conduct high-level monitoring. However, you may also use the command line, the admin socket or the Ceph API to monitor the storage cluster.

3.1.1. Interactive Mode

To run the **ceph** utility in interactive mode, type **ceph** at the command line with no arguments, for example:

```
# ceph
ceph> health
ceph> status
ceph> quorum_status
ceph> mon_status
```

3.1.2. Checking Cluster Health

After you start the Ceph storage cluster, and before you start reading and/or writing data, check the storage cluster's health first. You can check on the health of the Ceph storage cluster with the following:

```
# ceph health
```

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

```
# ceph -c /path/to/conf -k /path/to/keyring health
```

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

3.1.3. Watching a Cluster

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

```
# ceph -w
```

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

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

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

The output provides:

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

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.

3.1.4. Checking a Cluster's Usage Statistics

To check a cluster's data usage and data distribution among pools, you can use the **df** option. It is similar to Linux **df**. Execute the following:

```
# ceph df
```

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

- **SIZE:** The overall storage capacity of the storage cluster.
- **AVAIL:** The amount of free space available in the storage cluster.
- **RAW USED:** The amount of raw storage used.
- **% RAW USED:** The percentage of raw storage 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 1MB of data, the notional usage will be 1MB, but the actual usage may be 3MB or more depending on the number of replicas (e.g., **size = 3**, clones and snapshots.

- **NAME:** The name of the pool.
- **ID:** The pool ID.
- **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.
- **Objects:** The notional number of objects stored per pool.

NOTE

The numbers in the **POOLS** section are notional. They are not inclusive of the number of replicas, shapshots 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. See How Ceph Calculates Data Usage for details.

3.1.5. Checking a Cluster's Status

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

```
# ceph status
```

Or:

```
# ceph -s
```

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

```
ceph> status
```

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

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

3.1.6. Checking Monitor Status

If the storage cluster has multiple monitors, which is required for high availability for production Ceph storage clusters. You should check the Ceph Monitor quorum status after you start the Ceph storage cluster before reading and/or writing data. A quorum must be present when multiple monitors are running. You should also check Ceph Monitor status periodically to ensure that they are running. If there is a problem with the Monitor, that prevents an agreement on the state of the storage cluster, the fault may prevent Ceph clients from reading and writing data.

- To display the monitor map, execute the following:

  ```
  # ceph mon stat
  ```

  or

  ```
  # ceph mon dump
  ```

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

  ```
  # ceph quorum_status -f json-pretty
  ```

  Ceph will return the quorum status. For example, a Ceph storage cluster consisting of three monitors may return the following:

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

```
                "name": "c",
                "addr": "127.0.0.1:6791\/0"}
            ]
        }
    }
```

3.1.7. Using the 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 relaying on Monitors. This is useful when Monitors are **down**.
- Dump historic operations
- Dump the operation priority queue state
- Dump operations without rebooting
- Dump performance counters

In addition, using the socket is helpful when troubleshooting problems related to Monitors or OSDs. For details, see the Troubleshooting Guide for Red Hat Ceph Storage 3.

To use the socket:

```
ceph daemon <type>.<id> <command>
```

Replace:

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

For example, to view a Monitor status of a Monitor named **mon.0**:

```
# ceph daemon mon.0 mon_status
```

Alternatively, specify the daemon by using its socket file.

```
ceph daemon /var/run/ceph/<socket-file> <command>
```

For example, to view the status of an OSD named **osd.2**:

```
# ceph daemon /var/run/ceph/ceph-osd.2.asok status
```

To list all socket files for the Ceph processes:

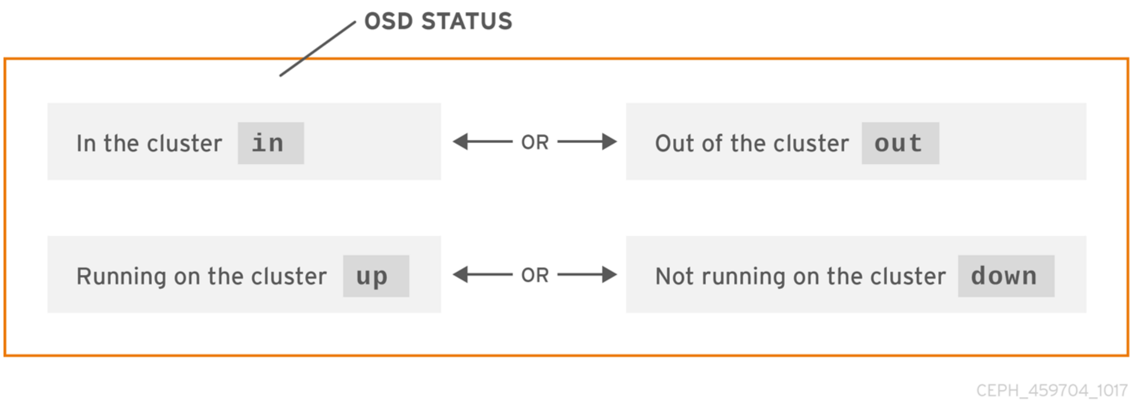
```
$ ls /var/run/ceph
```

3.1.8. Checking OSD Status

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

NOTE

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



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

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

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

```
# ceph osd stat
```

or

```
# ceph osd dump
```

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

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

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

```
# ceph osd tree
```

Example output:

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

TIP

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

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

```
# systemctl start ceph-osd@<osd_id>
```

3.2. LOW-LEVEL MONITORING

Lower-level monitoring typically involves ensuring that OSDs are peering. When faults occur, placement groups operate in a degraded state. This can be due to many things such as failed hardware, hung or crashed daemon, network latency or outage among other things.

3.2.1. Placement Group Sets

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

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

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

- To retrieve a list of placement groups:

  ```
  # ceph pg dump
  ```

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

  ```
  # ceph pg map <pg-num>
  ```

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

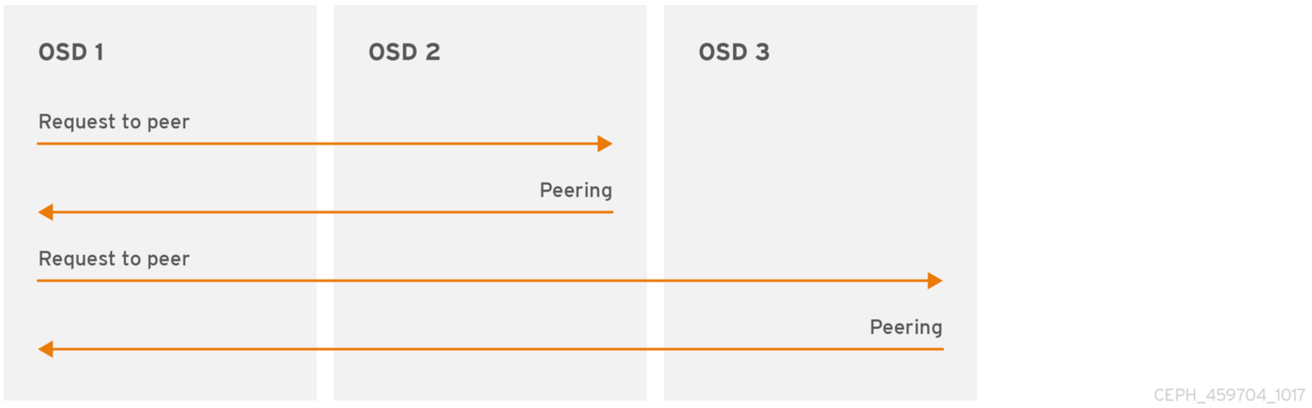
  ```
  osdmap eNNN pg <pg-num> -> up [0,1,2] acting [0,1,2]
  ```

  NOTE

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

3.2.2. 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 (i.e., the first OSD in the acting set), peers with the secondary and tertiary OSDs to establish agreement on the current state of the placement group (assuming a pool with 3 replicas of the PG).



3.2.3. Monitoring 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:

```
# 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
```

NOTE

See the miscellaneous OSD settings in the Red Hat Ceph Storage 3 Configuration Guide for more details on the snapshot trimming settings.

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

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

Placement Group IDs

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

```
<pool_num>.<pg_id>
```

Example output:

```
0.1f
```

- To retrieve a list of placement groups:

  ```
  # ceph pg dump
  ```

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

  ```
  # ceph pg dump -o <file_name> --format=json
  ```

- To query a particular placement group:

  ```
  # ceph pg <pool_num>.<pg_id> query
  ```

  Example output in JSON format:

  ```
  {
    "state": "active+clean",
    "up": [
      1,
      0
    ],
    "acting": [
      1,
      0
    ],
    "info": {
      "pgid": "1.e",
      "last_update": "4'1",
      "last_complete": "4'1",
      "log_tail": "0'0",
      "last_backfill": "MAX",
      "purged_snaps": "[]",
      "history": {
        "epoch_created": 1,
        "last_epoch_started": 537,
        "last_epoch_clean": 537,
        "last_epoch_split": 534,
        "same_up_since": 536,
        "same_interval_since": 536,
        "same_primary_since": 536,
        "last_scrub": "4'1",
        "last_scrub_stamp": "2013-01-25 10:12:23.828174"
      },
      "stats": {
  ```

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

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

      ],
      "scrub": {
```

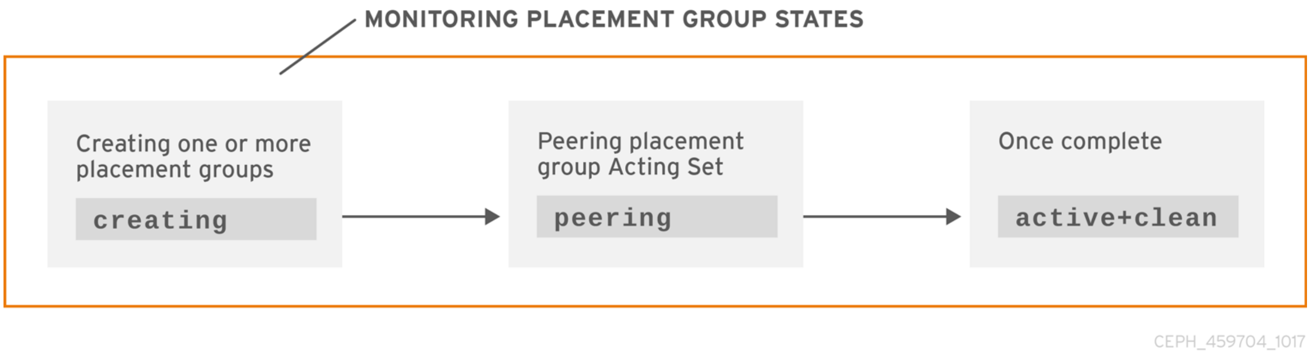
```
                "scrub_epoch_start": "536",
                "scrub_active": 0,
                "scrub_block_writes": 0,
                "finalizing_scrub": 0,
                "scrub_waiting_on": 0,
                "scrub_waiting_on_whom": [

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

The following subsections describe common states in greater detail.

3.2.3.1. Creating

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


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

3.2.3.2. Peering

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.

3.2.3.3. Active

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.

3.2.3.4. Clean

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.

3.2.3.5. Degraded

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

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

If an OSD is **down** and the **degraded** condition persists, Ceph may mark the **down** OSD as **out** of the cluster and remap the data from the **down** OSD to another OSD. The time between being marked **down** and being marked **out** is controlled by **mon osd down out interval**, which is set to **300** seconds by default.

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

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

3.2.3.6. Recovering

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

Recovery isn't always trivial, because a hardware failure might cause a cascading failure of multiple OSDs. For example, a network switch for a rack or cabinet may fail, which can cause the OSDs of a number of host machines to fall behind the current state of the cluster. Each one of the OSDs must recover once the fault is resolved.

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

max active setting limits the number of recovery requests an OSD will entertain simultaneously to prevent the OSD from failing to serve . The **osd recovery max chunk** setting limits the size of the recovered data chunks to prevent network congestion.

3.2.3.7. Backfilling

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

During the backfill operations, you may see one of several states: **backfill_wait** indicates that a backfill operation is pending, but isn't underway yet; **backfill** indicates that a backfill operation is underway; and, **backfill_too_full** indicates that a backfill operation was requested, but couldn't be completed due to insufficient storage capacity. When a placement group cannot be backfilled, it may be considered **incomplete**.

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

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

3.2.3.8. Remapped

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.

3.2.3.9. Stale

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.

3.2.3.10. Misplaced

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.

Lets say 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]
```

Here, [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.

3.2.3.11. Incomplete

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.

3.2.4. Identifying Troubled Placement Groups

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

- **Unclean**: Placement groups contain objects that are not replicated the desired number of times. They should be recovering.
- **Inactive**: Placement groups cannot process reads or writes because they are waiting for an OSD with the most up-to-date data to come back **up**.

- **Stale**: Placement groups are in an unknown state, because the OSDs that host them have not reported to the monitor cluster in a while, and can be configured with the **mon osd report timeout** setting.

To identify stuck placement groups, execute the following:

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

3.2.5. Finding an Object Location

To store object data in the Ceph Object Store, a Ceph client must:

1. Set an object name
2. Specify a pool

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

```
# ceph osd map <pool_name> <object_name>
```

3.3. MONITORING A CEPH STORAGE CLUSTER WITH THE RED HAT CEPH STORAGE DASHBOARD

The Red Hat Ceph Storage Dashboard provides a monitoring dashboard to visualize the state of a Ceph Storage Cluster. Also, the Red Hat Ceph Storage Dashboard architecture provides a framework for additional modules to add functionality to the storage cluster.

- To learn about the Dashboard, see Section 3.3.1, "The Red Hat Ceph Storage Dashboard" .
- To install the Dashboard, see Section 3.3.2, "Installing the Red Hat Ceph Storage Dashboard" .
- To access the Dashboard, see Section 3.3.3, "Accessing the Red Hat Ceph Storage Dashboard" .
- To change the default password after installing the Dashboard, see Section 3.3.4, "Changing the default Red Hat Ceph Storage dashboard password".
- To learn about the Prometheus plugin, see Section 3.3.5, "The Prometheus plugin for Red Hat Ceph Storage".
- To learn about the Red Hat Ceph Storage Dashboard alerts and how to configure them, see Section 3.3.6, "The Red Hat Ceph Storage Dashboard alerts" .

Prerequisites

- A running Red Hat Ceph Storage cluster

3.3.1. The Red Hat Ceph Storage Dashboard

The Red Hat Ceph Storage Dashboard provides a monitoring dashboard for Ceph clusters to visualize the storage cluster state. The dashboard is accessible from a web browser and provides a number of metrics and graphs about the state of the cluster, Monitors, OSDs, Pools, or the network.

As of Red Hat Ceph Storage 3.2, the only supported installation of the dashboard is containerized. Trying to install the dashboard as non-containerized in RHCS 3.2 or later will cause the Ansible playbook to fail. Note that this is regardless of whether Ceph itself is running in containers or not.

The introduction of Prometheus as the monitoring data source simplifies deployment and operational management of the Red Hat Ceph Storage Dashboard solution, along with reducing the overall hardware requirements. By sourcing the Ceph monitoring data directly, the Red Hat Ceph Storage Dashboard solution is better able to support Ceph clusters deployed in containers.

NOTE

With this change in architecture, there is no migration path for monitoring data from Red Hat Ceph Storage 2.x and 3.0 to Red Hat Ceph Storage 3.3.

The Red Hat Ceph Storage Dashboard uses the following utilities:

- The Ansible automation application for deployment.
- The embedded Prometheus **ceph-mgr** plugin.
- The Prometheus **node-exporter** daemon, running on each node of the storage cluster.
- The Grafana platform to provide a user interface and alerting.

The Red Hat Ceph Storage Dashboard supports the following features:

General Features

- Support for Red Hat Ceph Storage 3.1 and higher
- SELinux support
- Support for FileStore and BlueStore OSD back ends
- Support for encrypted and non-encrypted OSDs
- Support for Monitor, OSD, the Ceph Object Gateway, and iSCSI roles
- Initial support for the Metadata Servers (MDS)
- Drill down and dashboard links
- 15 second granularity
- Support for Hard Disk Drives (HDD), Solid-state Drives (SSD), Non-volatile Memory Express (NVMe) interface, and Intel® Cache Acceleration Software (Intel® CAS)

Node Metrics

- CPU and RAM usage
- Network load

Configurable Alerts

- Out-of-Band (OOB) alerts and triggers
- Notification channel is automatically defined during the installation
- The Ceph Health Summary dashboard created by default
  See the Red Hat Ceph Storage Dashboard Alerts section for details.

Cluster Summary

- OSD configuration summary
- OSD FileStore and BlueStore summary
- Cluster versions breakdown by role
- Disk size summary
- Host size by capacity and disk count
- Placement Groups (PGs) status breakdown
- Pool counts
- Device class summary, HDD vs. SSD

Cluster Details

- Cluster flags status (**noout**, **nodown**, and others)
- OSD or Ceph Object Gateway hosts **up** and **down** status
- Per pool capacity usage
- Raw capacity utilization
- Indicators for active scrub and recovery processes
- Growth tracking and forecast (raw capacity)
- Information about OSDs that are **down** or **near full**, including the OSD host and disk
- Distribution of PGs per OSD
- OSDs by PG counts, highlighting the over or under utilized OSDs

OSD Performance

- Information about I/O operations per second (IOPS) and throughput by pool
- OSD performance indicators
- Disk statistics per OSD
- Cluster wide disk throughput
- Read/write ratio (client IOPS)

- Disk utilization heat map
- Network load by Ceph role

The Ceph Object Gateway Details

- Aggregated load view
- Per host latency and throughput
- Workload breakdown by HTTP operations

The Ceph iSCSI Gateway Details

- Aggregated views
- Configuration
- Performance
- Per Gateway resource utilization
- Per client load and configuration
- Per Ceph Block Device image performance

3.3.2. Installing the Red Hat Ceph Storage Dashboard

The Red Hat Ceph Storage Dashboard provides a visual dashboard to monitor various metrics in a running Ceph Storage Cluster.

NOTE

For information on upgrading the Red Hat Ceph Storage Dashboard see Upgrading Red Hat Ceph Storage Dashboard in the Installation Guide for Red Hat Enterprise Linux .

IMPORTANT

On the container host node, you must add all the target nodes to the **/etc/hosts** file in the Prometheus container for name resolution.

Prerequisites

- The storage cluster nodes use Red Hat Enterprise Linux 7.
- A separate node, the Red Hat Ceph Storage Dashboard node, for receiving data from the cluster nodes and providing the Red Hat Ceph Storage Dashboard.
- Prepare the Red Hat Ceph Storage Dashboard node:
  - Enable the Tools repository on all nodes.
    For details, see the *Enabling the Red Hat Ceph Storage Repositories* section in the Red Hat Ceph Storage 3 *Installation Guide for Red Hat Enterprise Linux*.
  - If using a firewall, then ensure that the following TCP ports are open:

Table 3.1. TCP Port Requirements

Port	Use	Where?
3000	Grafana	The Red Hat Ceph Storage Dashboard node.
9090	Basic Prometheus graphs	The Red Hat Ceph Storage Dashboard node.
9100	Prometheus' **node-exporter** daemon	All storage cluster nodes.
9283	Gathering Ceph data	All **ceph-mgr** nodes.
9287	Ceph iSCSI gateway data	All Ceph iSCSI gateway nodes.

For more details see the *Using Firewalls* chapter in the *Security Guide* for Red Hat Enterprise Linux 7.

There are two installation processes for the Red Hat Ceph Storage Dashboard:

- Installing the Red Hat Ceph Storage Dashboard in an online environment
- Installing the Red Hat Ceph Storage Dashboard in an offline environment

3.3.2.1. Installing the Red Hat Ceph Storage Dashboard in an online environment

To install the Red Hat Ceph Storage Dashboard in an environment with internet access:

Procedure

Run the following commands on the Ansible administration node as the **root** user.

1. Install the **cephmetrics-ansible** package.

   ```
   [root@admin ~]# yum install cephmetrics-ansible
   ```

2. Using the Ceph Ansible inventory as a base, add the Red Hat Ceph Storage Dashboard node under the **[ceph-grafana]** section of the Ansible inventory file, by default located at **/etc/ansible/hosts**.

   ```
   [ceph-grafana]
   $HOST_NAME
   ```

   Replace:

   - **$HOST_NAME** with the name of the Red Hat Ceph Storage Dashboard node

   For example:

   ```
   [ceph-grafana]
   node0
   ```

–

3. Change to the **/usr/share/cephmetrics-ansible/** directory.

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

4. Run the Ansible playbook.

```
[root@admin cephmetrics-ansible]# ansible-playbook -v playbook.yml
```

IMPORTANT

Every time you update the cluster configuration, for example, you add or remove a MON or OSD node, you must re-run the **cephmetrics** Ansible playbook.

NOTE

The **cephmetrics** Ansible playbook does the following actions:

- Updates the **ceph-mgr** instance to enable the prometheus plugin and opens TCP port 9283.
- Deploys the Prometheus **node-exporter** daemon to each node in the storage cluster.
  - Opens TCP port 9100.
  - Starts the **node-exporter** daemon.
- Deploys Grafana and Prometheus containers under Docker/systemd on the Red Hat Ceph Storage Dashboard node.
  - Prometheus is configured to gather data from the ceph-mgr nodes and the node-exporters running on each ceph host
  - Opens TCP port 3000.
  - The dashboards, themes and user accounts are all created in Grafana.
  - Outputs the URL of Grafana for the administrator.

3.3.2.2. Installing the Red Hat Ceph Storage Dashboard in an offline environment

To install the Red Hat Ceph Storage Dashboard in an environment that cannot access the internet during installation, you must first follow these steps on a system with internet access:

Procedure

1. On a node that has access to registry.access.redhat.com, install the latest Docker RPM package:

```
[root@admin ~]# yum install docker
```

2. Start a local Docker registry:

```
[root@admin ~]# docker run -d -p 5000:5000 --restart=always --name registry registry:2
```

3. Pull the Red Hat Ceph Storage 3.x images of Prometheus and Dashboard from the Red Hat Customer Portal:

   ```
   [root@admin ~]# docker pull registry.access.redhat.com/rhceph/rhceph-3-dashboard-rhel7:3
   [root@admin ~]# docker pull registry.access.redhat.com/openshift3/prometheus:v3.9
   ```

4. Tag the images:

   ```
   [root@admin ~]# docker tag registry.access.redhat.com/rhceph/rhceph-3-rhel7:3 <local-host-
   fqdn>:5000/rhceph/rhceph-3-dashboard-rhel7:3
   [root@admin ~]# docker tag registry.access.redhat.com/openshift3/prometheus:v3.9 <local-
   host-fqdn>:5000/openshift3/prometheus:v3.9
   ```

   Replace **<local-host-fqdn>** with your local host FQDN.

5. Push the images to the local Docker registry you started:

   ```
   [root@admin ~]#docker push <local-host-fqdn>:5000/rhceph/rhceph-3-dashboard-rhel7:3
   [root@admin ~]#docker push <local-host-fqdn>:5000/openshift3/prometheus:v3.9
   ```

   Replace **<local-host-fqdn>** with your local host FQDN.

   NOTE

   If you run into an error when pushing the images to the local Docker registry, you may need to mention the local host in the **insecure-registries**. To do so:

   1. Edit the file **/etc/docker/daemon.json**
   2. Add the host FQDN with the port in the file, and save:

      ```
      { "insecure-registries":["<local-host-fqdn>:5000"] }
      ```

   3. Restart the docker service:

      ```
      systemctl restart docker
      ```

6. Install the **cephmetrics-ansible** package.

   ```
   [root@admin ~]# yum install cephmetrics-ansible
   ```

7. Using the Ceph Ansible inventory as a base, add the Red Hat Ceph Storage Dashboard node under the **[ceph-grafana]** section of the Ansible inventory file, by default located at **/etc/ansible/hosts**.

   ```
   [ceph-grafana]
   $HOST_NAME
   ```

   Replace:

   - **$HOST_NAME** with the name of the Red Hat Ceph Storage Dashboard node

   For example:

```
[ceph-grafana]
node0
```

8. Change to the **/usr/share/cephmetrics-ansible/** directory.

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

9. Create a new copy of the **all.yml.sample** file :

```
cp /usr/share/cephmetrics-ansible/group_vars/all.yml.sample /usr/share/cephmetrics-
ansible/group_vars/all.yml
```

10. Edit **all.yml** to point to your local registry, where **<local-host-fqdn>:5000** will be FQDN of your local docker registry node:

```
vi  /usr/share/cephmetrics-ansible/group_vars/all.yml
~~~
dummy:

#cluster_name: ceph

#containerized: true

# Set the backend options, mgr+prometheus or cephmetrics+graphite
#backend:
#  metrics: mgr  # mgr, cephmetrics
#  storage: prometheus  # prometheus, graphite

# Turn on/off devel_mode
#devel_mode: true

# Set grafana admin user and password
# You need to change these in the web UI on an already deployed machine, first
# New deployments work fine
#grafana:
#  admin_user: admin
#  admin_password: admin

grafana:
  container_name: <local-host-fqdn>:5000/rhceph/rhceph-3-dashboard-rhel7
  version: 3

prometheus:
  container_name: <local-host-fqdn>:5000/openshift3/prometheus
  version: v3.9
```

11. Run the Ansible playbook.

```
[root@admin cephmetrics-ansible]# ansible-playbook -v playbook.yml
```

3.3.3. Accessing the Red Hat Ceph Storage Dashboard

Accessing the Red Hat Ceph Storage Dashboard gives you access to the web-based management tool for administrating Red Hat Ceph Storage clusters.

Prerequisites

- Install the Red Hat Ceph Storage Dashboard .
- Ensure that NTP is synchronizing clocks properly because a time lag can occur among the Ceph Storage Dashboard node, cluster nodes, and a browser when the nodes are not properly synced. See the *Configuring the Network Time Protocol for Red Hat Ceph Storage* section in the Red Hat Ceph Storage 3 Installation Guide for Red Hat Enterprise Linux or Ubuntu.

Procedure

1. Enter the following URL to a web browser:

   ```
   http://$HOST_NAME:3000
   ```

   Replace:

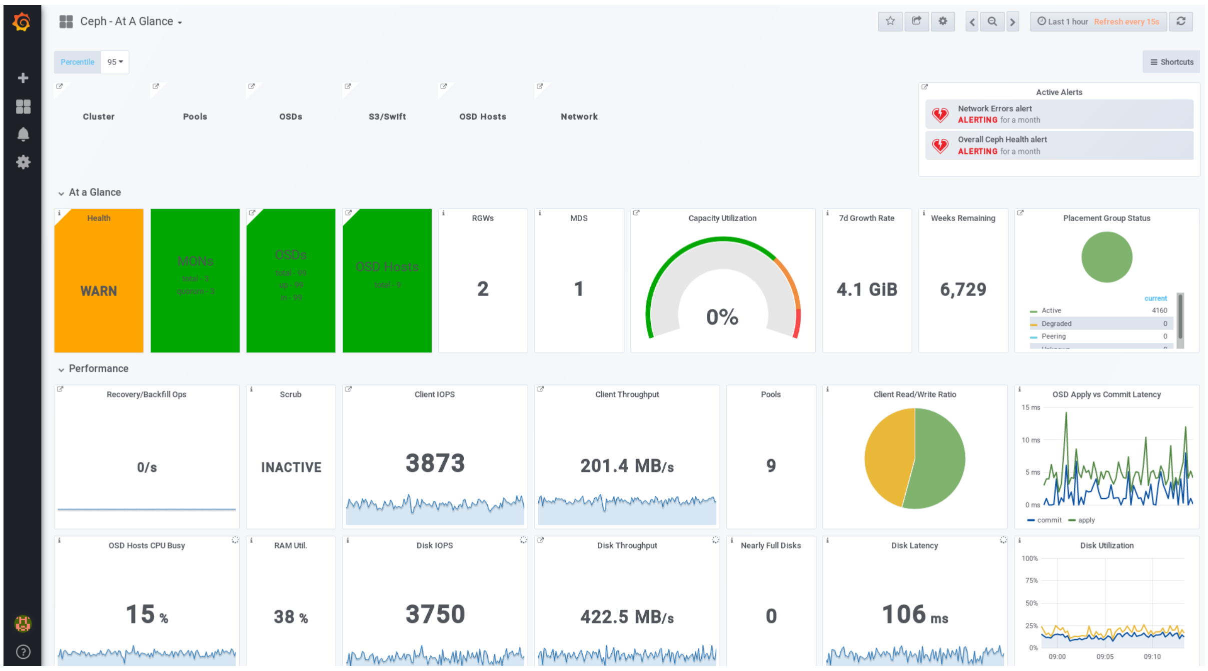
   - **$HOST_NAME** with the name of the Red Hat Ceph Storage Dashboard node

   For example:

   ```
   http://cephmetrics:3000
   ```

2. Enter the password for the **admin** user. If you did not set the password during the installation, use **admin**, which is the default password.
   Once logged in, you are automatically placed on the *Ceph At a Glance* dashboard. The *Ceph At a Glance* dashboard provides a high-level overviews of capacity, performance, and node-level performance information.

Example



Additional Resources

- See the Changing the Default Red Hat Ceph Storage Dashboard Password section in the Red Hat Ceph Storage Administration Guide.

3.3.4. Changing the default Red Hat Ceph Storage dashboard password

The default user name and password for accessing the Red Hat Ceph Storage Dashboard is set to **admin** and **admin**. For security reasons, you might want to change the password after the installation.

NOTE

If you redeploy the Red Hat Ceph Storage dashboard using Ceph Ansible, then the password will be reset to the default value. Update the Ceph Ansible inventory file (**/etc/ansible/hosts**) with the custom password to prevent the password from resetting to the default value.

Prerequisites

- Install the Red Hat Ceph Storage Dashboard .
- Log in to the Red Hat Ceph Storage Dashboard .

Procedure

1. Click the Grafana icon in the upper-left corner.
2. Hover over the user name you want to modify the password for. In this case **admin**.
3. Click **Preferences**.
4. Click **Change Password**.
5. Enter the new password twice and click **Change Password**.

Additional Resource

- If you forgot the password, follow the Reset admin password procedure on the Grafana web pages.

3.3.5. The Prometheus plugin for Red Hat Ceph Storage

As a storage administrator, you can gather performance data, export that data using the Prometheus plugin module for the Red Hat Ceph Storage Dashboard, and then perform queries on this data. The Prometheus module allows **ceph-mgr** to expose Ceph related state and performance data to a Prometheus server.

3.3.5.1. Prerequisites

- Running Red Hat Ceph Storage 3.1 or higher.
- Installation of the Red Hat Ceph Storage Dashboard.

3.3.5.2. The Prometheus plugin

The Prometheus plugin provides an exporter to pass on Ceph performance counters from the collection point in **ceph-mgr**. The Red Hat Ceph Storage Dashboard receives **MMgrReport** messages from all **MgrClient** processes, such as Ceph Monitors and OSDs. A circular buffer of the last number of samples contains the performance counter schema data and the actual counter data. This plugin creates an HTTP endpoint and retrieves the latest sample of every counter when polled. The HTTP path and query parameters are ignored; all extant counters for all reporting entities are returned in a text exposition format.

Additional Resources

- See the Prometheus documentation for more details on the text exposition format.

3.3.5.3. Managing the Prometheus environment

To monitor a Ceph storage cluster with Prometheus you can configure and enable the Prometheus exporter so the metadata information about the Ceph storage cluster can be collected.

Prerequisites

- A running Red Hat Ceph Storage 3.1 cluster
- Installation of the Red Hat Ceph Storage Dashboard

Procedure

1. As the **root** user, open and edit the **/etc/prometheus/prometheus.yml** file.

   a. Under the **global** section, set the **scrape_interval** and **evaluation_interval** options to 15 seconds.

      Example

      ```
      global:
        scrape_interval:     15s
        evaluation_interval: 15s
      ```

   b. Under the **scrape_configs** section, add the **honor_labels: true** option, and edit the **targets**, and **instance** options for each of the **ceph-mgr** nodes.

      Example

      ```
      scrape_configs:
        - job_name: 'node'
          honor_labels: true
          static_configs:
          - targets: [ 'node1.example.com:9100' ]
            labels:
              instance: "node1.example.com"
          - targets: ['node2.example.com:9100']
            labels:
              instance: "node2.example.com"
      ```

NOTE

Using the **honor_labels** option enables Ceph to output properly-labelled data relating to any node in the Ceph storage cluster. This allows Ceph to export the proper **instance** label without Prometheus overwriting it.

c. To add a new node, simply add the **targets**, and **instance** options in the following format:

Example

```
- targets: [ 'new-node.example.com:9100' ]
  labels:
    instance: "new-node"
```

NOTE

The **instance** label has to match what appears in Ceph's OSD metadata **instance** field, which is the short host name of the node. This helps to correlate Ceph stats with the node's stats.

2. Add Ceph targets to the **/etc/prometheus/ceph_targets.yml** file in the following format.

Example

```
[
    {
        "targets": [ "cephnode1.example.com:9283" ],
        "labels": {}
    }
]
```

3. Enable the Prometheus module:

```
# ceph mgr module enable prometheus
```

3.3.5.4. Using the Prometheus expression browser

Use the builtin Prometheus expression browser to run queries against the collected data.

Prerequisites

- A running Red Hat Ceph Storage 3.1 cluster
- Installation of the Red Hat Ceph Storage Dashboard

Procedure

1. Enter the URL for the Prometh the web browser:

```
http://$DASHBOARD_SEVER_NAME:9090/graph
```

Replace…

- **$DASHBOARD_SEVER_NAME** with the name of the Red Hat Ceph Storage Dashboard server.

2. Click on *Graph*, then type in or paste the query into the query window and press the *Execute* button.
   a. View the results in the console window.
3. Click on *Graph* to view the rendered data.

Additional Resources

- See the Prometheus expression browser documentation on the Prometheus web site for more information.

3.3.5.5. Working with the Prometheus data and queries

The statistic names are exactly as Ceph names them, with illegal characters translated to underscores, and **ceph_** prefixed to all names. All Ceph daemon statistics have a **ceph_daemon** label that identifies the type and ID of the daemon they come from, for example: **osd.123**. Some statistics can come from different types of daemons, so when querying you will want to filter on Ceph daemons starting with **osd** to avoid mixing in the Ceph Monitor and RocksDB stats. The global Ceph storage cluster statistics have labels appropriate to what they report on. For example, metrics relating to pools have a **pool_id** label. The long running averages that represent the histograms from core Ceph are represented by a pair of sum and count performance metrics.

The following example queries can be used in the Prometheus expression browser:

Show the physical disk utilization of an OSD

```
(irate(node_disk_io_time_ms[1m]) /10) and on(device,instance)
ceph_disk_occupation{ceph_daemon="osd.1"}
```

Show the physical IOPS of an OSD as seen from the operating system

```
irate(node_disk_reads_completed[1m]) + irate(node_disk_writes_completed[1m]) and on (device,
instance) ceph_disk_occupation{ceph_daemon="osd.1"}
```

Pool and OSD metadata series

Special data series are output to enable the displaying and the querying on certain metadata fields. Pools have a **ceph_pool_metadata** field, for example:

```
ceph_pool_metadata{pool_id="2",name="cephfs_metadata_a"} 1.0
```

OSDs have a **ceph_osd_metadata** field, for example:

```
ceph_osd_metadata{cluster_addr="172.21.9.34:6802/19096",device_class="ssd",ceph_daemon="osd.0
",public_addr="172.21.9.34:6801/19096",weight="1.0"} 1.0
```

Correlating drive statistics with node_exporter

The Prometheus output from Ceph is designed to be used in conjunction with the generic node monitoring from the Prometheus node exporter. Correlation of Ceph OSD statistics with the generic node monitoring drive statistics, special data series are output, for example:

```
ceph_disk_occupation{ceph_daemon="osd.0",device="sdd", exported_instance="node1"}
```

To get disk statistics by an OSD ID, use either the **and** operator or the asterisk (*) operator in the Prometheus query. All metadata metrics have the value of **1** so they act neutral with asterisk operator. Using asterisk operator allows to use **group_left** and **group_right** grouping modifiers, so that the resulting metric has additional labels from one side of the query. For example:

```
rate(node_disk_bytes_written[30s]) and on (device,instance)
ceph_disk_occupation{ceph_daemon="osd.0"}
```

Using label_replace

The **label_replace** function can add a label to, or alter a label of, a metric within a query. To correlate an OSD and its disks write rate, the following query can be used:

```
label_replace(rate(node_disk_bytes_written[30s]), "exported_instance", "$1", "instance", "(.*):.*") and
on (device,exported_instance) ceph_disk_occupation{ceph_daemon="osd.0"}
```

Additional Resources

- See Prometheus querying basics for more information on constructing queries.
- See Prometheus' **label_replace** documentation for more information.

3.3.5.6. Additional Resources

- The Prometheus website

3.3.6. The Red Hat Ceph Storage Dashboard alerts

This section includes information about alerting in the Red Hat Ceph Storage Dashboard.

- To learn about the Red Hat Ceph Storage Dashboard alerts, see Section 3.3.6.2, “About Alerts”.
- To view the alerts, see Section 3.3.6.3, “Accessing the Alert Status dashboard” .
- To configure the notification target, see Section 3.3.6.4, “Configuring the Notification Target”.
- To change the default alerts or add new ones, see Section 3.3.6.5, “Changing the Default Alerts and Adding New Ones”.

3.3.6.1. Prerequisites

- Install the Red Hat Ceph Storage Dashboard .
- Log in to the Red Hat Ceph Storage Dashboard .

3.3.6.2. About Alerts

The Red Hat Ceph Storage Dashboard supports alerting mechanism that is provided by the Grafana platform. You can configure the dashboard to send you a notification when a metric that you are interested in reaches certain value. Such metrics are in the **Alert Status** dashboard.

By default, **Alert Status** already includes certain metrics, such as *Overall Ceph Health*, *OSDs Down*, or *Pool Capacity*. You can add metrics that you are interested in to this dashboard or change their trigger values.

Here is a list of the pre-defined alerts that are included with Red Hat Ceph Storage Dashboard:

- Overall Ceph Health
- Disks Near Full (>85%)
- OSD Down
- OSD Host Down
- PG's Stuck Inactive
- OSD Host Less - Free Capacity Check
- OSD's With High Response Times
- Network Errors
- Pool Capacity High
- Monitors Down
- Overall Cluster Capacity Low
- OSDs With High PG Count

3.3.6.3. Accessing the Alert Status dashboard

Certain Red Hat Ceph Storage Dashboard alerts are configured by default in the **Alert Status** dashboard. This section shows two ways to access it.

Procedure
To access the dashboard:

- In the main **At the Glance** dashboard, click the **Active Alerts** panel in the upper-right corner.

Or..

- Click the dashboard menu from in the upper-left corner next to the Grafana icon. Select **Alert Status**.

3.3.6.4. Configuring the Notification Target

A notification channel called **cephmetrics** is automatically created during installation. All preconfigured alerts reference the **cephmetrics** channel but before you can receive the alerts, complete the notification channel definition by selecting the desired notification type. The Grafana platform supports a number of different notification types including email, Slack, and PagerDuty.

Procedure

- To configure the notification channel, follow the instructions in the Alert Notifications section on the Grafana web page.

3.3.6.5. Changing the Default Alerts and Adding New Ones

This section explains how to change the trigger value on already configured alerts and how to add new alerts to the **Alert Status** dashboard.

Procedure

- To change the trigger value on alerts or to add new alerts, follow the Alerting Engine & Rules Guide on the Grafana web pages.

IMPORTANT

To prevent overriding custom alerts, the **Alert Status** dashboard will not be updated when upgrading the Red Hat Ceph Storage Dashboard packages when you change the trigger values or add new alerts.

Additional Resources

- The Grafana web page

3.4. USING CEPH-MEDIC TO DIAGNOSE A CEPH STORAGE CLUSTER

The **ceph-medic** utility performs checks against a running Ceph Storage Cluster to identify potential problems.

The **ceph-medics** utility for example checks:

- The correct ownership of files and directories
- If the **fsid** is the same for all nodes in the storage cluster
- If the secret key in the keyring is different than other nodes in the storage cluster

3.4.1. Prerequisites

- A working Red Hat Ceph Storage cluster
- SSH and **sudo** access to storage nodes

3.4.2. Installing the ceph-medic Utility

Prerequisite

- Access to the Red Hat Ceph Storage 3 software repositories

Procedure
Do the following steps on the Ansible administration node, as the **root** user.

1. Install the **ceph-medic** package:

```
[root@admin ~]# yum install ceph-medic
```

2. Verify the installation of **ceph-medic**:

```
[root@admin ~]# ceph-medic --help
```

Additional Resources

- The *Enabling the Red Hat Ceph Storage Repositories* section in the Red Hat Ceph Storage 3 *Installation Guide for Red Hat Enterprise Linux*

3.4.3. Running a Diagnostic Check

A basic check for potential problems with a Ceph Storage Cluster.

Procedure

Do the following step from the Ansible administration node, as a normal user.

- Use the **ceph-medic check** command:

```
[admin@admin ~]$ ceph-medic check
Host: mon0                  connection: [connected  ]
Host: mon1                  connection: [connected  ]
Host: mon2                  connection: [connected  ]
Host: osd0                  connection: [connected  ]
Host: osd1                  connection: [connected  ]
Host: osd2                  connection: [connected  ]
Collection completed!

=======================  Starting remote check session
========================
Version: 1.0.2    Cluster Name: "example"
Total hosts: [6]
OSDs:    3    MONs:    3     Clients:    0
MDSs:    0    RGWs:    0     MGRs:       0

==========================================================================
=====

------------ osds ------------
 osd0
 osd1
 osd2

------------ mons ------------
 mon0
 mon1
 mon2

17 passed, 0 errors, on 6 hosts
```

Additional Resources

- *Error Code Definitions for `ceph-medic`*

3.4.4. Using a Custom Inventory File

The **ceph-medic** utility must know the storage cluster topology. By default, **ceph-medic** uses the Ansible inventory file (**/etc/ansible/hosts**) to detect nodes. However, you can instruct **ceph-medic** to use a custom inventory file.

Procedure

Do the following steps on the Ansible administration node, as a user.

1. Create a custom **hosts** file:

   ```
   [admin@admin ~]$ touch ~/example/hosts
   ```

2. Open the **hosts** file for editing. Add the nodes in the storage cluster under the appropriate node group type. The **ceph-medic** tool supports the following node group types: **mons**, **osds**, **rgws**, **mdss**, **mgrs** and **clients**.
   For example:

   ```
   [mons]
   mon0
   mon1
   mon2

   [osds]
   osd0
   osd1
   osd2
   ```

3. When doing a diagnostic check, to specify a custom inventory file, use the **--inventory** option:

   ```
   ceph-medic --inventory $PATH_TO_HOSTS_FILE check
   ```

   Replace

   - **$PATH_TO_HOSTS_FILE** with the full path to the **hosts** file.
     For example:

     ```
     [admin@admin ~]$ ceph-medic --inventory ~/example/hosts check
     ```

3.4.5. Configuring a Custom Logging Path

The **ceph-medic** log file contains more verbosity than the command output on the terminal. The **ceph-medic** utility by default writes logs to the current working directory. However, you can configure **ceph-medic** to use a custom logging path.

Procedure

Do the following step on the Ansible administration node, as a normal user.

1. Open the **~/.cephmedic.conf** file for editing.

2. Change the **--log-path** option, from a **.** to a custom log location.
   For example:

   ```
   --log-path = /var/log/ceph-medic/
   ```

CHAPTER 4. OVERRIDES

By default, Ceph will reflect the current status of OSDs and perform normal operations such as rebalancing, recovering, and scrubbing. From time to time, it may be advantageous to override Ceph's default behavior.

4.1. SETTING AND UNSETTING OVERRIDES

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

```
# ceph osd set <flag>
```

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

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

```
# ceph osd unset <flag>
```

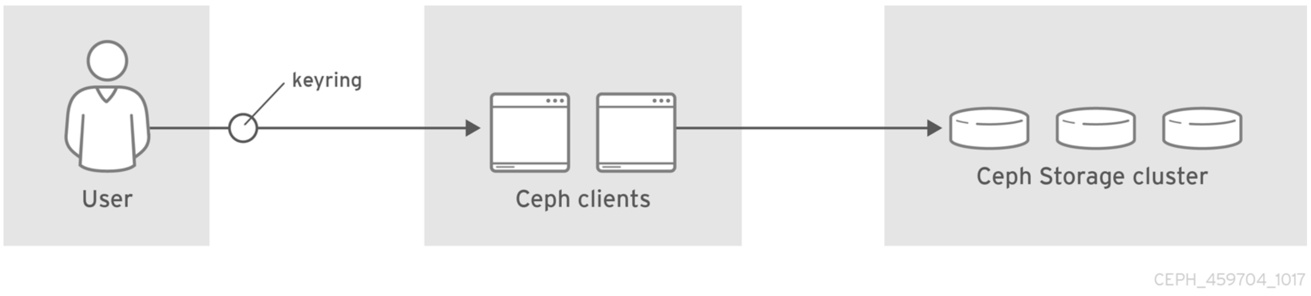
Flag	Description
noin	Prevents OSDs from being treated as **in** the cluster.
noout	Prevents OSDs from being treated as **out** of the cluster.
noup	Prevents OSDs from being treated as **up** and running.
nodown	Prevents OSDs from being treated as **down**.
full	Makes a cluster appear to have reached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

4.2. 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, (e.g., upgrading daemons), you can set **nobackfill** so that Ceph will not backfill while the OSD(s) 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 (e.g., to reduce overhead during high loads, recovery, backfilling, rebalancing, etc.), 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. USER MANAGEMENT

This section describes Ceph client users, and their authentication and authorization with the Red Hat Ceph Storage cluster. Users are either individuals or system actors such as applications, which use Ceph clients to interact with the Red Hat Ceph Storage cluster daemons.



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

```
# ceph health
```

Ceph interprets the command like this:

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

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

For details on configuring the Red Hat Ceph Storage cluster to use authentication, see Configuration Guide for Red Hat Ceph Storage 3.

5.1. BACKGROUND

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 will help you understand Ceph user management.

5.1.1. User

A user of the Red Hat Ceph Storage cluster is either an individual or a system actor such as an application. Creating users allows you to control who (or what) can access the storage cluster, its pools, and the data within 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**. We recommend 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 Storage 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.

5.1.2. Authorization (Capabilities)

Ceph uses the term "capabilities" (caps) to describe authorizing an authenticated user to exercise the functionality of the 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:

```
<daemon_type> 'allow <capability>' [<daemon_type> 'allow <capability>']
```

- **Monitor Caps:** Monitor capabilities include **r**, **w**, **x**, **allow profile <cap>**, and **profile rbd**. For example:

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

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

  ```
  osd 'allow <capability>' [pool=<pool_name>] [namespace=<namespace_name>]
  ```

NOTE

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

The following entries describe each capability.

allow	Precedes access settings for a daemon.
r	Gives the user read access. Required with monitors to retrieve the CRUSH map.
w	Gives the user write access to objects.
x	Gives the user the capability to call class methods (that is, both read and write) and to conduct **auth** operations on monitors.

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

5.1.3. 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/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.

5.1.4. 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.

5.2. MANAGING USERS

User management functionality provides system administrators with the ability to create, update and delete Red Hat Ceph Storage cluster users.

When you create or delete users in a Red Hat Ceph Storage cluster, you may need to distribute keys to clients so that they can be added to keyrings. See Keyring Management for details.

5.2.1. List Users

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

```
# ceph auth list
```

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

```
installed auth entries:

osd.0
    key: AQCvCbtToC6MDhAATtuT70Sl+DymPCfDSsyV4w==
    caps: [mon] allow profile osd
    caps: [osd] allow *
osd.1
    key: AQC4CbtTCFJBChAAVq5spj0ff4eHZICxIOVZeA==
    caps: [mon] allow profile osd
    caps: [osd] allow *
client.admin
    key: AQBHCbtT6APDHhAA5W00cBchwkQjh3dkKsyPjw==
    caps: [mds] allow
    caps: [mon] allow *
    caps: [osd] allow *
client.bootstrap-mds
    key: AQBICbtTOK9uGBAAdbe5zcIGHZL3T/u2g6EBww==
    caps: [mon] allow profile bootstrap-mds
client.bootstrap-osd
    key: AQBHCbtT4GxqORAADE5u7RkpCN/oo4e5W0uBtw==
    caps: [mon] allow profile bootstrap-osd
```

Note that 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.

5.2.2. Get a User

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

Syntax

```
# ceph auth get <TYPE.ID>
```

Example

```
# ceph auth get client.admin
```

You may also use the **-o <file_name>** option with **ceph auth get** to save the output to a file. Developers may also execute the following:

Syntax

```
# ceph auth export <TYPE.ID>
```

Example

```
# ceph auth export client.admin
```

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

5.2.3. Add a 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 auth add client.john mon 'allow r' osd 'allow rw pool=liverpool'
# ceph auth get-or-create client.paul mon 'allow r' osd 'allow rw pool=liverpool'
# ceph auth get-or-create client.george mon 'allow r' osd 'allow rw pool=liverpool' -o george.keyring
# ceph auth get-or-create-key client.ringo mon 'allow r' osd 'allow rw pool=liverpool' -o ringo.key
```

IMPORTANT

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

5.2.4. Modify User Capabilities

The **ceph auth caps** command allows you to specify a user and change the user's capabilties. 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 auth caps client.john mon 'allow r' osd 'allow rw pool=liverpool'
# ceph auth caps client.paul mon 'allow rw' osd 'allow rwx pool=liverpool'
# ceph auth caps client.brian-manager mon 'allow *' osd 'allow *'
```

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. For example:

```
# ceph auth caps client.ringo mon ' ' osd ' '
```

See Section 5.1.2, "Authorization (Capabilities)" for additional details on capabilities.

5.2.5. Delete a User

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

```
# ceph auth del {TYPE}.{ID}
```

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

5.2.6. Print a User's Key

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

```
# ceph auth print-key <TYPE>.<ID>
```

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

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

```
# mount -t ceph <hostname>:/<mount_point> -o name=client.user,secret=`ceph auth print-key
client.user`
```

5.2.7. Import a User

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

Syntax

```
# ceph auth import -i </path/to/keyring>
```

Example

```
# ceph auth import -i /etc/ceph/ceph.keyring
```

> **NOTE**
>
> The ceph storage cluster will add new users, their keys and their capabilities and will update existing users, their keys and their capabilities.

5.3. KEYRING MANAGEMENT

When you access Ceph by using a Ceph client, the Ceph client will look for a local keyring. Ceph presets the **keyring** setting with the following four keyring names by default so you don't have to set them in the Ceph configuration file unless you want to override the defaults, which is not recommended:

- **/etc/ceph/$cluster.$name.keyring**
- **/etc/ceph/$cluster.keyring**
- **/etc/ceph/keyring**
- **/etc/ceph/keyring.bin**

The **$cluster** metavariable is the Ceph storage cluster name as defined by the name of the Ceph configuration file, that is, **ceph.conf** means the cluster name is **ceph**; thus, **ceph.keyring**. The **$name** metavariable is the user type and user ID, for example, **client.admin**; thus, **ceph.client.admin.keyring**.

> **NOTE**
>
> When executing commands that read or write to **/etc/ceph**, you may need to use **sudo** to execute the command as **root**.

After you create a user, for example, **client.ringo**, you must get the key and add it to a keyring on a Ceph client so that the user can access the Ceph Storage Cluster.

See Chapter 5, *User Management* for details on how to list, get, add, modify and delete users directly in the Ceph Storage Cluster. However, Ceph also provides the **ceph-authtool** utility to allow you to manage keyrings from a Ceph client.

5.3.1. Create a Keyring

When you use the procedures in the Managing Users_ section to create users, you need to provide user keys to the Ceph client(s) so that the Ceph client can retrieve the key for the specified user and authenticate with the Ceph Storage Cluster. Ceph Clients access keyrings to lookup a user name and retrieve the user’s key.

The **ceph-authtool** utility allows you to create a keyring. To create an empty keyring, use **--create-keyring** or **-C**. For example:

```
# ceph-authtool --create-keyring /path/to/keyring
```

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

```
# ceph-authtool -C /etc/ceph/ceph.keyring
```

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

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

5.3.2. Add a User to a Keyring

When you add a user to the Ceph storage cluster, you can use the **get** procedure to retrieve a user, key and capabilities, then save the user to a keyring file.

When you only want to use one user per keyring, the Get a User_ procedure with the **-o** option will save the output in the keyring file format. For example, to create a keyring for the **client.admin** user, execute the following:

```
# ceph auth get client.admin -o /etc/ceph/ceph.client.admin.keyring
```

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

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

```
# ceph-authtool /etc/ceph/ceph.keyring --import-keyring /etc/ceph/ceph.client.admin.keyring
```

5.3.3. Create a User

Ceph provides the Add a User_ function to create a user directly in the Ceph Storage Cluster. However, you can also create a user, keys and capabilities directly on a Ceph client keyring. Then, you can import the user to the Ceph Storage Cluster. For example:

```
# ceph-authtool -n client.ringo --cap osd 'allow rwx' --cap mon 'allow rwx' /etc/ceph/ceph.keyring
```

See Section 5.1.2, “Authorization (Capabilities)” for additional details on capabilities.

You can also create a keyring and add a new user to the keyring simultaneously. For example:

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

In the foregoing scenarios, the new user **client.ringo** is only in the keyring. To add the new user to the Ceph Storage Cluster, you must still add the new user to the Ceph Storage Cluster. :

```
# ceph auth add client.ringo -i /etc/ceph/ceph.keyring
```

5.3.4. Modify a User

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

```
# ceph-authtool /etc/ceph/ceph.keyring -n client.ringo --cap osd 'allow rwx' --cap mon 'allow rwx'
```

To update the user to the Ceph storage cluster, you must update the user in the keyring to the user entry in the the Ceph storage cluster:

```
# ceph auth import -i /etc/ceph/ceph.keyring
```

See Section 5.2.7, "Import a User" for details on updating a Ceph Storage Cluster user from a keyring.

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

5.4. COMMAND LINE USAGE

Ceph supports the following usage for user name and secret:

--id | **--user**

Description

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

```
# ceph --id foo --keyring /path/to/keyring health
# ceph --user foo --keyring /path/to/keyring health
```

--name | **-n**

Description

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

```
# ceph --name client.foo --keyring /path/to/keyring health
# ceph -n client.foo --keyring /path/to/keyring health
```

--keyring

Description

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

```
# rbd map foo --pool rbd myimage --id client.foo --keyring /path/to/keyring
```

5.5. LIMITATIONS

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

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

IMPORTANT

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

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

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

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

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

CHAPTER 6. USING THE CEPH-VOLUME UTILITY TO DEPLOY OSDS

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

IMPORTANT

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

6.1. USING THE 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:

```
ceph-volume lvm
```

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

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

NOTE

Using the **create** subcommand combines the **prepare** and **activate** subcommands into one subcommand. See the **create** subcommand section for more details.

6.1.1. Preparing OSDs

The **prepare** subcommand prepares an OSD backend object store and consumes logical volumes for both the OSD data and journal. There is no default object storage type. The object storage type requires either the **--filestore** or **--bluestore** option to be set at preparation time. Starting with Red Hat Ceph Storage 3.2, support for the BlueStore object storage type is available. The **prepare** subcommand will not create or modify the logical volumes, except for adding some extra metadata using LVM tags.

LVM tags makes volumes easier to discover later, and help identify them as part of a Ceph system, and what role they have. The **ceph-volume lvm prepare** command adds the following list of LVM tags:

- **cluster_fsid**

- **data_device**
- **journal_device**
- **encrypted**
- **osd_fsid**
- **osd_id**
- **journal_uuid**

The **prepare** process is very strict, it requires two logical volumes that are ready for use, and requires the minimum size for an OSD data and journal. The journal device can be either a logical volume or a partition.

Here is the **prepare** workflow process:

1. Accept logical volumes for data and journal
2. Generate a UUID for the OSD
3. Ask the Ceph Monitor to get an OSD identifier reusing the generated UUID
4. OSD data directory is created and data volume mounted
5. Journal is symlinked from data volume to journal location
6. The **monmap** is fetched for activation
7. Device is mounted and the data directory is populated by **ceph-osd**
8. LVM tags are assigned to theOSD data and journal volumes

Do the following step on an OSD node, and as the **root** user, to prepare a simple OSD deployment using LVM:

```
ceph-volume lvm prepare --bluestore --data $VG_NAME/$LV_NAME
```

For example:

```
# ceph-volume lvm prepare --bluestore --data example_vg/data_lv
```

For BlueStore, you can also specify the **--block.db** and **--block.wal** options, if you want to use a separate device for RocksDB.

Here is an example of using FileStore with a partition as a journal device:

```
# ceph-volume lvm prepare --filestore --data example_vg/data_lv --journal /dev/sdc1
```

When using a partition, it must contain a **PARTUUID** discoverable by the **blkid** command, this way it can be identified correctly regardless of the device name or path.

IMPORTANT

The **ceph-volume** LVM plugin does not create partitions on a raw disk device. Creating this partition has to be done before using a partition for the OSD journal device.

6.1.2. Activating OSDs

Once the prepare process is done, the OSD is ready to go active. The activation process enables a Systemd unit at boot time which allows the correct OSD identifier and its UUID to be enabled and mounted.

Here is the **activate** workflow process:

1. Requires both OSD id and OSD uuid
2. Enable the systemd unit with matching OSD id and OSD uuid
3. The systemd unit will ensure all devices are ready and mounted
4. The matching **ceph-osd** systemd unit will get started

Do the following step on an OSD node, and as the **root** user, to activate an OSD:

```
ceph-volume lvm activate --filestore $OSD_ID $OSD_UUID
```

For example:

```
# ceph-volume lvm activate --filestore 0 0263644D-0BF1-4D6D-BC34-28BD98AE3BC8
```

NOTE

There are no side-effects when running this command multiple times.

6.1.3. Creating OSDs

The **create** subcommand wraps the two-step process to deploy a new OSD by calling the **prepare** subcommand and then calling the **activate** subcommand into a single subcommand. The reason to use **prepare** and then **activate** separately is to gradually introduce new OSDs into a storage cluster, and avoiding large amounts of data being rebalanced. There is nothing different to the process except the OSD will become *up* and *in* immediately after completion.

Do the following step, for FileStore, on an OSD node, and as the **root** user:

```
ceph-volume lvm create --filestore --data $VG_NAME/$LV_NAME --journal $JOURNAL_DEVICE
```

For example:

```
# ceph-volume lvm create --filestore --data example_vg/data_lv --journal example_vg/journal_lv
```

Do the following step, for BlueStore, on an OSD node, and as the **root** user:

```
# ceph-volume lvm create --bluestore --data <device>
```

For example:

```
# ceph-volume lvm create --bluestore --data /dev/sda
```

6.1.4. Using batch mode

The **batch** subcommand automates the creation of multiple OSDs when single devices are provided. The **ceph-volume** command decides the best method in creating the OSDs based on drive type. This best method is dependant on the object store format, BlueStore or FileStore.

BlueStore is the default object store type for OSDs. When using BlueStore, OSD optimization depends on three different scenarios based on the devices being used. If all devices are traditional hard drives, then one OSD per device is created. If all devices are solid state drives, then two OSDs per device are created. If there is a mix of traditional hard drives and solid state drives, then data is put on the traditional hard drives, and the **block.db** is created as large as possible on the solid state drive.

NOTE

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

BlusStore example

```
# ceph-volume lvm batch --bluestore /dev/sda /dev/sdb /dev/nvme0n1
```

When using FileStore, OSD optimization depends on two different scenarios based on the devices being used. If all devices are traditional hard drives or are solid state drives, then one OSD per device is created, collocating the journal on the same device. If there is a mix of traditional hard drives and solid state drives, then data is put on the traditional hard drives, and the journal is created on the solid state drive using the sizing parameters specified in the Ceph configuration file, by default **ceph.conf**, with a default journal size of 5 GB.

FileStore example

```
# ceph-volume lvm batch --filestore /dev/sda /dev/sdb
```

CHAPTER 7. BENCHMARKING PERFORMANCE

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

7.2. STORAGE CLUSTER

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:

```
# echo 3 | sudo tee /proc/sys/vm/drop_caches && sudo sync
```

1. Create a new storage pool:

```
# ceph osd pool create testbench 100 100
```

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

```
# rados bench -p testbench 10 write --no-cleanup
```

Example Output

```
Maintaining 16 concurrent writes of 4194304 bytes for up to 10 seconds or 0 objects
 Object prefix: benchmark_data_cephn1.home.network_10510
   sec Cur ops   started  finished  avg MB/s  cur MB/s  last lat   avg lat
     0       0         0         0         0         0         -         0
     1      16        16         0         0         0         -         0
     2      16        16         0         0         0         -         0
     3      16        16         0         0         0         -         0
     4      16        17         1  0.998879         1   3.19824   3.19824
     5      16        18         2   1.59849         4   4.56163   3.87993
     6      16        18         2   1.33222         0         -   3.87993
     7      16        19         3   1.71239         2   6.90712     4.889
     8      16        25         9   4.49551        24   7.75362   6.71216
```

```
    9      16       25        9   3.99636        0         -   6.71216
   10      16       27       11   4.39632        4   9.65085   7.18999
   11      16       27       11   3.99685        0         -   7.18999
   12      16       27       11   3.66397        0         -   7.18999
   13      16       28       12   3.68975   1.33333   12.8124   7.65853
   14      16       28       12   3.42617        0         -   7.65853
   15      16       28       12   3.19785        0         -   7.65853
   16      11       28       17   4.24726   6.66667   12.5302   9.27548
   17      11       28       17   3.99751        0         -   9.27548
   18      11       28       17   3.77546        0         -   9.27548
   19      11       28       17   3.57683        0         -   9.27548
 Total time run:         19.505620
Total writes made:      28
Write size:             4194304
Bandwidth (MB/sec):     5.742

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

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

```
# rados bench -p testbench 10 seq
```

Example Output

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

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

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

```
# rados bench -p testbench 10 rand
```

Example Output

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

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

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

```
# rados bench -p testbench 10 write -t 4 --run-name client1
```

Example Output

```
Maintaining 4 concurrent writes of 4194304 bytes for up to 10 seconds or 0 objects
 Object prefix: benchmark_data_node1_12631
   sec Cur ops   started  finished  avg MB/s  cur MB/s  last lat   avg lat
     0       0         0         0         0         0         -         0
     1       4         4         0         0         0         -         0
     2       4         6         2   3.99099         4   1.94755   1.93361
     3       4         8         4   5.32498         8     2.978   2.44034
     4       4         8         4   3.99504         0         -   2.44034
     5       4        10         6   4.79504         4   2.92419    2.4629
     6       3        10         7   4.64471         4   3.02498    2.5432
     7       4        12         8   4.55287         4   3.12204   2.61555
     8       4        14        10    4.9821         8   2.55901   2.68396
     9       4        16        12   5.31621         8   2.68769   2.68081
    10       4        17        13   5.18488         4   2.11937   2.63763
    11       4        17        13   4.71431         0         -   2.63763
    12       4        18        14   4.65486         2    2.4836   2.62662
    13       4        18        14   4.29757         0         -   2.62662
Total time run:         13.123548
Total writes made:      18
Write size:             4194304
Bandwidth (MB/sec):     5.486

Stddev Bandwidth:       3.0991
Max bandwidth (MB/sec): 8
Min bandwidth (MB/sec): 0
```

```
Average Latency:       2.91578
Stddev Latency:        0.956993
Max latency:           5.72685
Min latency:           1.91967
```

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

```
# rados -p testbench cleanup
```

7.3. BLOCK DEVICE

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. For more information on the **rbd** command, see the Block Device Commands section in the Ceph Block Device Guide for Red Hat Ceph Storage 3.

Creating a Ceph Block Device

1. As **root**, load the **rbd** kernel module, if not already loaded:

```
# modprobe rbd
```

2. As **root**, create a 1 GB **rbd** image file in the **testbench** pool:

```
# rbd create image01 --size 1024 --pool testbench
```

> **NOTE**
>
> When creating a block device image these features are enabled by default: **layering**, **object-map**, **deep-flatten**, **journaling**, **exclusive-lock**, and **fast-diff**.
>
> On Red Hat Enterprise Linux 7.2 and Ubuntu 16.04, users utilizing the kernel RBD client will not be able to map the block device image. You must first disable all these features, except, **layering**.
>
> **Syntax**
>
> ```
> # rbd feature disable <image_name> <feature_name>
> ```
>
> **Example**
>
> ```
> # rbd feature disable image1 journaling deep-flatten exclusive-lock fast-diff
> object-map
> ```
>
> Using the **--image-feature layering** option on the **rbd create** command will only enable **layering** on newly created block device images.
>
> This is a known issue, see the Red Hat Ceph Storage 3.3 Release Notes. for more details.
>
> All these features will work for users utilizing the user-space RBD client to access the block device images.

3. As **root**, map the image file to a device file:

   ```
   # rbd map image01 --pool testbench --name client.admin
   ```

4. As **root**, create an **ext4** file system on the block device:

   ```
   # mkfs.ext4 /dev/rbd/testbench/image01
   ```

5. As **root**, create a new directory:

   ```
   # mkdir /mnt/ceph-block-device
   ```

6. As **root**, mount the block device under **/mnt/ceph-block-device/**:

   ```
   # mount /dev/rbd/testbench/image01 /mnt/ceph-block-device
   ```

Execute the write performance test against the block device

```
# rbd bench-write image01 --pool=testbench
```

Example

```
bench-write  io_size 4096 io_threads 16 bytes 1073741824 pattern seq
  SEC       OPS   OPS/SEC   BYTES/SEC
    2     11127   5479.59  22444382.79
```

```
3     11692   3901.91  15982220.33
4     12372   2953.34  12096895.42
5     12580   2300.05  9421008.60
6     13141   2101.80  8608975.15
7     13195    356.07  1458459.94
8     13820    390.35  1598876.60
9     14124    325.46  1333066.62
..
```

CHAPTER 8. PERFORMANCE COUNTERS

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. ACCESS

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
- Filestore Metrics - Displays information on all related filestore statistics
- 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

8.2. SCHEMA

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

To view the metric's schema:

```
# ceph daemon <daemon_name> perf schema
```

NOTE

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

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

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

Example

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

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

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

Example

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

Table 8.1. The bit field value definitions

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

Each value will have bit 1 or 2 set to indicate the type, either a floating point or an integer value. When bit 4 is set, there will be two values to read, a sum and a count. See Section 8.3.1, "Average Count and Sum" for more details. 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.

8.3. DUMP

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

To view the current metric data:

```
# ceph daemon {daemon_name} perf dump
```

NOTE

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

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

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

Example

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

To view a short description of each Monitor metric available, please see the table below.

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

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

Example

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

8.3.1. 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.

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

8.3.2. Monitor Metrics Description Tables

- 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
	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

Collection Name	Metric Name	Bit Field Value	Short Description
	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
	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

Collection Name	Metric Name	Bit Field Value	Short Description
	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
	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

Collection Name	Metric Name	Bit Field Value	Short Description
	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
	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

Collection Name	Metric Name	Bit Field Value	Short Description
	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

8.3.3. OSD Metrics Description Tables

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

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. Filestore Metrics Table

Collection Name	Metric Name	Bit Field Value	Short Description
filestore	**journal_queue_max_ops**	2	Max operations in journal queue
	journal_queue_ops	2	Operations in journal queue
	journal_ops	10	Total journal entries written
	journal_queue_max_bytes	2	Max data in journal queue
	journal_queue_bytes	2	Size of journal queue
	journal_bytes	10	Total operations size in journal
	journal_latency	5	Average journal queue completing latency
	journal_wr	10	Journal write IOs
	journal_wr_bytes	6	Journal data written
	journal_full	10	Journal writes while full
	committing	2	Is currently committing
	commitcycle	10	Commit cycles
	commitcycle_interval	5	Average interval between commits

Collection Name	Metric Name	Bit Field Value	Short Description
	commitcycle_latency	5	Average latency of commit
	op_queue_max_ops	2	Max operations count in queue
	op_queue_ops	2	Operations count in queue
	ops	10	Operations
	op_queue_max_bytes	2	Max size of queue
	op_queue_bytes	2	Size of queue
	bytes	10	Data written to store
	apply_latency	5	Apply latency
	queue_transaction_latency_avg	5	Store operation queue latency

Table 8.9. 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 8.10. Objecter Metrics Table

Collection Name	Metric Name	Bit Field Value	Short Description
objecter	**op_active**	2	Active operations

Collection Name	Metric Name	Bit Field Value	Short Description
	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

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

Table 8.11. 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

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

Collection Name	Metric Name	Bit Field Value	Short Description
	op_rw_latency	5	Latency of read-modify-write operation (including queue time)
	op_rw_process_latency	5	Latency of read-modify-write operation (excluding queue time)
	subop	10	Suboperations
	subop_in_bytes	10	Suboperations total size
	subop_latency	5	Suboperations latency
	subop_w	10	Replicated writes
	subop_w_in_bytes	10	Replicated written data size
	subop_w_latency	5	Replicated writes latency
	subop_pull	10	Suboperations pull requests
	subop_pull_latency	5	Suboperations pull latency
	subop_push	10	Suboperations push messages
	subop_push_in_bytes	10	Suboperations pushed size
	subop_push_latency	5	Suboperations push latency
	pull	10	Pull requests sent
	push	10	Push messages sent
	push_out_bytes	10	Pushed size
	push_in	10	Inbound push messages
	push_in_bytes	10	Inbound pushed size
	recovery_ops	10	Started recovery operations
	loadavg	2	CPU load

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

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

Table 8.12. Recovery State Metrics Table

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

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

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

Table 8.13. 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

8.3.4. The Ceph Object Gateway Metrics Tables

- RADOS Gateway Client Table
- Objecter Metrics Table

- RADOS Gateway Throttle Metrics Table

Table 8.14. RADOS Client Metrics Table

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

Table 8.15. 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

Collection Name	Metric Name	Bit Field Value	Short Description
	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
	osdop_clonerange	10	Clone range operations
	osdop_getxattr	10	Get xattr operations

Collection Name	Metric Name	Bit Field Value	Short Description
	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
	poolstat_send	10	Pool stat operations sent

Collection Name	Metric Name	Bit Field Value	Short Description
	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 8.16. RADOS 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

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

CHAPTER 9. BLUESTORE

BlueStore is a new back-end object store for the OSD daemons. The original object store, FileStore, requires a file system on top of raw block devices. Objects are then written to the file system. BlueStore does not require an initial file system, because BlueStore puts objects directly on the block device.

IMPORTANT

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

9.1. ABOUT BLUESTORE

BlueStore is a new back end for the OSD daemons. Unlike the original FileStore back end, BlueStore stores objects directly on the block devices without any file system interface, which improves the performance of the cluster.

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

Direct management of storage devices

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

Metadata management with RocksDB

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

Full data and metadata checksumming

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

Efficient copy-on-write

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

No large double-writes

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

Multi-device support

BlueStore can use multiple block devices for storing different data, for example: Hard Disk Drive (HDD) for the data, Solid-state Drive (SSD) for metadata, Non-volatile Memory (NVM) or Non-volatile random-access memory (NVRAM) or persistent memory for the RocksDB write-ahead log (WAL). See Section 9.2, "BlueStore Devices" for details.

NOTE

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

Efficient block device usage

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

9.2. BLUESTORE DEVICES

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

BlueStore manages either one, two, or (in certain cases) 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 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.

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 places 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. 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 BLUESTORE

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

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

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

NOTE

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

9.5. ADDING OSDS THAT USE BLUESTORE

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

Prerequisites

- A working Ceph cluster. See the Installation Guide for Red Hat Enterprise Linux or Ubuntu.

Procedure

Use the following commands on the Ansible administration node.

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

```
[osds]
node1
node2
node3
<hostname>
```

Replace:

- **<hostname>** with the name of the OSD node

For example:

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

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

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

3. Create the **host_vars** directory.

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

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

```
[root@admin ceph-ansible] touch host_vars/<hostname>.yml
```

Replace:

- **<hostname>** with the host name of the newly added OSD

For example:

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

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

```
osd_objectstore: bluestore
```

NOTE

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

6. Optional. If you want to store the **block.wal** and **block.db** partitions on dedicated devices, edit the **host_vars/<hostname>.yml** file as follows.

   a. To use dedicated devices for **block.wal**:

   ```
   osd_scenario: non-collocated

   bluestore_wal_devices:
     - <device>
     - <device>
   ```

   Replace:

   - **<device>** with the path to the device

   For example:

   ```
   osd_scenario: non-collocated
   ```

```
bluestore_wal_devices:
  - /dev/sdf
  - /dev/sdg
```

b. To use dedicated devices for **block.db**:

```
osd_scenario: non-collocated

dedicated_devices:
  - <device>
  - <device>
```

Replace:

- **<device>** with the path to the device

For example:

```
osd_scenario: non-collocated

dedicated_devices:
  - /dev/sdh
  - /dev/sdi
```

NOTE

If you use the **osd_scenario: collocated** parameter, the **block.wal** and **block.db** partitions will use the same device as specified with the **devices** parameter. For details, see the *Installing a Red Hat Ceph Storage Cluster* section in the Red Hat Ceph Storage 3 | Installation Guide for Red Hat Enterprise Linux or Ubuntu.

NOTE

To use BlueStore for all OSDs, add the aforementioned parameters to the **group_vars/osds.yml** file.

c. To override the **block.db** and **block.wal** default size in the **group_vars/all.yml** file:

```
ceph_conf_overrides:
  osd:
    bluestore_block_db_size: <value>
    bluestore_block_wal_size: <value>
```

Replace:

- **<value>** with the size in bytes.

For example:

```
ceph_conf_overrides:
  osd:
    bluestore_block_db_size: 14336000000
    bluestore_block_wal_size: 2048000000
```

7. To configure LVM based BlueStore OSDs, use **osd_scenario: lvm** in **host_vars/<hostname>.yml**:

```
osd_scenario: lvm
lvm_volumes:
  - data: <datalv>
    data_vg: <datavg>
```

Replace:

- **<datalv>** with the data logical volume name
- **<datavg>** with the data logical volume group name

For example:

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

NOTE

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

8. If the setting is **osd_scenario: lvm**, then on the target OSD node, as **root**, create the volume groups and logical volumes for the OSDs.

NOTE

If you use spinning drives and solid state drives, Red Hat recommends to place **block.db** on the faster device while **block** is on the slower device.

a. Create volume groups for the **block** and the **block.db** devices:

```
$ vgcreate <vg_name> <device>
```

For example:

```
$ vgcreate ceph-block-0 /dev/sda
```

b. Create logical volumes for the **block** and the **block.db** devices:

```
$ lvcreate -L <size(GB)> -n <lv_name> <vg_name>
```

For example:

```
$ lvcreate -l 100GB -n block-0 ceph-block-0
```

c. Create the OSDs with **ceph-volume**:

```
$ ceph-volume lvm create --bluestore --data <lv_name_block> --block.db
<lv_name_block.db>
```

For example:

```
$ ceph-volume lvm create --bluestore --data ceph-block-0/block-0 --block.db ceph-db-
0/db-0
```

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

```
osd_scenario: lvm
lvm_volumes:
  - data: <datalv>
    wal: <wallv>
    wal_vg: <vg>
    db: <dblv>
    db_vg: <vg>
```

Replace:

- <datalv> with the logical volume where the data should be contained
- <wallv> with the logical volume where the write-ahead-log should be contained
- <vg> with the volume group the WAL and/or DB device LVs are on
- <dblv> with the logical volume the BlueStore internal metadata should be contained

For example:

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

NOTE

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

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

NOTE

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

IMPORTANT

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

11. Use the **ansible-playbook**:

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

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

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

Additional Resources

- Adding an OSD with Ansible

9.6. TUNING BLUESTORE FOR SMALL WRITES WITH BLUESTORE_MIN_ALLOC_SIZE

In BlueStore, the raw partition is allocated and managed in chunks of **bluestore_min_alloc_size**. By default, **bluestore_min_alloc_size** is 64KiB for HDDs, and 16KiB for SSDs. The unwritten area in each chunk is filled with zeroes when it is written to the raw partition. This can lead to wasted unused space when not properly sized for your workload, for example when writing small objects.

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

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

bluestore_min_alloc_size = 4096

NOTE

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

Prerequisites

- A running Red Hat Ceph Storage cluster.
- New servers that can be freshly provisioned as OSD nodes, or:
- OSD nodes that can be redeployed.

Procedure

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

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

   For example:

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

2. If redeploying an existing OSD node, wipe the OSD drives and reinstall the OS. See the Installation Guide for Red Hat Enterprise Linux or the Installation Guide for Ubuntu for more information.

3. Prepare the node for OSD provisioning using Ansible. This includes things like enabling Red Hat Ceph Storage repositories, adding an Ansible user, and enabling Password-less SSH login. See the Installation Guide or the Installation Guide for Ubuntu for more information.

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

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

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

   ```
   [osds]
   NODENAME
   ```

   For example:

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

6. Provision the OSD node using Ansible:

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

   For example:

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

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

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

   For example:

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

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

Additional Resources

- Installation Guide for Red Hat Enterprise Linux
- Installation Guide for Ubuntu

9.7. THE BLUESTORE FRAGMENTATION TOOL

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

9.7.1. Prerequisites

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

9.7.2. What is the BlueStore fragmentation tool?

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

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

Table 9.1. Fragmentation scores' meaning

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

IMPORTANT

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

9.7.3. Checking for fragmentation

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

Prerequisites

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

Online BlueStore fragmentation score

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

    Syntax

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

    Example

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

    b. A more detailed report:

    Syntax

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

    Example

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

Offline BlueStore fragmentation score

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

    Syntax

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

    Example

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

    b. A more detailed report:

    Syntax

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

    Example

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

Additional Resources

- See the Red Hat Ceph Storage 3.3 Administration Guide for details on the fragmentation score.

APPENDIX A. ERROR CODE DEFINITIONS FOR CEPH-MEDIC

The **ceph-medic** utility displays an error code and message for any failing checks. These error codes are a warning or an error and apply to common issues specific to the Ceph daemons. The error codes start with *E* and warning codes start with *W*.

Common Error Messages

ECOM1

A Ceph configuration file can not be found at **/etc/ceph/$CLUSTER_NAME.conf**.

ECOM2

The **ceph** executable was not found.

ECOM3

The **/var/lib/ceph/** directory does not exist or could not be accessed.

ECOM4

The **/var/lib/ceph/** directory is not owned by the **ceph** user.

ECOM5

The **fsid** in the Ceph configuration is different between the nodes in the storage cluster.

ECOM6

The installed version of Ceph is different than the nodes in the storage cluster.

ECOM7

The installed version of Ceph is different than that of the running socket.

ECOM8

The **fsid** was not found in the Ceph configuration.

ECOM9

Cluster FSIDs from running sockets is different than the nodes in the storage cluster.

ECOM10

There are multiple running monitors.

Monitor Error Messages

EMON1

The secret key used in the keyring is different between the nodes in the storage cluster.

Monitor Warning Messages

WMON1

Multiple Monitor directories are found on the same node.

WMON2

Collocated OSDs where found on the Monitors nodes.

OSD Warning Messages

WOSD1

Multiple **ceph_fsid** values where found in the **/var/lib/ceph/osd/** directory.