



Red Hat Ceph Storage

1.2.3

Red Hat Ceph Administration Guide

Red Hat Ceph Storage Administration

Red Hat Customer Content
Services

Legal Notice

Copyright © 2017 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

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

Table of Contents

PREFACE	4
CHAPTER 1. CEPH PROCESS MANAGEMENT	5
1.1. RUNNING CEPH WITH SYSVINIT	6
1.2. RUNNING CEPH WITH UPSTART	7
1.3. RUNNING CEPH AS A SERVICE	9
PART I. MONITORING CEPH	11
CHAPTER 2. HIGH LEVEL MONITORING	12
2.1. INTERACTIVE MODE	12
2.2. CHECKING CLUSTER HEALTH	12
2.3. WATCHING A CLUSTER	12
2.4. CHECKING A CLUSTER'S USAGE STATS	13
2.5. CHECKING A CLUSTER'S STATUS	14
2.6. CHECKING MONITOR STATUS	15
2.7. USING THE ADMIN SOCKET	16
2.8. CHECKING OSD STATUS	16
CHAPTER 3. LOW-LEVEL MONITORING	18
3.1. PG SETS	18
3.2. PEERING	18
3.3. MONITORING PLACEMENT GROUP STATES	19
3.4. IDENTIFYING TROUBLED PGS	25
3.5. FINDING AN OBJECT LOCATION	25
PART II. CEPH OVERRIDES	26
CHAPTER 4. SETTING/UNSETTING OVERRIDES	27
CHAPTER 5. USE CASES	29
PART III. USER MANAGEMENT	30
CHAPTER 6. BACKGROUND	31
6.1. USER	31
6.2. AUTHORIZATION (CAPABILITIES)	31
6.3. POOL	33
6.4. NAMESPACE	33
CHAPTER 7. MANAGING USERS	34
7.1. LIST USERS	34
7.2. GET A USER	34
7.3. ADD A USER	35
7.4. MODIFY USER CAPABILITIES	36
7.5. DELETE A USER	36
7.6. PRINT A USER'S KEY	36
7.7. IMPORT A USER(S)	37
CHAPTER 8. KEYRING MANAGEMENT	38
8.1. CREATE A KEYRING	38
8.2. ADD A USER TO A KEYRING	39
8.3. CREATE A USER	39
8.4. MODIFY A USER	39
.....	..

CHAPTER 9. COMMAND LINE USAGE	41
CHAPTER 10. LIMITATIONS	42
PART IV. MANAGING CLUSTER SIZE	43
CHAPTER 11. ADD A MONITOR	44
11.1. HOST CONFIGURATION	44
11.2. CEPH-DEPLOY	45
11.3. MANUAL	45
CHAPTER 12. REMOVE A MONITOR	48
12.1. CEPH-DEPLOY	48
12.2. MANUAL	48
12.3. REMOVING MONITORS FROM AN UNHEALTHY CLUSTER	49
CHAPTER 13. ADD AN OSD	50
13.1. CEPH-DEPLOY	50
13.2. MANUAL	51
CHAPTER 14. CONNECT OSD HOSTS TO CALAMARI	53
CHAPTER 15. REMOVING OSDS (MANUAL)	54
15.1. TAKE THE OSD OUT OF THE CLUSTER	54
15.2. OBSERVE THE DATA MIGRATION	54
15.3. STOPPING THE OSD	54
15.4. REMOVING THE OSD	54
PART V. PERFORMANCE COUNTERS	56
CHAPTER 16. OVERVIEW	57
CHAPTER 17. ACCESS	58
17.1. SCHEMA	58
17.2. DUMP	60

PREFACE

The Red Hat Ceph Storage cluster is the foundation for all Ceph deployments. Based upon RADOS (Reliable Autonomic Distributed Object Store), Ceph Storage Clusters consist of two types of daemons:

- ✦ A Ceph OSD (OSD) stores data as objects within placement groups assigned to the OSD
- ✦ A Ceph monitor maintains a master copy of the cluster map.

A Red Hat Ceph Storage cluster may contain thousands of storage nodes. A minimal system will have at least one Ceph monitor and three Ceph OSDs for peering and object durability. A production system will have three or more monitors for high availability and typically a minimum of 50 OSDs for acceptable load balancing, re-balancing and recovery. See the installation guide to get a minimum Ceph cluster up and running.

CHAPTER 1. CEPH PROCESS MANAGEMENT

Each time you to **start**, **restart**, and **stop** Ceph daemons (or your entire cluster) you must specify at least one option and one command. You may also specify a daemon type or a daemon instance. :

```
{commandline} [options] [commands] [daemons]
```

The **ceph** options include:

Option	Shortcut	Description
--verbose	-v	Use verbose logging.
--allhosts	-a	Execute on all nodes in the Ceph configuration. Otherwise, it only executes on localhost .
--restart	N/A	Automatically restart daemon if it core dumps.
--norestart	N/A	Don't restart a daemon if it core dumps.
--conf	-c	Use an alternate configuration file.

The **ceph** commands include:

Command	Description
start	Start the daemon(s).
stop	Stop the daemon(s).
forcestop	Force the daemon(s) to stop. Same as kill -9
killall	Kill all daemons of a particular type.
cleanlogs	Cleans out the log directory.

Command	Description
cleanalllogs	Cleans out everything in the log directory.

For subsystem operations, the **ceph** service can target specific daemon types by adding a particular daemon type for the **[daemons]** option. Daemon types include:

- ✦ **mon**
- ✦ **osd**

1.1. RUNNING CEPH WITH SYSVINIT

Using System V Init (**sysvinit**) is the common way to run Ceph with RHEL 6/CentOS 6 and RHEL 7/CentOS 7 for Ceph Storage v1.2.3.

1.1.1. Starting all Daemons

To start your Ceph cluster, execute **ceph** with the **start** command. Use the following syntax:

```
sudo /etc/init.d/ceph [options] [start|restart] [daemonType|daemonID]
```

The following examples illustrates a typical use case:

```
sudo /etc/init.d/ceph -a start
```

Once you execute with **-a** (i.e., execute on all nodes), Ceph should begin operating.

1.1.2. Stopping all Daemons

To stop your Ceph cluster, execute **ceph** with the **stop** command. Use the following syntax:

```
sudo /etc/init.d/ceph [options] stop [daemonType|daemonID]
```

The following examples illustrates a typical use case:

```
sudo /etc/init.d/ceph -a stop
```

Once you execute with **-a** (i.e., execute on all nodes), Ceph should stop operating.

1.1.3. Starting all Daemons by Type

To start all Ceph daemons of a particular type on the local Ceph node, use the following syntax:

```
sudo /etc/init.d/ceph start {daemon-type}
sudo /etc/init.d/ceph start osd
```

To start all Ceph daemons of a particular type on another node, use the following syntax:

```
sudo /etc/init.d/ceph -a start {daemon-type}
sudo /etc/init.d/ceph -a start osd
```

1.1.4. Stopping all Daemons by Type

To stop all Ceph daemons of a particular type on the local Ceph node, use the following syntax:

```
sudo /etc/init.d/ceph stop {daemon-type}
sudo /etc/init.d/ceph stop osd
```

To stop all Ceph daemons of a particular type on another node, use the following syntax:

```
sudo /etc/init.d/ceph -a stop {daemon-type}
sudo /etc/init.d/ceph -a stop osd
```

1.1.5. Starting a Daemon

To start a Ceph daemon on the local Ceph node, use the following syntax:

```
sudo /etc/init.d/ceph start {daemon-type}.{instance}
sudo /etc/init.d/ceph start osd.0
```

To start a Ceph daemon on another node, use the following syntax:

```
sudo /etc/init.d/ceph -a start {daemon-type}.{instance}
sudo /etc/init.d/ceph -a start osd.0
```

1.1.6. Stopping a Daemon

To stop a Ceph daemon on the local Ceph node, use the following syntax:

```
sudo /etc/init.d/ceph stop {daemon-type}.{instance}
sudo /etc/init.d/ceph stop osd.0
```

To stop a Ceph daemon on another node, use the following syntax:

```
sudo /etc/init.d/ceph -a stop {daemon-type}.{instance}
sudo /etc/init.d/ceph -a stop osd.0
```

1.2. RUNNING CEPH WITH UPSTART

On Ubuntu 14.04 (Trusty), you can run Ceph using the event-based [Upstart](#). Upstart does not require you to define daemon instances in the Ceph configuration file.

To list the Ceph Upstart jobs and instances on a node, execute:

```
sudo initctl list | grep ceph
```

See [initctl](#) for additional details.

1.2.1. Starting all Daemons

To start all daemons on a Ceph Node (irrespective of type), execute the following:

```
sudo start ceph-all
```

1.2.2. Stopping all Daemons

To stop all daemons on a Ceph Node (irrespective of type), execute the following:

```
sudo stop ceph-all
```

1.2.3. Starting all Daemons by Type

To start all daemons of a particular type on a Ceph Node, execute one of the following:

```
sudo start ceph-osd-all
sudo start ceph-mon-all
sudo start ceph-mds-all
```

1.2.4. Stopping all Daemons by Type

To stop all daemons of a particular type on a Ceph Node, execute one of the following:

```
sudo stop ceph-osd-all
sudo stop ceph-mon-all
sudo stop ceph-mds-all
```

1.2.5. Starting a Daemon

To start a specific daemon instance on a Ceph Node, execute one of the following:

```
sudo start ceph-osd id={id}
sudo start ceph-mon id={hostname}
sudo start ceph-mds id={hostname}
```

For example:

```
sudo start ceph-osd id=1
sudo start ceph-mon id=ceph-server
sudo start ceph-mds id=ceph-server
```

1.2.6. Stopping a Daemon

To stop a specific daemon instance on a Ceph Node, execute one of the following:

```
sudo stop ceph-osd id={id}
sudo stop ceph-mon id={hostname}
sudo stop ceph-mds id={hostname}
```

For example:

```
sudo stop ceph-osd id=1
sudo start ceph-mon id=ceph-server
sudo start ceph-mds id=ceph-server
```

1.3. RUNNING CEPH AS A SERVICE

You may operate Ceph as a service.

1.3.1. Starting all Daemons

To start your Ceph cluster, execute **ceph** with the **start** command. Use the following syntax:

```
sudo service ceph [options] [start|restart] [daemonType|daemonID]
```

The following examples illustrates a typical use case:

```
sudo service ceph -a start
```

Once you execute with **-a** (i.e., execute on all nodes), Ceph should begin operating.

1.3.2. Stopping all Daemons

To stop your Ceph cluster, execute **ceph** with the **stop** command. Use the following syntax:

```
sudo service ceph [options] stop [daemonType|daemonID]
```

For example:

```
sudo service ceph -a stop
```

Once you execute with **-a** (i.e., execute on all nodes), Ceph should shut down.

1.3.3. Starting all Daemons by Type

To start all Ceph daemons of a particular type on the local Ceph node, use the following syntax:

```
sudo service ceph start {daemon-type}
sudo service ceph start osd
```

To start all Ceph daemons of a particular type on all nodes, use the following syntax:

```
sudo service ceph -a start {daemon-type}
sudo service ceph -a start osd
```

1.3.4. Stopping all Daemons by Type

To stop all Ceph daemons of a particular type on the local Ceph node, use the following syntax:

```
sudo service ceph stop {daemon-type}
sudo service ceph stop osd
```

To stop all Ceph daemons of a particular type on all nodes, use the following syntax:

```
sudo service ceph -a stop {daemon-type}
sudo service ceph -a stop osd
```

1.3.5. Starting a Daemon

To start a Ceph daemon on the local Ceph node, use the following syntax:

```
sudo service ceph start {daemon-type}.{instance}
sudo service ceph start osd.0
```

To start a Ceph daemon on another node, use the following syntax:

```
sudo service ceph -a start {daemon-type}.{instance}
sudo service ceph -a start osd.0
```

1.3.6. Stopping a Daemon

To stop a Ceph daemon on the local Ceph node, use the following syntax:

```
sudo service ceph stop {daemon-type}.{instance}
sudo service ceph stop osd.0
```

To stop a Ceph daemon on another node, use the following syntax:

```
sudo service ceph -a stop {daemon-type}.{instance}
sudo service ceph -a stop osd.0
```

PART I. MONITORING CEPH

Once you have a running cluster, you may begin monitoring your cluster to ensure that your monitor and OSD daemons are running (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.

CHAPTER 2. HIGH LEVEL MONITORING

High level monitoring of a 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 your cluster capacity to ensure that the cluster doesn't exceed its **full ratio**. The Calamari instance on your administration node is the most common way to conduct high-level monitoring. However, you may also use CLI, the admin socket or the Ceph API to monitor your cluster.

2.1. INTERACTIVE MODE

To run the **ceph** tool 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
```

2.2. CHECKING CLUSTER HEALTH

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

```
ceph health
```

If you specified non-default locations for your 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 your cluster is ready, **ceph health** should return a message such as **HEALTH_OK**. At that point, it is okay to begin using the cluster.

2.3. 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; and,
- ✦ 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 number) 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.

2.4. CHECKING A CLUSTER'S USAGE STATS

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 your cluster uses for your data.

- ✦ **SIZE:** The overall storage capacity of the cluster.
- ✦ **AVAIL:** The amount of free space available in the 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 your 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, snapshots or clones. As a result, the sum of the **USED** and **%USED** amounts will not add up to the **RAW USED** and **%RAW USED** amounts in the **GLOBAL** section of the output. See [How Ceph Calculates Data Usage](#) for details.

2.5. 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

```

2.6. CHECKING MONITOR STATUS

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

To see display the monitor map, execute the following:

```
ceph mon stat
```

Or:

```
ceph mon dump
```

To check the quorum status for the monitor cluster, execute the following:

```
ceph quorum_status
```

Ceph will return the quorum status. For example, a Ceph 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"}
    ]
  }
}

```

2.7. USING THE ADMIN SOCKET

The Ceph admin socket allows you to query a daemon via a socket interface. By default, Ceph sockets reside under `/var/run/ceph`. To access a daemon via the admin socket, login to the host running the daemon and use the following command:

```
ceph --admin-daemon /var/run/ceph/{socket-name}
```

To view the available admin socket commands, execute the following command:

```
ceph --admin-daemon /var/run/ceph/{socket-name} help
```

The admin socket command enables you to show and set your configuration at runtime.

Additionally, you can set configuration values at runtime directly (i.e., the admin socket bypasses the monitor, unlike `ceph tell {daemon-type}.{id} injectargs`, which relies on the monitor but doesn't require you to login directly to the host in question).

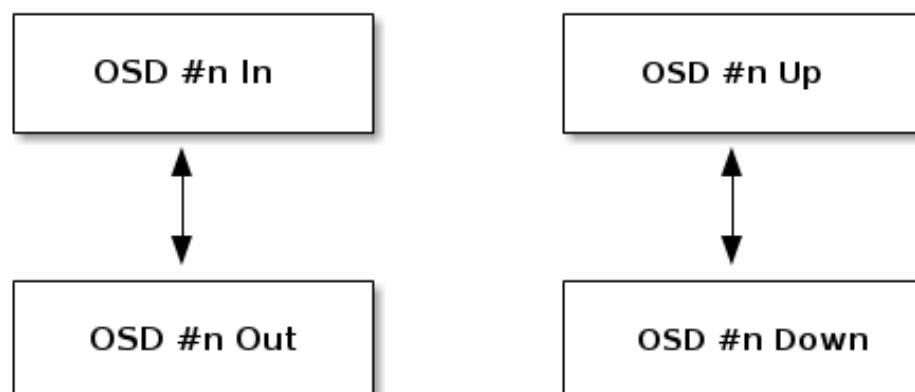
2.8. CHECKING OSD STATUS

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 (**down**). If an OSD is **up**, it may be either **in** the cluster (you can read and write data) or it is **out** of the 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:

1. You haven't started the cluster yet (it won't respond).
2. 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.
3. You just added or removed an OSD.
4. You just have modified your 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 are **up** (y) and how many are **in** (z).

```
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
```

The output should look something like this:

```
# 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 your cluster by identifying the physical locations faster.

If an OSD is **down**, connect to the node and start it. You can use Calamari to restart the node, or you can use the CLI. For example:

```
sudo /etc/init.d/ceph start osd.1
```

CHAPTER 3. 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.

3.1. PG 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 your 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 (i.e., Ceph usually echoes a "HEALTH WARN" state with a "stuck stale" message in such scenarios).

To retrieve a list of placement groups, execute:

```
ceph pg dump
```

To view which OSDs are within the Acting Set or the Up Set for a given placement group, execute:

```
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]
```

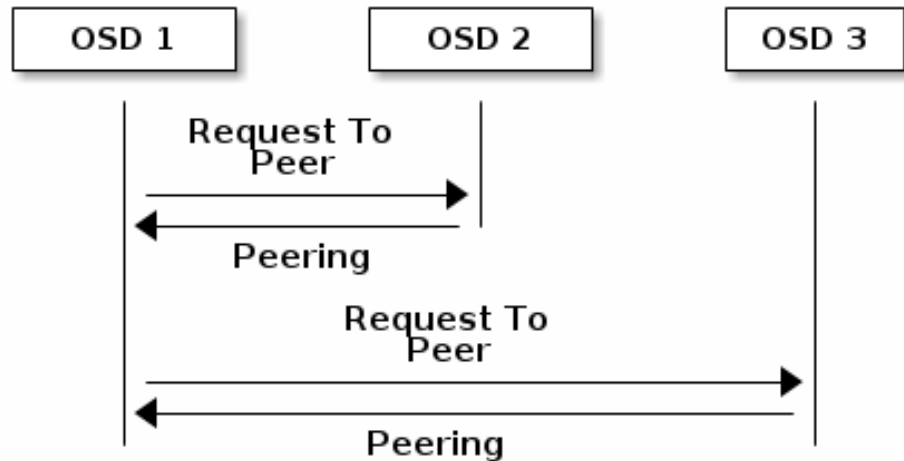


Note

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

3.2. 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).



3.3. 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:

1. You have just created a pool and placement groups haven't peered yet.
2. The placement groups are recovering.
3. You have just added an OSD to or removed an OSD from the cluster.
4. You have just modified your CRUSH map and your placement groups are migrating.
5. There is inconsistent data in different replicas of a placement group.
6. Ceph is scrubbing a placement group's replicas.
7. 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 are in a particular state such as **active+clean** (y). :

```
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.

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

Placement Group IDs

Placement group IDs consist of the pool number (not 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}
```

And it typically looks like this:

```
0.1f
```

To retrieve a list of placement groups, execute the following:

```
ceph pg dump
```

You can also format the output in JSON format and save it to a file:

```
ceph pg dump -o {filename} --format=json
```

To query a particular placement group, execute the following:

```
ceph pg {poolnum}.{pg-id} query
```

Ceph will output the query 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": [

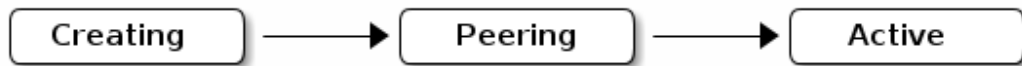
            ]
        }
    },
    {
        "name": "Started",
        "enter_time": "2013-01-23 09:35:31.581160"
    }
]
}

```

The following subsections describe common states in greater detail.

3.3.1. 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.



3.3.2. 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.

3.3.3. 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.

3.3.4. 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.

3.3.5. 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.

3.3.6. 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 (1 thread by default). The **osd recovery thread timeout** sets a thread timeout, because multiple OSDs may fail, restart and re-peer at staggered rates. The **osd recovery max active** setting limits the number of recovery requests an OSD will entertain simultaneously to prevent the OSD from failing to serve . The **osd recovery max chunk** setting limits the size of the recovered data chunks to prevent network congestion.

3.3.7. Back Filling

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. Back filling 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 can't 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 (85%, by default). If an OSD refuses a backfill request, the **osd backfill retry interval** enables an OSD to retry the request (after 10 seconds, by default). OSDs can also set **osd backfill scan min** and **osd backfill scan max** to manage scan intervals (64 and 512, by default).

3.3.8. 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.

3.3.9. 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 (e.g., a temporary network fault). By default, OSD daemons report their placement group, up thru, boot and failure statistics every half second (i.e., **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 your cluster, it is common to see the **stale** state until the peering process completes. After your 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.

3.4. IDENTIFYING TROUBLED PGS

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 (configured by **mon osd report timeout**).

To identify stuck placement groups, execute the following:

```
ceph pg dump_stuck [unclean|inactive|stale|undersized|degraded]
```

3.5. 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 {poolname} {object-name}
```

PART II. CEPH 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.

CHAPTER 4. SETTING/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.

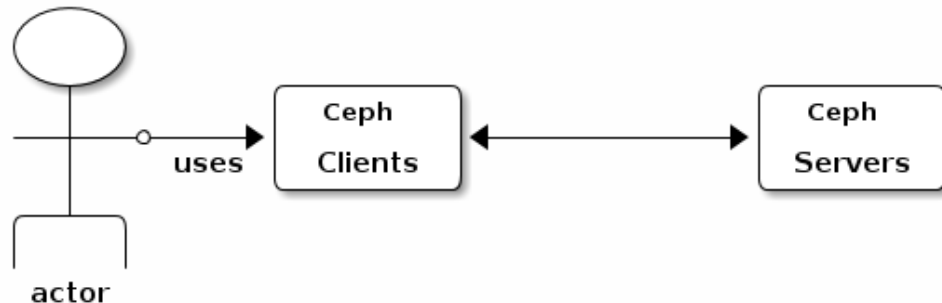
Flag	Description
nodeep-scrub	Ceph will prevent new deep scrubbing operations.
notieragent	Ceph will disable the process that is looking for cold/dirty objects to flush and evict.

CHAPTER 5. 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**.

PART III. 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 via 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 via 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 the Red Hat Ceph Configuration Guide.

CHAPTER 6. BACKGROUND

Irrespective of the type of Ceph client (e.g., Block Device, Object Storage, Filesystem, native API, CLI, etc.), 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.

6.1. 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 your 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 your 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.

6.2. 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** and **allow profile {cap}**. For example:

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

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

```
osd 'allow {capability}' [pool={poolname}] [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**Description**

Precedes access settings for a daemon.

r**Description**

Gives the user read access. Required with monitors to retrieve the CRUSH map.

w**Description**

Gives the user write access to objects.

x**Description**

Gives the user the capability to call class methods (i.e., both read and write) and to conduct **auth** operations on monitors.

class-read**Descriptions**

Gives the user the capability to call class read methods. Subset of **x**.

class-write**Description**

Gives the user the capability to call class write methods. Subset of **x**.

*******Description**

Gives the user read, write and execute permissions for a particular daemon/pool, and the ability to execute admin commands.

profile osd

Description

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**Description**

Gives a user permissions to bootstrap an OSD. Conferred on deployment tools such as **ceph-disk**, **ceph-deploy**, etc. so that they have permissions to add keys, etc. when bootstrapping an OSD.

6.3. 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 (e.g., cloud volumes/images, object storage, hot storage, cold storage, etc.). For example, when deploying Ceph as a backend for OpenStack, a typical deployment would have pools for volumes, images, backups and virtual machines, and users such as **client.glance**, **client.cinder**, etc.

6.4. 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.

CHAPTER 7. 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.

7.1. LIST USERS

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

```
ceph auth list
```

Ceph will list out all users in your cluster. For example, in a two-node exemplary 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: AQBHCbtT6APDhAA5W00cBchwKQjh3dkKsyPjw==
  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** (i.e., 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 {filename}** option with **ceph auth list** to save the output to a file.

7.2. GET A USER

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

```
ceph auth get {TYPE.ID}
```

For example:

■

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

You may also use the **-o {filename}** option with **ceph auth get** to save the output to a file. Developers may also execute the following:

```
ceph auth export {TYPE.ID}
```

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

7.3. ADD A USER

Adding a user creates a username (i.e., **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 {filename}** 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 (e.g., libvirt). If the user already exists, this command simply returns the key. You may use the **-o {filename}** 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!

7.4. MODIFY USER CAPABILITIES

The **ceph auth caps** command allows you to specify a user and change the user's capabilities. To add capabilities, use the form:

```
ceph auth caps USERTYPE.USERID {daemon} 'allow [r|w|x|*|...] [pool={pool-name}] [namespace={namespace-name}]'
```

For 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 [Authorization \(Capabilities\)](#) for additional details on capabilities.

7.5. 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.

7.6. 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 (e.g., `libvirt`):

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


7.7. IMPORT A USER(S)

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

```
ceph auth import -i /path/to/keyring
```

For example:

```
sudo 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.

CHAPTER 8. KEYRING MANAGEMENT

When you access Ceph via 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 your Ceph configuration file unless you want to override the defaults (not recommended):

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

The **\$cluster** metavariable is your Ceph cluster name as defined by the name of the Ceph configuration file (i.e., **ceph.conf** means the cluster name is **ceph**; thus, **ceph.keyring**). The **\$name** metavariable is the user type and user ID (e.g., **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 (e.g., **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.

The User Management_ section details 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.

8.1. 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 (e.g., **\$cluster.keyring**) for the keyring filename 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 your Ceph configuration file. For example, create **ceph.keyring** by executing the following:

```
sudo 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.

8.2. ADD A USER TO A KEYRING

When you Add a User_ to the Ceph Storage Cluster, you can use the Get a User_ procedure to retrieve a user, key and capabilities and save the user to a keyring.

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:

```
sudo 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:

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

8.3. 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:

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

See Authorization (Capabilities)_ for additional details on capabilities.

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

```
sudo 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. :

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

8.4. 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:

```
sudo 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. :

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

See [Import a User\(s\)](#) for details on updating a Ceph Storage Cluster user from a keyring.

You may also [Modify User Capabilities](#) directly in the cluster, store the results to a keyring file; then, import the keyring into your main **ceph.keyring** file.

CHAPTER 9. 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: +

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

CHAPTER 10. 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 your 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 your environment using a method that provides sufficient security for your 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. Further, Ceph does not include options to encrypt user data in the object store. Users can hand-encrypt and store their own data in the Ceph object store, of course, but Ceph provides no features to perform object encryption itself. Those storing sensitive data in Ceph should consider encrypting their data before providing it to the Ceph system.

PART IV. MANAGING CLUSTER SIZE

Managing cluster size generally involves adding or removing Ceph monitors or OSDs. If you are bootstrapping a cluster for the first time, see the Red Hat Ceph Installation Guide.

CHAPTER 11. ADD A MONITOR

Ceph monitors are light-weight processes that maintain a master copy of the cluster map. All Ceph clients contact a Ceph monitor and retrieve the current copy of the cluster map, enabling them to bind to pool and read/write data.

When you have a cluster up and running, you may add or remove monitors from the cluster at runtime. You can run a cluster with 1 monitor. We recommend at least 3 monitors for a production cluster. Ceph monitors use a variation of the Paxos protocol to establish consensus about maps and other critical information across the cluster. Due to the nature of Paxos, Ceph requires a majority of monitors running to establish a quorum (thus establishing consensus).

We recommend deploying an odd-number of monitors, but it is not mandatory. An odd-number of monitors has a higher resiliency to failures than an even-number of monitors. To maintain a quorum on a 2 monitor deployment, Ceph cannot tolerate any failures in order; with 3 monitors, one failure; with 4 monitors, one failure; with 5 monitors, two failures. This is why an odd-number is advisable. Summarizing, Ceph needs a majority of monitors to be running (and able to communicate with each other), but that majority can be achieved using a single monitor, or 2 out of 2 monitors, 2 out of 3, 3 out of 4, etc.

For an initial deployment of a multi-node Ceph cluster, we recommend to deploying three monitors, increasing the number two at a time if a valid need for more than three exists.

Since monitors are light-weight, it is possible to run them on the same host as an OSD; however, we recommend running them on separate hosts, because **fsync** issues with the kernel may impair performance.



Note

A majority of monitors in your cluster must be able to reach each other in order to establish a quorum.

11.1. HOST CONFIGURATION

When adding Ceph monitors to your cluster, you should be deploying them on separate hosts. Running Ceph monitors on the same host does not provide any additional high availability assurance if a host fails. Ideally, the host hardware should be uniform throughout your monitor cluster.

For details on the minimum recommendations for Ceph monitor hardware, see Hardware Recommendations.

For installation, see the Red Hat Ceph Installation Guide and be sure to address the pre-installation requirements.

Add your monitor host to a rack in your cluster, connect it to the network and ensure that it has network connectivity.



Important

You must install NTP, and you must open port 6789.

11.2. CEPH-DEPLOY

From your admin node in the directory where you keep your Ceph cluster configuration, install Red Hat Ceph Storage.

```
ceph-deploy install --release <version> <ceph-node> [<ceph-node>]
```

For example, to install Ceph on two new monitor hosts **node5** and **node6**, you would execute the following:

```
ceph-deploy install --release ice1.2.3 node5 node6
```

Once you have installed Ceph, you can create new monitors.

```
ceph-deploy mon create <ceph-node> [<ceph-node>]
```

For example, to add Ceph monitors on monitor hosts **node5** and **node6**, you would execute the following:

```
ceph-deploy mon create node5 node6
```

Check to see that your monitors have joined the quorum.

```
ceph quorum_status --format json-pretty
```

To ensure the cluster identifies the monitors on start/restart, add the monitor hostname and IP address to your Ceph configuration file.

```
[mon.node5]
host = node5
addr = 10.0.0.1:6789

[mon.node6]
host = node6
addr = 10.0.0.2:6789
```

Then, push a new copy of the Ceph configuration file to your Ceph nodes.

Finally, connect your monitors to Calamari.

```
ceph-deploy calamari connect <ceph-node>[<ceph-node> ...]
```

For example, using the exemplary **node5** and **node6** from above, you would execute:

```
ceph-deploy calamari connect node5 node6
```

11.3. MANUAL

This section is intended for users who wish to use a third party deployment tool (e.g., Puppet, Chef, Juju).

This procedure creates a **ceph-mon** data directory, retrieves the monitor map and monitor keyring, and adds a **ceph-mon** daemon to your cluster. If this results in only two monitor daemons, you may add more monitors by repeating this procedure until you have a sufficient number of **ceph-mon** daemons to achieve a quorum.

At this point you should define your monitor's id. Traditionally, monitors have been named with single letters (**a**, **b**, **c**, ...), but you are free to define the id as you see fit. For the purpose of this document, please take into account that **{mon-id}** should be the id you chose, without the **mon.** prefix (i.e., **{mon-id}** should be the **a** on **mon.a**).

1. Create the default directory on the machine that will host your new monitor.

```
ssh {new-mon-host}
sudo mkdir /var/lib/ceph/mon/ceph-{mon-id}
```

2. Create a temporary directory **{tmp}** to keep the files needed during this process. This directory should be different from the monitor's default directory created in the previous step, and can be removed after all the steps are executed.

```
mkdir {tmp}
```

3. Retrieve the keyring for your monitors, where **{tmp}** is the path to the retrieved keyring, and **{key-filename}** is the name of the file containing the retrieved monitor key.

```
ceph auth get mon. -o {tmp}/{key-filename}
```

4. Retrieve the monitor map, where **{tmp}** is the path to the retrieved monitor map, and **{map-filename}** is the name of the file containing the retrieved monitor monitor map.

```
ceph mon getmap -o {tmp}/{map-filename}
```

5. Prepare the monitor's data directory created in the first step. You must specify the path to the monitor map so that you can retrieve the information about a quorum of monitors and their **fsid**. You must also specify a path to the monitor keyring:

```
sudo ceph-mon -i {mon-id} --mkfs --monmap {tmp}/{map-filename} --
keyring {tmp}/{key-filename}
```

6. Add the new monitor to the list of monitors for you cluster (runtime). This enables other nodes to use this monitor during their initial startup.

```
ceph mon add <mon-id> <ip>[:<port>]
```

7. Start the new monitor and it will automatically join the cluster. The daemon needs to know which address to bind to, either via **--public-addr {ip:port}** or by setting **mon addr** in the appropriate section of **ceph.conf**. For example:

```
ceph-mon -i {mon-id} --public-addr {ip:port}
```

8. To ensure the cluster identifies the monitor on start/restart, add the monitor hostname and IP address to your Ceph configuration file.

```
[mon.{mon-id}]  
host = {mon-id}  
addr = {ip:port}
```

Then, push a new copy of the Ceph configuration file to your Ceph nodes.

9. Finally, from the admin node in the directory where you keep your cluster's Ceph configuration, connect your monitor to Calamari.

```
ceph-deploy calamari connect <ceph-node>[<ceph-node> ...]
```

CHAPTER 12. REMOVE A MONITOR

When you remove monitors from a cluster, consider that Ceph monitors use PAXOS to establish consensus about the master cluster map. You must have a sufficient number of monitors to establish a quorum for consensus about the cluster map.

12.1. CEPH-DEPLOY

To remove a monitor from your cluster, use the **mon destroy** command.

```
ceph-deploy mon destroy <ceph-node> [<ceph-node>]
```

For example, to remove Ceph monitors on monitor hosts **node5** and **node6**, you would execute the following:

```
ceph-deploy mon destroy node5 node6
```

Check to see that your monitors have left the quorum.

```
ceph quorum_status --format json-pretty
```



Important

Ensure that you remove any references to this monitor in your Ceph configuration file; then, push a new copy of the Ceph configuration file to your Ceph nodes.

Ideally, you should remove the monitor host from Calamari. Get the cluster ID:

```
http://{calamari-fqdn}/api/v2/cluster
```

Then, remove the monitor host from Calamari.

```
http://{calamari-fqdn}/api/v2/server
http://{calamari-fqdn}/api/v2/key/{host-fqdn}
```

12.2. MANUAL

This procedure removes a **ceph-mon** daemon from your cluster. If this procedure results in only two monitor daemons, you may add or remove another monitor until you have a number of **ceph-mon** daemons that can achieve a quorum.

1. Stop the monitor. :

```
service ceph -a stop mon.{mon-id}
```

2. Remove the monitor from the cluster. :

```
ceph mon remove {mon-id}
```

3. Remove the monitor entry from **ceph.conf**.

12.3. REMOVING MONITORS FROM AN UNHEALTHY CLUSTER

This procedure removes a **ceph-mon** daemon from an unhealthy cluster—i.e., a cluster that has placement groups that are persistently not **active + clean**.

1. Identify a surviving monitor and log in to that host. :

```
ceph mon dump
ssh {mon-host}
```

2. Stop the **ceph-mon** daemon and extract a copy of the monap file. :

```
service ceph stop mon || stop ceph-mon-all
ceph-mon -i {mon-id} --extract-monmap {map-path}
# for example,
ceph-mon -i a --extract-monmap /tmp/monmap
```

3. Remove the non-surviving monitors. For example, if you have three monitors, **mon.a**, **mon.b**, and **mon.c**, where only **mon.a** will survive, follow the example below:

```
monmaptool {map-path} --rm {mon-id}
# for example,
monmaptool /tmp/monmap --rm b
monmaptool /tmp/monmap --rm c
```

4. Inject the surviving map with the removed monitors into the surviving monitors. For example, to inject a map into monitor **mon.a**, follow the example below:

```
ceph-mon -i {mon-id} --inject-monmap {map-path}
# for example,
ceph-mon -i a --inject-monmap /tmp/monmap
```

CHAPTER 13. ADD AN OSD

When you have a cluster up and running, you may add OSDs or remove OSDs from the cluster at runtime.

A Ceph OSD generally consists of one **ceph-osd** daemon for one storage drive (and its associated journal) within a host machine. If your host has multiple storage drives, you may map one **ceph-osd** daemon for each drive.

Generally, it's a good idea to check the capacity of your cluster to see if you are reaching the upper end of its capacity. As your cluster reaches its **near full** ratio, you should add one or more OSDs to expand your cluster's capacity.



Important

Do not let your cluster reach its **full ratio** before adding an OSD. OSD failures that occur after the cluster reaches its **near full** ratio may cause the cluster to exceed its **full ratio**. Ceph will block write access to protect your data until you resolve storage capacity issues. **DO NOT REMOVE** OSDs without considering the impact on the **full ratio** first.

OSDs and their supporting hardware should be similarly configured as a storage strategy for the pool(s) that will use the OSDs. Ceph prefers uniform hardware across pools for a consistent performance profile. For best performance, consider a CRUSH hierarchy with drives of the same type/size. View the Storage Strategies guide for details.

If you are adding drives of dissimilar size, you can adjust their weights. When you add the OSD to the CRUSH map, consider the weight you give to the new OSD. Hard drive capacity grows approximately 40% per year, so newer OSD hosts may have larger hard drives than older hosts in the cluster (i.e., they may have greater weight).

13.1. CEPH-DEPLOY

Before creating OSDs, consider the following:

- ✦ We recommend using the XFS filesystem (default).
- ✦ We recommend using SSDs for journals. It is common to partition SSDs to serve multiple OSDs. Ensure that the number of SSD partitions does not exceed your SSD's sequential write limits. Also, ensure that SSD partitions are properly aligned, or their write performance will suffer.
- ✦ We recommend using **ceph-deploy disk zap** on a Ceph OSD drive before executing **ceph-deploy osd create**. For example:

```
ceph-deploy disk zap <ceph-node>:<data-drive>
```

From your admin node, use **ceph-deploy** to prepare the OSDs.

```
ceph-deploy osd prepare <ceph-node>:<data-drive> <journal-partition>
[<ceph-node>:<data-drive> <journal-partition>]
```

For example:

■

```
ceph-deploy osd prepare node2:sdb ssdb node3:sdd ssdb node4:sdd ssdb
```

In the foregoing example, **sdb** is a spinning hard drive. Ceph will use the entire drive for OSD data. **ssdb** is a partition on an SSD drive, which Ceph will use to store the journal for the OSD.

Once you prepare OSDs, use **ceph-deploy** to activate the OSDs.

```
ceph-deploy osd activate <ceph-node>:<data-drive> <journal-partition>
[<ceph-node>:<data-drive> <journal-partition>]
```

For example:

```
ceph-deploy osd activate node2:sdb ssdb node3:sdd ssdb node4:sdd ssdb
```

To achieve an **active + clean** state, you must add as many OSDs as the value of **osd pool default size = <n>** from your Ceph configuration file.

13.2. MANUAL

This procedure sets up a **ceph-osd** daemon, configures it to use one drive, and configures the cluster to distribute data to the OSD. If your host has multiple drives, you may add an OSD for each drive by repeating this procedure.

To add an OSD, create a data directory for it, mount a drive to that directory, add the OSD to the cluster, and then add it to the CRUSH map.

1. Create the OSD. If no UUID is given, it will be set automatically when the OSD starts up. The following command will output the OSD number, which you will need for subsequent steps. :

```
ceph osd create [{uuid}]
```

2. Create the default directory on your new OSD. :

```
ssh {new-osd-host}
sudo mkdir /var/lib/ceph/osd/ceph-{osd-number}
```

3. If the OSD is for a drive other than the OS drive, prepare it for use with Ceph, and mount it to the directory you just created:

```
ssh {new-osd-host}
sudo mkfs -t {fstype} /dev/{drive}
sudo mount -o user_xattr /dev/{hdd} /var/lib/ceph/osd/ceph-{osd-number}
```

4. Initialize the OSD data directory. :

```
ssh {new-osd-host}
ceph-osd -i {osd-num} --mkfs --mkkey
```

The directory must be empty before you can run **ceph-osd**.

5. Register the OSD authentication key. The value of **ceph** for **ceph-{osd-num}** in the path is the **\$cluster-\$id**. If your cluster name differs from **ceph**, use your cluster name instead.:

```
ceph auth add osd.{osd-num} osd 'allow *' mon 'allow rwx' -i /var/lib/ceph/osd/ceph-{osd-num}/keyring
```

6. Add the OSD to the CRUSH map so that the OSD can begin receiving data. The **ceph osd crush add** command allows you to add OSDs to the CRUSH hierarchy wherever you wish. If you specify at least one bucket, the command will place the OSD into the most specific bucket you specify, *and* it will move that bucket underneath any other buckets you specify. **Important:** If you specify only the root bucket, the command will attach the OSD directly to the root, but CRUSH rules expect OSDs to be inside of hosts.

Execute the following:

```
ceph osd crush add {id-or-name} {weight} [{bucket-type}={bucket-name} ...]
```

You may also decompile the CRUSH map, add the OSD to the device list, add the host as a bucket (if it's not already in the CRUSH map), add the device as an item in the host, assign it a weight, recompile it and set it.

13.2.1. Starting the OSD

After you add an OSD to Ceph, the OSD is in your configuration. However, it is not yet running. The OSD is **down** and **in**. You must start your new OSD before it can begin receiving data. For `sysvinit`, execute:

```
sudo /etc/init.d/ceph start osd.{osd-num}
```

Once you start your OSD, it is **up** and **in**.

13.2.2. Observe the Data Migration

Once you have added your new OSD to the CRUSH map, Ceph will begin rebalancing the server by migrating placement groups to your new OSD. You can observe this process with the `ceph` tool. :

```
ceph -w
```

You should see the placement group states change from **active+clean** to **active**, **some degraded objects**, and finally **active+clean** when migration completes. (Control-c to exit.)

CHAPTER 14. CONNECT OSD HOSTS TO CALAMARI

Once you have added the initial OSDs, you need to connect the OSD hosts to Calamari.

```
ceph-deploy calamari connect <ceph-node>[<ceph-node> ...]
```

For example, using the exemplary **node2**, **node3** and **node4** from above, you would execute:

```
ceph-deploy calamari connect node2 node3 node4
```

As you expand your cluster with additional OSD hosts, you will have to connect the hosts that contain them to Calamari, too.

CHAPTER 15. REMOVING OSDS (MANUAL)

When you want to reduce the size of a cluster or replace hardware, you may remove an OSD at runtime. With Ceph, an OSD is generally one Ceph **ceph-osd** daemon for one storage drive within a host machine. If your host has multiple storage drives, you may need to remove one **ceph-osd** daemon for each drive. Generally, it's a good idea to check the capacity of your cluster to see if you are reaching the upper end of its capacity. Ensure that when you remove an OSD that your cluster is not at its **near full** ratio.

Warning

Do not let your cluster reach its **full ratio** when removing an OSD. Removing OSDs could cause the cluster to reach or exceed its **full ratio**.

15.1. TAKE THE OSD OUT OF THE CLUSTER

Before you remove an OSD, it is usually **up** and **in**. You need to take it out of the cluster so that Ceph can begin rebalancing and copying its data to other OSDs.

```
ceph osd out {osd-num}
```

15.2. OBSERVE THE DATA MIGRATION

Once you have taken your OSD **out** of the cluster, Ceph will begin rebalancing the cluster by migrating placement groups out of the OSD you removed. You can observe this process with the **ceph** CLI tool.

```
ceph -w
```

You should see the placement group states change from **active+clean** to **active, some degraded objects**, and finally **active+clean** when migration completes. (Control-c to exit.)

15.3. STOPPING THE OSD

After you take an OSD out of the cluster, it may still be running. That is, the OSD may be **up** and **out**. You must stop your OSD before you remove it from the configuration.

```
ssh {osd-host}  
sudo /etc/init.d/ceph stop osd.{osd-num}
```

Once you stop your OSD, it is **down**.

15.4. REMOVING THE OSD

This procedure removes an OSD from a cluster map, removes its authentication key, removes the OSD from the OSD map, and removes the OSD from the **ceph.conf** file. If your host has multiple drives, you may need to remove an OSD for each drive by repeating this procedure.

1. Remove the OSD from the CRUSH map so that it no longer receives data. You may also decompile the CRUSH map, remove the OSD from the device list, remove the device as an item in the host bucket or remove the host bucket (if it's in the CRUSH map and you intend to remove the host), recompile the map and set it. See the Storage Strategies guide for details.

```
ceph osd crush remove {name}
```

2. Remove the OSD authentication key.

```
ceph auth del osd.{osd-num}
```

The value of **ceph** for **ceph-{osd-num}** in the path is the **\$cluster-\$id**. If your cluster name differs from **ceph**, use your cluster name instead.

3. Remove the OSD.

```
ceph osd rm {osd-num}
#for example
ceph osd rm 1
```

4. Navigate to the host where you keep the master copy of the cluster's **ceph.conf** file.

```
ssh {admin-host}
cd /etc/ceph
vim ceph.conf
```

5. Remove the OSD entry from your **ceph.conf** file (if it exists). :

```
[osd.1]
host = {hostname}
```

6. From the host where you keep the master copy of the cluster's **ceph.conf** file, copy the updated **ceph.conf** file to the **/etc/ceph** directory of other hosts in your cluster.

PART V. PERFORMANCE COUNTERS

CHAPTER 16. OVERVIEW

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.

CHAPTER 17. 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 collection names represent a subsystem or an instance of a subsystem. For example, the **throttle** mechanism reports statistics on how it is throttling.

Example output of the **throttle** collection:

```
throttle-filestore_bytes ..
throttle-filestore_ops ..
throttle-msgr_dispatch_throttler-client ..
throttle-msgr_dispatch_throttler-cluster ..
throttle-msgr_dispatch_throttler-hb_back_server ..
throttle-msgr_dispatch_throttler-hb_front_server ..
throttle-msgr_dispatch_throttler-hbclient ..
throttle-msgr_dispatch_throttler-ms_objecter ..
throttle-objecter_bytes ..
throttle-objecter_ops ..
throttle-osd_client_bytes ..
throttle-osd_client_messages ..
```

17.1. 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:

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

Example output:

```
{
  "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:

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

Example output:

```

...
"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 17.1. The bit field value definitions

Bit	Meaning
1	Floating point value
2	Unsigned 64-bit integer value

Bit	Meaning
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 (average) is set, there will be two values to read, a sum and a count. Please see the [Average Count and Sum](#) section below for more details. When bit 8 (counter) 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.

17.2. 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:

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

Example output:

```
{
  "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:

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

Example output:

```
...
"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
  },
  ...
}
```

17.2.1. 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.

17.2.2. Monitor Metrics Description Tables

- ✧ [Cluster Metrics Table](#)
- ✧ [Level Database Metrics Table](#)
- ✧ [General Monitor Metrics Table](#)
- ✧ [Paxos Metrics Table](#)
- ✧ [Throttle Metrics Table](#)

Table 17.2. 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

Collection Name	Metric Name	Bit Field Value	Short Description
	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 17.3. Level Database Metrics Table

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

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

Table 17.4. General Monitor Metrics Table

Collection Name	Metric Name	Bit Field Value	Short Description
mon	num_sessions	2	Current number of opened monitor sessions
	session_add	10	Number of created monitor sessions
	session_rm	10	Number of remove_session calls in monitor
	session_trim	10	Number of trimmed 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 17.5. 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
	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

Collection Name	Metric Name	Bit Field Value	Short Description
	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

Collection Name	Metric Name	Bit Field Value	Short Description
	new_pn	10	New proposal number queries
	new_pn_latency	5	New proposal number getting latency

Table 17.6. Throttle Metrics Table

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

Collection Name	Metric Name	Bit Field Value	Short Description
	<code>wait</code>	5	Waiting latency

17.2.3. OSD Metrics Description Tables

- ✎ [Write Back Throttle Table](#)
- ✎ [File Store Table](#)
- ✎ [Level Database Table](#)
- ✎ [Objecter Table](#)
- ✎ [Read and Write Operations Table](#)
- ✎ [Recovery State Table](#)
- ✎ [OSD Throttle Table](#)

Table 17.7. Write Back Throttle Table

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

Table 17.8. File Store 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
	commitcycle_latency	5	Average latency of commit

Collection Name	Metric Name	Bit Field Value	Short Description
	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 17.9. Level Database Table

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

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

Table 17.10. Objecter Table

Collection Name	Metric Name	Bit Field Value	Short Description
objecter	op_active	2	Active operations
	op_laggy	2	Laggy operations
	op_send	10	Sent operations
	op_send_bytes	10	Sent data
	op_resend	10	Resent operations
	op_ack	10	Commit callbacks
	op_commit	10	Operation commits
	op	10	Operation
	op_r	10	Read operations
	op_w	10	Write operations

Collection Name	Metric Name	Bit Field Value	Short Description
	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_clone_range	10	Clone range operations

Collection Name	Metric Name	Bit Field Value	Short Description
	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

Collection Name	Metric Name	Bit Field Value	Short Description
	linger_active	2	Active lingering operations
	linger_send	10	Sent lingering operations
	linger_resend	10	Resent lingering operations
	linger_ping	10	Sent pings to lingering operations
	poolop_active	2	Active pool operations
	poolop_send	10	Sent pool operations
	poolop_resend	10	Resent pool operations
	poolstat_active	2	Active get pool stat operations
	poolstat_send	10	Pool stat operations sent
	poolstat_resend	10	Resent pool stats
	statfs_active	2	Statfs operations
	statfs_send	10	Sent FS stats
	statfs_resend	10	Resent FS stats
	command_active	2	Active commands

Collection Name	Metric Name	Bit Field Value	Short Description
	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 17.11. Read and Write Operations Table

Collection Name	Metric Name	Bit Field Value	Short Description
osd	op_wip	2	Replication operations currently being processed (primary)
	op_in_bytes	10	Client operations total write size
	op_out_bytes	10	Client operations total read size

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

Collection Name	Metric Name	Bit Field Value	Short Description
	op_rw_out_bytes	10	Client read-modify-write operations read out
	op_rw_rlat	5	Client read-modify-write operation readable/applied latency
	op_rw_latency	5	Latency of read-modify-write operation (including queue time)
	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

Collection Name	Metric Name	Bit Field Value	Short Description
	subop_push	10	Suboperations push messages
	subop_push_in_bytes	10	Suboperations pushed size
	subop_push_latency	5	Suboperations push latency
	pull	10	Pull requests sent
	push	10	Push messages sent
	push_out_bytes	10	Pushed size
	push_in	10	Inbound push messages
	push_in_bytes	10	Inbound pushed size
	recovery_ops	10	Started recovery operations
	loadavg	2	CPU load
	buffer_bytes	2	Total allocated buffer size
	numpg	2	Placement groups
	numpg_primary	2	Placement groups for which this osd is primary
	numpg_replica	2	Placement groups for which this osd is replica

Collection Name	Metric Name	Bit Field Value	Short Description
	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 <i>copy-from</i> operations
	tier_promote	10	Tier promotions
	tier_flush	10	Tier flushes
	tier_flush_fail	10	Failed tier flushes

Collection Name	Metric Name	Bit Field Value	Short Description
	tier_try_flush	10	Tier flush attempts
	tier_try_flush_fail	10	Failed tier flush attempts
	tier_evict	10	Tier evictions
	tier_whiteout	10	Tier whiteouts
	tier_dirty	10	Dirty tier flag set
	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 17.12. Recovery State Table

Collection Name	Metric Name	Bit Field Value	Short Description
recoverystate_perf	initial_latency	5	Initial recovery state latency
	started_latency	5	Started recovery state latency
	reset_latency	5	Reset recovery state latency
	start_latency	5	Start recovery state latency
	primary_latency	5	Primary recovery state latency
	peering_latency	5	Peering recovery state latency
	backfilling_latency	5	Backfilling recovery state latency
	waitremotebackfillreserved_latency	5	Wait remote backfill reserved recovery state latency
	waitlocalbackfillreserved_latency	5	Wait local backfill reserved recovery state latency
	notbackfilling_latency	5	Notbackfilling recovery state latency

Collection Name	Metric Name	Bit Field Value	Short Description
	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

Collection Name	Metric Name	Bit Field Value	Short Description
	replicaactive_latency	5	Replicaactive recovery state latency
	stray_latency	5	Stray recovery state latency
	getinfo_latency	5	Getinfo recovery state latency
	getlog_latency	5	Getlog recovery state latency
	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 17.13. OSD Throttle 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