Top 10 Essential Ceph Commands To Manage Your Cluster

Improve your Ceph storage command-line skills with these top tips

By Danny Abukalam
Product Engineering Lead at SoftIron
Whether you’re administering your storage cluster with support, or have decided to Do It Yourself, these tips will undoubtedly come in handy at some point, no matter the size of your estate. Either way, if you follow the best practices for deployment and maintenance, Ceph becomes a much easier beast to tame and operate. Here’s a look at some of the most fundamental and useful Ceph commands we use on a day to day basis at SoftIron to manage our own internal Ceph clusters, and those of our customers.

1. status

First and foremost is ceph -s, or ceph status, which is typically the first command you’ll want to run on any Ceph cluster. The output consolidates many other command outputs into one single pane of glass that provides an instant view into cluster health, size, usage, activity, and any immediate issues that may be occurring. 

HEALTH_OK is the one to look for; it’s an immediate sign that you can sleep at night, as opposed to HEALTH_WARN or HEALTH_ERR, which could indicate drive or node failure or worse.

Other key things to look for are how many OSDs you have in vs out, how many other services you have running, such as rgw or cephfs, and how they’re doing.

```
$ ceph -s
cluster:
  id:    7c9d43ce-c945-449a-8a66-5f1407c7e47f
  health: HEALTH_OK

services:
  mon:  1 daemons, quorum danny-mon (age 2h)
  mgr:  danny-mgr(active, since 2h)
  osd:  36 osds: 36 up (since 2h), 36 in (since 2h)
  rgw:  1 daemon active (danny-mgr)

  task status:

  data:
    pools:  6 pools, 2208 pgs
    objects: 187 objects, 1.2 KiB
    usage: 2.3 TiB used, 327 TiB / 330 TiB avail
    pgs: 2208 active+clean
```
Next up is `ceph osd tree`, which provides a list of every OSD and also the class, weight, status, which node it’s in, and any reweight or priority. In the case of an OSD failure this is the first place you’ll want to look, as if you need to look at OSD logs or local node failure, this will send you in the right direction. OSDs are typically weighted against each other based on size, so a 1TB OSD will have twice the weight of a 500GB OSD, in order to ensure that the cluster is filling up the OSDs at an equal rate.

If there’s an issue with a particular OSD in your tree, or you are running a very large cluster and want to quickly check a single OSD’s details without grep-ing or scrolling through a wall of text first, you can also use `osd find`. This command will enable you to identify an OSD’s ip address, rack location and more with a single command.

```
$ ceph osd tree

ID   CLASS   WEIGHT   TYPE  NAME  STATUS  REWEIGHT PRI-AFF
-1     329.69476 root   default
-3     109.89825 host   danny-1
 0   hdd   9.15819   osd.0  up   1.00000  1.00000
 1   hdd   9.15819   osd.1  up   1.00000  1.00000
 3   hdd   9.15819   osd.2  up   1.00000  1.00000
 4   hdd   9.15819   osd.3  up   1.00000  1.00000
 5   hdd   9.15819   osd.4  up   1.00000  1.00000
 6   hdd   9.15819   osd.5  up   1.00000  1.00000
-7     109.89825 host   danny-2
12  hdd   9.15819   osd.12 up   1.00000  1.00000
13  hdd   9.15819   osd.13 up   1.00000  1.00000
15  hdd   9.15819   osd.14 up   1.00000  1.00000
16  hdd   9.15819   osd.15 up   1.00000  1.00000
17  hdd   9.15819   osd.16 up   1.00000  1.00000
-5     109.89825 host   danny-3
24  hdd   9.15819   osd.24 up   1.00000  1.00000
25  hdd   9.15819   osd.25 up   1.00000  1.00000
26  hdd   9.15819   osd.26 up   1.00000  1.00000
27  hdd   9.15819   osd.27 up   1.00000  1.00000
28  hdd   9.15819   osd.28 up   1.00000  1.00000

$ ceph osd find 37

{  
  "osd": 37,  
  "ip": "172.16.4.68:6880/636",  
  "crush_location": {  
    "datacenter": "pa2.ssdr",  
    "host": "lxc-ceph-main-front-osd-03.ssdr",  
    "physical-host": "store-front-03.ssdr",  
    "rack": "pa2-104.ssdr",  
    "root": "ssdr"  
  }  
}
3. df

Similar to the *nix df command, that tells us how much space is free on most unix and linux systems, Ceph has its own df command, `ceph df`, which provides an overview and breakdown of the amount of storage we have in our cluster, how much is used vs how much is available, and how that breaks down across our pools and storage classes.

Filling a cluster to the brim is a very bad idea with Ceph - you should add more storage well before you get to the 90% mark, and ensure that you add it in a sensible way to allow for redistribution. This is particularly important if your cluster has lots of client activity on a regular basis.

```
$ ceph df
RAW STORAGE:
CLASS  SIZE  AVAIL  USED  RAW USED  %RAW USED
hdd 330 TiB 327 TiB 2.3 TiB 2.3 TiB   0.69
TOTAL 330 TiB 327 TiB 2.3 TiB 2.3 TiB   0.69
POOLS:
POOL            ID  PGS  STORED  OBJECTS  USED  %USED  MAX
   .rgw.root       1  32  1.2 KiB  4   768 KiB  0  104 TiB
 default.rgw.control  2  32    0 B  8    0 B    0  104 TiB
 default.rgw.meta  3  32    0 B    0    0 B    0  104 TiB
 default.rgw.log  4  32    0 B  175    0 B    0  104 TiB
 default.rgw.buckets.index  5  32    0 B    0    0 B    0  104 TiB
 default.rgw.buckets.data  6 2048    0 B    0    0 B    0  104 TiB
```

4. osd pool ls detail

This is a useful one for getting a quick view of pools, but with a lot more information about their particular configuration. Ideally we need to know if a pool is erasure coded or triple-replicated, what crush rule we have in place, what the min_size is, how many placement groups are in a pool, and what application we’re using this particular pool for.

```
$ ceph osd pool ls detail
pool 1 '.rgw.root' replicated size 3 min_size 2 crush_rule 0 object_hash rjenkins pg_num 32 ppp_num 32 autoscale_mode warn last_change 64 flags hashpspool stripe_width 0 application rgw
pool 2 'default.rgw.control' replicated size 3 min_size 2 crush_rule 0 object_hash rjenkins pg_num 32 ppp_num 32 autoscale_mode warn last_change 68 flags hashpspool stripe_width 0 application rgw
pool 3 'default.rgw.meta' replicated size 3 min_size 2 crush_rule 0 object_hash rjenkins pg_num 32 ppp_num 32 autoscale_mode warn last_change 73 flags hashpspool stripe_width 0 application rgw
pool 4 'default.rgw.log' replicated size 3 min_size 2 crush_rule 0 object_hash rjenkins pg_num 32 ppp_num 32 autoscale_mode warn last_change 71 flags hashpspool stripe_width 0 application rgw
pool 5 'default.rgw.buckets.index' replicated size 3 min_size 2 crush_rule 0 object_hash rjenkins pg_num 32 ppp_num 32 autoscale_mode warn last_change 76 flags hashpspool stripe_width 0 application rgw
pool 6 'default.rgw.buckets.data' replicated size 3 min_size 2 crush_rule 0 object_hash rjenkins pg_num 2048 ppp_num 2048 autoscale_mode warn last_change 83 lfor 0/0/81 flags hashpspool stripe_width 0 application rgw
```
5. osd crush rule dump

At the heart of any Ceph cluster are the CRUSH rules. CRUSH is Ceph’s placement algorithm, and the rules help us define how we want to place data across the cluster - be it drives, nodes, racks, data centres. For example, if we require at least one copy of data at each one of our sites for our image store, we’d assign a CRUSH rule to our image store pool that mandated that behaviour, regardless of how many nodes we may have on each side.

`crush rule dump` is a good way to quickly get a list of our crush rules and how we’ve defined them in the cluster. If we want to then make changes, we have a whole host of crush commands we can use to make modifications, or we can download and decompile the crush map to manually edit it, recompile it and push it back up to our cluster.

```
$ ceph osd crush rule dump
{
  "rule_id": 0,
  "rule_name": "replicated_rule",
  "ruleset": 0,
  "type": 1,
  "min_size": 1,
  "max_size": 10,
  "steps": [
    {
      "op": "take",
      "item": -1,
      "item_name": "default"
    },
    {
      "op": "chooseleaf_firstn",
      "num": 0,
      "type": "host"
    },
    {
      "op": 'emit'
    }
  ]
}
```
6. versions

With a distributed cluster running in production, upgrading everything at once and praying for the best is clearly not the best approach. For this reason, each cluster-wide daemon in Ceph has its own version and can be upgraded independently. This means that we can upgrade daemons on a gradual basis and bring our cluster up to date with little or no disruption to service.

As long as we keep our versions somewhat close to one another, daemons with differing versions will work alongside each other perfectly happily. This does mean that we could potentially have hundreds of different daemons and respective versions to manage during an upgrade process. Enter `ceph versions` - a very easy way to get a look at how many instances of a daemon running a specific version are running.

```
$ ceph versions
{
  "mon": {
    "ceph version 14.2.15-2-g7407245e7b (7407245e7b329ac9d475f61e2cbf9f8c616505d6) nautilus (stable)": 1
  },
  "mgr": {
    "ceph version 14.2.15-2-g7407245e7b (7407245e7b329ac9d475f61e2cbf9f8c616505d6) nautilus (stable)": 1
  },
  "osd": {
    "ceph version 14.2.15-2-g7407245e7b (7407245e7b329ac9d475f61e2cbf9f8c616505d6) nautilus (stable)": 36
  },
  "mds": {},
  "rgw": {
    "ceph version 14.2.15-2-g7407245e7b (7407245e7b329ac9d475f61e2cbf9f8c616505d6) nautilus (stable)": 1
  },
  "overall": {
    "ceph version 14.2.15-2-g7407245e7b (7407245e7b329ac9d475f61e2cbf9f8c616505d6) nautilus (stable)": 39
  }
}
```

7. auth print-key

If we have lots of different clients using our cluster, we’ll need to get our keys off the cluster so they can authenticate. `ceph auth print-key` is a pretty handy way of quickly viewing any key, rather than fishing through configuration files. Another useful and related command is `ceph auth list`, which will show us a full list of all the authentication keys across the cluster for both clients and daemons, and what their respective capabilities are.

```
$ ceph auth print-key client.admin
AQDgrLhg3qY1ChAAzzZPHCw2tYz/o+2RkpaS1g==d
```
8. crash ls

Daemon crashed? There could be all sorts of reasons why this may have happened, but `ceph crash ls` is the first place we want to look. We’ll get an idea of what’s crashed and where, so we’ll be able to diagnose further. Often these will be minor warnings or easy to address errors, but crashes can also indicate more serious problems. Related useful commands are `ceph crash info <id>` which gives more info on the crash ID in question, and `ceph crash archive-all` which will archive all of our crashes if they’re warnings we’re not worried about, or issues that we’ve already dealt with.

```
$ ceph crash ls
1 daemons have recently crashed
osd.9 crashed on host danny-1 at 2021-03-06 07:28:12.665310Z
```

9. osd flags

There are a number of OSD flags that are incredibly useful. For a full list, see `OSDMAP_FLAGS`, but the most common ones are:

- `pauserd, pausewr` - Read and Write requests will no longer be answered.
- `noout` - Ceph won’t consider OSDs as out of the cluster in case the daemon fails for some reason.
- `nobackfill, norecover, norebalance` - Recovery and rebalancing is disabled

We can see how to set these flags below with the `ceph osd set` command, and also how this impacts our health messaging. Another useful and related command is the ability to take out multiple OSDs with a simple bash expansion.

```
$ ceph osd out {7..11}
marked out osd.7. marked out osd.8. marked out osd.9. marked out osd.10. marked out osd.11.
$ ceph osd set noout
noout is set
$ ceph osd set nobackfill
nobackfill is set
$ ceph osd set norecover
norecover is set
$ ceph osd set norebalance
norebalance is set
$ ceph osd set nodown
nodown is set
$ ceph osd set pause
pauserd,pausewr is set
$ ceph health detail
HEALTH_WARN pauserd,pausewr,nodown,noout,nobackfill,norecover norebalance flag(s) set
OSDMAP_FLAGS pauserd,pausewr,nodown,noout,nobackfill,norebalance,norecover flag(s) set
```
10. pg dump

All data is placed into Ceph, which provides an abstraction layer – a bit like data buckets (not S3 buckets) – for our storage, and allows the cluster to easily decide how to distribute data and best react to failures. It’s often useful to get a granular look at how our placement groups are mapped across our OSDs, or the other way around. We can do both with `pg dump`, and while many of the placement group commands can be very verbose and difficult to read, `ceph pg dump osds` does a good job of distilling this into a single pane.
With these essential commands, you’re well-equipped to handle daily Ceph cluster management. And with HyperDrive Storage Manager, you can make life even easier.

Just as kids learn how to add, subtract, divide and multiply on paper before being given the convenience of a calculator, it’s important for any Ceph administrator to understand these critical Ceph commands. But once they’re under your belt, then why not make cluster management even simpler with HyperDrive Storage Manager.

HyperDrive Storage Manager is a powerful, unified and intuitive management tool that radically simplifies management of all your Ceph software and storage hardware.

Book a demo at softiron.com/storage/storagemanager to learn more.