Ceph

Release dev

May 07, 2015

Ceph uniquely delivers **object, block, and file storage in one unified system**.

- RESTful Interface
- S3- and Swift-compliant APIs
- S3-style subdomains
- Unified S3/Swift namespace
- User management
- Usage tracking
- Striped objects
- Cloud solution integration
- Multi-site deployment
- Disaster recovery
- Thin-provisioned
- Images up to 16 exabytes
- Configurable striping
- In-memory caching
- Snapshots
- Copy-on-write cloning
- Kernel driver support
- KVM/libvirt support
- Back-end for cloud solutions
- Incremental backup
- POSIX-compliant semantics
- Separates metadata from data
- Dynamic rebalancing
- Subdirectory snapshots
- Configurable striping
- Kernel driver support
- FUSE support
- NFS/CIFS deployable
- Use with Hadoop (replace HDFS)

See Ceph Object Store for additional details.

See Ceph Block Device for additional details.

See Ceph Filesystem for additional details.

Ceph is highly reliable, easy to manage, and free. The power of Ceph can transform your company's IT infrastructure and your ability to manage vast amounts of data. To try Ceph, see our Getting Started guides. To learn more about Ceph, see our Architecture section.

CHAPTER

ONE

INTRO TO CEPH

Whether you want to provide *Ceph Object Storage* and/or *Ceph Block Device* services to *Cloud Platforms*, deploy a *Ceph Filesystem* or use Ceph for another purpose, all *Ceph Storage Cluster* deployments begin with setting up each *Ceph Node*, your network and the Ceph Storage Cluster. A Ceph Storage Cluster requires at least one Ceph Monitor and at least two Ceph OSD Daemons. The Ceph Metadata Server is essential when running Ceph Filesystem clients.

- **Ceph OSDs**: A *Ceph OSD Daemon* (Ceph OSD) stores data, handles data replication, recovery, backfilling, rebalancing, and provides some monitoring information to Ceph Monitors by checking other Ceph OSD Daemons for a heartbeat. A Ceph Storage Cluster requires at least two Ceph OSD Daemons to achieve an `active + clean` state when the cluster makes two copies of your data (Ceph makes 2 copies by default, but you can adjust it).
- **Monitors**: A *Ceph Monitor* maintains maps of the cluster state, including the monitor map, the OSD map, the Placement Group (PG) map, and the CRUSH map. Ceph maintains a history (called an "epoch") of each state change in the Ceph Monitors, Ceph OSD Daemons, and PGs.
- **MDSs**: A *Ceph Metadata Server* (MDS) stores metadata on behalf of the *Ceph Filesystem* (i.e., Ceph Block Devices and Ceph Object Storage do not use MDS). Ceph Metadata Servers make it feasible for POSIX file system users to execute basic commands like `ls`, `find`, etc. without placing an enormous burden on the Ceph Storage Cluster.

Ceph stores a client's data as objects within storage pools. Using the CRUSH algorithm, Ceph calculates which placement group should contain the object, and further calculates which Ceph OSD Daemon should store the placement group. The CRUSH algorithm enables the Ceph Storage Cluster to scale, rebalance, and recover dynamically.

To begin using Ceph in production, you should review our hardware recommendations and operating system recommendations.

1.1 Hardware Recommendations

Ceph was designed to run on commodity hardware, which makes building and maintaining petabyte-scale data clusters economically feasible. When planning out your cluster hardware, you will need to balance a number of considerations, including failure domains and potential performance issues. Hardware planning should include distributing Ceph daemons and other processes that use Ceph across many hosts. Generally, we recommend running Ceph daemons of a specific type on a host configured for that type of daemon. We recommend using other hosts for processes that utilize your data cluster (e.g., OpenStack, CloudStack, etc).

Tip: Check out the Ceph blog too. Articles like Ceph Write Throughput 1, Ceph Write Throughput 2, Argonaut v. Bobtail Performance Preview, Bobtail Performance - I/O Scheduler Comparison and others are an excellent source of information.

1.1.1 CPU

Ceph metadata servers dynamically redistribute their load, which is CPU intensive. So your metadata servers should have significant processing power (e.g., quad core or better CPUs). Ceph OSDs run the *RADOS* service, calculate data placement with *CRUSH*, replicate data, and maintain their own copy of the cluster map. Therefore, OSDs should have a reasonable amount of processing power (e.g., dual core processors). Monitors simply maintain a master copy of the cluster map, so they are not CPU intensive. You must also consider whether the host machine will run CPU-intensive processes in addition to Ceph daemons. For example, if your hosts will run computing VMs (e.g., OpenStack Nova), you will need to ensure that these other processes leave sufficient processing power for Ceph daemons. We recommend running additional CPU-intensive processes on separate hosts.

1.1.2 RAM

Metadata servers and monitors must be capable of serving their data quickly, so they should have plenty of RAM (e.g., 1GB of RAM per daemon instance). OSDs do not require as much RAM for regular operations (e.g., 500MB of RAM per daemon instance); however, during recovery they need significantly more RAM (e.g., ~1GB per 1TB of storage per daemon). Generally, more RAM is better.

1.1.3 Data Storage

Plan your data storage configuration carefully. There are significant cost and performance tradeoffs to consider when planning for data storage. Simultaneous OS operations, and simultaneous request for read and write operations from multiple daemons against a single drive can slow performance considerably. There are also file system limitations to consider: btrfs is not quite stable enough for production, but it has the ability to journal and write data simultaneously, whereas XFS and ext4 do not.

Important: Since Ceph has to write all data to the journal before it can send an ACK (for XFS and EXT4 at least), having the journal and OSD performance in balance is really important!

Hard Disk Drives

OSDs should have plenty of hard disk drive space for object data. We recommend a minimum hard disk drive size of 1 terabyte. Consider the cost-per-gigabyte advantage of larger disks. We recommend dividing the price of the hard disk drive by the number of gigabytes to arrive at a cost per gigabyte, because larger drives may have a significant impact on the cost-per-gigabyte. For example, a 1 terabyte hard disk priced at $75.00 has a cost of $0.07 per gigabyte (i.e., $75 / 1024 = 0.0732). By contrast, a 3 terabyte hard disk priced at $150.00 has a cost of $0.05 per gigabyte (i.e., $150 / 3072 = 0.0488). In the foregoing example, using the 1 terabyte disks would generally increase the cost per gigabyte by 40%–rendering your cluster substantially less cost efficient. Also, the larger the storage drive capacity, the more memory per Ceph OSD Daemon you will need, especially during rebalancing, backfilling and recovery. A general rule of thumb is ~1GB of RAM for 1TB of storage space.

Tip: Running multiple OSDs on a single disk–irrespective of partitions–is **NOT** a good idea.

Tip: Running an OSD and a monitor or a metadata server on a single disk–irrespective of partitions–is **NOT** a good idea either.

Storage drives are subject to limitations on seek time, access time, read and write times, as well as total throughput. These physical limitations affect overall system performance–especially during recovery. We recommend using a dedicated drive for the operating system and software, and one drive for each Ceph OSD Daemon you run on the host. Most "slow OSD" issues arise due to running an operating system, multiple OSDs, and/or multiple journals on the same drive. Since the cost of troubleshooting performance issues on a small cluster likely exceeds the cost of the extra disk drives, you can accelerate your cluster design planning by avoiding the temptation to overtax the OSD storage drives.

You may run multiple Ceph OSD Daemons per hard disk drive, but this will likely lead to resource contention and diminish the overall throughput. You may store a journal and object data on the same drive, but this may increase the time it takes to journal a write and ACK to the client. Ceph must write to the journal before it can ACK the write. The btrfs filesystem can write journal data and object data simultaneously, whereas XFS and ext4 cannot.

Ceph best practices dictate that you should run operating systems, OSD data and OSD journals on separate drives.

Solid State Drives

One opportunity for performance improvement is to use solid-state drives (SSDs) to reduce random access time and read latency while accelerating throughput. SSDs often cost more than 10x as much per gigabyte when compared to a hard disk drive, but SSDs often exhibit access times that are at least 100x faster than a hard disk drive.

SSDs do not have moving mechanical parts so they aren't necessarily subject to the same types of limitations as hard disk drives. SSDs do have significant limitations though. When evaluating SSDs, it is important to consider the performance of sequential reads and writes. An SSD that has 400MB/s sequential write throughput may have much better performance than an SSD with 120MB/s of sequential write throughput when storing multiple journals for multiple OSDs.

Important: We recommend exploring the use of SSDs to improve performance. However, before making a significant investment in SSDs, we **strongly recommend** both reviewing the performance metrics of an SSD and testing the SSD in a test configuration to gauge performance.

Since SSDs have no moving mechanical parts, it makes sense to use them in the areas of Ceph that do not use a lot of storage space (e.g., journals). Relatively inexpensive SSDs may appeal to your sense of economy. Use caution. Acceptable IOPS are not enough when selecting an SSD for use with Ceph. There are a few important performance considerations for journals and SSDs:

- **Write-intensive semantics:** Journaling involves write-intensive semantics, so you should ensure that the SSD you choose to deploy will perform equal to or better than a hard disk drive when writing data. Inexpensive SSDs may introduce write latency even as they accelerate access time, because sometimes high performance hard drives can write as fast or faster than some of the more economical SSDs available on the market!
- **Sequential Writes:** When you store multiple journals on an SSD you must consider the sequential write limitations of the SSD too, since they may be handling requests to write to multiple OSD journals simultaneously.
- **Partition Alignment:** A common problem with SSD performance is that people like to partition drives as a best practice, but they often overlook proper partition alignment with SSDs, which can cause SSDs to transfer data much more slowly. Ensure that SSD partitions are properly aligned.

While SSDs are cost prohibitive for object storage, OSDs may see a significant performance improvement by storing an OSD's journal on an SSD and the OSD's object data on a separate hard disk drive. The `osd journal` configuration setting defaults to `/var/lib/ceph/osd/$cluster-$id/journal`. You can mount this path to an SSD or to an SSD partition so that it is not merely a file on the same disk as the object data.

One way Ceph accelerates CephFS filesystem performance is to segregate the storage of CephFS metadata from the storage of the CephFS file contents. Ceph provides a default `metadata` pool for CephFS metadata. You will never have to create a pool for CephFS metadata, but you can create a CRUSH map hierarchy for your CephFS metadata pool that points only to a host's SSD storage media. See Mapping Pools to Different Types of OSDs for details.

Controllers

Disk controllers also have a significant impact on write throughput. Carefully, consider your selection of disk controllers to ensure that they do not create a performance bottleneck.

Tip: The Ceph blog is often an excellent source of information on Ceph performance issues. See Ceph Write Throughput 1 and Ceph Write Throughput 2 for additional details.

Additional Considerations

You may run multiple OSDs per host, but you should ensure that the sum of the total throughput of your OSD hard disks doesn't exceed the network bandwidth required to service a client's need to read or write data. You should also consider what percentage of the overall data the cluster stores on each host. If the percentage on a particular host is large and the host fails, it can lead to problems such as exceeding the `full ratio`, which causes Ceph to halt operations as a safety precaution that prevents data loss.

When you run multiple OSDs per host, you also need to ensure that the kernel is up to date. See OS Recommendations for notes on `glibc` and `syncfs(2)` to ensure that your hardware performs as expected when running multiple OSDs per host.

Hosts with high numbers of OSDs (e.g., > 20) may spawn a lot of threads, especially during recovery and rebalancing. Many Linux kernels default to a relatively small maximum number of threads (e.g., 32k). If you encounter problems starting up OSDs on hosts with a high number of OSDs, consider setting `kernel.pid_max` to a higher number of threads. The theoretical maximum is 4,194,303 threads. For example, you could add the following to the `/etc/sysctl.conf` file:

```
kernel.pid_max = 4194303
```

1.1.4 Networks

We recommend that each host have at least two 1Gbps network interface controllers (NICs). Since most commodity hard disk drives have a throughput of approximately 100MB/second, your NICs should be able to handle the traffic for the OSD disks on your host. We recommend a minimum of two NICs to account for a public (front-side) network and a cluster (back-side) network. A cluster network (preferably not connected to the internet) handles the additional load for data replication and helps stop denial of service attacks that prevent the cluster from achieving `active + clean` states for placement groups as OSDs replicate data across the cluster. Consider starting with a 10Gbps network in your racks. Replicating 1TB of data across a 1Gbps network takes 3 hours, and 3TBs (a typical drive configuration) takes 9 hours. By contrast, with a 10Gbps network, the replication times would be 20 minutes and 1 hour respectively. In a petabyte-scale cluster, failure of an OSD disk should be an expectation, not an exception. System administrators will appreciate PGs recovering from a `degraded` state to an `active + clean` state as rapidly as possible, with price / performance tradeoffs taken into consideration. Additionally, some deployment tools (e.g., Dell's Crowbar) deploy with five different networks, but employ VLANs to make hardware and network cabling more manageable. VLANs using 802.1q protocol require VLAN-capable NICs and Switches. The added hardware expense may be offset by the operational cost savings for network setup and maintenance. When using VLANs to handle VM traffic between between the cluster and compute stacks (e.g., OpenStack, CloudStack, etc.), it is also worth considering using 10G Ethernet. Top-of-rack routers for each network also need to be able to communicate with spine routers that have even faster throughput–e.g., 40Gbps to 100Gbps.

Your server hardware should have a Baseboard Management Controller (BMC). Administration and deployment tools may also use BMCs extensively, so consider the cost/benefit tradeoff of an out-of-band network for administration.

Hypervisor SSH access, VM image uploads, OS image installs, management sockets, etc. can impose significant loads on a network. Running three networks may seem like overkill, but each traffic path represents a potential capacity, throughput and/or performance bottleneck that you should carefully consider before deploying a large scale data cluster.

1.1.5 Failure Domains

A failure domain is any failure that prevents access to one or more OSDs. That could be a stopped daemon on a host; a hard disk failure, an OS crash, a malfunctioning NIC, a failed power supply, a network outage, a power outage, and so forth. When planning out your hardware needs, you must balance the temptation to reduce costs by placing too many responsibilities into too few failure domains, and the added costs of isolating every potential failure domain.

1.1.6 Minimum Hardware Recommendations

Ceph can run on inexpensive commodity hardware. Small production clusters and development clusters can run successfully with modest hardware.

Process	Criteria	Minimum Recommended
`ceph-osd`	Processor	• 1x 64-bit AMD-64 • 1x 32-bit ARM dual-core or better • 1x i386 dual-core
	RAM	~1GB for 1TB of storage per daemon
	Volume Storage	1x storage drive per daemon
	Journal	1x SSD partition per daemon (optional)
	Network	2x 1GB Ethernet NICs
`ceph-mon`	Processor	• 1x 64-bit AMD-64/i386 • 1x 32-bit ARM dual-core or better • 1x i386 dual-core
	RAM	1 GB per daemon
	Disk Space	10 GB per daemon
	Network	2x 1GB Ethernet NICs
`ceph-mds`	Processor	• 1x 64-bit AMD-64 quad-core • 1x 32-bit ARM quad-core • 1x i386 quad-core
	RAM	1 GB minimum per daemon
	Disk Space	1 MB per daemon
	Network	2x 1GB Ethernet NICs

Tip: If you are running an OSD with a single disk, create a partition for your volume storage that is separate from the partition containing the OS. Generally, we recommend separate disks for the OS and the volume storage.

1.1.7 Production Cluster Examples

Production clusters for petabyte scale data storage may also use commodity hardware, but should have considerably more memory, processing power and data storage to account for heavy traffic loads.

Dell Example

A recent (2012) Ceph cluster project is using two fairly robust hardware configurations for Ceph OSDs, and a lighter configuration for monitors.

Configuration	Criteria	Minimum Recommended
Dell PE R510	Processor	2x 64-bit quad-core Xeon CPUs
	RAM	16 GB
	Volume Storage	8x 2TB drives. 1 OS, 7 Storage
	Client Network	2x 1GB Ethernet NICs
	OSD Network	2x 1GB Ethernet NICs
	Mgmt. Network	2x 1GB Ethernet NICs
Dell PE R515	Processor	1x hex-core Opteron CPU
	RAM	16 GB
	Volume Storage	12x 3TB drives. Storage
	OS Storage	1x 500GB drive. Operating System.
	Client Network	2x 1GB Ethernet NICs
	OSD Network	2x 1GB Ethernet NICs
	Mgmt. Network	2x 1GB Ethernet NICs

1.2 OS Recommendations

1.2.1 Ceph Dependencies

As a general rule, we recommend deploying Ceph on newer releases of Linux. We also recommend deploying on releases with long-term support.

Linux Kernel

- **Ceph Kernel Client**

  We currently recommend:

  - v3.16.3 or later (rbd deadlock regression in v3.16.[0-2])
  - *NOT* v3.15.* (rbd deadlock regression)
  - v3.14.*
  - v3.10.*

  These are considered pretty old, but if you must:

  - v3.6.6 or later in the v3.6 stable series
  - v3.4.20 or later in the v3.4 stable series

  firefly (CRUSH_TUNABLES3) tunables are supported starting with v3.15. See CRUSH Tunables for more details.

- **btrfs**

  If you use the `btrfs` file system with Ceph, we recommend using a recent Linux kernel (v3.14 or later).

glibc

- **fdatasync(2)**: With Firefly v0.80 and beyond, use `fdatasync(2)` instead of `fsync(2)` to improve performance.
- **syncfs(2)**: For non-btrfs filesystems such as XFS and ext4 where more than one `ceph-osd` daemon is used on a single server, Ceph performs significantly better with the `syncfs(2)` system call (added in kernel 2.6.39 and glibc 2.14). New versions of Ceph (v0.55 and later) do not depend on glibc support.

1.2.2 Platforms

The charts below show how Ceph's requirements map onto various Linux platforms. Generally speaking, there is very little dependence on specific distributions aside from the kernel and system initialization package (i.e., sysvinit, upstart, systemd).

Firefly (0.80)

Distro	Release	Code Name	Kernel	Notes	Testing
CentOS	6	N/A	linux-2.6.32	1, 2	B, I
CentOS	7		linux-3.10.0		B
Debian	6.0	Squeeze	linux-2.6.32	1, 2, 3	B
Debian	7.0	Wheezy	linux-3.2.0	1, 2	B
Fedora	19	Schrödinger's Cat	linux-3.10.0		B
Fedora	20	Heisenbug	linux-3.14.0		B
RHEL	6		linux-2.6.32	1, 2	B, I, C
RHEL	7		linux-3.10.0		B, I, C
Ubuntu	12.04	Precise Pangolin	linux-3.2.0	1, 2	B, I, C
Ubuntu	14.04	Trusty Tahr	linux-3.13.0		B, I, C

NOTE: Ceph also supports `Quantal`, `Raring` and `Saucy`. However, we recommend using LTS releases.

Emperor (0.72)

The Ceph Emperor release, version 0.72, is no longer supported, and Emperor users should update to Firefly (version 0.80).

Dumpling (0.67)

Distro	Release	Code Name	Kernel	Notes	Testing
CentOS	6.3	N/A	linux-2.6.32	1, 2	B, I
Debian	6.0	Squeeze	linux-2.6.32	1, 2, 3	B
Debian	7.0	Wheezy	linux-3.2.0	1, 2	B
Fedora	18	Spherical Cow	linux-3.6.0		B
Fedora	19	Schrödinger’s Cat	linux-3.10.0		B
OpenSuse	12.2	N/A	linux-3.4.0	2	B
RHEL	6.3		linux-2.6.32	1, 2	B, I
Ubuntu	12.04	Precise Pangolin	linux-3.2.0	1, 2	B, I, C
Ubuntu	12.10	Quantal Quetzal	linux-3.5.4	2	B
Ubuntu	13.04	Raring Ringtail	linux-3.8.5		B

Argonaut (0.48), Bobtail (0.56), and Cuttlefish (0.61)

The Ceph Argonaut, Bobtail, and Cuttlefish releases are no longer supported, and users should update to the latest stable release (Dumpling or Firefly).

Notes

- **1**: The default kernel has an older version of `btrfs` that we do not recommend for `ceph-osd` storage nodes. Upgrade to a recommended kernel or use `XFS` or `ext4`.
- **2**: The default kernel has an old Ceph client that we do not recommend for kernel client (kernel RBD or the Ceph file system). Upgrade to a recommended kernel.
- **3**: The default kernel or installed version of `glibc` does not support the `syncfs(2)` system call. Putting multiple `ceph-osd` daemons using `XFS` or `ext4` on the same host will not perform as well as they could.

Testing

- **B**: We build release packages for this platform. For some of these platforms, we may also continuously build all ceph branches and exercise basic unit tests.
- **I**: We do basic installation and functionality tests of releases on this platform.
- **C**: We run a comprehensive functional, regression, and stress test suite on this platform on a continuous basis. This includes development branches, pre-release, and released code.

1.3 Get Involved in the Ceph Community!

These are exciting times in the Ceph community! Get involved!

Channel	Description	Contact Info
Blog	Check the Ceph Blog periodically to keep track of Ceph progress and important announcements.	http://ceph.com/community/blog/
Planet Ceph	Check the blog aggregation on Planet Ceph for interesting stories, information and experiences from the community.	http://ceph.com/community/planet-ceph/
Wiki	Check the Ceph Wiki is a source for more community and development related topics. You can find there information about blueprints, meetups, the Ceph Developer Summits and more.	https://wiki.ceph.com/
IRC	As you delve into Ceph, you may have questions or feedback for the Ceph development team. Ceph developers are often available on the `#ceph` IRC channel particularly during daytime hours in the US Pacific Standard Time zone. While `#ceph` is a good starting point for cluster operators and users, there is also `#ceph-devel` dedicated for Ceph developers.	• **Domain:** `irc.oftc.net` • **Channels:** `#ceph` and `#ceph-devel`
User List	Ask and answer user-related questions by subscribing to the email list at ceph-users@ceph.com. You can opt out of the email list at any time by unsubscribing. A simple email is all it takes! If you would like to view the archives, go to Gmane.	• User Subscribe • User Unsubscribe • Gmane for Users
Devel List	Keep in touch with developer activity by subscribing to the email list at ceph-devel@vger.kernel.org. You can opt out of the email list at any time by unsubscribing. A simple email is all it takes! If you would like to view the archives, go to Gmane.	• Devel Subscribe • Devel Unsubscribe • Gmane for Developers
Commit List	Subscribe to ceph-commit@ceph.com to get commit notifications via email. You can opt out of the email list at any time by unsubscribing. A simple email is all it takes!	• Commit Subscribe • Commit Unsubscribe • Mailing list archives
QA List	For Quality Assurance (QA) related activities subscribe to this list. You can opt out of the email list at any time by unsubscribing. A simple email is all it takes!	• QA Subscribe • QA Unsubscribe • Mailing list archives
Community List	For all discussions related to the Ceph User Committee and other community topics. You can opt out of the email list at any time by unsubscribing. A simple email is all it takes!	• Community Subscribe • Community Unsubscribe • Mailing list archives
Bug Tracker	You can help keep Ceph production worthy by filing and tracking bugs, and providing feature requests using the Bug Tracker.	http://tracker.ceph.com/projects/ceph

1.4 Documenting Ceph

The **easiest way** to help the Ceph project is to contribute to the documentation. As the Ceph user base grows and the development pace quickens, an increasing number of people are updating the documentation and adding new information. Even small contributions like fixing spelling errors or clarifying instructions will help the Ceph project immensely.

The Ceph documentation source resides in the `ceph/docs` directory of the Ceph repository, and Python Sphinx renders the source into HTML and manpages. The http://ceph.com/docs link currenly displays the `master` branch by default, but you may view documentation for older branches (e.g., `argonaut`) or future branches (e.g., `next`) as well as work-in-progress branches by substituting `master` with the branch name you prefer.

1.4.1 Making Contributions

Making a documentation contribution generally involves the same procedural sequence as making a code contribution, except that you must build documentation source instead of compiling program source. The sequence includes the following steps:

1. *Get the Source*
2. *Select a Branch*
3. *Make a Change*
4. *Build the Source*
5. *Commit the Change*
6. *Push the Change*
7. *Make a Pull Request*
8. *Notify the Relevant Person*

Get the Source

Ceph documentation lives in the Ceph repository right alongside the Ceph source code under the `ceph/doc` directory. For details on github and Ceph, see *Get Involved in the Ceph Community!*.

The most common way to make contributions is to use the Fork and Pull approach. You must:

1. Install git locally. For Debian/Ubuntu, execute:

```
sudo apt-get install git
```

For Fedora, execute:

```
sudo yum install git
```

For CentOS/RHEL, execute:

```
sudo yum install git
```

2. Ensure your `.gitconfig` file has your name and email address.

```
[user]
   email = {your-email-address}
   name = {your-name}
```

For example:

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```

3. Create a github account (if you don't have one).
4. Fork the Ceph project. See https://github.com/ceph/ceph.
5. Clone your fork of the Ceph project to your local host.

Ceph organizes documentation into an information architecture primarily by its main components.

- **Ceph Storage Cluster:** The Ceph Storage Cluster documentation resides under the `doc/rados` directory.
- **Ceph Block Device:** The Ceph Block Device documentation resides under the `doc/rbd` directory.
- **Ceph Object Storage:** The Ceph Object Storage documentation resides under the `doc/radosgw` directory.
- **Ceph Filesystem:** The Ceph Filesystem documentation resides under the `doc/cephfs` directory.
- **Installation (Quick):** Quick start documentation resides under the `doc/start` directory.
- **Installation (Manual):** Manual installation documentation resides under the `doc/install` directory.
- **Manpage:** Manpage source resides under the `doc/man` directory.
- **Developer:** Developer documentation resides under the `doc/dev` directory.
- **Images:** If you include images such as JPEG or PNG files, you should store them under the `doc/images` directory.

Select a Branch

When you make small changes to the documentation, such as fixing typographical errors or clarifying explanations, use the `master` branch (default). You should also use the `master` branch when making contributions to features that are in the current release. `master` is the most commonly used branch.

```
git checkout master
```

When you make changes to documentation that affect an upcoming release, use the `next` branch. `next` is the second most commonly used branch.

```
git checkout next
```

When you are making substantial contributions such as new features that are not yet in the current release; if your contribution is related to an issue with a tracker ID; or, if you want to see your documentation rendered on the Ceph.com website before it gets merged into the `master` branch, you should create a branch. To distinguish branches that include only documentation updates, we prepend them with `wip-doc` by convention, following the form `wip-doc-{your-branch-name}`. If the branch relates to an issue filed in http://tracker.ceph.com/issues, the branch name incorporates the issue number. For example, if a documentation branch is a fix for issue #4000, the branch name should be `wip-doc-4000` by convention and the relevant tracker URL will be http://tracker.ceph.com/issues/4000.

Note: Please do not mingle documentation contributions and source code contributions in a single pull request. Editors review the documentation and engineers review source code changes. When you keep documentation pull requests separate from source code pull requests, it simplifies the process and we won't have to ask you to resubmit the requests separately.

Before you create your branch name, ensure that it doesn't already exist in the local or remote repository.

```
git branch -a | grep wip-doc-{your-branch-name}
```

If it doesn't exist, create your branch:

```
git checkout -b wip-doc-{your-branch-name}
```

Make a Change

Modifying a document involves opening a restructuredText file, changing its contents, and saving the changes. See *Documentation Style Guide* for details on syntax requirements.

Adding a document involves creating a new restructuredText file under the `doc` directory or its subdirectories and saving the file with a `*.rst` file extension. You must also include a reference to the document: a hyperlink or a table of contents entry. The `index.rst` file of a top-level directory usually contains a TOC, where you can add the new file name. All documents must have a title. See *Headings* for details.

Your new document doesn't get tracked by `git` automatically. When you want to add the document to the repository, you must use `git add {path-to-filename}`. For example, from the top level directory of the repository, adding an `example.rst` file to the `rados` subdirectory would look like this:

```
git add doc/rados/example.rst
```

Deleting a document involves removing it from the repository with `git rm {path-to-filename}`. For example:

```
git rm doc/rados/example.rst
```

You must also remove any reference to a deleted document from other documents.

Build the Source

To build the documentation, navigate to the `ceph` repository directory:

```
cd ceph
```

To build the documentation on Debian/Ubuntu, execute:

```
admin/build-doc
```

To build the documentation on Fedora, execute:

```
admin/build-doc
```

To build the documentation on CentOS/RHEL, execute:

```
admin/build-doc
```

Executing `admin/build-doc` will create a `build-doc` directory under `ceph`. You may need to create a directory under `ceph/build-doc` for output of Javadoc files.

```
mkdir -p output/html/api/libcephfs-java/javadoc
```

The build script `build-doc` will produce an output of errors and warnings. You MUST fix errors in documents you modified before committing a change, and you SHOULD fix warnings that are related to syntax you modified.

Important: You must validate ALL HYPERLINKS. If a hyperlink is broken, it automatically breaks the build!

Once you build the documentation set, you may navigate to the source directory to view it:

```
cd build-doc/output
```

There should be an `html` directory and a `man` directory containing documentation in HTML and manpage formats respectively.

Build the Source (First Time)

Ceph uses Python Sphinx, which is generally distribution agnostic. The first time you build Ceph documentation, it will generate a doxygen XML tree, which is a bit time consuming.

Python Sphinx does have some dependencies that vary across distributions. The first time you build the documentation, the script will notify you if you do not have the dependencies installed. To run Sphinx and build documentation successfully, the following packages are required:

- gcc
- python-dev
- python-pip
- python-virtualenv
- python-sphinx
- libxml2-dev
- libxslt1-dev
- doxygen
- graphviz
- ant
- ditaa
- gcc
- python-devel
- python-pip
- python-virtualenv
- python-docutils
- python-jinja2
- python-pygments
- python-sphinx
- libxml2-devel
- libxslt1-devel
- doxygen
- graphviz
- ant
- ditaa
- gcc
- python-devel

- python-pip
- python-virtualenv
- python-docutils
- python-jinja2
- python-pygments
- python-sphinx
- libxml2-dev
- libxslt1-dev
- doxygen
- graphviz
- ant

Install each dependency that isn't installed on your host. For Debian/Ubuntu distributions, execute the following:

```
sudo apt-get install gcc python-dev python-pip python-virtualenv libxml2-dev libxslt-dev doxygen grap
sudo apt-get install python-sphinx
```

For Fedora distributions, execute the following:

```
sudo yum install gcc python-devel python-pip python-virtualenv libxml2-devel libxslt-devel doxygen gr
sudo pip install html2text
sudo yum install python-jinja2 python-pygments python-docutils python-sphinx
sudo yum install jericho-html ditaa
```

For CentOS/RHEL distributions, it is recommended to have `epel` (Extra Packages for Enterprise Linux) repository as it provides some extra packages which are not available in the default repository. To install `epel`, execute the following:

```
wget http://ftp.riken.jp/Linux/fedora/epel/7/x86_64/e/epel-release-7-2.noarch.rpm
sudo yum install epel-release-7-2.noarch.rpm
```

For CentOS/RHEL distributions, execute the following:

```
sudo yum install gcc python-devel python-pip python-virtualenv libxml2-devel libxslt-devel doxygen gr
sudo pip install html2text
```

For CentOS/RHEL distributions, the remaining python packages are not available in the default and `epel` repositories. So, use http://rpmfind.net/ to find the packages. Then, download them from a mirror and install them. For example:

```
wget ftp://rpmfind.net/linux/centos/7.0.1406/os/x86_64/Packages/python-jinja2-2.7.2-2.el7.noarch.rpm
sudo yum install python-jinja2-2.7.2-2.el7.noarch.rpm
wget ftp://rpmfind.net/linux/centos/7.0.1406/os/x86_64/Packages/python-pygments-1.4-9.el7.noarch.rpm
sudo yum install python-pygments-1.4-9.el7.noarch.rpm
wget ftp://rpmfind.net/linux/centos/7.0.1406/os/x86_64/Packages/python-docutils-0.11-0.2.20130715svn7
sudo yum install python-docutils-0.11-0.2.20130715svn7687.el7.noarch.rpm
wget ftp://rpmfind.net/linux/centos/7.0.1406/os/x86_64/Packages/python-sphinx-1.1.3-8.el7.noarch.rpm
sudo yum install python-sphinx-1.1.3-8.el7.noarch.rpm
```

Ceph documentation makes extensive use of ditaa, which isn't presently built for CentOS/RHEL7. You must install `ditaa` if you are making changes to `ditaa` diagrams so that you can verify that they render properly before you commit new or modified `ditaa` diagrams. You may retrieve compatible required packages for CentOS/RHEL distributions and install them manually. To run `ditaa` on CentOS/RHEL7, following dependencies are required:

- jericho-html

- jai-imageio-core
- batik

Use http://rpmfind.net/ to find compatible `ditaa` and the dependencies. Then, download them from a mirror and install them. For example:

```
wget ftp://rpmfind.net/linux/fedora/linux/releases/20/Everything/x86_64/os/Packages/j/jericho-html-3
sudo yum install jericho-html-3.2-6.fc20.noarch.rpm
wget ftp://rpmfind.net/linux/centos/7.0.1406/os/x86_64/Packages/jai-imageio-core-1.2-0.14.20100217cvs
sudo yum install jai-imageio-core-1.2-0.14.20100217cvs.el7.noarch.rpm
wget ftp://rpmfind.net/linux/centos/7.0.1406/os/x86_64/Packages/batik-1.8-0.12.svn1230816.el7.noarch
sudo yum install batik-1.8-0.12.svn1230816.el7.noarch.rpm
wget ftp://rpmfind.net/linux/fedora/linux/releases/20/Everything/x86_64/os/Packages/d/ditaa-0.9-10.r7
sudo yum install ditaa-0.9-10.r74.fc20.noarch.rpm
```

Important: Do not install the `fc21` rpm for `ditaa` as it uses a `JRE` newer than the default installed in CentOS/RHEL7 which causes a conflict, throws an `Exception` and doesn't allow the application to run.

Once you have installed all these packages, build the documentation by following the steps given in `Build the Source`.

Commit the Change

Ceph documentation commits are simple, but follow a strict convention:

- A commit SHOULD have 1 file per commit (it simplifies rollback). You MAY commit multiple files with related changes. Unrelated changes SHOULD NOT be put into the same commit.
- A commit MUST have a comment.
- A commit comment MUST be prepended with `doc:`. (strict)
- The comment summary MUST be one line only. (strict)
- Additional comments MAY follow a blank line after the summary, but should be terse.
- A commit MAY include `Fixes: #{bug number}`.
- Commits MUST include `Signed-off-by: Firstname Lastname <email>`. (strict)

Tip: Follow the foregoing convention particularly where it says `(strict)` or you will be asked to modify your commit to comply with this convention.

The following is a common commit comment (preferred):

```
doc: Fixes a spelling error and a broken hyperlink.

Signed-off-by: John Doe <john.doe@gmail.com>
```

The following comment includes a reference to a bug.

```
doc: Fixes a spelling error and a broken hyperlink.

Fixes: #1234

Signed-off-by: John Doe <john.doe@gmail.com>
```

The following comment includes a terse sentence following the comment summary. There is a carriage return between the summary line and the description:

```
doc: Added mon setting to monitor config reference

Describes 'mon setting', which is a new setting added
to config_opts.h.

Signed-off-by: John Doe <john.doe@gmail.com>
```

To commit changes, execute the following:

```
git commit -a
```

An easy way to manage your documentation commits is to use visual tools for `git`. For example, `gitk` provides a graphical interface for viewing the repository history, and `git-gui` provides a graphical interface for viewing your uncommitted changes, staging them for commit, committing the changes and pushing them to your forked Ceph repository.

For Debian/Ubuntu, execute:

```
sudo apt-get install gitk git-gui
```

For Fedora, execute:

```
sudo yum install gitk git-gui
```

In CentOS/RHEL7, `gitk` and `git-gui` are not available in default or `epel` repository. So, use http://rpmfind.net/ to find them. Then, download them from a mirror and install them. For example:

```
wget ftp://rpmfind.net/linux/centos/7.0.1406/os/x86_64/Packages/gitk-1.8.3.1-4.el7.noarch.rpm
sudo yum install gitk-1.8.3.1-4.el7.noarch.rpm
wget ftp://rpmfind.net/linux/centos/7.0.1406/os/x86_64/Packages/git-gui-1.8.3.1-4.el7.noarch.rpm
sudo yum install git-gui-1.8.3.1-4.el7.noarch.rpm
```

Then, execute:

```
cd {git-ceph-repo-path}
gitk
```

Finally, select **File->Start git gui** to activate the graphical user interface.

Push the Change

Once you have one or more commits, you must push them from the local copy of the repository to `github`. A graphical tool like `git-gui` provides a user interface for pushing to the repository. If you created a branch previously:

```
git push origin wip-doc-{your-branch-name}
```

Otherwise:

```
git push
```

Make a Pull Request

As noted earlier, you can make documentation contributions using the Fork and Pull approach.

Notify the Relevant Person

After you make a pull request, notify the relevant person. For general documentation pull requests, notify John Wilkins.

1.4.2 Documentation Style Guide

One objective of the Ceph documentation project is to ensure the readability of the documentation in both native reStructuredText format and its rendered formats such as HTML. Navigate to your Ceph repository and view a document in its native format. You may notice that it is generally as legible in a terminal as it is in its rendered HTML format. Additionally, you may also notice that diagrams in `ditaa` format also render reasonably well in text mode.

```
cat doc/architecture.rst | less
```

Review the following style guides to maintain this consistency.

Headings

1. **Document Titles:** Document titles use the = character overline and underline with a leading and trailing space on the title text line. See Document Title for details.
2. **Section Titles:** Section tiles use the = character underline with no leading or trailing spaces for text. Two carriage returns should precede a section title (unless an inline reference precedes it). See Sections for details.
3. **Subsection Titles:** Subsection titles use the _ character underline with no leading or trailing spaces for text. Two carriage returns should precede a subsection title (unless an inline reference precedes it).

Text Body

As a general rule, we prefer text to wrap at column 80 so that it is legible in a command line interface without leading or trailing white space. Where possible, we prefer to maintain this convention with text, lists, literal text (exceptions allowed), tables, and `ditaa` graphics.

1. **Paragraphs**: Paragraphs have a leading and a trailing carriage return, and should be 80 characters wide or less so that the documentation can be read in native format in a command line terminal.
2. **Literal Text:** To create an example of literal text (e.g., command line usage), terminate the preceding paragraph with `::` or enter a carriage return to create an empty line after the preceding paragraph; then, enter `::` on a separate line followed by another empty line. Then, begin the literal text with tab indentation (preferred) or space indentation of 3 characters.
3. **Indented Text:** Indented text such as bullet points (e.g., `- some text`) may span multiple lines. The text of subsequent lines should begin at the same character position as the text of the indented text (less numbers, bullets, etc.).

   Indented text may include literal text examples. Whereas, text indentation should be done with spaces, literal text examples should be indented with tabs. This convention enables you to add an additional indented paragraph following a literal example by leaving a blank line and beginning the subsequent paragraph with space indentation.
4. **Numbered Lists:** Numbered lists should use autonumbering by starting a numbered indent with `#.` instead of the actual number so that numbered paragraphs can be repositioned without requiring manual renumbering.
5. **Code Examples:** Ceph supports the use of the `.. code-block::<language>` role, so that you can add highlighting to source examples. This is preferred for source code. However, use of this tag will cause

autonumbering to restart at 1 if it is used as an example within a numbered list. See Showing code examples for details.

Paragraph Level Markup

The Ceph project uses paragraph level markup to highlight points.

1. **Tip:** Use the `.. tip::` directive to provide additional information that assists the reader or steers the reader away from trouble.
2. **Note**: Use the `.. note::` directive to highlight an important point.
3. **Important:** Use the `.. important::` directive to highlight important requirements or caveats (e.g., anything that could lead to data loss). Use this directive sparingly, because it renders in red.
4. **Version Added:** Use the `.. versionadded::` directive for new features or configuration settings so that users know the minimum release for using a feature.
5. **Version Changed:** Use the `.. versionchanged::` directive for changes in usage or configuration settings.
6. **Deprecated:** Use the `.. deprecated::` directive when CLI usage, a feature or a configuration setting is no longer preferred or will be discontinued.
7. **Topic:** Use the `.. topic::` directive to encapsulate text that is outside the main flow of the document. See the topic directive for additional details.

TOC and Hyperlinks

All documents must be linked from another document or a table of contents, otherwise you will receive a warning when building the documentation.

The Ceph project uses the `.. toctree::` directive. See The TOC tree for details. When rendering a TOC, consider specifying the `:maxdepth:` parameter so the rendered TOC is reasonably terse.

Document authors should prefer to use the `:ref:` syntax where a link target contains a specific unique identifier (e.g., `.. _unique-target-id:`), and a reference to the target specifically references the target (e.g., `:ref:‘unique-target-id‘`) so that if source files are moved or the information architecture changes, the links will still work. See Cross referencing arbitrary locations for details.

Ceph documentation also uses the backtick (accent grave) character followed by the link text, another backtick and an underscore. Sphinx allows you to incorporate the link destination inline; however, we prefer to use the use the `.. _Link Text: ../path` convention at the bottom of the document, because it improves the readability of the document in a command line interface.

CHAPTER

TWO

INSTALLATION (QUICK)

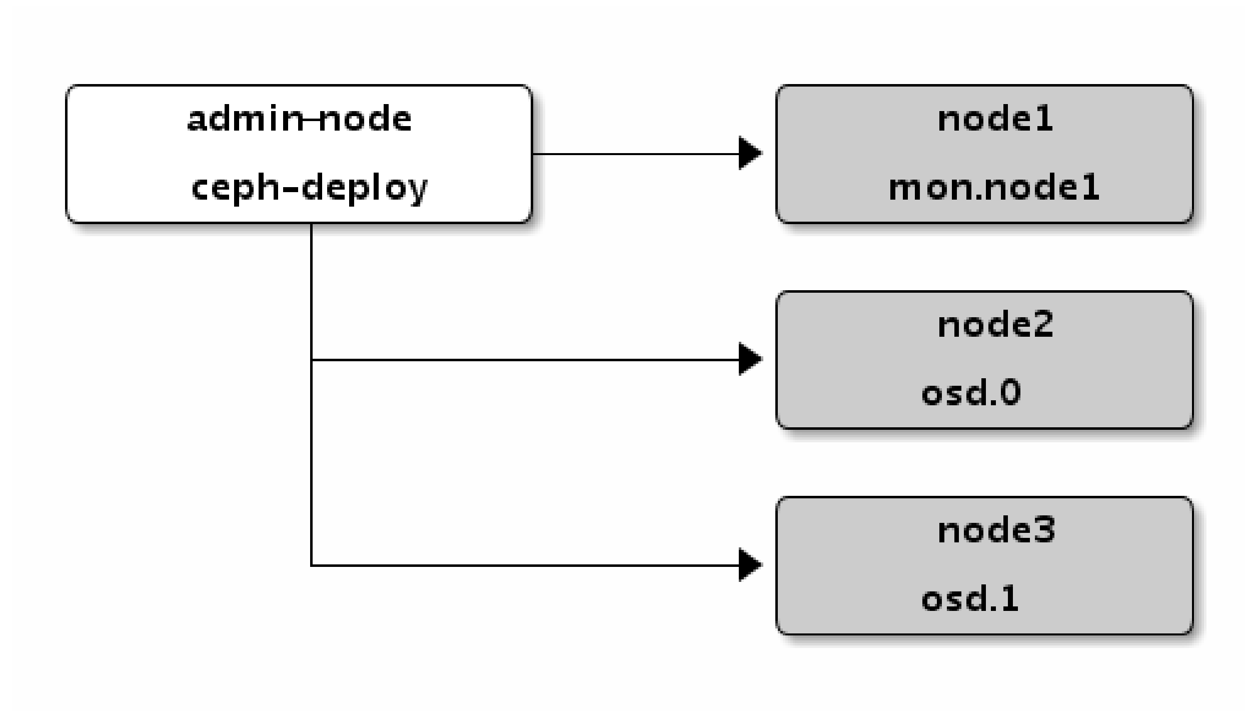
A *Ceph Client* and a *Ceph Node* may require some basic configuration work prior to deploying a Ceph Storage Cluster. You can also avail yourself of help by getting involved in the Ceph community.

2.1 Preflight Checklist

New in version 0.60.

Thank you for trying Ceph! We recommend setting up a `ceph-deploy` admin *node* and a 3-node *Ceph Storage Cluster* to explore the basics of Ceph. This **Preflight Checklist** will help you prepare a `ceph-deploy` admin node and three Ceph Nodes (or virtual machines) that will host your Ceph Storage Cluster. Before proceeding any further, see OS Recommendations to verify that you have a supported distribution and version of Linux. When you use a single Linux distribution and version across the cluster, it will make it easier for you to troubleshoot issues that arise in production.

In the descriptions below, *Node* refers to a single machine.



2.1.1 Ceph Deploy Setup

Add Ceph repositories to the `ceph-deploy` admin node. Then, install `ceph-deploy`.

Advanced Package Tool (APT)

For Debian and Ubuntu distributions, perform the following steps:

1. Add the release key:

```
wget -q -O- 'https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc' | sudo apt-key ad
```

2. Add the Ceph packages to your repository. Replace `{ceph-stable-release}` with a stable Ceph release (e.g., `cuttlefish`, `dumpling`, `emperor`, `firefly`, etc.). For example:

```
echo deb http://ceph.com/debian-{ceph-stable-release}/ $(lsb_release -sc) main | sudo tee /etc/a
```

3. Update your repository and install `ceph-deploy`:

```
sudo apt-get update && sudo apt-get install ceph-deploy
```

Note: You can also use the EU mirror eu.ceph.com for downloading your packages. Simply replace `http://ceph.com/` by `http://eu.ceph.com/`

Red Hat Package Manager (RPM)

For Red Hat(rhel6, rhel7), CentOS (el6, el7), and Fedora 19-20 (f19-f20) perform the following steps:

1. Add the package to your repository. Open a text editor and create a Yellowdog Updater, Modified (YUM) entry. Use the file path `/etc/yum.repos.d/ceph.repo`. For example:

```
sudo vim /etc/yum.repos.d/ceph.repo
```

Paste the following example code. Replace `{ceph-release}` with the recent major release of Ceph (e.g., `firefly`). Replace `{distro}` with your Linux distribution (e.g., `el6` for CentOS 6, `el7` for CentOS 7, `rhel6` for Red Hat 6.5, `rhel7` for Red Hat 7, and `fc19` or `fc20` for Fedora 19 or Fedora 20. Finally, save the contents to the `/etc/yum.repos.d/ceph.repo` file.

```
[ceph-noarch]
name=Ceph noarch packages
baseurl=http://ceph.com/rpm-{ceph-release}/{distro}/noarch
enabled=1
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc
```

2. Update your repository and install `ceph-deploy`:

```
sudo yum update && sudo yum install ceph-deploy
```

Note: You can also use the EU mirror eu.ceph.com for downloading your packages. Simply replace `http://ceph.com/` by `http://eu.ceph.com/`

2.1.2 Ceph Node Setup

The admin node must be have password-less SSH access to Ceph nodes. When ceph-deploy logs in to a Ceph node as a user, that particular user must have passwordless `sudo` privileges.

Install NTP

We recommend installing NTP on Ceph nodes (especially on Ceph Monitor nodes) to prevent issues arising from clock drift. See Clock for details.

On CentOS / RHEL, execute:

```
sudo yum install ntp ntpdate ntp-doc
```

On Debian / Ubuntu, execute:

```
sudo apt-get install ntp
```

Ensure that you enable the NTP service. Ensure that each Ceph Node uses the same NTP time server. See NTP for details.

Install SSH Server

For **ALL** Ceph Nodes perform the following steps:

1. Install an SSH server (if necessary) on each Ceph Node:

```
    sudo apt-get install openssh-server
```

   or:

```
    sudo yum install openssh-server
```

2. Ensure the SSH server is running on **ALL** Ceph Nodes.

Create a Ceph User

The `ceph-deploy` utility must login to a Ceph node as a user that has passwordless `sudo` privileges, because it needs to install software and configuration files without prompting for passwords.

Recent versions of `ceph-deploy` support a `--username` option so you can specify any user that has password-less `sudo` (including `root`, although this is **NOT** recommended). To use `ceph-deploy --username {username}`, the user you specify must have password-less SSH access to the Ceph node, as `ceph-deploy` will not prompt you for a password.

We recommend creating a Ceph user on **ALL** Ceph nodes in the cluster. A uniform user name across the cluster may improve ease of use (not required), but you should avoid obvious user names, because hackers typically use them with brute force hacks (e.g., `root`, `admin`, `{productname}`). The following procedure, substituting `{username}` for the user name you define, describes how to create a user with passwordless `sudo`.

1. Create a user on each Ceph Node.

```
    ssh user@ceph-server
    sudo useradd -d /home/{username} -m {username}
    sudo passwd {username}
```

2. For the user you added to each Ceph node, ensure that the user has `sudo` privileges.

```
echo "{username} ALL = (root) NOPASSWD:ALL" | sudo tee /etc/sudoers.d/{username}
sudo chmod 0440 /etc/sudoers.d/{username}
```

Enable Password-less SSH

Since `ceph-deploy` will not prompt for a password, you must generate SSH keys on the admin node and distribute the public key to each Ceph node. `ceph-deploy` will attempt to generate the SSH keys for initial monitors.

1. Generate the SSH keys, but do not use `sudo` or the `root` user. Leave the passphrase empty:

```
ssh-keygen

Generating public/private key pair.
Enter file in which to save the key (/ceph-admin/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /ceph-admin/.ssh/id_rsa.
Your public key has been saved in /ceph-admin/.ssh/id_rsa.pub.
```

2. Copy the key to each Ceph Node, replacing `{username}` with the user name you created with *Create a Ceph User*.

```
ssh-copy-id {username}@node1
ssh-copy-id {username}@node2
ssh-copy-id {username}@node3
```

3. (Recommended) Modify the `~/.ssh/config` file of your `ceph-deploy` admin node so that `ceph-deploy` can log in to Ceph nodes as the user you created without requiring you to specify `--username {username}` each time you execute `ceph-deploy`. This has the added benefit of streamlining `ssh` and `scp` usage. Replace `{username}` with the user name you created:

```
Host node1
   Hostname node1
   User {username}
Host node2
   Hostname node2
   User {username}
Host node3
   Hostname node3
   User {username}
```

Enable Networking On Bootup

Ceph OSDs peer with each other and report to Ceph Monitors over the network. If networking is `off` by default, the Ceph cluster cannot come online during bootup until you enable networking.

The default configuration on some distributions (e.g., CentOS) has the networking interface(s) off by default. Ensure that, during boot up, your network interface(s) turn(s) on so that your Ceph daemons can communicate over the network. For example, on Red Hat and CentOS, navigate to `/etc/sysconfig/network-scripts` and ensure that the `ifcfg-{iface}` file has `ONBOOT` set to `yes`.

Ensure Connectivity

Ensure connectivity using `ping` with short hostnames (`hostname -s`). Address hostname resolution issues as necessary.

Note: Hostnames should resolve to a network IP address, not to the loopback IP address (e.g., hostnames should resolve to an IP address other than `127.0.0.1`). If you use your admin node as a Ceph node, you should also ensure that it resolves to its hostname and IP address (i.e., not its loopback IP address).

Open Required Ports

Ceph Monitors communicate using port `6789` by default. Ceph OSDs communicate in a port range of `6800:7300` by default. See the Network Configuration Reference for details. Ceph OSDs can use multiple network connections to communicate with clients, monitors, other OSDs for replication, and other OSDs for heartbeats.

On some distributions (e.g., RHEL), the default firewall configuration is fairly strict. You may need to adjust your firewall settings allow inbound requests so that clients in your network can communicate with daemons on your Ceph nodes.

For `firewalld` on RHEL 7, add port `6789` for Ceph Monitor nodes and ports `6800:7100` for Ceph OSDs to the public zone and ensure that you make the setting permanent so that it is enabled on reboot. For example:

```
sudo firewall-cmd --zone=public --add-port=6789/tcp --permanent
```

For `iptables`, add port `6789` for Ceph Monitors and ports `6800:7100` for Ceph OSDs. For example:

```
sudo iptables -A INPUT -i {iface} -p tcp -s {ip-address}/{netmask} --dport 6789 -j ACCEPT
```

Once you have finished configuring `iptables`, ensure that you make the changes persistent on each node so that they will be in effect when your nodes reboot. For example:

```
/sbin/service iptables save
```

TTY

On CentOS and RHEL, you may receive an error while trying to execute `ceph-deploy` commands. If `requiretty` is set by default on your Ceph nodes, disable it by executing `sudo visudo` and locate the `Defaults requiretty` setting. Change it to `Defaults:ceph !requiretty` or comment it out to ensure that `ceph-deploy` can connect using the user you created with *Create a Ceph User*.

Note: If editing, `/etc/sudoers`, ensure that you use `sudo visudo` rather than a text editor.

SELinux

On CentOS and RHEL, SELinux is set to `Enforcing` by default. To streamline your installation, we recommend setting SELinux to `Permissive` or disabling it entirely and ensuring that your installation and cluster are working properly before hardening your configuration. To set SELinux to `Permissive`, execute the following:

```
sudo setenforce 0
```

To configure SELinux persistently (recommended if SELinux is an issue), modify the configuration file at `/etc/selinux/config`.

Priorities/Preferences

Ensure that your package manager has priority/preferences packages installed and enabled. On CentOS, you may need to install EPEL. On RHEL, you may need to enable optional repositories.

```
sudo yum install yum-plugin-priorities
```

For example, on RHEL 7 server, execute the following to install `yum-plugin-priorities` and enable the `rhel-7-server-optional-rpms` repository:

```
sudo yum install yum-plugin-priorities --enablerepo=rhel-7-server-optional-rpms
```

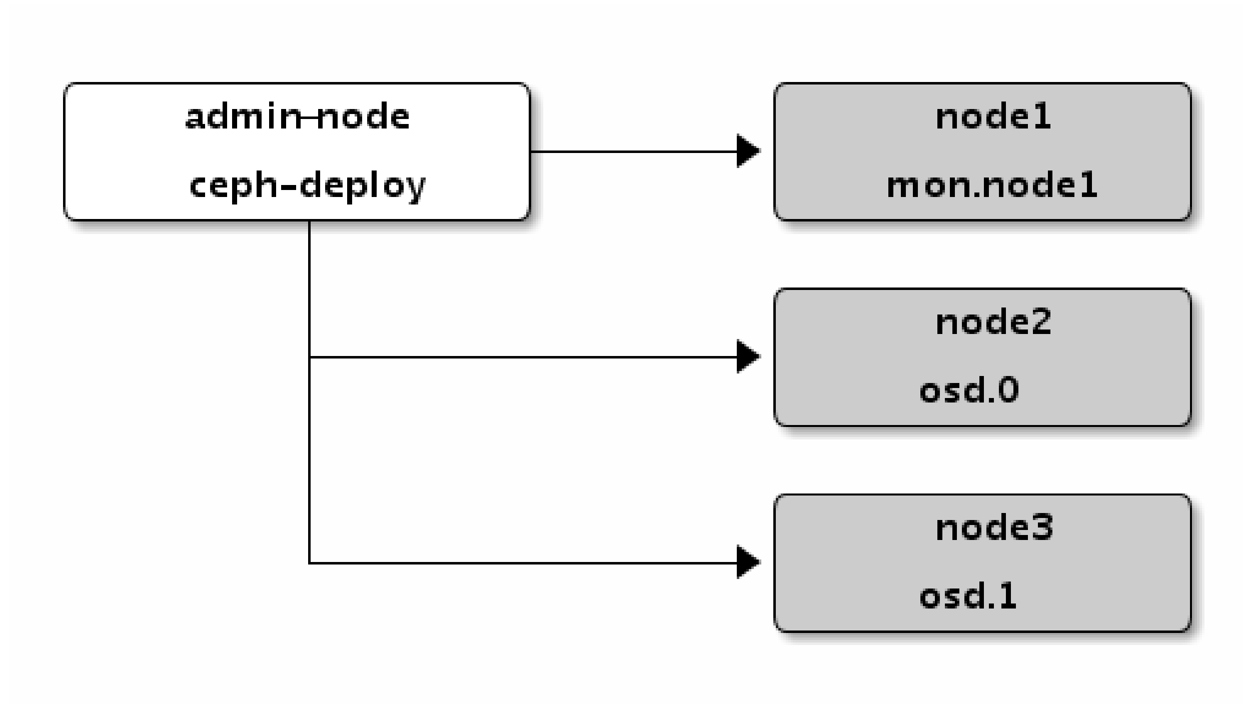
2.1.3 Summary

This completes the Quick Start Preflight. Proceed to the Storage Cluster Quick Start.

Once you've completed your preflight checklist, you should be able to begin deploying a Ceph Storage Cluster.

2.2 Storage Cluster Quick Start

If you haven't completed your Preflight Checklist, do that first. This **Quick Start** sets up a *Ceph Storage Cluster* using `ceph-deploy` on your admin node. Create a three Ceph Node cluster so you can explore Ceph functionality.



As a first exercise, create a Ceph Storage Cluster with one Ceph Monitor and two Ceph OSD Daemons. Once the cluster reaches a `active + clean` state, expand it by adding a third Ceph OSD Daemon, a Metadata Server and two more Ceph Monitors. For best results, create a directory on your admin node node for maintaining the configuration files and keys that `ceph-deploy` generates for your cluster.

```
mkdir my-cluster
cd my-cluster
```

The `ceph-deploy` utility will output files to the current directory. Ensure you are in this directory when executing `ceph-deploy`.

Important: Do not call `ceph-deploy` with `sudo` or run it as `root` if you are logged in as a different user, because it will not issue `sudo` commands needed on the remote host.

Disable `requiretty`

On some distributions (e.g., CentOS), you may receive an error while trying to execute `ceph-deploy` commands. If `requiretty` is set by default, disable it by executing `sudo visudo` and locate the `Defaults requiretty` setting. Change it to `Defaults:ceph !requiretty` to ensure that `ceph-deploy` can connect using the `ceph` user and execute commands with `sudo`.

2.2.1 Create a Cluster

If at any point you run into trouble and you want to start over, execute the following to purge the configuration:

```
ceph-deploy purgedata {ceph-node} [{ceph-node}]
ceph-deploy forgetkeys
```

To purge the Ceph packages too, you may also execute:

```
ceph-deploy purge {ceph-node} [{ceph-node}]
```

If you execute `purge`, you must re-install Ceph.

On your admin node from the directory you created for holding your configuration details, perform the following steps using `ceph-deploy`.

1. Create the cluster.

   ```
   ceph-deploy new {initial-monitor-node(s)}
   ```

   For example:

   ```
   ceph-deploy new node1
   ```

   Check the output of `ceph-deploy` with `ls` and `cat` in the current directory. You should see a Ceph configuration file, a monitor secret keyring, and a log file for the new cluster. See ceph-deploy new -h for additional details.

2. Change the default number of replicas in the Ceph configuration file from `3` to `2` so that Ceph can achieve an `active + clean` state with just two Ceph OSDs. Add the following line under the `[global]` section:

   ```
   osd pool default size = 2
   ```

3. If you have more than one network interface, add the `public network` setting under the `[global]` section of your Ceph configuration file. See the Network Configuration Reference for details.

   ```
   public network = {ip-address}/{netmask}
   ```

4. Install Ceph.

   ```
   ceph-deploy install {ceph-node}[{ceph-node} ...]
   ```

   For example:

   ```
   ceph-deploy install admin-node node1 node2 node3
   ```

   The `ceph-deploy` utility will install Ceph on each node. **NOTE**: If you use `ceph-deploy purge`, you must re-execute this step to re-install Ceph.

5. Add the initial monitor(s) and gather the keys:

```
ceph-deploy mon create-initial
```

Once you complete the process, your local directory should have the following keyrings:

- `{cluster-name}.client.admin.keyring`
- `{cluster-name}.bootstrap-osd.keyring`
- `{cluster-name}.bootstrap-mds.keyring`
- `{cluster-name}.bootstrap-rgw.keyring`

Note: The bootstrap-rgw keyring is only created during installation of clusters running Hammer or newer

1. Add two OSDs. For fast setup, this quick start uses a directory rather than an entire disk per Ceph OSD Daemon. See ceph-deploy osd for details on using separate disks/partitions for OSDs and journals. Login to the Ceph Nodes and create a directory for the Ceph OSD Daemon.

```
ssh node2
sudo mkdir /var/local/osd0
exit

ssh node3
sudo mkdir /var/local/osd1
exit
```

Then, from your admin node, use `ceph-deploy` to prepare the OSDs.

```
ceph-deploy osd prepare {ceph-node}:/path/to/directory
```

For example:

```
ceph-deploy osd prepare node2:/var/local/osd0 node3:/var/local/osd1
```

Finally, activate the OSDs.

```
ceph-deploy osd activate {ceph-node}:/path/to/directory
```

For example:

```
ceph-deploy osd activate node2:/var/local/osd0 node3:/var/local/osd1
```

2. Use `ceph-deploy` to copy the configuration file and admin key to your admin node and your Ceph Nodes so that you can use the `ceph` CLI without having to specify the monitor address and `ceph.client.admin.keyring` each time you execute a command.

```
ceph-deploy admin {admin-node} {ceph-node}
```

For example:

```
ceph-deploy admin admin-node node1 node2 node3
```

When `ceph-deploy` is talking to the local admin host (`admin-node`), it must be reachable by its hostname. If necessary, modify `/etc/hosts` to add the name of the admin host.

3. Ensure that you have the correct permissions for the `ceph.client.admin.keyring`.

```
sudo chmod +r /etc/ceph/ceph.client.admin.keyring
```

4. Check your cluster's health.

```
    ceph health
```

Your cluster should return an `active + clean` state when it has finished peering.

2.2.2 Operating Your Cluster

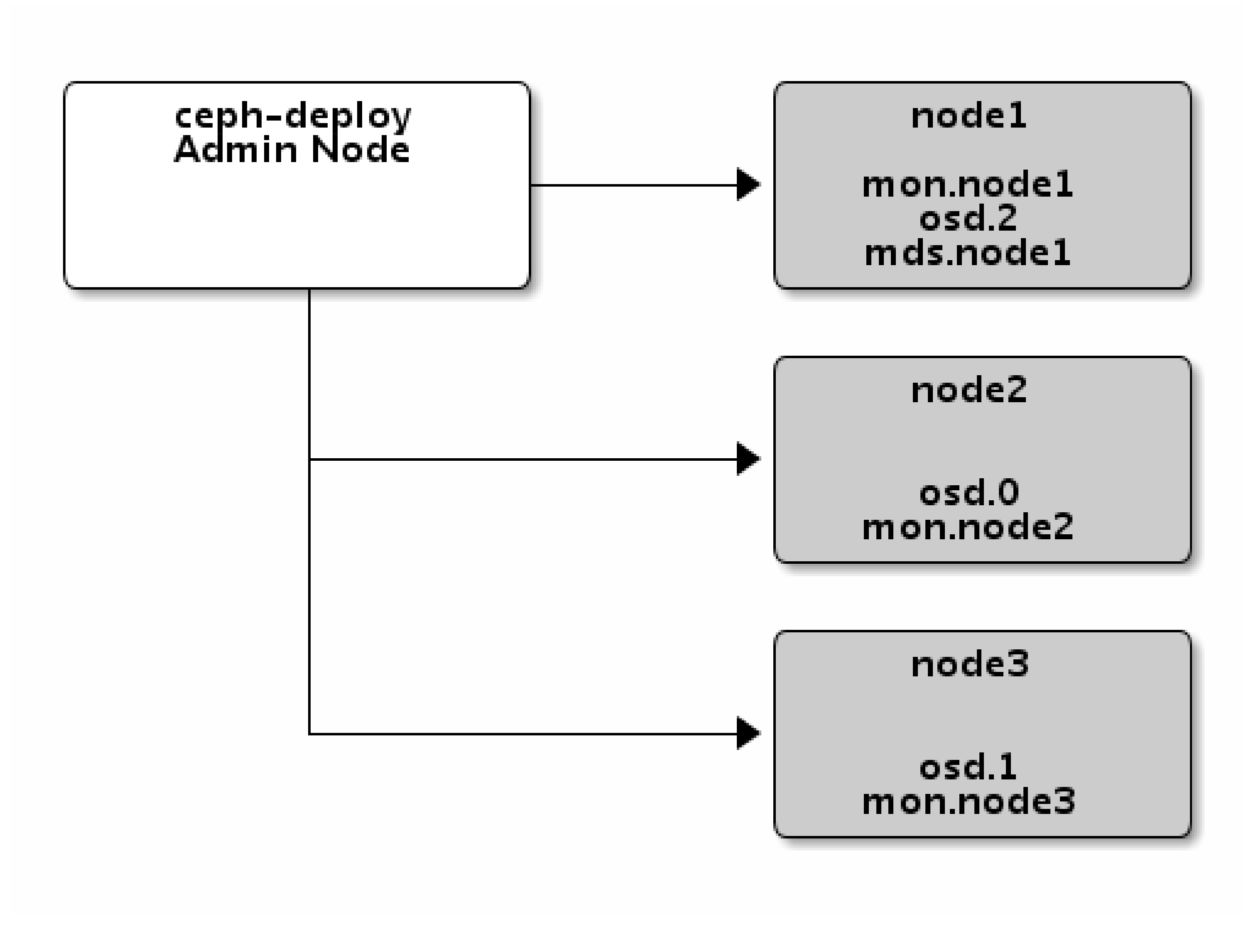
Deploying a Ceph cluster with `ceph-deploy` automatically starts the cluster. To operate the cluster daemons with Debian/Ubuntu distributions, see Running Ceph with Upstart. To operate the cluster daemons with CentOS, Red Hat, Fedora, and SLES distributions, see Running Ceph with sysvinit.

To learn more about peering and cluster health, see Monitoring a Cluster. To learn more about Ceph OSD Daemon and placement group health, see Monitoring OSDs and PGs. To learn more about managing users, see User Management.

Once you deploy a Ceph cluster, you can try out some of the administration functionality, the `rados` object store command line, and then proceed to Quick Start guides for Ceph Block Device, Ceph Filesystem, and the Ceph Object Gateway.

2.2.3 Expanding Your Cluster

Once you have a basic cluster up and running, the next step is to expand cluster. Add a Ceph OSD Daemon and a Ceph Metadata Server to `node1`. Then add a Ceph Monitor to `node2` and `node3` to establish a quorum of Ceph Monitors.



Adding an OSD

Since you are running a 3-node cluster for demonstration purposes, add the OSD to the monitor node.

```
ssh node1
sudo mkdir /var/local/osd2
exit
```

Then, from your `ceph-deploy` node, prepare the OSD.

```
ceph-deploy osd prepare {ceph-node}:/path/to/directory
```

For example:

```
ceph-deploy osd prepare node1:/var/local/osd2
```

Finally, activate the OSDs.

```
ceph-deploy osd activate {ceph-node}:/path/to/directory
```

For example:

```
ceph-deploy osd activate node1:/var/local/osd2
```

Once you have added your new OSD, Ceph will begin rebalancing the cluster by migrating placement groups to your new OSD. You can observe this process with the `ceph` CLI.

```
ceph -w
```

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

Add a Metadata Server

To use CephFS, you need at least one metadata server. Execute the following to create a metadata server:

```
ceph-deploy mds create {ceph-node}
```

For example:

```
ceph-deploy mds create node1
```

Note: Currently Ceph runs in production with one metadata server only. You may use more, but there is currently no commercial support for a cluster with multiple metadata servers.

Add an RGW Instance

To use the *Ceph Object Gateway* component of Ceph, you must deploy an instance of *RGW*. Execute the following to create an new instance of RGW:

```
ceph-deploy rgw create {gateway-node}
```

For example:

```
ceph-deploy rgw create node1
```

Note: This functionality is new with the **Hammer** release, and also with `ceph-deploy` v1.5.23.

By default, the *RGW* instance will listen on port 7480. This can be changed by editing ceph.conf on the node running the *RGW* as follows:

```
[client]
rgw frontends = civetweb port=80
```

To use an IPv6 address, use:

```
[client]
rgw frontends = civetweb port=[::]:80
```

Adding Monitors

A Ceph Storage Cluster requires at least one Ceph Monitor to run. For high availability, Ceph Storage Clusters typically run multiple Ceph Monitors so that the failure of a single Ceph Monitor will not bring down the Ceph Storage Cluster. Ceph uses the Paxos algorithm, which requires a majority of monitors (i.e., 1, 2:3, 3:4, 3:5, 4:6, etc.) to form a quorum.

Add two Ceph Monitors to your cluster.

```
ceph-deploy mon create {ceph-node}
```

For example:

```
ceph-deploy mon create node2 node3
```

Once you have added your new Ceph Monitors, Ceph will begin synchronizing the monitors and form a quorum. You can check the quorum status by executing the following:

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

Tip: When you run Ceph with multiple monitors, you SHOULD install and configure NTP on each monitor host. Ensure that the monitors are NTP peers.

2.2.4 Storing/Retrieving Object Data

To store object data in the Ceph Storage Cluster, 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 a Ceph OSD Daemon 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}
```

Exercise: Locate an Object

As an exercise, lets create an object. Specify an object name, a path to a test file containing some object data and a pool name using the `rados put` command on the command line. For example:

```
echo {Test-data} > testfile.txt
rados put {object-name} {file-path} --pool=data
rados put test-object-1 testfile.txt --pool=data
```

To verify that the Ceph Storage Cluster stored the object, execute the following:

```
rados -p data ls
```

Now, identify the object location:

```
ceph osd map {pool-name} {object-name}
ceph osd map data test-object-1
```

Ceph should output the object's location. For example:

```
osdmap e537 pool 'data' (0) object 'test-object-1' -> pg 0.d1743484 (0.4) -> up [1,0] acting [1,0]
```

To remove the test object, simply delete it using the `rados rm` command. For example:

```
rados rm test-object-1 --pool=data
```

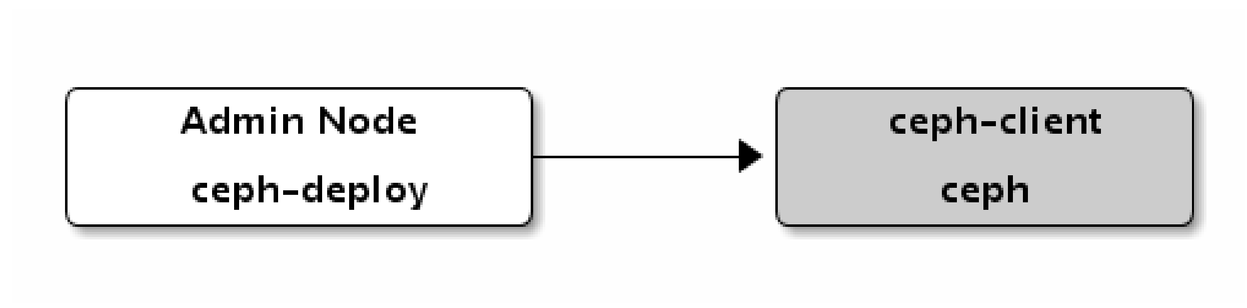
As the cluster evolves, the object location may change dynamically. One benefit of Ceph's dynamic rebalancing is that Ceph relieves you from having to perform the migration manually.

Most Ceph users don't store objects directly in the Ceph Storage Cluster. They typically use at least one of Ceph Block Devices, the Ceph Filesystem, and Ceph Object Storage.

2.3 Block Device Quick Start

To use this guide, you must have executed the procedures in the Storage Cluster Quick Start guide first. Ensure your *Ceph Storage Cluster* is in an `active + clean` state before working with the *Ceph Block Device*.

Note: The Ceph Block Device is also known as *RBD* or *RADOS* Block Device.



You may use a virtual machine for your `ceph-client` node, but do not execute the following procedures on the same physical node as your Ceph Storage Cluster nodes (unless you use a VM). See FAQ for details.

2.3.1 Install Ceph

1. Verify that you have an appropriate version of the Linux kernel. See OS Recommendations for details.

```
lsb_release -a
uname -r
```

2. On the admin node, use `ceph-deploy` to install Ceph on your `ceph-client` node.

```
ceph-deploy install ceph-client
```

3. On the admin node, use `ceph-deploy` to copy the Ceph configuration file and the `ceph.client.admin.keyring` to the `ceph-client`.

```
ceph-deploy admin ceph-client
```

   The `ceph-deploy` utility copies the keyring to the `/etc/ceph` directory. Ensure that the keyring file has appropriate read permissions (e.g., `sudo chmod +r /etc/ceph/ceph.client.admin.keyring`).

2.3.2 Configure a Block Device

1. On the `ceph-client` node, create a block device image.

```
rbd create foo --size 4096 [-m {mon-IP}] [-k /path/to/ceph.client.admin.keyring]
```

2. On the `ceph-client` node, map the image to a block device.

```
sudo rbd map foo --pool rbd --name client.admin [-m {mon-IP}] [-k /path/to/ceph.client.admin.key
```

3. Use the block device by creating a file system on the `ceph-client` node.

```
sudo mkfs.ext4 -m0 /dev/rbd/rbd/foo

This may take a few moments.
```

4. Mount the file system on the `ceph-client` node.

```
sudo mkdir /mnt/ceph-block-device
sudo mount /dev/rbd/rbd/foo /mnt/ceph-block-device
cd /mnt/ceph-block-device
```

See block devices for additional details.

2.4 Ceph FS Quick Start

To use the *Ceph FS* Quick Start guide, you must have executed the procedures in the Storage Cluster Quick Start guide first. Execute this quick start on the Admin Host.

2.4.1 Prerequisites

1. Verify that you have an appropriate version of the Linux kernel. See OS Recommendations for details.

```
lsb_release -a
uname -r
```

2. On the admin node, use `ceph-deploy` to install Ceph on your `ceph-client` node.

```
ceph-deploy install ceph-client
```

3. Ensure that the *Ceph Storage Cluster* is running and in an `active + clean` state. Also, ensure that you have at least one *Ceph Metadata Server* running.

```
ceph -s [-m {monitor-ip-address}] [-k {path/to/ceph.client.admin.keyring}]
```

2.4.2 Create a Filesystem

You have already created an MDS (Storage Cluster Quick Start) but it will not become active until you create some pools and a filesystem. See Create a Ceph filesystem.

```
ceph osd pool create cephfs_data <pg_num>
ceph osd pool create cephfs_metadata <pg_num>
ceph fs new <fs_name> cephfs_metadata cephfs_data
```

2.4.3 Create a Secret File

The Ceph Storage Cluster runs with authentication turned on by default. You should have a file containing the secret key (i.e., not the keyring itself). To obtain the secret key for a particular user, perform the following procedure:

1. Identify a key for a user within a keyring file. For example:

```
cat ceph.client.admin.keyring
```

2. Copy the key of the user who will be using the mounted Ceph FS filesystem. It should look something like this:

```
[client.admin]
   key = AQCj2YpRiAe6CxAA7/ETt7Hcl9IyxyYciVs47w==
```

3. Open a text editor.

4. Paste the key into an empty file. It should look something like this:

```
AQCj2YpRiAe6CxAA7/ETt7Hcl9IyxyYciVs47w==
```

5. Save the file with the user `name` as an attribute (e.g., `admin.secret`).

6. Ensure the file permissions are appropriate for the user, but not visible to other users.

2.4.4 Kernel Driver

Mount Ceph FS as a kernel driver.

```
sudo mkdir /mnt/mycephfs
sudo mount -t ceph {ip-address-of-monitor}:6789:/ /mnt/mycephfs
```

The Ceph Storage Cluster uses authentication by default. Specify a user `name` and the `secretfile` you created in the *Create a Secret File* section. For example:

```
sudo mount -t ceph 192.168.0.1:6789:/ /mnt/mycephfs -o name=admin,secretfile=admin.secret
```

Note: Mount the Ceph FS filesystem on the admin node, not the server node. See FAQ for details.

2.4.5 Filesystem in User Space (FUSE)

Mount Ceph FS as a Filesystem in User Space (FUSE).

```
sudo mkdir ~/mycephfs
sudo ceph-fuse -m {ip-address-of-monitor}:6789 ~/mycephfs
```

The Ceph Storage Cluster uses authentication by default. Specify a keyring if it is not in the default location (i.e., `/etc/ceph`):

```
sudo ceph-fuse -k ./ceph.client.admin.keyring -m 192.168.0.1:6789 ~/mycephfs
```

2.4.6 Additional Information

See Ceph FS for additional information. Ceph FS is not quite as stable as the Ceph Block Device and Ceph Object Storage. See Troubleshooting if you encounter trouble.

2.5 Quick Ceph Object Storage

To use the *Ceph Object Storage* Quick Start guide, you must have executed the procedures in the Storage Cluster Quick Start guide first. Make sure that you have at least one *RGW* instance running.

2.5.1 Configure new RGW instance

The *RGW* instance created by the Storage Cluster Quick Start will run using the embedded CivetWeb webserver. `ceph-deploy` will create the instance and start it automatically with default parameters.

To administer the *RGW* instance, see details in the the RGW Admin Guide.

Additional details may be found in the Configuring Ceph Object Gateway guide, but the steps specific to Apache are no longer needed.

Note: Deploying RGW using `ceph-deploy` and using the CivetWeb webserver instead of Apache is new functionality as of **Hammer** release.

CHAPTER

THREE

INSTALLATION (MANUAL)

3.1 Get Software

There are several methods for getting Ceph software. The easiest and most common method is to get packages by adding repositories for use with package management tools such as the Advanced Package Tool (APT) or Yellowdog Updater, Modified (YUM). You may also retrieve pre-compiled packages from the Ceph repository. Finally, you can retrieve tarballs or clone the Ceph source code repository and build Ceph yourself.

3.1.1 Get Packages

To install Ceph and other enabling software, you need to retrieve packages from the Ceph repository. Follow this guide to get packages; then, proceed to the Install Ceph Object Storage.

Getting Packages

There are two ways to get packages:

- **Add Repositories:** Adding repositories is the easiest way to get packages, because package management tools will retrieve the packages and all enabling software for you in most cases. However, to use this approach, each *Ceph Node* in your cluster must have internet access.
- **Download Packages Manually:** Downloading packages manually is a convenient way to install Ceph if your environment does not allow a *Ceph Node* to access the internet.

Requirements

All Ceph deployments require Ceph packages (except for development). You should also add keys and recommended packages.

- **Keys: (Recommended)** Whether you add repositories or download packages manually, you should download keys to verify the packages. If you do not get the keys, you may encounter security warnings. There are two keys: one for releases (common) and one for development (programmers and QA only). Choose the key that suits your needs. See *Add Keys* for details.
- **Ceph Extras: (Required)** The Ceph Extras repository provides newer Ceph-enabled versions of packages which are already provided in your Linux distribution, but where newer versions are required to support Ceph. Examples of newer versions of available packages include QEMU for CentOS/RHEL distribution and iSCSI among others. If you intend to use any of the foregoing packages, you must add the Ceph Extras repository or download the packages manually. This repository also contains Ceph dependencies for those who intend to install Ceph manually. See *Add Ceph Extras* for details.

- **Ceph: (Required)** All Ceph deployments require Ceph release packages, except for deployments that use development packages (development, QA, and bleeding edge deployments only). See *Add Ceph* for details.
- **Ceph Development: (Optional)** If you are developing for Ceph, testing Ceph development builds, or if you want features from the bleeding edge of Ceph development, you may get Ceph development packages. See *Add Ceph Development* for details.
- **Apache/FastCGI: (Optional)** If you are deploying a *Ceph Object Storage* service, you must install Apache and FastCGI. Ceph provides Apache and FastCGI builds that are identical to those available from Apache, but with 100-continue support. If you want to enable *Ceph Object Gateway* daemons with 100-continue support, you must retrieve Apache/FastCGI packages from the Ceph repository. See *Add Apache/FastCGI* for details.

If you intend to download packages manually, see Section *Download Packages*.

Add Keys

Add a key to your system's list of trusted keys to avoid a security warning. For major releases (e.g., `dumpling`, `emperor`, `firefly`) and development releases (`release-name-rc1`, `release-name-rc2`), use the `release.asc` key. For development testing packages, use the `autobuild.asc` key (developers and QA).

APT

To install the `release.asc` key, execute the following:

```
wget -q -O- 'https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc' | sudo apt-key add -
```

To install the `autobuild.asc` key, execute the following (QA and developers only):

```
wget -q -O- 'https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/autobuild.asc' | sudo apt-key add
```

RPM

To install the `release.asc` key, execute the following:

```
sudo rpm --import 'https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc'
```

To install the `autobuild.asc` key, execute the following (QA and developers only):

```
sudo rpm --import 'https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/autobuild.asc'
```

Add Ceph Extras

Some Ceph deployments require newer Ceph-enabled versions of packages that are already available in your Linux distribution. For example, Ceph Extras contains newer Ceph-enabled packages for the SCSI target framework and QEMU packages for RPMs. The repository also contains `curl`, `leveldb` and other Ceph dependencies. Add the Ceph Extras repository to ensure you obtain these additional packages from the Ceph repository.

Debian Packages

Add our Ceph Extras package repository to your system's list of APT sources.

```
echo deb http://ceph.com/packages/ceph-extras/debian $(lsb_release -sc) main | sudo tee /etc/apt/sou
```

RPM Packages

Note: ceph-extras on RPM-based systems is only needed on EL6-based distributions (RHEL 6, CentOS 6, Scientific Linux 6). It is not needed for Fedora or RHEL 7+.

For RPM packages, add our package repository to your `/etc/yum.repos.d` repos (e.g., `ceph-extras.repo`). Some Ceph packages (e.g., QEMU) must take priority over standard packages, so you must ensure that you set `priority=2`.

```
[ceph-extras]
name=Ceph Extras Packages
baseurl=http://ceph.com/packages/ceph-extras/rpm/{distro}/$basearch
enabled=1
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc

[ceph-extras-noarch]
name=Ceph Extras noarch
baseurl=http://ceph.com/packages/ceph-extras/rpm/{distro}/noarch
enabled=1
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc

[ceph-extras-source]
name=Ceph Extras Sources
baseurl=http://ceph.com/packages/ceph-extras/rpm/{distro}/SRPMS
enabled=1
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc
```

Add Ceph

Release repositories use the `release.asc` key to verify packages. To install Ceph packages with the Advanced Package Tool (APT) or Yellowdog Updater, Modified (YUM), you must add Ceph repositories.

You may find releases for Debian/Ubuntu (installed with APT) at:

```
http://ceph.com/debian-{release-name}
```

You may find releases for CentOS/RHEL and others (installed with YUM) at:

```
http://ceph.com/rpm-{release-name}
```

The major releases of Ceph include:

- **Giant:** Giant is the most recent major release of Ceph. These packages are recommended for anyone deploying Ceph in a production environment. Critical bug fixes are backported and point releases are made as necessary.
- **Firefly:** Firefly is the sixth major release of Ceph. These packages are recommended for anyone deploying Ceph in a production environment. Firefly is a long-term stable release, so critical bug fixes are backported and point releases are made as necessary.

- **Emperor:** Emperor is the fifth major release of Ceph. These packages are are old and no longer supported, so we recommend that users upgrade to Firefly immediately.
- **Dumpling:** Dumpling is the fourth major release of Ceph. These packages are older and not recommended for new users, but critical bug fixes are still backported as necessary. We encourage all Dumpling users to update to Firefly as soon as they are able to do so.
- **Argonaut, Bobtail, Cuttlefish:** These are the first three releases of Ceph. These packages are old and no longer supported, so we recommend that users upgrade to a supported version.

Tip: For European users, there is also a mirror in the Netherlands at: http://eu.ceph.com/

Debian Packages

Add a Ceph package repository to your system's list of APT sources. For newer versions of Debian/Ubuntu, call `lsb_release -sc` on the command line to get the short codename, and replace `{codename}` in the following command.

```
sudo apt-add-repository 'deb http://ceph.com/debian-firefly/ {codename} main'
```

For early Linux distributions, you may execute the following command:

```
echo deb http://ceph.com/debian-firefly/ $(lsb_release -sc) main | sudo tee /etc/apt/sources.list.d/c
```

For earlier Ceph releases, replace `{release-name}` with the name with the name of the Ceph release. You may call `lsb_release -sc` on the command line to get the short codename, and replace `{codename}` in the following command.

```
sudo apt-add-repository 'deb http://ceph.com/debian-{release-name}/ {codename} main'
```

For older Linux distributions, replace `{release-name}` with the name of the release:

```
echo deb http://ceph.com/debian-{release-name}/ $(lsb_release -sc) main | sudo tee /etc/apt/sources.l
```

Ceph on ARM processors requires Google's memory profiling tools (`google-perftools`). The Ceph repository should have a copy at http://ceph.com/packages/google-perftools/debian.

```
echo deb http://ceph.com/packages/google-perftools/debian  $(lsb_release -sc) main | sudo tee /etc/ap
```

For development release packages, add our package repository to your system's list of APT sources. See the testing Debian repository for a complete list of Debian and Ubuntu releases supported.

```
echo deb http://ceph.com/debian-testing/ $(lsb_release -sc) main | sudo tee /etc/apt/sources.list.d/c
```

RPM Packages

For major releases, you may add a Ceph entry to the `/etc/yum.repos.d` directory. Create a `ceph.repo` file. In the example below, replace `{ceph-release}` with a major release of Ceph (e.g., `dumpling`, `emperor`, etc.) and `{distro}` with your Linux distribution (e.g., `el6`, `rhel6`, etc.). You may view http://ceph.com/rpm-{ceph-release}/ directory to see which distributions Ceph supports. Some Ceph packages (e.g., EPEL) must take priority over standard packages, so you must ensure that you set `priority=2`.

```
[ceph]
name=Ceph packages for $basearch
baseurl=http://ceph.com/rpm-{ceph-release}/{distro}/$basearch
```

```
enabled=1
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc

[ceph-noarch]
name=Ceph noarch packages
baseurl=http://ceph.com/rpm-{ceph-release}/{distro}/noarch
enabled=1
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc

[ceph-source]
name=Ceph source packages
baseurl=http://ceph.com/rpm-{ceph-release}/{distro}/SRPMS
enabled=0
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc
```

For development release packages, you may specify the repository for development releases instead.

```
[ceph]
name=Ceph packages for $basearch/$releasever
baseurl=http://ceph.com/rpm-testing/{distro}/$basearch
enabled=1
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc

[ceph-noarch]
name=Ceph noarch packages
baseurl=http://ceph.com/rpm-testing/{distro}/noarch
enabled=1
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc

[ceph-source]
name=Ceph source packages
baseurl=http://ceph.com/rpm-testing/{distro}/SRPMS
enabled=0
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc
```

For specific packages, you may retrieve them by specifically downloading the release package by name. Our development process generates a new release of Ceph every 3-4 weeks. These packages are faster-moving than the major releases. Development packages have new features integrated quickly, while still undergoing several weeks of QA prior to release.

The repository package installs the repository details on your local system for use with `yum` or `up2date`. Replace `{distro}` with your Linux distribution, and `{release}` with the specific release of Ceph:

```
su -c 'rpm -Uvh http://ceph.com/rpms/{distro}/x86_64/ceph-{release}.el6.noarch.rpm'
```

You can download the RPMs directly from:

```
http://ceph.com/rpm-testing
```

Add Ceph Development

Development repositories use the `autobuild.asc` key to verify packages. If you are developing Ceph and need to deploy and test specific Ceph branches, ensure that you remove repository entries for major releases first.

Debian Packages

We automatically build Debian and Ubuntu packages for current development branches in the Ceph source code repository. These packages are intended for developers and QA only.

Add our package repository to your system's list of APT sources, but replace `{BRANCH}` with the branch you'd like to use (e.g., chef-3, wip-hack, master). See the gitbuilder page for a complete list of distributions we build.

```
echo deb http://gitbuilder.ceph.com/ceph-deb-$(lsb_release -sc)-x86_64-basic/ref/{BRANCH} $(lsb_relea
```

RPM Packages

For current development branches, you may add a Ceph entry to the `/etc/yum.repos.d` directory. Create a `ceph.repo` file. In the example below, replace `{distro}` with your Linux distribution (e.g., `centos6`, `rhel6`, etc.), and `{branch}` with the name of the branch you want to install.

```
[ceph-source]
name=Ceph source packages
baseurl=http://gitbuilder.ceph.com/ceph-rpm-{distro}-x86_64-basic/ref/{branch}/SRPMS
enabled=0
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/autobuild.asc
```

You may view http://gitbuilder.ceph.com directory to see which distributions Ceph supports.

Add Apache/FastCGI

Ceph Object Gateway works with ordinary Apache and FastCGI libraries. However, Ceph builds Apache and FastCGI packages that support 100-continue. To use the Ceph Apache and FastCGI packages, add them to your repository.

Debian Packages

Add our Apache and FastCGI packages to your system's list of APT sources if you intend to use 100-continue.

```
echo deb http://gitbuilder.ceph.com/apache2-deb-$(lsb_release -sc)-x86_64-basic/ref/master $(lsb_rele
echo deb http://gitbuilder.ceph.com/libapache-mod-fastcgi-deb-$(lsb_release -sc)-x86_64-basic/ref/mas
```

RPM Packages

You may add a Ceph entry to the `/etc/yum.repos.d` directory. Create a `ceph-apache.repo` file. In the example below, replace `{distro}` with your Linux distribution (e.g., `el6`, `rhel6`, etc.). You may view http://gitbuilder.ceph.com directory to see which distributions Ceph supports.

```
[apache2-ceph-noarch]
name=Apache noarch packages for Ceph
baseurl=http://gitbuilder.ceph.com/apache2-rpm-{distro}-x86_64-basic/ref/master
enabled=1
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/autobuild.asc

[apache2-ceph-source]
name=Apache source packages for Ceph
baseurl=http://gitbuilder.ceph.com/apache2-rpm-{distro}-x86_64-basic/ref/master
enabled=0
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/autobuild.asc
```

Repeat the forgoing process by creating a `ceph-fastcgi.repo` file.

```
[fastcgi-ceph-basearch]
name=FastCGI basearch packages for Ceph
baseurl=http://gitbuilder.ceph.com/mod_fastcgi-rpm-{distro}-x86_64-basic/ref/master
enabled=1
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/autobuild.asc

[fastcgi-ceph-noarch]
name=FastCGI noarch packages for Ceph
baseurl=http://gitbuilder.ceph.com/mod_fastcgi-rpm-{distro}-x86_64-basic/ref/master
enabled=1
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/autobuild.asc

[fastcgi-ceph-source]
name=FastCGI source packages for Ceph
baseurl=http://gitbuilder.ceph.com/mod_fastcgi-rpm-{distro}-x86_64-basic/ref/master
enabled=0
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/autobuild.asc
```

Download Packages

If you are attempting to install behind a firewall in an environment without internet access, you must retrieve the packages (mirrored with all the necessary dependencies) before attempting an install.

Debian Packages

Ceph requires additional additional third party libraries.

- libaio1
- libsnappy1
- libcurl3
- curl
- libgoogle-perftools4
- google-perftools
- libleveldb1

The repository package installs the repository details on your local system for use with `apt`. Replace `{release}` with the latest Ceph release. Replace `{version}` with the latest Ceph version number. Replace `{distro}` with your Linux distribution codename. Replace `{arch}` with the CPU architecture.

```
wget -q http://ceph.com/debian-{release}/pool/main/c/ceph/ceph_{version}{distro}_{arch}.deb
```

RPM Packages

Ceph requires additional additional third party libraries. To add the EPEL repository, execute the following:

```
su -c 'rpm -Uvh http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm'
```

Ceph requires the following packages:

- snappy
- leveldb
- gdisk
- python-argparse
- gperftools-libs

Packages are currently built for the RHEL/CentOS6 (`el6`), Fedora 18 and 19 (`f18` and `f19`), OpenSUSE 12.2 (`opensuse12.2`), and SLES (`sles11`) platforms. The repository package installs the repository details on your local system for use with `yum` or `up2date`. Replace `{distro}` with your distribution.

```
su -c 'rpm -Uvh http://ceph.com/rpm-firefly/{distro}/noarch/ceph-{version}.{distro}.noarch.rpm'
```

For example, for CentOS 6 (`el6`):

```
su -c 'rpm -Uvh http://ceph.com/rpm-firefly/el6/noarch/ceph-release-1-0.el6.noarch.rpm'
```

You can download the RPMs directly from:

```
http://ceph.com/rpm-firefly
```

For earlier Ceph releases, replace `{release-name}` with the name with the name of the Ceph release. You may call `lsb_release -sc` on the command line to get the short codename.

```
su -c 'rpm -Uvh http://ceph.com/rpm-{release-name}/{distro}/noarch/ceph-{version}.{distro}.noarch.rpm
```

3.1.2 Downloading a Ceph Release Tarball

As Ceph development progresses, the Ceph team releases new versions of the source code. You may download source code tarballs for Ceph releases here:

Ceph Release Tarballs

Ceph Release Tarballs (EU mirror)

3.1.3 Cloning the Ceph Source Code Repository

You may clone a Ceph branch of the Ceph source code by going to github Ceph Repository, selecting a branch (`master` by default), and clicking the **Download ZIP** button.

To clone the entire git repository, install and configure `git`.

Install Git

To install `git` on Debian/Ubuntu, execute:

```
sudo apt-get install git
```

To install `git` on CentOS/RHEL, execute:

```
sudo yum install git
```

You must also have a `github` account. If you do not have a `github` account, go to github.com and register. Follow the directions for setting up git at Set Up Git.

Add SSH Keys (Optional)

If you intend to commit code to Ceph or to clone using SSH (`git@github.com:ceph/ceph.git`), you must generate SSH keys for github.

Tip: If you only intend to clone the repository, you may use `git clone --recursive https://github.com/ceph/ceph.git` without generating SSH keys.

To generate SSH keys for `github`, execute:

```
ssh-keygen
```

Get the key to add to your `github` account (the following example assumes you used the default file path):

```
cat .ssh/id_rsa.pub
```

Copy the public key.

Go to your your `github` account, click on "Account Settings" (i.e., the 'tools' icon); then, click "SSH Keys" on the left side navbar.

Click "Add SSH key" in the "SSH Keys" list, enter a name for the key, paste the key you generated, and press the "Add key" button.

Clone the Source

To clone the Ceph source code repository, execute:

```
git clone --recursive https://github.com/ceph/ceph.git
```

Once `git clone` executes, you should have a full copy of the Ceph repository.

Tip: Make sure you maintain the latest copies of the submodules included in the repository. Running `git status` will tell you if the submodules are out of date.

```
cd ceph
git status
```

If your submodules are out of date, run:

```
git submodule update --force --init --recursive
```

Choose a Branch

Once you clone the source code and submodules, your Ceph repository will be on the `master` branch by default, which is the unstable development branch. You may choose other branches too.

- `master`: The unstable development branch.
- `stable`: The bugfix branch.
- `next`: The release candidate branch.

```
git checkout master
```

3.1.4 Build Ceph

You can get Ceph software by retrieving Ceph source code and building it yourself. To build Ceph, you need to set up a development environment, compile Ceph, and then either install in user space or build packages and install the packages.

Build Prerequisites

Tip: Check this section to see if there are specific prerequisites for your Linux/Unix distribution.

Before you can build Ceph source code, you need to install several libraries and tools:

```
./install-deps.sh
```

Note: Some distributions that support Google's memory profiler tool may use a different package name (e.g., `libgoogle-perftools4`).

Build Ceph

Ceph provides `automake` and `configure` scripts to streamline the build process. To build Ceph, navigate to your cloned Ceph repository and execute the following:

```
cd ceph
./autogen.sh
./configure
make
```

Hyperthreading

You can use `make -j` to execute multiple jobs depending upon your system. For example, `make -j4` for a dual core processor may build faster.

See Installing a Build to install a build in user space.

Build Ceph Packages

To build packages, you must clone the Ceph repository. You can create installation packages from the latest code using `dpkg-buildpackage` for Debian/Ubuntu or `rpmbuild` for the RPM Package Manager.

Tip: When building on a multi-core CPU, use the `-j` and the number of cores * 2. For example, use `-j4` for a dual-core processor to accelerate the build.

Advanced Package Tool (APT)

To create `.deb` packages for Debian/Ubuntu, ensure that you have cloned the Ceph repository, installed the *Build Prerequisites* and installed `debhelper`:

```
sudo apt-get install debhelper
```

Once you have installed debhelper, you can build the packages:

```
sudo dpkg-buildpackage
```

For multi-processor CPUs use the `-j` option to accelerate the build.

RPM Package Manager

To create `.rpm` packages, ensure that you have cloned the Ceph repository, installed the *Build Prerequisites* and installed `rpm-build` and `rpmdevtools`:

```
yum install rpm-build rpmdevtools
```

Once you have installed the tools, setup an RPM compilation environment:

```
rpmdev-setuptree
```

Fetch the source tarball for the RPM compilation environment:

```
wget -P ~/rpmbuild/SOURCES/ http://ceph.com/download/ceph-<version>.tar.bz2
```

Or from the EU mirror:

```
wget -P ~/rpmbuild/SOURCES/ http://eu.ceph.com/download/ceph-<version>.tar.bz2
```

Extract the specfile:

```
tar --strip-components=1 -C ~/rpmbuild/SPECS/ --no-anchored -xvjf ~/rpmbuild/SOURCES/ceph-<version>.t
```

Build the RPM packages:

```
rpmbuild -ba ~/rpmbuild/SPECS/ceph.spec
```

For multi-processor CPUs use the `-j` option to accelerate the build.

3.2 Install Software

Once you have the Ceph software (or added repositories), installing the software is easy. To install packages on each *Ceph Node* in your cluster. You may use `ceph-deploy` to install Ceph for your storage cluster, or use package management tools. You should install Yum Priorities for RHEL/CentOS and other distributions that use Yum if you intend to install the Ceph Object Gateway or QEMU.

3.2.1 Install Ceph Deploy

The `ceph-deploy` tool enables you to set up and tear down Ceph clusters for development, testing and proof-of-concept projects.

APT

To install `ceph-deploy` with `apt`, execute the following:

```
sudo apt-get update && sudo apt-get install ceph-deploy
```

RPM

To install `ceph-deploy` with `yum`, execute the following:

```
sudo yum install ceph-deploy
```

3.2.2 Install Ceph Storage Cluster

This guide describes installing Ceph packages manually. This procedure is only for users who are not installing with a deployment tool such as `ceph-deploy`, `chef`, `juju`, etc.

Tip: You can also use `ceph-deploy` to install Ceph packages, which may be more convenient since you can install `ceph` on multiple hosts with a single command.

Installing with APT

Once you have added either release or development packages to APT, you should update APT's database and install Ceph:

```
sudo apt-get update && sudo apt-get install ceph ceph-mds
```

Installing with RPM

To install Ceph with RPMs, execute the following steps:

1. Install `yum-plugin-priorities`.

```
sudo yum install yum-plugin-priorities
```

2. Ensure `/etc/yum/pluginconf.d/priorities.conf` exists.
3. Ensure `priorities.conf` enables the plugin.

```
[main]
enabled = 1
```

4. Ensure your YUM `ceph.repo` entry includes `priority=2`. See Get Packages for details:

```
[ceph]
name=Ceph packages for $basearch
baseurl=http://ceph.com/rpm-{ceph-release}/{distro}/$basearch
enabled=1
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc

[ceph-noarch]
name=Ceph noarch packages
baseurl=http://ceph.com/rpm-{ceph-release}/{distro}/noarch
enabled=1
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc

[ceph-source]
name=Ceph source packages
baseurl=http://ceph.com/rpm-{ceph-release}/{distro}/SRPMS
enabled=0
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc
```

5. Install pre-requisite packages:

```
sudo yum install snappy leveldb gdisk python-argparse gperftools-libs
```

Once you have added either release or development packages, or added a `ceph.repo` file to `/etc/yum.repos.d`, you can install Ceph packages.

```
sudo yum install ceph
```

Installing a Build

If you build Ceph from source code, you may install Ceph in user space by executing the following:

```
sudo make install
```

If you install Ceph locally, `make` will place the executables in `usr/local/bin`. You may add the Ceph configuration file to the `usr/local/bin` directory to run Ceph from a single directory.

3.2.3 Install Ceph Object Gateway

Note: To run the Ceph object gateway service, you should have a running Ceph cluster, the gateway host should have access to storage and public networks, and SELinux should be in permissive mode in rpm-based distros.

The *Ceph Object Gateway* daemon runs on Apache and FastCGI.

To run a *Ceph Object Storage* service, you must install Apache and Ceph Object Gateway daemon on the host that is going to provide the gateway service, i.e, the `gateway host`. If you plan to run a Ceph Object Storage service with a federated architecture (multiple regions and zones), you must also install the synchronization agent.

Note: Previous versions of Ceph shipped with `mod_fastcgi`. The current version ships with `mod_proxy_fcgi` instead.

In distros that ship Apache 2.4 (such as RHEL 7, CentOS 7 or Ubuntu 14.04 `Trusty`), `mod_proxy_fcgi` is already present. When you install the `httpd` package with `yum` or the `apache2` package with `apt-get`, `mod_proxy_fcgi` becomes available for use on your server.

In distros that ship Apache 2.2 (such as RHEL 6, CentOS 6 or Ubuntu 12.04 `Precise`), `mod_proxy_fcgi` comes as a separate package. In **RHEL 6/CentOS 6**, it is available in `EPEL 6` repo and can be installed with `yum install mod_proxy_fcgi`. For **Ubuntu 12.04**, a backport for `mod_proxy_fcgi` is in progress and a bug has been filed for the same. See: ceph radosgw needs mod-proxy-fcgi for apache 2.2

Install Apache

To install Apache on the `gateway host`, execute the following:

On Debian-based distros, run:

```
sudo apt-get install apache2
```

On RPM-based distros, run:

```
sudo yum install httpd
```

Configure Apache

Make the following changes in Apache's configuration on the `gateway host`:

Debian-based distros

1. Add a line for the `ServerName` in `/etc/apache2/apache2.conf`. Provide the fully qualified domain name of the server machine (e.g., `hostname -f`):

```
    ServerName {fqdn}
```

2. Load `mod_proxy_fcgi` module.

   Execute:

```
sudo a2enmod proxy_fcgi
```

3. Start Apache service:

```
sudo service apache2 start
```

RPM-based distros

1. Open the `httpd.conf` file:

```
sudo vim /etc/httpd/conf/httpd.conf
```

2. Uncomment `#ServerName` in the file and add the name of your server. Provide the fully qualified domain name of the server machine (e.g., `hostname -f`):

```
ServerName {fqdn}
```

3. Update `/etc/httpd/conf/httpd.conf` to load `mod_proxy_fcgi` module. Add the following to the file:

```
<IfModule !proxy_fcgi_module>
LoadModule proxy_fcgi_module modules/mod_proxy_fcgi.so
</IfModule>
```

4. Edit the line `Listen 80` in `/etc/httpd/conf/httpd.conf` with the public IP address of the host that you are configuring as a gateway server. Write `Listen {IP ADDRESS}:80` in place of `Listen 80`.

5. Start httpd service

   Execute:

```
sudo service httpd start
```

   Or:

```
sudo systemctl start httpd
```

Enable SSL

Some REST clients use HTTPS by default. So you should consider enabling SSL for Apache. Use the following procedures to enable SSL.

Note: You can use self-certified certificates. Some client APIs check for a trusted certificate authority. You may need to obtain a SSL certificate from a trusted authority to use those client APIs.

Debian-based distros

To enable SSL on Debian-based distros, execute the following steps:

1. Ensure that you have installed the dependencies:

```
sudo apt-get install openssl ssl-cert
```

2. Enable the SSL module:

```
sudo a2enmod ssl
```

3. Generate a certificate:

```
sudo mkdir /etc/apache2/ssl
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/apache2/ssl/apache.key -ou
```

4. Restart Apache:

```
sudo service apache2 restart
```

See the Ubuntu Server Guide for additional details.

RPM-based distros

To enable SSL on RPM-based distros, execute the following steps:

1. Ensure that you have installed the dependencies:

```
sudo yum install mod_ssl openssl
```

2. Generate private key:

```
openssl genrsa -out ca.key 2048
```

3. Generate CSR:

```
openssl req -new -key ca.key -out ca.csr
```

4. Generate a certificate:

```
openssl x509 -req -days 365 -in ca.csr -signkey ca.key -out ca.crt
```

5. Copy the files to appropriate locations:

```
sudo cp ca.crt /etc/pki/tls/certs
sudo cp ca.key /etc/pki/tls/private/ca.key
sudo cp ca.csr /etc/pki/tls/private/ca.csr
```

6. Update the Apache SSL configuration file `/etc/httpd/conf.d/ssl.conf`.

   Give the correct location of `SSLCertificateFile`:

```
SSLCertificateFile /etc/pki/tls/certs/ca.crt
```

   Give the correct location of `SSLCertificateKeyFile`:

```
SSLCertificateKeyFile /etc/pki/tls/private/ca.key
```

   Save the changes.

7. Restart Apache.

   Execute:

```
sudo service httpd restart
```

   Or:

```
sudo systemctl restart httpd
```

See Setting up an SSL secured Webserver with CentOS for additional details.

Install Ceph Object Gateway Daemon

Ceph Object Storage services use the Ceph Object Gateway daemon (`radosgw`) to enable the gateway. For federated architectures, the synchronization agent (`radosgw-agent`) provides data and metadata synchronization between zones and regions.

Debian-based distros

To install the Ceph Object Gateway daemon on the *gateway host*, execute the following:

```
sudo apt-get install radosgw
```

To install the Ceph Object Gateway synchronization agent, execute the following:

```
sudo apt-get install radosgw-agent
```

RPM-based distros

To install the Ceph Object Gateway daemon on the `gateway host`, execute the following:

```
sudo yum install ceph-radosgw
```

To install the Ceph Object Gateway synchronization agent, execute the following:

```
sudo yum install radosgw-agent
```

Configure The Gateway

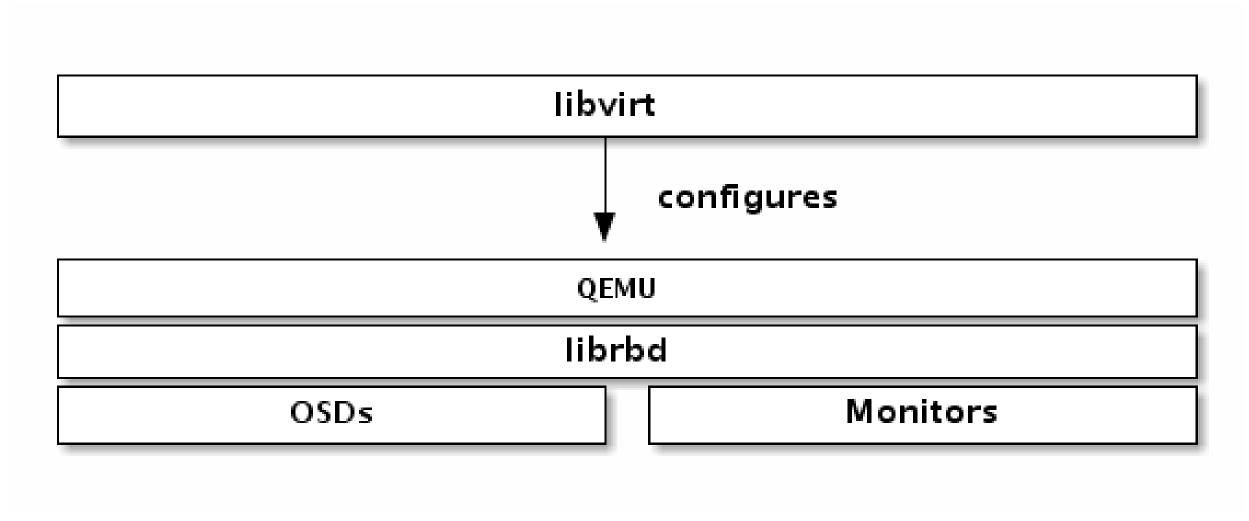
Once you have installed the Ceph Object Gateway packages, the next step is to configure your Ceph Object Gateway. There are two approaches:

- **Simple:** A simple Ceph Object Gateway configuration implies that you are running a Ceph Object Storage service in a single data center. So you can configure the Ceph Object Gateway without regard to regions and zones.
- **Federated:** A federated Ceph Object Gateway configuration implies that you are running a Ceph Object Storage service in a geographically distributed manner for fault tolerance and failover. This involves configuring your Ceph Object Gateway instances with regions and zones.

Choose the approach that best reflects your cluster.

3.2.4 Install Virtualization for Block Device

If you intend to use Ceph Block Devices and the Ceph Storage Cluster as a backend for Virtual Machines (VMs) or *Cloud Platforms* the QEMU/KVM and `libvirt` packages are important for enabling VMs and cloud platforms. Examples of VMs include: QEMU/KVM, XEN, VMWare, LXC, VirtualBox, etc. Examples of Cloud Platforms include OpenStack, CloudStack, OpenNebula, etc.



Install QEMU

QEMU KVM can interact with Ceph Block Devices via `librbd`, which is an important feature for using Ceph with cloud platforms. Once you install QEMU, see QEMU and Block Devices for usage.

Debian Packages

QEMU packages are incorporated into Ubuntu 12.04 Precise Pangolin and later versions. To install QEMU, execute the following:

```
sudo apt-get install qemu
```

RPM Packages

To install QEMU, execute the following:

1. Install `yum-plugin-priorities`.

```
sudo yum install yum-plugin-priorities
```

2. Ensure `/etc/yum/pluginconf.d/priorities.conf` exists.
3. Ensure `priorities.conf` enables the plugin.

```
[main]
enabled = 1
```

Note: ceph-extras on RPM-based systems is only needed on EL6-based distributions (RHEL 6, CentOS 6, Scientific Linux 6). It is not needed for Fedora or RHEL 7+.

1. Create a `/etc/yum.repos.d/ceph-extras.repo` file with the following contents, and replace `{distro}` with your Linux distribution. Follow the `baseurl` path below to see which distributions Ceph supports:

```
[ceph-extras]
name=Ceph Extras
baseurl=http://ceph.com/packages/ceph-extras/rpm/{distro}/$basearch
enabled=1
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc

[ceph-qemu-source]
name=Ceph Extras Sources
baseurl=http://ceph.com/packages/ceph-extras/rpm/{distro}/SRPMS
enabled=1
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc
```

2. Update your repositories.

```
sudo yum update
```

3. Ensure that non-priority versions are removed.

```
sudo yum remove qemu-kvm qemu-kvm-tools qemu-img
sudo yum clean all
```

4. Install QEMU for Ceph.

```
sudo yum install qemu-kvm qemu-kvm-tools qemu-img
```

5. Install additional QEMU packages (optional):

```
sudo yum install qemu-guest-agent qemu-guest-agent-win32
```

Building QEMU

To build QEMU from source, use the following procedure:

```
cd {your-development-directory}
git clone git://git.qemu.org/qemu.git
cd qemu
./configure --enable-rbd
make; make install
```

Install libvirt

To use `libvirt` with Ceph, you must have a running Ceph Storage Cluster, and you must have installed and configured QEMU. See Using libvirt with Ceph Block Device for usage.

Debian Packages

`libvirt` packages are incorporated into Ubuntu 12.04 Precise Pangolin and later versions of Ubuntu. To install `libvirt` on these distributions, execute the following:

```
sudo apt-get update && sudo apt-get install libvirt-bin
```

RPM Packages

To use `libvirt` with a Ceph Storage Cluster, you must have a running Ceph Storage Cluster and you must also install a version of QEMU with `rbd` format support. See *Install QEMU* for details.

`libvirt` packages are incorporated into the recent CentOS/RHEL distributions. To install `libvirt`, execute the following:

```
sudo yum install libvirt
```

Building `libvirt`

To build `libvirt` from source, clone the `libvirt` repository and use AutoGen to generate the build. Then, execute `make` and `make install` to complete the installation. For example:

```
git clone git://libvirt.org/libvirt.git
cd libvirt
./autogen.sh
make
sudo make install
```

See libvirt Installation for details.

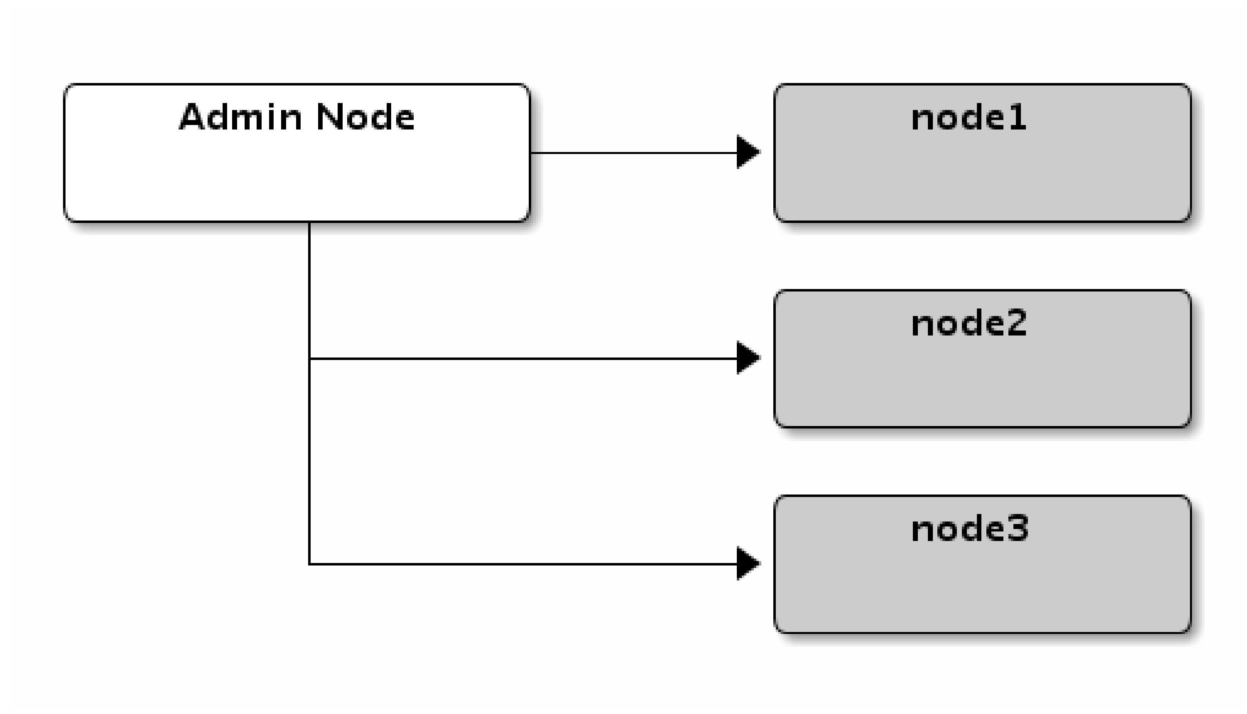
3.3 Deploy a Cluster Manually

Once you have Ceph installed on your nodes, you can deploy a cluster manually. The manual procedure is primarily for exemplary purposes for those developing deployment scripts with Chef, Juju, Puppet, etc.

3.3.1 Manual Deployment

All Ceph clusters require at least one monitor, and at least as many OSDs as copies of an object stored on the cluster. Bootstrapping the initial monitor(s) is the first step in deploying a Ceph Storage Cluster. Monitor deployment also sets important criteria for the entire cluster, such as the number of replicas for pools, the number of placement groups per OSD, the heartbeat intervals, whether authentication is required, etc. Most of these values are set by default, so it's useful to know about them when setting up your cluster for production.

Following the same configuration as Installation (Quick), we will set up a cluster with `node1` as the monitor node, and `node2` and `node3` for OSD nodes.



Monitor Bootstrapping

Bootstrapping a monitor (a Ceph Storage Cluster, in theory) requires a number of things:

- **Unique Identifier:** The `fsid` is a unique identifier for the cluster, and stands for File System ID from the days when the Ceph Storage Cluster was principally for the Ceph Filesystem. Ceph now supports native interfaces, block devices, and object storage gateway interfaces too, so `fsid` is a bit of a misnomer.
- **Cluster Name:** Ceph clusters have a cluster name, which is a simple string without spaces. The default cluster name is `ceph`, but you may specify a different cluster name. Overriding the default cluster name is especially useful when you are working with multiple clusters and you need to clearly understand which cluster your are working with.

  For example, when you run multiple clusters in a federated architecture, the cluster name (e.g., `us-west`, `us-east`) identifies the cluster for the current CLI session. **Note:** To identify the cluster name on the command line interface, specify the a Ceph configuration file with the cluster name (e.g., `ceph.conf`, `us-west.conf`, `us-east.conf`, etc.). Also see CLI usage (`ceph --cluster {cluster-name}`).
- **Monitor Name:** Each monitor instance within a cluster has a unique name. In common practice, the Ceph Monitor name is the host name (we recommend one Ceph Monitor per host, and no commingling of Ceph OSD Daemons with Ceph Monitors). You may retrieve the short hostname with `hostname -s`.
- **Monitor Map:** Bootstrapping the initial monitor(s) requires you to generate a monitor map. The monitor map requires the `fsid`, the cluster name (or uses the default), and at least one host name and its IP address.
- **Monitor Keyring**: Monitors communicate with each other via a secret key. You must generate a keyring with a monitor secret and provide it when bootstrapping the initial monitor(s).
- **Administrator Keyring**: To use the `ceph` CLI tools, you must have a `client.admin` user. So you must generate the admin user and keyring, and you must also add the `client.admin` user to the monitor keyring.

The foregoing requirements do not imply the creation of a Ceph Configuration file. However, as a best practice, we recommend creating a Ceph configuration file and populating it with the `fsid`, the `mon initial members` and the `mon host` settings.

You can get and set all of the monitor settings at runtime as well. However, a Ceph Configuration file may contain only those settings that override the default values. When you add settings to a Ceph configuration file, these settings override the default settings. Maintaining those settings in a Ceph configuration file makes it easier to maintain your cluster.

The procedure is as follows:

1. Log in to the initial monitor node(s):

```
ssh {hostname}
```

   For example:

```
ssh node1
```

2. Ensure you have a directory for the Ceph configuration file. By default, Ceph uses `/etc/ceph`. When you install `ceph`, the installer will create the `/etc/ceph` directory automatically.

```
ls /etc/ceph
```

   Note: Deployment tools may remove this directory when purging a cluster (e.g., `ceph-deploy purgedata {node-name}`, `ceph-deploy purge {node-name}`).

3. Create a Ceph configuration file. By default, Ceph uses `ceph.conf`, where `ceph` reflects the cluster name.

```
sudo vim /etc/ceph/ceph.conf
```

4. Generate a unique ID (i.e., `fsid`) for your cluster.

```
uuidgen
```

5. Add the unique ID to your Ceph configuration file.

```
fsid = {UUID}
```

   For example:

```
fsid = a7f64266-0894-4f1e-a635-d0aeaca0e993
```

6. Add the initial monitor(s) to your Ceph configuration file.

```
mon initial members = {hostname}[,{hostname}]
```

   For example:

```
mon initial members = node1
```

7. Add the IP address(es) of the initial monitor(s) to your Ceph configuration file and save the file.

```
mon host = {ip-address}[,{ip-address}]
```

   For example:

```
mon host = 192.168.0.1
```

   Note: You may use IPv6 addresses too, but you must set `ms bind ipv6` to `true`. See Network Configuration Reference for details about network configuration.

8. Create a keyring for your cluster and generate a monitor secret key.

```
ceph-authtool --create-keyring /tmp/ceph.mon.keyring --gen-key -n mon. --cap mon 'allow *'
```

9. Generate an administrator keyring, generate a `client.admin` user and add the user to the keyring.

```
ceph-authtool --create-keyring /etc/ceph/ceph.client.admin.keyring --gen-key -n client.admin --s
```

10. Add the `client.admin` key to the `ceph.mon.keyring`.

```
ceph-authtool /tmp/ceph.mon.keyring --import-keyring /etc/ceph/ceph.client.admin.keyring
```

11. Generate a monitor map using the hostname(s), host IP address(es) and the FSID. Save it as `/tmp/monmap`:

```
monmaptool --create --add {hostname} {ip-address} --fsid {uuid} /tmp/monmap
```

    For example:

```
monmaptool --create --add node1 192.168.0.1 --fsid a7f64266-0894-4f1e-a635-d0aeaca0e993 /tmp/mon
```

12. Create a default data directory (or directories) on the monitor host(s).

```
sudo mkdir /var/lib/ceph/mon/{cluster-name}-{hostname}
```

    For example:

```
sudo mkdir /var/lib/ceph/mon/ceph-node1
```

    See Monitor Config Reference - Data for details.

13. Populate the monitor daemon(s) with the monitor map and keyring.

```
ceph-mon [--cluster {cluster-name}] --mkfs -i {hostname} --monmap /tmp/monmap --keyring /tmp/cep
```

    For example:

```
ceph-mon --mkfs -i node1 --monmap /tmp/monmap --keyring /tmp/ceph.mon.keyring
```

14. Consider settings for a Ceph configuration file. Common settings include the following:

```
[global]
fsid = {cluster-id}
mon initial members = {hostname}[, {hostname}]
mon host = {ip-address}[, {ip-address}]
public network = {network}[, {network}]
cluster network = {network}[, {network}]
auth cluster required = cephx
auth service required = cephx
auth client required = cephx
osd journal size = {n}
filestore xattr use omap = true
osd pool default size = {n}  # Write an object n times.
osd pool default min size = {n} # Allow writing n copy in a degraded state.
osd pool default pg num = {n}
osd pool default pgp num = {n}
osd crush chooseleaf type = {n}
```

    In the foregoing example, the `[global]` section of the configuration might look like this:

```
[global]
fsid = a7f64266-0894-4f1e-a635-d0aeaca0e993
mon initial members = node1
mon host = 192.168.0.1
public network = 192.168.0.0/24
auth cluster required = cephx
auth service required = cephx
auth client required = cephx
```

```
    osd journal size = 1024
    filestore xattr use omap = true
    osd pool default size = 2
    osd pool default min size = 1
    osd pool default pg num = 333
    osd pool default pgp num = 333
    osd crush chooseleaf type = 1
```

15. Touch the `done` file.

    Mark that the monitor is created and ready to be started:

```
sudo touch /var/lib/ceph/mon/ceph-node1/done
```

16. Start the monitor(s).

    For Ubuntu, use Upstart:

```
sudo start ceph-mon id=node1 [cluster={cluster-name}]
```

    In this case, to allow the start of the daemon at each reboot you must create two empty files like this:

```
sudo touch /var/lib/ceph/mon/{cluster-name}-{hostname}/upstart
```

    For example:

```
sudo touch /var/lib/ceph/mon/ceph-node1/upstart
```

    For Debian/CentOS/RHEL, use sysvinit:

```
sudo /etc/init.d/ceph start mon.node1
```

17. Verify that Ceph created the default pools.

```
ceph osd lspools
```

    You should see output like this:

```
0 data,1 metadata,2 rbd,
```

18. Verify that the monitor is running.

```
ceph -s
```

    You should see output that the monitor you started is up and running, and you should see a health error indicating that placement groups are stuck inactive. It should look something like this:

```
cluster a7f64266-0894-4f1e-a635-d0aeaca0e993
  health HEALTH_ERR 192 pgs stuck inactive; 192 pgs stuck unclean; no osds
  monmap e1: 1 mons at {node1=192.168.0.1:6789/0}, election epoch 1, quorum 0 node1
  osdmap e1: 0 osds: 0 up, 0 in
  pgmap v2: 192 pgs, 3 pools, 0 bytes data, 0 objects
     0 kB used, 0 kB / 0 kB avail
     192 creating
```

    Note: Once you add OSDs and start them, the placement group health errors should disappear. See the next section for details.

Adding OSDs

Once you have your initial monitor(s) running, you should add OSDs. Your cluster cannot reach an `active + clean` state until you have enough OSDs to handle the number of copies of an object (e.g., `osd pool default size = 2` requires at least two OSDs). After bootstrapping your monitor, your cluster has a default CRUSH map; however, the CRUSH map doesn't have any Ceph OSD Daemons mapped to a Ceph Node.

Short Form

Ceph provides the `ceph-disk` utility, which can prepare a disk, partition or directory for use with Ceph. The `ceph-disk` utility creates the OSD ID by incrementing the index. Additionally, `ceph-disk` will add the new OSD to the CRUSH map under the host for you. Execute `ceph-disk -h` for CLI details. The `ceph-disk` utility automates the steps of the *Long Form* below. To create the first two OSDs with the short form procedure, execute the following on `node2` and `node3`:

1. Prepare the OSD.

```
ssh {node-name}
sudo ceph-disk prepare --cluster {cluster-name} --cluster-uuid {uuid} --fs-type {ext4|xfs|btrfs}
```

For example:

```
ssh node1
sudo ceph-disk prepare --cluster ceph --cluster-uuid a7f64266-0894-4f1e-a635-d0aeaca0e993 --fs-t
```

2. Activate the OSD:

```
sudo ceph-disk activate {data-path} [--activate-key {path}]
```

For example:

```
sudo ceph-disk activate /dev/hdd1
```

Note: Use the `--activate-key` argument if you do not have a copy of `/var/lib/ceph/bootstrap-osd/{cluster}.keyring` on the Ceph Node.

Long Form

Without the benefit of any helper utilities, creating an OSD and adding it to the cluster and CRUSH map the following procedure. To create the first two OSDs with the long form procedure, execute the following on `node2` and `node3`:

1. Connect to the OSD host.

```
ssh {node-name}
```

2. Generate a UUID for the OSD.

```
uuidgen
```

3. 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}]
```

4. Create the default directory on your new OSD.

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

5. 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/{hdd}
sudo mount -o user_xattr /dev/{hdd} /var/lib/ceph/osd/{cluster-name}-{osd-number}
```

6. Initialize the OSD data directory.

```
ssh {new-osd-host}
sudo ceph-osd -i {osd-num} --mkfs --mkkey --osd-uuid [{uuid}]
```

The directory must be empty before you can run `ceph-osd` with the `--mkkey` option. In addition, the ceph-osd tool requires specification of custom cluster names with the `--cluster` option.

7. 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.:

```
sudo ceph auth add osd.{osd-num} osd 'allow *' mon 'allow profile osd' -i /var/lib/ceph/osd/{clu
```

8. Add your Ceph Node to the CRUSH map.

```
ceph [--cluster {cluster-name}] osd crush add-bucket {hostname} host
```

For example:

```
ceph osd crush add-bucket node1 host
```

9. Place the Ceph Node under the root `default`.

```
ceph osd crush move node1 root=default
```

10. Add the OSD to the CRUSH map so that it can begin receiving data. 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.

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

For example:

```
ceph osd crush add osd.0 1.0 host=node1
```

11. 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 Ubuntu, use Upstart:

```
sudo start ceph-osd id={osd-num} [cluster={cluster-name}]
```

For example:

```
sudo start ceph-osd id=0
sudo start ceph-osd id=1
```

For Debian/CentOS/RHEL, use sysvinit:

```
sudo /etc/init.d/ceph start osd.{osd-num} [--cluster {cluster-name}]
```

For example:

```
    sudo /etc/init.d/ceph start osd.0
    sudo /etc/init.d/ceph start osd.1
```

In this case, to allow the start of the daemon at each reboot you must create an empty file like this:

```
    sudo touch /var/lib/ceph/osd/{cluster-name}-{osd-num}/sysvinit
```

For example:

```
    sudo touch /var/lib/ceph/osd/ceph-0/sysvinit
    sudo touch /var/lib/ceph/osd/ceph-1/sysvinit
```

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

Summary

Once you have your monitor and two OSDs up and running, you can watch the placement groups peer by executing the following:

```
ceph -w
```

To view the tree, execute the following:

```
ceph osd tree
```

You should see output that looks something like this:

```
# id    weight  type name       up/down reweight
-1      2       root default
-2      2               host node1
0       1                       osd.0   up      1
-3      1               host node2
1       1                       osd.1   up      1
```

To add (or remove) additional monitors, see Add/Remove Monitors. To add (or remove) additional Ceph OSD Daemons, see Add/Remove OSDs.

3.4 Upgrade Software

As new versions of Ceph become available, you may upgrade your cluster to take advantage of new functionality. Read the upgrade documentation before you upgrade your cluster. Sometimes upgrading Ceph requires you to follow an upgrade sequence.

3.4.1 Upgrading Ceph

Each release of Ceph may have additional steps. Refer to the release-specific sections in this document and the release notes document to identify release-specific procedures for your cluster before using the upgrade procedures.

Summary

You can upgrade daemons in your Ceph cluster while the cluster is online and in service! Certain types of daemons depend upon others. For example, Ceph Metadata Servers and Ceph Object Gateways depend upon Ceph Monitors and Ceph OSD Daemons. We recommend upgrading in this order:

1. *Ceph Deploy*
2. Ceph Monitors
3. Ceph OSD Daemons
4. Ceph Metadata Servers
5. Ceph Object Gateways

As a general rule, we recommend upgrading all the daemons of a specific type (e.g., all `ceph-mon` daemons, all `ceph-osd` daemons, etc.) to ensure that they are all on the same release. We also recommend that you upgrade all the daemons in your cluster before you try to exercise new functionality in a release.

The *Upgrade Procedures* are relatively simple, but please look at distribution-specific sections before upgrading. The basic process involves three steps:

1. Use `ceph-deploy` on your admin node to upgrade the packages for multiple hosts (using the `ceph-deploy install` command), or login to each host and upgrade the Ceph package manually. For example, when *Upgrading Monitors*, the `ceph-deploy` syntax might look like this:

```
    ceph-deploy install --release {release-name} ceph-node1[ ceph-node2]
    ceph-deploy install --release firefly mon1 mon2 mon3
```

   Note: The `ceph-deploy install` command will upgrade the packages in the specified node(s) from the old release to the release you specify. There is no `ceph-deploy upgrade` command.

2. Login in to each Ceph node and restart each Ceph daemon. See Operating a Cluster for details.
3. Ensure your cluster is healthy. See Monitoring a Cluster for details.

Important: Once you upgrade a daemon, you cannot downgrade it.

Ceph Deploy

Before upgrading Ceph daemons, upgrade the `ceph-deploy` tool.

```
sudo pip install -U ceph-deploy
```

Or:

```
sudo apt-get install ceph-deploy
```

Or:

```
sudo yum install ceph-deploy python-pushy
```

Argonaut to Bobtail

When upgrading from Argonaut to Bobtail, you need to be aware of several things:

1. Authentication now defaults to **ON**, but used to default to **OFF**.
2. Monitors use a new internal on-wire protocol.
3. RBD `format2` images require upgrading all OSDs before using it.

Ensure that you update package repository paths. For example:

```
sudo rm /etc/apt/sources.list.d/ceph.list
echo deb http://ceph.com/debian-bobtail/ $(lsb_release -sc) main | sudo tee /etc/apt/sources.list.d/c
```

See the following sections for additional details.

Authentication

The Ceph Bobtail release enables authentication by default. Bobtail also has finer-grained authentication configuration settings. In previous versions of Ceph (i.e., actually v 0.55 and earlier), you could simply specify:

```
auth supported = [cephx | none]
```

This option still works, but is deprecated. New releases support `cluster`, `service` and `client` authentication settings as follows:

```
auth cluster required = [cephx | none]  # default cephx
auth service required = [cephx | none] # default cephx
auth client required = [cephx | none] # default cephx,none
```

Important: If your cluster does not currently have an `auth supported` line that enables authentication, you must explicitly turn it off in Bobtail using the settings below.:

```
auth cluster required = none
auth service required = none
```

This will disable authentication on the cluster, but still leave clients with the default configuration where they can talk to a cluster that does enable it, but do not require it.

Important: If your cluster already has an `auth supported` option defined in the configuration file, no changes are necessary.

See Ceph Authentication - Backward Compatibility for details.

Monitor On-wire Protocol

We recommend upgrading all monitors to Bobtail. A mixture of Bobtail and Argonaut monitors will not be able to use the new on-wire protocol, as the protocol requires all monitors to be Bobtail or greater. Upgrading only a majority of the nodes (e.g., two out of three) may expose the cluster to a situation where a single additional failure may compromise availability (because the non-upgraded daemon cannot participate in the new protocol). We recommend not waiting for an extended period of time between `ceph-mon` upgrades.

RBD Images

The Bobtail release supports `format 2` images! However, you should not create or use `format 2` RBD images until after all `ceph-osd` daemons have been upgraded. Note that `format 1` is still the default. You can use the new `ceph osd ls` and `ceph tell osd.N version` commands to doublecheck your cluster. `ceph osd ls` will give a list of all OSD IDs that are part of the cluster, and you can use that to write a simple shell loop to display all the OSD version strings:

```
for i in $(ceph osd ls); do
    ceph tell osd.${i} version
done
```

Argonaut to Cuttlefish

To upgrade your cluster from Argonaut to Cuttlefish, please read this section, and the sections on upgrading from Argonaut to Bobtail and upgrading from Bobtail to Cuttlefish carefully. When upgrading from Argonaut to Cuttlefish, **YOU MUST UPGRADE YOUR MONITORS FROM ARGONAUT TO BOBTAIL v0.56.5 FIRST!!!**. All other Ceph daemons can upgrade from Argonaut to Cuttlefish without the intermediate upgrade to Bobtail.

Important: Ensure that the repository specified points to Bobtail, not Cuttlefish.

For example:

```
sudo rm /etc/apt/sources.list.d/ceph.list
echo deb http://ceph.com/debian-bobtail/ $(lsb_release -sc) main | sudo tee /etc/apt/sources.list.d/c
```

We recommend upgrading all monitors to Bobtail before proceeding with the upgrade of the monitors to Cuttlefish. A mixture of Bobtail and Argonaut monitors will not be able to use the new on-wire protocol, as the protocol requires all monitors to be Bobtail or greater. Upgrading only a majority of the nodes (e.g., two out of three) may expose the cluster to a situation where a single additional failure may compromise availability (because the non-upgraded daemon cannot participate in the new protocol). We recommend not waiting for an extended period of time between `ceph-mon` upgrades. See *Upgrading Monitors* for details.

Note: See the *Authentication* section and the Ceph Authentication - Backward Compatibility for additional information on authentication backward compatibility settings for Bobtail.

Once you complete the upgrade of your monitors from Argonaut to Bobtail, and have restarted the monitor daemons, you must upgrade the monitors from Bobtail to Cuttlefish. Ensure that you have a quorum before beginning this upgrade procedure. Before upgrading, remember to replace the reference to the Bobtail repository with a reference to the Cuttlefish repository. For example:

```
sudo rm /etc/apt/sources.list.d/ceph.list
echo deb http://ceph.com/debian-cuttlefish/ $(lsb_release -sc) main | sudo tee /etc/apt/sources.list
```

See *Upgrading Monitors* for details.

The architecture of the monitors changed significantly from Argonaut to Cuttlefish. See Monitor Config Reference and Joao's blog post for details. Once you complete the monitor upgrade, you can upgrade the OSD daemons and the MDS daemons using the generic procedures. See *Upgrading an OSD* and *Upgrading a Metadata Server* for details.

Bobtail to Cuttlefish

Upgrading your cluster from Bobtail to Cuttlefish has a few important considerations. First, the monitor uses a new architecture, so you should upgrade the full set of monitors to use Cuttlefish. Second, if you run multiple metadata servers in a cluster, ensure the metadata servers have unique names. See the following sections for details.

Replace any `apt` reference to older repositories with a reference to the Cuttlefish repository. For example:

```
sudo rm /etc/apt/sources.list.d/ceph.list
echo deb http://ceph.com/debian-cuttlefish/ $(lsb_release -sc) main | sudo tee /etc/apt/sources.list
```

Monitor

The architecture of the monitors changed significantly from Bobtail to Cuttlefish. See Monitor Config Reference and Joao's blog post for details. This means that v0.59 and pre-v0.59 monitors do not talk to each other (Cuttlefish is v.0.61). When you upgrade each monitor, it will convert its local data store to the new format. Once you upgrade a majority of monitors, the monitors form a quorum using the new protocol and the old monitors will be blocked until they get upgraded. For this reason, we recommend upgrading the monitors in immediate succession.

Important: Do not run a mixed-version cluster for an extended period.

MDS Unique Names

The monitor now enforces that MDS names be unique. If you have multiple metadata server daemons that start with the same ID (e.g., mds.a) the second metadata server will implicitly mark the first metadata server as `failed`. Multi-MDS configurations with identical names must be adjusted accordingly to give daemons unique names. If you run your cluster with one metadata server, you can disregard this notice for now.

ceph-deploy

The `ceph-deploy` tool is now the preferred method of provisioning new clusters. For existing clusters created via the obsolete `mkcephfs` tool that would like to transition to the new tool, there is a migration path, documented at *Transitioning to ceph-deploy*.

Cuttlefish to Dumpling

When upgrading from Cuttlefish (v0.61-v0.61.7) you may perform a rolling upgrade. However, there are a few important considerations. First, you must upgrade the `ceph` command line utility, because it has changed significantly. Second, you must upgrade the full set of monitors to use Dumpling, because of a protocol change.

Replace any reference to older repositories with a reference to the Dumpling repository. For example, with `apt` perform the following:

```
sudo rm /etc/apt/sources.list.d/ceph.list
echo deb http://ceph.com/debian-dumpling/ $(lsb_release -sc) main | sudo tee /etc/apt/sources.list.d/
```

With CentOS/Red Hat distributions, remove the old repository.

```
sudo rm /etc/yum.repos.d/ceph.repo
```

Then add a new `ceph.repo` repository entry with the following contents.

```
[ceph]
name=Ceph Packages and Backports $basearch
baseurl=http://ceph.com/rpm/el6/$basearch
enabled=1
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc
```

Note: Ensure you use the correct URL for your distribution. Check the http://ceph.com/rpm directory for your distribution.

Note: Since you can upgrade using `ceph-deploy` you will only need to add the repository on Ceph Client nodes where you use the `ceph` command line interface or the `ceph-deploy` tool.

Dumpling to Emperor

When upgrading from Dumpling (v0.64) you may perform a rolling upgrade.

Replace any reference to older repositories with a reference to the Emperor repository. For example, with `apt` perform the following:

```
sudo rm /etc/apt/sources.list.d/ceph.list
echo deb http://ceph.com/debian-emperor/ $(lsb_release -sc) main | sudo tee /etc/apt/sources.list.d/c
```

With CentOS/Red Hat distributions, remove the old repository.

```
sudo rm /etc/yum.repos.d/ceph.repo
```

Then add a new `ceph.repo` repository entry with the following contents and replace `{distro}` with your distribution (e.g., `el6`, `rhel6`, etc).

```
[ceph]
name=Ceph Packages and Backports $basearch
baseurl=http://ceph.com/rpm-emperor/{distro}/$basearch
enabled=1
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc
```

Note: Ensure you use the correct URL for your distribution. Check the http://ceph.com/rpm directory for your distribution.

Note: Since you can upgrade using `ceph-deploy` you will only need to add the repository on Ceph Client nodes where you use the `ceph` command line interface or the `ceph-deploy` tool.

Command Line Utility

In V0.65, the `ceph` commandline interface (CLI) utility changed significantly. You will not be able to use the old CLI with Dumpling. This means that you must upgrade the `ceph-common` library on all nodes that access the Ceph Storage Cluster with the `ceph` CLI before upgrading Ceph daemons.

```
sudo apt-get update && sudo apt-get install ceph-common
```

Ensure that you have the latest version (v0.67 or later). If you do not, you may need to uninstall, auto remove dependencies and reinstall.

See v0.65 for details on the new command line interface.

Monitor

Dumpling (v0.67) `ceph-mon` daemons have an internal protocol change. This means that v0.67 daemons cannot talk to v0.66 or older daemons. Once you upgrade a majority of monitors, the monitors form a quorum using the new protocol and the old monitors will be blocked until they get upgraded. For this reason, we recommend upgrading all monitors at once (or in relatively quick succession) to minimize the possibility of downtime.

Important: Do not run a mixed-version cluster for an extended period.

Dumpling to Firefly

If your existing cluster is running a version older than v0.67 Dumpling, please first upgrade to the latest Dumpling release before upgrading to v0.80 Firefly.

Monitor

Dumpling (v0.67) `ceph-mon` daemons have an internal protocol change. This means that v0.67 daemons cannot talk to v0.66 or older daemons. Once you upgrade a majority of monitors, the monitors form a quorum using the new protocol and the old monitors will be blocked until they get upgraded. For this reason, we recommend upgrading all monitors at once (or in relatively quick succession) to minimize the possibility of downtime.

Important: Do not run a mixed-version cluster for an extended period.

Ceph Config File Changes

We recommand adding the following to the `[mon]` section of your `ceph.conf` prior to upgrade:

```
mon warn on legacy crush tunables = false
```

This will prevent health warnings due to the use of legacy CRUSH placement. Although it is possible to rebalance existing data across your cluster, we do not normally recommend it for production environments as a large amount of data will move and there is a significant performance impact from the rebalancing.

Command Line Utility

In V0.65, the `ceph` commandline interface (CLI) utility changed significantly. You will not be able to use the old CLI with Firefly. This means that you must upgrade the `ceph-common` library on all nodes that access the Ceph Storage Cluster with the `ceph` CLI before upgrading Ceph daemons.

For Debian/Ubuntu, execute:

```
sudo apt-get update && sudo apt-get install ceph-common
```

For CentOS/RHEL, execute:

```
sudo yum install ceph-common
```

Ensure that you have the latest version. If you do not, you may need to uninstall, auto remove dependencies and reinstall.

See v0.65 for details on the new command line interface.

Upgrade Sequence

Replace any reference to older repositories with a reference to the Firely repository. For example, with `apt` perform the following:

```
sudo rm /etc/apt/sources.list.d/ceph.list
echo deb http://ceph.com/debian-firefly/ $(lsb_release -sc) main | sudo tee /etc/apt/sources.list.d/c
```

With CentOS/Red Hat distributions, remove the old repository.

```
sudo rm /etc/yum.repos.d/ceph.repo
```

Then add a new `ceph.repo` repository entry with the following contents and replace `{distro}` with your distribution (e.g., `el6`, `rhel6`, `rhel7`, etc.).

```
[ceph]
name=Ceph Packages and Backports $basearch
baseurl=http://ceph.com/rpm-firefly/{distro}/$basearch
enabled=1
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc
```

Upgrade daemons in the following order:

1. **Monitors:** If the `ceph-mon` daemons are not restarted prior to the `ceph-osd` daemons, the monitors will not correctly register their new capabilities with the cluster and new features may not be usable until the monitors are restarted a second time.
2. **OSDs**
3. **MDSs:** If the `ceph-mds` daemon is restarted first, it will wait until all OSDs have been upgraded before finishing its startup sequence.
4. **Gateways:** Upgrade `radosgw` daemons together. There is a subtle change in behavior for multipart uploads that prevents a multipart request that was initiated with a new `radosgw` from being completed by an old `radosgw`.

Note: Make sure you upgrade your **ALL** of your Ceph monitors **AND** restart them **BEFORE** upgrading and restarting OSDs, MDSs, and gateways!

Emperor to Firefly

If your existing cluster is running a version older than v0.67 Dumpling, please first upgrade to the latest Dumpling release before upgrading to v0.80 Firefly. Please refer to *Cuttlefish to Dumpling* and the Firefly release notes for details. To upgrade from a post-Emperor point release, see the Firefly release notes for details.

Ceph Config File Changes

We recommand adding the following to the `[mon]` section of your `ceph.conf` prior to upgrade:

```
mon warn on legacy crush tunables = false
```

This will prevent health warnings due to the use of legacy CRUSH placement. Although it is possible to rebalance existing data across your cluster, we do not normally recommend it for production environments as a large amount of data will move and there is a significant performance impact from the rebalancing.

Upgrade Sequence

Replace any reference to older repositories with a reference to the Firefly repository. For example, with `apt` perform the following:

```
sudo rm /etc/apt/sources.list.d/ceph.list
echo deb http://ceph.com/debian-firefly/ $(lsb_release -sc) main | sudo tee /etc/apt/sources.list.d/c
```

With CentOS/Red Hat distributions, remove the old repository.

```
sudo rm /etc/yum.repos.d/ceph.repo
```

Then add a new `ceph.repo` repository entry with the following contents, but replace `{distro}` with your distribution (e.g., `el6`, `rhel6`, `rhel7`, etc.).

```
[ceph]
name=Ceph Packages and Backports $basearch
baseurl=http://ceph.com/rpm/{distro}/$basearch
enabled=1
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc
```

Note: Ensure you use the correct URL for your distribution. Check the http://ceph.com/rpm directory for your distribution.

Note: Since you can upgrade using `ceph-deploy` you will only need to add the repository on Ceph Client nodes where you use the `ceph` command line interface or the `ceph-deploy` tool.

Upgrade daemons in the following order:

1. **Monitors:** If the `ceph-mon` daemons are not restarted prior to the `ceph-osd` daemons, the monitors will not correctly register their new capabilities with the cluster and new features may not be usable until the monitors are restarted a second time.
2. **OSDs**
3. **MDSs:** If the `ceph-mds` daemon is restarted first, it will wait until all OSDs have been upgraded before finishing its startup sequence.
4. **Gateways:** Upgrade `radosgw` daemons together. There is a subtle change in behavior for multipart uploads that prevents a multipart request that was initiated with a new `radosgw` from being completed by an old `radosgw`.

Upgrade Procedures

The following sections describe the upgrade process.

Important: Each release of Ceph may have some additional steps. Refer to release-specific sections for details **BEFORE** you begin upgrading daemons.

Upgrading Monitors

To upgrade monitors, perform the following steps:

1. Upgrade the Ceph package for each daemon instance.

   You may use `ceph-deploy` to address all monitor nodes at once. For example:

```
    ceph-deploy install --release {release-name} ceph-node1[ ceph-node2]
    ceph-deploy install --release dumpling mon1 mon2 mon3
```

   You may also use the package manager for your Linux distribution on each individual node. To upgrade packages manually on each Debian/Ubuntu host, perform the following steps .

```
ssh {mon-host}
sudo apt-get update && sudo apt-get install ceph
```

On CentOS/Red Hat hosts, perform the following steps:

```
ssh {mon-host}
sudo yum update && sudo yum install ceph
```

2. Restart each monitor. For Ubuntu distributions, use:

```
sudo restart ceph-mon id={hostname}
```

For CentOS/Red Hat/Debian distributions, use:

```
sudo /etc/init.d/ceph restart {mon-id}
```

For CentOS/Red Hat distributions deployed with `ceph-deploy`, the monitor ID is usually `mon.{hostname}`.

3. Ensure each monitor has rejoined the quorum.

```
ceph mon stat
```

Ensure that you have completed the upgrade cycle for all of your Ceph Monitors.

Upgrading an OSD

To upgrade a Ceph OSD Daemon, perform the following steps:

1. Upgrade the Ceph OSD Daemon package.

   You may use `ceph-deploy` to address all Ceph OSD Daemon nodes at once. For example:

```
ceph-deploy install --release {release-name} ceph-node1[ ceph-node2]
ceph-deploy install --release dumpling mon1 mon2 mon3
```

You may also use the package manager on each node to upgrade packages manually. For Debian/Ubuntu hosts, perform the following steps on each host.

```
ssh {osd-host}
sudo apt-get update && sudo apt-get install ceph
```

For CentOS/Red Hat hosts, perform the following steps:

```
ssh {osd-host}
sudo yum update && sudo yum install ceph
```

2. Restart the OSD, where `N` is the OSD number. For Ubuntu, use:

```
sudo restart ceph-osd id=N
```

For multiple OSDs on a host, you may restart all of them with Upstart.

```
sudo restart ceph-osd-all
```

For CentOS/Red Hat/Debian distributions, use:

```
sudo /etc/init.d/ceph restart N
```

3. Ensure each upgraded Ceph OSD Daemon has rejoined the cluster:

```
ceph osd stat
```

Ensure that you have completed the upgrade cycle for all of your Ceph OSD Daemons.

Upgrading a Metadata Server

To upgrade a Ceph Metadata Server, perform the following steps:

1. Upgrade the Ceph Metadata Server package. You may use `ceph-deploy` to address all Ceph Metadata Server nodes at once, or use the package manager on each node. For example:

```
ceph-deploy install --release {release-name} ceph-node1[ ceph-node2]
ceph-deploy install --release dumpling mon1 mon2 mon3
```

   To upgrade packages manually, perform the following steps on each Debian/Ubuntu host.

```
ssh {mon-host}
sudo apt-get update && sudo apt-get install ceph-mds
```

   Or the following steps on CentOS/Red Hat hosts:

```
ssh {mon-host}
sudo yum update && sudo yum install ceph-mds
```

2. Restart the metadata server. For Ubuntu, use:

```
sudo restart ceph-mds id={hostname}
```

   For CentOS/Red Hat/Debian distributions, use:

```
sudo /etc/init.d/ceph restart mds.{hostname}
```

   For clusters deployed with `ceph-deploy`, the name is usually either the name you specified on creation or the hostname.

3. Ensure the metadata server is up and running:

```
ceph mds stat
```

Upgrading a Client

Once you have upgraded the packages and restarted daemons on your Ceph cluster, we recommend upgrading `ceph-common` and client libraries (`librbd1` and `librados2`) on your client nodes too.

1. Upgrade the package:

```
ssh {client-host}
apt-get update && sudo apt-get install ceph-common librados2 librbd1 python-rados python-rbd
```

2. Ensure that you have the latest version:

```
ceph --version
```

If you do not have the latest version, you may need to uninstall, auto remove dependencies and reinstall.

Transitioning to ceph-deploy

If you have an existing cluster that you deployed with `mkcephfs` (usually Argonaut or Bobtail releases), you will need to make a few changes to your configuration to ensure that your cluster will work with `ceph-deploy`.

Monitor Keyring

You will need to add `caps mon = "allow *"` to your monitor keyring if it is not already in the keyring. By default, the monitor keyring is located under `/var/lib/ceph/mon/ceph-$id/keyring`. When you have added the `caps` setting, your monitor keyring should look something like this:

```
[mon.]
        key = AQBJIHhRuHCwDRAAZjBTSJcIBIoGpdOR9ToiyQ==
        caps mon = "allow *"
```

Adding `caps mon = "allow *"` will ease the transition from `mkcephfs` to `ceph-deploy` by allowing `ceph-create-keys` to use the `mon.` keyring file in `$mon_data` and get the caps it needs.

Use Default Paths

Under the `/var/lib/ceph` directory, the `mon` and `osd` directories need to use the default paths.

- **OSDs**: The path should be `/var/lib/ceph/osd/ceph-$id`
- **MON**: The path should be `/var/lib/ceph/mon/ceph-$id`

Under those directories, the keyring should be in a file named `keyring`.

CHAPTER

FOUR

CEPH STORAGE CLUSTER

The *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 Daemon* (OSD) stores data as objects on a storage node; and a *Ceph Monitor* (MON) maintains a master copy of the cluster map. A Ceph Storage Cluster may contain thousands of storage nodes. A minimal system will have at least one Ceph Monitor and two Ceph OSD Daemons for data replication.

The Ceph Filesystem, Ceph Object Storage and Ceph Block Devices read data from and write data to the Ceph Storage Cluster.

Ceph Storage Clusters have a few required settings, but most configuration settings have default values. A typical deployment uses a deployment tool to define a cluster and bootstrap a monitor. See Deployment for details on `ceph-deploy`.

4.1 Configuration

Ceph can run with a cluster containing thousands of Object Storage Devices (OSDs). A minimal system will have at least two OSDs for data replication. To configure OSD clusters, you must provide settings in the configuration file. Ceph provides default values for many settings, which you can override in the configuration file. Additionally, you can make runtime modification to the configuration using command-line utilities.

When Ceph starts, it activates three daemons:

- `ceph-mon` (mandatory)
- `ceph-osd` (mandatory)
- `ceph-mds` (mandatory for cephfs only)

Each process, daemon or utility loads the host's configuration file. A process may have information about more than one daemon instance (*i.e.,* multiple contexts). A daemon or utility only has information about a single daemon instance (a single context).

Note: Ceph can run on a single host for evaluation purposes.

For general object store configuration, refer to the following:

4.1.1 Hard Disk and File System Recommendations

Hard Drive Prep

Ceph aims for data safety, which means that when the *Ceph Client* receives notice that data was written to a storage drive, that data was actually written to the storage drive. For old kernels (<2.6.33), disable the write cache if the journal is on a raw drive. Newer kernels should work fine.

Use `hdparm` to disable write caching on the hard disk:

```
sudo hdparm -W 0 /dev/hda 0
```

In production environments, we recommend running a *Ceph OSD Daemon* with separate drives for the operating system and the data. If you run data and an operating system on a single disk, we recommend creating a separate partition for your data.

Filesystems

Ceph OSD Daemons rely heavily upon the stability and performance of the underlying filesystem.

Note: We currently recommend `XFS` for production deployments. We recommend `btrfs` for testing, development, and any non-critical deployments. We believe that `btrfs` has the correct feature set and roadmap to serve Ceph in the long-term, but `XFS` and `ext4` provide the necessary stability for today's deployments. `btrfs` development is proceeding rapidly: users should be comfortable installing the latest released upstream kernels and be able to track development activity for critical bug fixes.

Ceph OSD Daemons depend on the Extended Attributes (XATTRs) of the underlying file system for various forms of internal object state and metadata. The underlying filesystem must provide sufficient capacity for XATTRs. `btrfs` does not bound the total xattr metadata stored with a file. `XFS` has a relatively large limit (64 KB) that most deployments won't encounter, but the `ext4` is too small to be usable.

You should always add the following line to the `[osd]` section of your `ceph.conf` file for `ext4` filesystems; you can optionally use it for `btrfs` and `XFS`.:

```
filestore xattr use omap = true
```

Filesystem Background Info

The `XFS`, `btrfs` and `ext4` file systems provide numerous advantages in highly scaled data storage environments when compared to `ext3`.

`XFS`, `btrfs` and `ext4` are journaling file systems, which means that they are more robust when recovering from crashes, power outages, etc. These filesystems journal all of the changes they will make before performing writes.

`XFS` was developed for Silicon Graphics, and is a mature and stable filesystem. By contrast, `btrfs` is a relatively new file system that aims to address the long-standing wishes of system administrators working with large scale data storage environments. `btrfs` has some unique features and advantages compared to other Linux filesystems.

`btrfs` is a copy-on-write filesystem. It supports file creation timestamps and checksums that verify metadata integrity, so it can detect bad copies of data and fix them with the good copies. The copy-on-write capability means that `btrfs` can support snapshots that are writable. `btrfs` supports transparent compression and other features.

`btrfs` also incorporates multi-device management into the file system, which enables you to support heterogeneous disk storage infrastructure, data allocation policies. The community also aims to provide `fsck`, deduplication, and data encryption support in the future. This compelling list of features makes `btrfs` the ideal choice for Ceph clusters.

4.1.2 Configuring Ceph

When you start the Ceph service, the initialization process activates a series of daemons that run in the background. A *Ceph Storage Cluster* runs two types of daemons:

- *Ceph Monitor* (`ceph-mon`)

- *Ceph OSD Daemon* (`ceph-osd`)

Ceph Storage Clusters that support the *Ceph Filesystem* run at least one *Ceph Metadata Server* (`ceph-mds`). Clusters that support *Ceph Object Storage* run Ceph Gateway daemons (`radosgw`). For your convenience, each daemon has a series of default values (*i.e.*, many are set by `ceph/src/common/config_opts.h`). You may override these settings with a Ceph configuration file.

The Configuration File

When you start a Ceph Storage Cluster, each daemon looks for a Ceph configuration file (i.e., `ceph.conf` by default) that provides the cluster's configuration settings. For manual deployments, you need to create a Ceph configuration file. For tools that create configuration files for you (*e.g.*, `ceph-deploy`, Chef, etc.), you may use the information contained herein as a reference. The Ceph configuration file defines:

- Cluster Identity
- Authentication settings
- Cluster membership
- Host names
- Host addresses
- Paths to keyrings
- Paths to journals
- Paths to data
- Other runtime options

The default Ceph configuration file locations in sequential order include:

1. `$CEPH_CONF` (*i.e.,* the path following the `$CEPH_CONF` environment variable)
2. `-c path/path` (*i.e.,* the `-c` command line argument)
3. `/etc/ceph/ceph.conf`
4. `~/.ceph/config`
5. `./ceph.conf` (*i.e.,* in the current working directory)

The Ceph configuration file uses an *ini* style syntax. You can add comments by preceding comments with a pound sign (#) or a semi-colon (;). For example:

```
# <--A number (#) sign precedes a comment.
; A comment may be anything.
# Comments always follow a semi-colon (;) or a pound (#) on each line.
# The end of the line terminates a comment.
# We recommend that you provide comments in your configuration file(s).
```

Config Sections

The configuration file can configure all Ceph daemons in a Ceph Storage Cluster, or all Ceph daemons of a particular type. To configure a series of daemons, the settings must be included under the processes that will receive the configuration as follows:

`[global]`

Description Settings under `[global]` affect all daemons in a Ceph Storage Cluster.

Example `auth supported = cephx`

`[osd]`

Description Settings under `[osd]` affect all `ceph-osd` daemons in the Ceph Storage Cluster, and override the same setting in `[global]`.

Example `osd journal size = 1000`

`[mon]`

Description Settings under `[mon]` affect all `ceph-mon` daemons in the Ceph Storage Cluster, and override the same setting in `[global]`.

Example `mon addr = 10.0.0.101:6789`

`[mds]`

Description Settings under `[mds]` affect all `ceph-mds` daemons in the Ceph Storage Cluster, and override the same setting in `[global]`.

Example `host = myserver01`

`[client]`

Description Settings under `[client]` affect all Ceph Clients (e.g., mounted Ceph Filesystems, mounted Ceph Block Devices, etc.).

Example `log file = /var/log/ceph/radosgw.log`

Global settings affect all instances of all daemon in the Ceph Storage Cluster. Use the `[global]` setting for values that are common for all daemons in the Ceph Storage Cluster. You can override each `[global]` setting by:

1. Changing the setting in a particular process type (*e.g.,* `[osd]`, `[mon]`, `[mds]`).
2. Changing the setting in a particular process (*e.g.,* `[osd.1]`).

Overriding a global setting affects all child processes, except those that you specifically override in a particular daemon.

A typical global setting involves activating authentication. For example:

```
[global]
#Enable authentication between hosts within the cluster.
#v 0.54 and earlier
auth supported = cephx

#v 0.55 and after
auth cluster required = cephx
auth service required = cephx
auth client required = cephx
```

You can specify settings that apply to a particular type of daemon. When you specify settings under `[osd]`, `[mon]` or `[mds]` without specifying a particular instance, the setting will apply to all OSDs, monitors or metadata daemons respectively.

A typical daemon-wide setting involves setting journal sizes, filestore settings, etc. For example:

```
[osd]
osd journal size = 1000
```

You may specify settings for particular instances of a daemon. You may specify an instance by entering its type, delimited by a period (.) and by the instance ID. The instance ID for a Ceph OSD Daemon is always numeric, but it may be alphanumeric for Ceph Monitors and Ceph Metadata Servers.

```
[osd.1]
# settings affect osd.1 only.

[mon.a]
# settings affect mon.a only.

[mds.b]
# settings affect mds.b only.
```

If the daemon you specify is a Ceph Gateway client, specify the daemon and the instance, delimited by a period (.). For example:

```
[client.radosgw.instance-name]
# settings affect client.radosgw.instance-name only.
```

Metavariables

Metavariables simplify Ceph Storage Cluster configuration dramatically. When a metavariable is set in a configuration value, Ceph expands the metavariable into a concrete value. Metavariables are very powerful when used within the `[global]`, `[osd]`, `[mon]`, `[mds]` or `[client]` sections of your configuration file. Ceph metavariables are similar to Bash shell expansion.

Ceph supports the following metavariables:

`$cluster`

Description Expands to the Ceph Storage Cluster name. Useful when running multiple Ceph Storage Clusters on the same hardware.

Example `/etc/ceph/$cluster.keyring`

Default `ceph`

`$type`

Description Expands to one of `mds`, `osd`, or `mon`, depending on the type of the instant daemon.

Example `/var/lib/ceph/$type`

`$id`

Description Expands to the daemon identifier. For `osd.0`, this would be `0`; for `mds.a`, it would be `a`.

Example `/var/lib/ceph/$type/$cluster-$id`

`$host`

Description Expands to the host name of the instant daemon.

`$name`

Description Expands to `$type.$id`.

Example `/var/run/ceph/$cluster-$name.asok`

Common Settings

The Hardware Recommendations section provides some hardware guidelines for configuring a Ceph Storage Cluster. It is possible for a single *Ceph Node* to run multiple daemons. For example, a single node with multiple drives may run one `ceph-osd` for each drive. Ideally, you will have a node for a particular type of process. For example, some nodes may run `ceph-osd` daemons, other nodes may run `ceph-mds` daemons, and still other nodes may run `ceph-mon` daemons.

Each node has a name identified by the `host` setting. Monitors also specify a network address and port (i.e., domain name or IP address) identified by the `addr` setting. A basic configuration file will typically specify only minimal settings for each instance of monitor daemons. For example:

```
[global]
mon_initial_members = ceph1
mon_host = 10.0.0.1
```

Important: The `host` setting is the short name of the node (i.e., not an fqdn). It is **NOT** an IP address either. Enter `hostname -s` on the command line to retrieve the name of the node. Do not use `host` settings for anything other than initial monitors unless you are deploying Ceph manually. You **MUST NOT** specify `host` under individual daemons when using deployment tools like `chef` or `ceph-deploy`, as those tools will enter the appropriate values for you in the cluster map.

Networks

See the Network Configuration Reference for a detailed discussion about configuring a network for use with Ceph.

Monitors

Ceph production clusters typically deploy with a minimum 3 *Ceph Monitor* daemons to ensure high availability should a monitor instance crash. At least three (3) monitors ensures that the Paxos algorithm can determine which version of the *Ceph Cluster Map* is the most recent from a majority of Ceph Monitors in the quorum.

Note: You may deploy Ceph with a single monitor, but if the instance fails, the lack of other monitors may interrupt data service availability.

Ceph Monitors typically listen on port `6789`. For example:

```
[mon.a]
host = hostName
mon addr = 150.140.130.120:6789
```

By default, Ceph expects that you will store a monitor's data under the following path:

```
/var/lib/ceph/mon/$cluster-$id
```

You or a deployment tool (e.g., `ceph-deploy`) must create the corresponding directory. With metavariables fully expressed and a cluster named "ceph", the foregoing directory would evaluate to:

```
/var/lib/ceph/mon/ceph-a
```

For additional details, see the Monitor Config Reference.

Authentication

New in version Bobtail: 0.56

For Bobtail (v 0.56) and beyond, you should expressly enable or disable authentication in the `[global]` section of your Ceph configuration file.

```
auth cluster required = cephx
auth service required = cephx
auth client required = cephx
```

Additionally, you should enable message signing. See Cephx Config Reference and Cephx Authentication for details.

Important: When upgrading, we recommend expressly disabling authentication first, then perform the upgrade. Once the upgrade is complete, re-enable authentication.

OSDs

Ceph production clusters typically deploy Ceph OSD Daemons where one node has one OSD daemon running a filestore on one storage drive. A typical deployment specifies a journal size. For example:

```
[osd]
osd journal size = 10000

[osd.0]
host = {hostname} #manual deployments only.
```

By default, Ceph expects that you will store a Ceph OSD Daemon's data with the following path:

```
/var/lib/ceph/osd/$cluster-$id
```

You or a deployment tool (e.g., `ceph-deploy`) must create the corresponding directory. With metavariables fully expressed and a cluster named "ceph", the foregoing directory would evaluate to:

```
/var/lib/ceph/osd/ceph-0
```

You may override this path using the `osd data` setting. We don't recommend changing the default location. Create the default directory on your OSD host.

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

The `osd data` path ideally leads to a mount point with a hard disk that is separate from the hard disk storing and running the operating system and daemons. If the OSD is for a disk other than the OS disk, prepare it for use with Ceph, and mount it to the directory you just created:

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

We recommend using the `xfs` file system or the `btrfs` file system when running **`mkfs`**.

See the OSD Config Reference for additional configuration details.

Heartbeats

During runtime operations, Ceph OSD Daemons check up on other Ceph OSD Daemons and report their findings to the Ceph Monitor. You do not have to provide any settings. However, if you have network latency issues, you may wish to modify the settings.

See Configuring Monitor/OSD Interaction for additional details.

Logs / Debugging

Sometimes you may encounter issues with Ceph that require modifying logging output and using Ceph's debugging. See Debugging and Logging for details on log rotation.

Example ceph.conf

```
[global]
fsid = {cluster-id}
mon initial members = {hostname}[, {hostname}]
mon host = {ip-address}[, {ip-address}]

#All clusters have a front-side public network.
#If you have two NICs, you can configure a back side cluster
#network for OSD object replication, heart beats, backfilling,
#recovery, etc.
public network = {network}[, {network}]
#cluster network = {network}[, {network}]

#Clusters require authentication by default.
auth cluster required = cephx
auth service required = cephx
auth client required = cephx

#Choose reasonable numbers for your journals, number of replicas
#and placement groups.
osd journal size = {n}
osd pool default size = {n}  # Write an object n times.
osd pool default min size = {n} # Allow writing n copy in a degraded state.
osd pool default pg num = {n}
osd pool default pgp num = {n}

#Choose a reasonable crush leaf type.
#0 for a 1-node cluster.
#1 for a multi node cluster in a single rack
#2 for a multi node, multi chassis cluster with multiple hosts in a chassis
#3 for a multi node cluster with hosts across racks, etc.
osd crush chooseleaf type = {n}
```

Runtime Changes

Ceph allows you to make changes to the configuration of a `ceph-osd`, `ceph-mon`, or `ceph-mds` daemon at runtime. This capability is quite useful for increasing/decreasing logging output, enabling/disabling debug settings, and even for runtime optimization. The following reflects runtime configuration usage:

```
ceph tell {daemon-type}.{id or *} injectargs --{name} {value} [--{name} {value}]
```

Replace `{daemon-type}` with one of `osd`, `mon` or `mds`. You may apply the runtime setting to all daemons of a particular type with `*`, or specify a specific daemon's ID (i.e., its number or letter). For example, to increase debug logging for a `ceph-osd` daemon named `osd.0`, execute the following:

```
ceph tell osd.0 injectargs --debug-osd 20 --debug-ms 1
```

In your `ceph.conf` file, you may use spaces when specifying a setting name. When specifying a setting name on the command line, ensure that you use an underscore or hyphen (_ or -) between terms (e.g., `debug osd` becomes `--debug-osd`).

Viewing a Configuration at Runtime

If your Ceph Storage Cluster is running, and you would like to see the configuration settings from a running daemon, execute the following:

```
ceph daemon {daemon-type}.{id} config show | less
```

If you are on a machine where osd.0 is running, the command would be:

```
ceph daemon osd.0 config show | less
```

Running Multiple Clusters

With Ceph, you can run multiple Ceph Storage Clusters on the same hardware. Running multiple clusters provides a higher level of isolation compared to using different pools on the same cluster with different CRUSH rulesets. A separate cluster will have separate monitor, OSD and metadata server processes. When running Ceph with default settings, the default cluster name is `ceph`, which means you would save your Ceph configuration file with the file name `ceph.conf` in the `/etc/ceph` default directory.

See ceph-deploy new for details. .. _ceph-deploy new:../ceph-deploy-new

When you run multiple clusters, you must name your cluster and save the Ceph configuration file with the name of the cluster. For example, a cluster named `openstack` will have a Ceph configuration file with the file name `openstack.conf` in the `/etc/ceph` default directory.

Important: Cluster names must consist of letters a-z and digits 0-9 only.

Separate clusters imply separate data disks and journals, which are not shared between clusters. Referring to *Metavariables*, the `$cluster` metavariable evaluates to the cluster name (i.e., `openstack` in the foregoing example). Various settings use the `$cluster` metavariable, including:

- `keyring`
- `admin socket`
- `log file`
- `pid file`
- `mon data`
- `mon cluster log file`
- `osd data`
- `osd journal`
- `mds data`
- `rgw data`

See General Settings, OSD Settings, Monitor Settings, MDS Settings, RGW Settings and Log Settings for relevant path defaults that use the `$cluster` metavariable.

When creating default directories or files, you should use the cluster name at the appropriate places in the path. For example:

```
sudo mkdir /var/lib/ceph/osd/openstack-0
sudo mkdir /var/lib/ceph/mon/openstack-a
```

Important: When running monitors on the same host, you should use different ports. By default, monitors use port 6789. If you already have monitors using port 6789, use a different port for your other cluster(s).

To invoke a cluster other than the default `ceph` cluster, use the `-c {filename}.conf` option with the `ceph` command. For example:

```
ceph -c {cluster-name}.conf health
ceph -c openstack.conf health
```

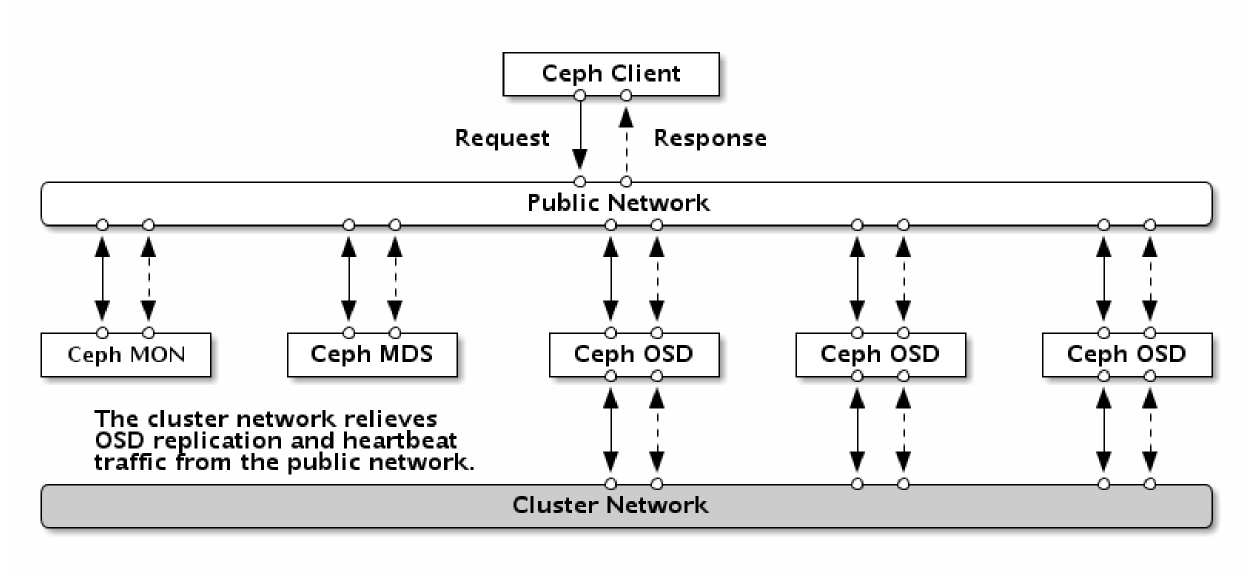
To optimize the performance of your cluster, refer to the following:

4.1.3 Network Configuration Reference

Network configuration is critical for building a high performance *Ceph Storage Cluster*. The Ceph Storage Cluster does not perform request routing or dispatching on behalf of the *Ceph Client*. Instead, Ceph Clients make requests directly to Ceph OSD Daemons. Ceph OSD Daemons perform data replication on behalf of Ceph Clients, which means replication and other factors impose additional loads on Ceph Storage Cluster networks.

Our Quick Start configurations provide a trivial Ceph configuration file that sets monitor IP addresses and daemon host names only. Unless you specify a cluster network, Ceph assumes a single "public" network. Ceph functions just fine with a public network only, but you may see significant performance improvement with a second "cluster" network in a large cluster.

We recommend running a Ceph Storage Cluster with two networks: a public (front-side) network and a cluster (back-side) network. To support two networks, each *Ceph Node* will need to have more than one NIC. See Hardware Recommendations - Networks for additional details.



There are several reasons to consider operating two separate networks:

1. **Performance:** Ceph OSD Daemons handle data replication for the Ceph Clients. When Ceph OSD Daemons replicate data more than once, the network load between Ceph OSD Daemons easily dwarfs the network load between Ceph Clients and the Ceph Storage Cluster. This can introduce latency and create a performance problem. Recovery and rebalancing can also introduce significant latency on the public network. See Scalability and High Availability for additional details on how Ceph replicates data. See Monitor / OSD Interaction for details on heartbeat traffic.

2. **Security**: While most people are generally civil, a very tiny segment of the population likes to engage in what's known as a Denial of Service (DoS) attack. When traffic between Ceph OSD Daemons gets disrupted, placement groups may no longer reflect an `active + clean` state, which may prevent users from reading and writing data. A great way to defeat this type of attack is to maintain a completely separate cluster network that doesn't connect directly to the internet. Also, consider using Message Signatures to defeat spoofing attacks.

IP Tables

By default, daemons *bind* to ports within the `6800:7300` range. You may configure this range at your discretion. Before configuring your IP tables, check the default `iptables` configuration.

> sudo iptables -L

Some Linux distributions include rules that reject all inbound requests except SSH from all network interfaces. For example:

```
REJECT all -- anywhere anywhere reject-with icmp-host-prohibited
```

You will need to delete these rules on both your public and cluster networks initially, and replace them with appropriate rules when you are ready to harden the ports on your Ceph Nodes.

Monitor IP Tables

Ceph Monitors listen on port `6789` by default. Additionally, Ceph Monitors always operate on the public network. When you add the rule using the example below, make sure you replace `{iface}` with the public network interface (e.g., `eth0`, `eth1`, etc.), `{ip-address}` with the IP address of the public network and `{netmask}` with the netmask for the public network.

```
sudo iptables -A INPUT -i {iface} -p tcp -s {ip-address}/{netmask} --dport 6789 -j ACCEPT
```

MDS IP Tables

A *Ceph Metadata Server* listens on the first available port on the public network beginning at port 6800. Ensure that you open one port beginning at port 6800 for each Ceph Metadata Server that runs on the Ceph Node. When you add the rule using the example below, make sure you replace `{iface}` with the public network interface (e.g., `eth0`, `eth1`, etc.), `{ip-address}` with the IP address of the public network and `{netmask}` with the netmask of the public network.

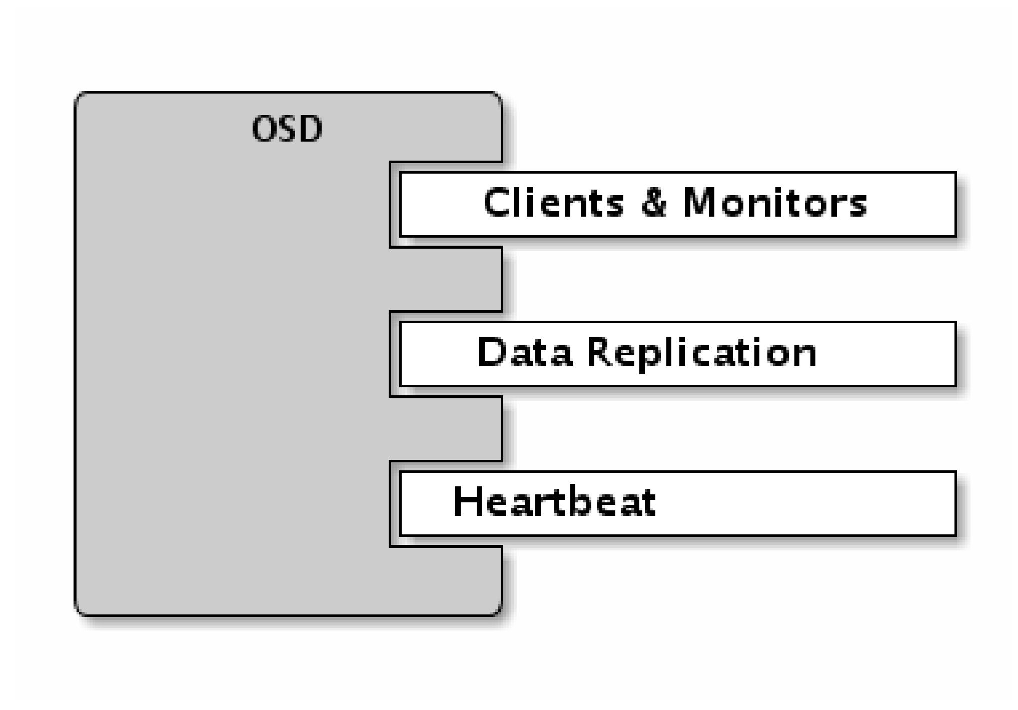
For example:

```
sudo iptables -A INPUT -i {iface} -m multiport -p tcp -s {ip-address}/{netmask} --dports 6800:6810 -
```

OSD IP Tables

By default, Ceph OSD Daemons *bind* to the first available ports on a Ceph Node beginning at port 6800. Ensure that you open at least three ports beginning at port 6800 for each OSD that runs on the host. Each Ceph OSD Daemon on a Ceph Node may use up to three ports:

1. One for talking to clients and monitors.
2. One for sending data to other OSDs.
3. One for heartbeating.



Ports are node-specific, so you don't need to open any more ports than the number of ports needed by Ceph daemons running on that Ceph Node. You may consider opening a few additional ports in case a daemon fails and restarts without letting go of the port such that the restarted daemon binds to a new port.

If you set up separate public and cluster networks, you must add rules for both the public network and the cluster network, because clients will connect using the public network and other Ceph OSD Daemons will connect using the cluster network. When you add the rule using the example below, make sure you replace `{iface}` with the network interface (e.g., `eth0`, `eth1`, etc.), `{ip-address}` with the IP address and `{netmask}` with the netmask of the public or cluster network. For example:

```
sudo iptables -A INPUT -i {iface}  -m multiport -p tcp -s {ip-address}/{netmask} --dports 6800:6810 -
```

Be sure to replace the "6810" in the above example with an upper bound that reflects the number of daemons you will be running on this host.

Tip: If you run Ceph Metadata Servers on the same Ceph Node as the Ceph OSD Daemons, you can consolidate the public network configuration step. Ensure that you open the number of ports required for each daemon per host.

Ceph Networks

To configure Ceph networks, you must add a network configuration to the `[global]` section of the configuration file. Our 5-minute Quick Start provides a trivial Ceph configuration file that assumes one public network with client and server on the same network and subnet. Ceph functions just fine with a public network only. However, Ceph allows you to establish much more specific criteria, including multiple IP network and subnet masks for your public network. You can also establish a separate cluster network to handle OSD heartbeat, object replication and recovery traffic. Don't confuse the IP addresses you set in your configuration with the public-facing IP addresses network clients may use to access your service. Typical internal IP networks are often `192.168.0.0` or `10.0.0.0`.

Tip: If you specify more than one IP address and subnet mask for either the public or the cluster network, the subnets within the network must be capable of routing to each other. Additionally, make sure you include each IP address/subnet in your IP tables and open ports for them as necessary.

Note: Ceph uses CIDR notation for subnets (e.g., `10.0.0.0/24`).

When you've configured your networks, you may restart your cluster or restart each daemon. Ceph daemons bind dynamically, so you do not have to restart the entire cluster at once if you change your network configuration.

Public Network

To configure a public network, add the following option to the `[global]` section of your Ceph configuration file.

```
[global]
        ...
        public network = {public-network/netmask}
```

Cluster Network

If you declare a cluster network, OSDs will route heartbeat, object replication and recovery traffic over the cluster network. This may improve performance compared to using a single network. To configure a cluster network, add the following option to the `[global]` section of your Ceph configuration file.

```
[global]
        ...
        cluster network = {cluster-network/netmask}
```

We prefer that the cluster network is **NOT** reachable from the public network or the Internet for added security.

Ceph Daemons

Ceph has one network configuration requirement that applies to all daemons: the Ceph configuration file **MUST** specify the `host` for each daemon. Ceph also requires that a Ceph configuration file specify the monitor IP address and its port.

Important: Some deployment tools (e.g., `ceph-deploy`, Chef) may create a configuration file for you. **DO NOT** set these values if the deployment tool does it for you.

Tip: The `host` setting is the short name of the host (i.e., not an fqdn). It is **NOT** an IP address either. Enter `hostname -s` on the command line to retrieve the name of the host.

```
[mon.a]

        host = {hostname}
        mon addr = {ip-address}:6789

[osd.0]
        host = {hostname}
```

You do not have to set the host IP address for a daemon. If you have a static IP configuration and both public and cluster networks running, the Ceph configuration file may specify the IP address of the host for each daemon. To set a static IP address for a daemon, the following option(s) should appear in the daemon instance sections of your `ceph.conf` file.

```
[osd.0]
        public addr = {host-public-ip-address}
        cluster addr = {host-cluster-ip-address}
```

One NIC OSD in a Two Network Cluster

Generally, we do not recommend deploying an OSD host with a single NIC in a cluster with two networks. However, you may accomplish this by forcing the OSD host to operate on the public network by adding a `public addr` entry to the `[osd.n]` section of the Ceph configuration file, where `n` refers to the number of the OSD with one NIC. Additionally, the public network and cluster network must be able to route traffic to each other, which we don't recommend for security reasons.

Network Config Settings

Network configuration settings are not required. Ceph assumes a public network with all hosts operating on it unless you specifically configure a cluster network.

Public Network

The public network configuration allows you specifically define IP addresses and subnets for the public network. You may specifically assign static IP addresses or override `public network` settings using the `public addr` setting for a specific daemon.

`public network`

Description The IP address and netmask of the public (front-side) network (e.g., `192.168.0.0/24`). Set in `[global]`. You may specify comma-delimited subnets.

Type `{ip-address}/{netmask} [, {ip-address}/{netmask}]`

Required No

Default N/A

`public addr`

Description The IP address for the public (front-side) network. Set for each daemon.

Type IP Address

Required No

Default N/A

Cluster Network

The cluster network configuration allows you to declare a cluster network, and specifically define IP addresses and subnets for the cluster network. You may specifically assign static IP addresses or override `cluster network` settings using the `cluster addr` setting for specific OSD daemons.

`cluster network`

Description The IP address and netmask of the cluster (back-side) network (e.g., `10.0.0.0/24`). Set in `[global]`. You may specify comma-delimited subnets.

Type `{ip-address}/{netmask} [, {ip-address}/{netmask}]`

Required No

Default N/A

`cluster addr`

Description The IP address for the cluster (back-side) network. Set for each daemon.

Type Address

Required No

Default N/A

Bind

Bind settings set the default port ranges Ceph OSD and MDS daemons use. The default range is `6800:7300`. Ensure that your *IP Tables* configuration allows you to use the configured port range.

You may also enable Ceph daemons to bind to IPv6 addresses.

`ms bind port min`

Description The minimum port number to which an OSD or MDS daemon will bind.

Type 32-bit Integer

Default `6800`

Required No

`ms bind port max`

Description The maximum port number to which an OSD or MDS daemon will bind.

Type 32-bit Integer

Default `7300`

Required No.

`ms bind ipv6`

Description Enables Ceph daemons to bind to IPv6 addresses.

Type Boolean

Default `false`

Required No

Hosts

Ceph expects at least one monitor declared in the Ceph configuration file, with a `mon addr` setting under each declared monitor. Ceph expects a `host` setting under each declared monitor, metadata server and OSD in the Ceph configuration file.

`mon addr`

Description A list of `{hostname}:{port}` entries that clients can use to connect to a Ceph monitor. If not set, Ceph searches `[mon.*]` sections.

Type String

Required No

Default N/A

`host`

Description The hostname. Use this setting for specific daemon instances (e.g., `[osd.0]`).

Type String

Required Yes, for daemon instances.

Default `localhost`

Tip: Do not use `localhost`. To get your host name, execute `hostname -s` on your command line and use the name of your host (to the first period, not the fully-qualified domain name).

Important: You should not specify any value for `host` when using a third party deployment system that retrieves the host name for you.

TCP

Ceph disables TCP buffering by default.

`tcp nodelay`

Description Ceph enables `tcp nodelay` so that each request is sent immediately (no buffering). Disabling Nagle's algorithm increases network traffic, which can introduce latency. If you experience large numbers of small packets, you may try disabling `tcp nodelay`.

Type Boolean

Required No

Default `true`

`tcp rcvbuf`

Description The size of the socket buffer on the receiving end of a network connection. Disable by default.

Type 32-bit Integer

Required No

Default `0`

`ms tcp read timeout`

Description If a client or daemon makes a request to another Ceph daemon and does not drop an unused connection, the `tcp read timeout` defines the connection as idle after the specified number of seconds.

Type Unsigned 64-bit Integer

Required No

Default `900` 15 minutes.

4.1.4 Cephx Config Reference

The `cephx` protocol is enabled by default. Cryptographic authentication has some computational costs, though they should generally be quite low. If the network environment connecting your client and server hosts is very safe and you cannot afford authentication, you can turn it off. **This is not generally recommended**.

Note: If you disable authentication, you are at risk of a man-in-the-middle attack altering your client/server messages, which could lead to disastrous security effects.

For creating users, see User Management. For details on the architecture of Cephx, see Architecture - High Availability Authentication.

Deployment Scenarios

There are two main scenarios for deploying a Ceph cluster, which impact how you initially configure Cephx. Most first time Ceph users use `ceph-deploy` to create a cluster (easiest). For clusters using other deployment tools (e.g., Chef, Juju, Puppet, etc.), you will need to use the manual procedures or configure your deployment tool to bootstrap your monitor(s).

ceph-deploy

When you deploy a cluster with `ceph-deploy`, you do not have to bootstrap the monitor manually or create the `client.admin` user or keyring. The steps you execute in the Storage Cluster Quick Start will invoke `ceph-deploy` to do that for you.

When you execute `ceph-deploy new {initial-monitor(s)}`, Ceph will create a monitor keyring for you (only used to bootstrap monitors), and it will generate an initial Ceph configuration file for you, which contains the following authentication settings, indicating that Ceph enables authentication by default:

```
auth_cluster_required = cephx
auth_service_required = cephx
auth_client_required = cephx
```

When you execute `ceph-deploy mon create-initial`, Ceph will bootstrap the initial monitor(s), retrieve a `ceph.client.admin.keyring` file containing the key for the `client.admin` user. Additionally, it will also retrieve keyrings that give `ceph-deploy` and `ceph-disk` utilities the ability to prepare and activate OSDs and metadata servers.

When you execute `ceph-deploy admin {node-name}` (**note:** Ceph must be installed first), you are pushing a Ceph configuration file and the `ceph.client.admin.keyring` to the `/etc/ceph` directory of the node. You will be able to execute Ceph administrative functions as `root` on the command line of that node.

Manual Deployment

When you deploy a cluster manually, you have to bootstrap the monitor manually and create the `client.admin` user and keyring. To bootstrap monitors, follow the steps in Monitor Bootstrapping. The steps for monitor bootstrapping are the logical steps you must perform when using third party deployment tools like Chef, Puppet, Juju, etc.

Enabling/Disabling Cephx

Enabling Cephx requires that you have deployed keys for your monitors, OSDs and metadata servers. If you are simply toggling Cephx on / off, you do not have to repeat the bootstrapping procedures.

Enabling Cephx

When `cephx` is enabled, Ceph will look for the keyring in the default search path, which includes `/etc/ceph/$cluster.$name.keyring`. You can override this location by adding a `keyring` option in the `[global]` section of your Ceph configuration file, but this is not recommended.

Execute the following procedures to enable `cephx` on a cluster with authentication disabled. If you (or your deployment utility) have already generated the keys, you may skip the steps related to generating keys.

1. Create a `client.admin` key, and save a copy of the key for your client host:

```
ceph auth get-or-create client.admin mon 'allow *' mds 'allow *' osd 'allow *' -o /etc/ceph/ceph
```

   Warning: This will clobber any existing `/etc/ceph/client.admin.keyring` file. Do not perform this step if a deployment tool has already done it for you. Be careful!

2. Create a keyring for your monitor cluster and generate a monitor secret key.

```
ceph-authtool --create-keyring /tmp/ceph.mon.keyring --gen-key -n mon. --cap mon 'allow *'
```

3. Copy the monitor keyring into a `ceph.mon.keyring` file in every monitor's `mon data` directory. For example, to copy it to `mon.a` in cluster `ceph`, use the following:

```
cp /tmp/ceph.mon.keyring /var/lib/ceph/mon/ceph-a/keyring
```

4. Generate a secret key for every OSD, where `{$id}` is the OSD number:

```
ceph auth get-or-create osd.{$id} mon 'allow rwx' osd 'allow *' -o /var/lib/ceph/osd/ceph-{$id}/
```

5. Generate a secret key for every MDS, where `{$id}` is the MDS letter:

```
ceph auth get-or-create mds.{$id} mon 'allow rwx' osd 'allow *' mds 'allow *' -o /var/lib/ceph/m
```

6. Enable `cephx` authentication by setting the following options in the `[global]` section of your Ceph configuration file:

```
auth cluster required = cephx
auth service required = cephx
auth client required = cephx
```

7. Start or restart the Ceph cluster. See Operating a Cluster for details.

For details on bootstrapping a monitor manually, see Manual Deployment.

Disabling Cephx

The following procedure describes how to disable Cephx. If your cluster environment is relatively safe, you can offset the computation expense of running authentication. **We do not recommend it.** However, it may be easier during setup and/or troubleshooting to temporarily disable authentication.

1. Disable `cephx` authentication by setting the following options in the `[global]` section of your Ceph configuration file:

```
auth cluster required = none
auth service required = none
auth client required = none
```

2. Start or restart the Ceph cluster. See Operating a Cluster for details.

Configuration Settings

Enablement

`auth cluster required`

Description If enabled, the Ceph Storage Cluster daemons (i.e., `ceph-mon`, `ceph-osd`, and `ceph-mds`) must authenticate with each other. Valid settings are `cephx` or `none`.

Type String

Required No

Default `cephx`.

`auth service required`

Description If enabled, the Ceph Storage Cluster daemons require Ceph Clients to authenticate with the Ceph Storage Cluster in order to access Ceph services. Valid settings are `cephx` or `none`.

Type String

Required No

Default `cephx`.

`auth client required`

Description If enabled, the Ceph Client requires the Ceph Storage Cluster to authenticate with the Ceph Client. Valid settings are `cephx` or `none`.

Type String

Required No

Default `cephx`.

Keys

When you run Ceph with authentication enabled, `ceph` administrative commands and Ceph Clients require authentication keys to access the Ceph Storage Cluster.

The most common way to provide these keys to the `ceph` administrative commands and clients is to include a Ceph keyring under the `/etc/ceph` directory. For Cuttlefish and later releases using `ceph-deploy`, the filename is usually `ceph.client.admin.keyring` (or `$cluster.client.admin.keyring`). If you include the keyring under the `/etc/ceph` directory, you don't need to specify a `keyring` entry in your Ceph configuration file.

We recommend copying the Ceph Storage Cluster's keyring file to nodes where you will run administrative commands, because it contains the `client.admin` key.

You may use `ceph-deploy admin` to perform this task. See Create an Admin Host for details. To perform this step manually, execute the following:

```
sudo scp {user}@{ceph-cluster-host}:/etc/ceph/ceph.client.admin.keyring /etc/ceph/ceph.client.admin.k
```

Tip: Ensure the `ceph.keyring` file has appropriate permissions set (e.g., `chmod 644`) on your client machine.

You may specify the key itself in the Ceph configuration file using the `key` setting (not recommended), or a path to a keyfile using the `keyfile` setting.

`keyring`

Description The path to the keyring file.

Type String

Required No

Default `/etc/ceph/$cluster.$name.keyring,/etc/ceph/$cluster.keyring,/etc/ceph/keyring,/e`

`keyfile`

Description The path to a key file (i.e,. a file containing only the key).

Type String

Required No

Default None

`key`

Description The key (i.e., the text string of the key itself). Not recommended.

Type String

Required No

Default None

Daemon Keyrings

Administrative users or deployment tools (e.g., `ceph-deploy`) may generate daemon keyrings in the same way as generating user keyrings. By default, Ceph stores daemons keyrings inside their data directory. The default keyring locations, and the capabilities necessary for the daemon to function, are shown below.

`ceph-mon`

Location `$mon_data/keyring`

Capabilities `mon 'allow *'`

`ceph-osd`

Location `$osd_data/keyring`

Capabilities `mon 'allow profile osd' osd 'allow *'`

`ceph-mds`

Location `$mds_data/keyring`

Capabilities `mds 'allow' mon 'allow profile mds' osd 'allow rwx'`

`radosgw`

Location `$rgw_data/keyring`

Capabilities `mon 'allow rwx' osd 'allow rwx'`

Note: The monitor keyring (i.e., `mon.`) contains a key but no capabilities, and is not part of the cluster `auth` database.

The daemon data directory locations default to directories of the form:

```
/var/lib/ceph/$type/$cluster-$id
```

For example, `osd.12` would be:

```
/var/lib/ceph/osd/ceph-12
```

You can override these locations, but it is not recommended.

Signatures

In Ceph Bobtail and subsequent versions, we prefer that Ceph authenticate all ongoing messages between the entities using the session key set up for that initial authentication. However, Argonaut and earlier Ceph daemons do not know how to perform ongoing message authentication. To maintain backward compatibility (e.g., running both Botbail and Argonaut daemons in the same cluster), message signing is **off** by default. If you are running Bobtail or later daemons exclusively, configure Ceph to require signatures.

Like other parts of Ceph authentication, Ceph provides fine-grained control so you can enable/disable signatures for service messages between the client and Ceph, and you can enable/disable signatures for messages between Ceph daemons.

`cephx require signatures`

Description If set to `true`, Ceph requires signatures on all message traffic between the Ceph Client and the Ceph Storage Cluster, and between daemons comprising the Ceph Storage Cluster.

Type Boolean

Required No

Default `false`

`cephx cluster require signatures`

Description If set to `true`, Ceph requires signatures on all message traffic between Ceph daemons comprising the Ceph Storage Cluster.

Type Boolean

Required No

Default `false`

`cephx service require signatures`

Description If set to `true`, Ceph requires signatures on all message traffic between Ceph Clients and the Ceph Storage Cluster.

Type Boolean

Required No

Default `false`

`cephx sign messages`

Description If the Ceph version supports message signing, Ceph will sign all messages so they cannot be spoofed.

Type Boolean

Default `true`

Time to Live

`auth service ticket ttl`

Description When the Ceph Storage Cluster sends a Ceph Client a ticket for authentication, the Ceph Storage Cluster assigns the ticket a time to live.

Type Double

Default `60*60`

Backward Compatibility

For Cuttlefish and earlier releases, see Cephx.

In Ceph Argonaut v0.48 and earlier versions, if you enable `cephx` authentication, Ceph only authenticates the initial communication between the client and daemon; Ceph does not authenticate the subsequent messages they send to each other, which has security implications. In Ceph Bobtail and subsequent versions, Ceph authenticates all ongoing messages between the entities using the session key set up for that initial authentication.

We identified a backward compatibility issue between Argonaut v0.48 (and prior versions) and Bobtail (and subsequent versions). During testing, if you attempted to use Argonaut (and earlier) daemons with Bobtail (and later) daemons, the Argonaut daemons did not know how to perform ongoing message authentication, while the Bobtail versions of the daemons insist on authenticating message traffic subsequent to the initial request/response–making it impossible for Argonaut (and prior) daemons to interoperate with Bobtail (and subsequent) daemons.

We have addressed this potential problem by providing a means for Argonaut (and prior) systems to interact with Bobtail (and subsequent) systems. Here's how it works: by default, the newer systems will not insist on seeing signatures from older systems that do not know how to perform them, but will simply accept such messages without authenticating them. This new default behavior provides the advantage of allowing two different releases to interact. **We do not recommend this as a long term solution**. Allowing newer daemons to forgo ongoing authentication has the unfortunate security effect that an attacker with control of some of your machines or some access to your network can disable session security simply by claiming to be unable to sign messages.

Note: Even if you don't actually run any old versions of Ceph, the attacker may be able to force some messages to be accepted unsigned in the default scenario. While running Cephx with the default scenario, Ceph still authenticates the initial communication, but you lose desirable session security.

If you know that you are not running older versions of Ceph, or you are willing to accept that old servers and new servers will not be able to interoperate, you can eliminate this security risk. If you do so, any Ceph system that is new enough to support session authentication and that has Cephx enabled will reject unsigned messages. To preclude new servers from interacting with old servers, include the following in the `[global]` section of your Ceph configuration file directly below the line that specifies the use of Cephx for authentication:

```
cephx require signatures = true    ; everywhere possible
```

You can also selectively require signatures for cluster internal communications only, separate from client-facing service:

```
cephx cluster require signatures = true    ; for cluster-internal communication
cephx service require signatures = true    ; for client-facing service
```

An option to make a client require signatures from the cluster is not yet implemented.

We recommend migrating all daemons to the newer versions and enabling the foregoing flag at the nearest practical time so that you may avail yourself of the enhanced authentication.

Note: Ceph kernel modules do not support signatures yet.

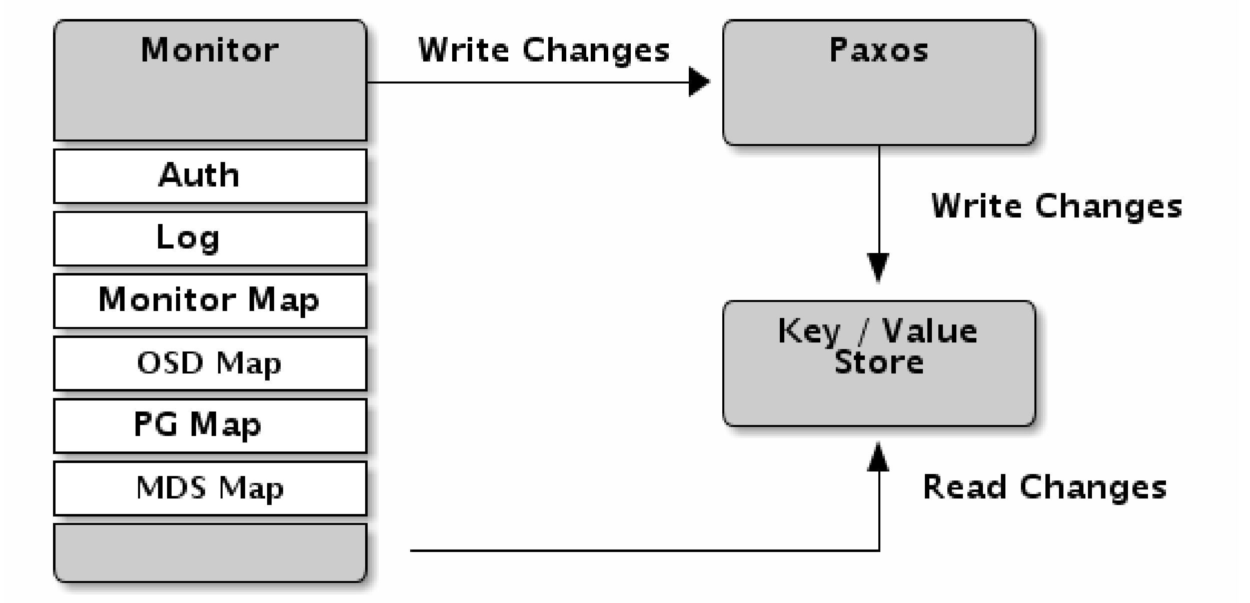
4.1.5 Monitor Config Reference

Understanding how to configure a *Ceph Monitor* is an important part of building a reliable *Ceph Storage Cluster*. **All Ceph Storage Clusters have at least one monitor**. A monitor configuration usually remains fairly consistent, but you can add, remove or replace a monitor in a cluster. See Adding/Removing a Monitor and Add/Remove a Monitor (ceph-deploy) for details.

Background

Ceph Monitors maintain a "master copy" of the *cluster map*, which means a *Ceph Client* can determine the location of all Ceph Monitors, Ceph OSD Daemons, and Ceph Metadata Servers just by connecting to one Ceph Monitor and retrieving a current cluster map. Before Ceph Clients can read from or write to Ceph OSD Daemons or Ceph Metadata Servers, they must connect to a Ceph Monitor first. With a current copy of the cluster map and the CRUSH algorithm, a Ceph Client can compute the location for any object. The ability to compute object locations allows a Ceph Client to talk directly to Ceph OSD Daemons, which is a very important aspect of Ceph's high scalability and performance. See Scalability and High Availability for additional details.

The primary role of the Ceph Monitor is to maintain a master copy of the cluster map. Ceph Monitors also provide authentication and logging services. Ceph Monitors write all changes in the monitor services to a single Paxos instance, and Paxos writes the changes to a key/value store for strong consistency. Ceph Monitors can query the most recent version of the cluster map during sync operations. Ceph Monitors leverage the key/value store's snapshots and iterators (using leveldb) to perform store-wide synchronization.



Deprecated since version version: 0.58

In Ceph versions 0.58 and earlier, Ceph Monitors use a Paxos instance for each service and store the map as a file.

Cluster Maps

The cluster map is a composite of maps, including the monitor map, the OSD map, the placement group map and the metadata server map. The cluster map tracks a number of important things: which processes are `in` the Ceph Storage

Cluster; which processes that are `in` the Ceph Storage Cluster are `up` and running or `down`; whether, the placement groups are `active` or `inactive`, and `clean` or in some other state; and, other details that reflect the current state of the cluster such as the total amount of storage space, and the amount of storage used.

When there is a significant change in the state of the cluster–e.g., a Ceph OSD Daemon goes down, a placement group falls into a degraded state, etc.–the cluster map gets updated to reflect the current state of the cluster. Additionally, the Ceph Monitor also maintains a history of the prior states of the cluster. The monitor map, OSD map, placement group map and metadata server map each maintain a history of their map versions. We call each version an "epoch."

When operating your Ceph Storage Cluster, keeping track of these states is an important part of your system administration duties. See Monitoring a Cluster and Monitoring OSDs and PGs for additional details.

Monitor Quorum

Our Getting Started section provides a trivial Ceph configuration file that provides for one monitor in the test cluster. A cluster will run fine with a single monitor; however, **a single monitor is a single-point-of-failure**. To ensure high availability in a production Ceph Storage Cluster, you should run Ceph with multiple monitors so that the failure of a single monitor **WILL NOT** bring down your entire cluster.

When a Ceph Storage Cluster runs multiple Ceph Monitors for high availability, Ceph Monitors use Paxos to establish consensus about the master cluster map. A consensus requires a majority of monitors running to establish a quorum for consensus about the cluster map (e.g., 1; 2 out of 3; 3 out of 5; 4 out of 6; etc.).

Consistency

When you add monitor settings to your Ceph configuration file, you need to be aware of some of the architectural aspects of Ceph Monitors. **Ceph imposes strict consistency requirements** for a Ceph monitor when discovering another Ceph Monitor within the cluster. Whereas, Ceph Clients and other Ceph daemons use the Ceph configuration file to discover monitors, monitors discover each other using the monitor map (monmap), not the Ceph configuration file.

A Ceph Monitor always refers to the local copy of the monmap when discovering other Ceph Monitors in the Ceph Storage Cluster. Using the monmap instead of the Ceph configuration file avoids errors that could break the cluster (e.g., typos in `ceph.conf` when specifying a monitor address or port). Since monitors use monmaps for discovery and they share monmaps with clients and other Ceph daemons, **the monmap provides monitors with a strict guarantee that their consensus is valid.**

Strict consistency also applies to updates to the monmap. As with any other updates on the Ceph Monitor, changes to the monmap always run through a distributed consensus algorithm called Paxos. The Ceph Monitors must agree on each update to the monmap, such as adding or removing a Ceph Monitor, to ensure that each monitor in the quorum has the same version of the monmap. Updates to the monmap are incremental so that Ceph Monitors have the latest agreed upon version, and a set of previous versions. Maintaining a history enables a Ceph Monitor that has an older version of the monmap to catch up with the current state of the Ceph Storage Cluster.

If Ceph Monitors discovered each other through the Ceph configuration file instead of through the monmap, it would introduce additional risks because the Ceph configuration files aren't updated and distributed automatically. Ceph Monitors might inadvertently use an older Ceph configuration file, fail to recognize a Ceph Monitor, fall out of a quorum, or develop a situation where Paxos isn't able to determine the current state of the system accurately.

Bootstrapping Monitors

In most configuration and deployment cases, tools that deploy Ceph may help bootstrap the Ceph Monitors by generating a monitor map for you (e.g., `ceph-deploy`, etc). A Ceph Monitor requires a few explicit settings:

- **Filesystem ID**: The `fsid` is the unique identifier for your object store. Since you can run multiple clusters on the same hardware, you must specify the unique ID of the object store when bootstrapping a monitor. Deployment tools usually do this for you (e.g., `ceph-deploy` can call a tool like `uuidgen`), but you may specify the `fsid` manually too.
- **Monitor ID**: A monitor ID is a unique ID assigned to each monitor within the cluster. It is an alphanumeric value, and by convention the identifier usually follows an alphabetical increment (e.g., `a`, `b`, etc.). This can be set in a Ceph configuration file (e.g., `[mon.a]`, `[mon.b]`, etc.), by a deployment tool, or using the `ceph` commandline.
- **Keys**: The monitor must have secret keys. A deployment tool such as `ceph-deploy` usually does this for you, but you may perform this step manually too. See Monitor Keyrings for details.

For additional details on bootstrapping, see Bootstrapping a Monitor.

Configuring Monitors

To apply configuration settings to the entire cluster, enter the configuration settings under `[global]`. To apply configuration settings to all monitors in your cluster, enter the configuration settings under `[mon]`. To apply configuration settings to specific monitors, specify the monitor instance (e.g., `[mon.a]`). By convention, monitor instance names use alpha notation.

```
[global]

[mon]

[mon.a]

[mon.b]

[mon.c]
```

Minimum Configuration

The bare minimum monitor settings for a Ceph monitor via the Ceph configuration file include a hostname and a monitor address for each monitor. You can configure these under `[mon]` or under the entry for a specific monitor.

```
[mon]
        mon host = hostname1,hostname2,hostname3
        mon addr = 10.0.0.10:6789,10.0.0.11:6789,10.0.0.12:6789
```

```
[mon.a]
        host = hostname1
        mon addr = 10.0.0.10:6789
```

See the Network Configuration Reference for details.

Note: This minimum configuration for monitors assumes that a deployment tool generates the `fsid` and the `mon.` key for you.

Once you deploy a Ceph cluster, you **SHOULD NOT** change the IP address of the monitors. However, if you decide to change the monitor's IP address, you must follow a specific procedure. See Changing a Monitor's IP Address for details.

Cluster ID

Each Ceph Storage Cluster has a unique identifier (`fsid`). If specified, it usually appears under the `[global]` section of the configuration file. Deployment tools usually generate the `fsid` and store it in the monitor map, so the value may not appear in a configuration file. The `fsid` makes it possible to run daemons for multiple clusters on the same hardware.

`fsid`

Description The cluster ID. One per cluster.

Type UUID

Required Yes.

Default N/A. May be generated by a deployment tool if not specified.

Note: Do not set this value if you use a deployment tool that does it for you.

Initial Members

We recommend running a production Ceph Storage Cluster with at least three Ceph Monitors to ensure high availability. When you run multiple monitors, you may specify the initial monitors that must be members of the cluster in order to establish a quorum. This may reduce the time it takes for your cluster to come online.

```
[mon]
        mon initial members = a,b,c
```

`mon initial members`

Description The IDs of initial monitors in a cluster during startup. If specified, Ceph requires an odd number of monitors to form an initial quorum (e.g., 3).

Type String

Default None

Note: A *majority* of monitors in your cluster must be able to reach each other in order to establish a quorum. You can decrease the initial number of monitors to establish a quorum with this setting.

Data

Ceph provides a default path where Ceph Monitors store data. For optimal performance in a production Ceph Storage Cluster, we recommend running Ceph Monitors on separate hosts and drives from Ceph OSD Daemons. Ceph Monitors do lots of `fsync()`, which can interfere with Ceph OSD Daemon workloads.

In Ceph versions 0.58 and earlier, Ceph Monitors store their data in files. This approach allows users to inspect monitor data with common tools like `ls` and `cat`. However, it doesn't provide strong consistency.

In Ceph versions 0.59 and later, Ceph Monitors store their data as key/value pairs. Ceph Monitors require ACID transactions. Using a data store prevents recovering Ceph Monitors from running corrupted versions through Paxos, and it enables multiple modification operations in one single atomic batch, among other advantages.

Generally, we do not recommend changing the default data location. If you modify the default location, we recommend that you make it uniform across Ceph Monitors by setting it in the `[mon]` section of the configuration file.

`mon data`

Description The monitor's data location.

Type String

Default `/var/lib/ceph/mon/$cluster-$id`

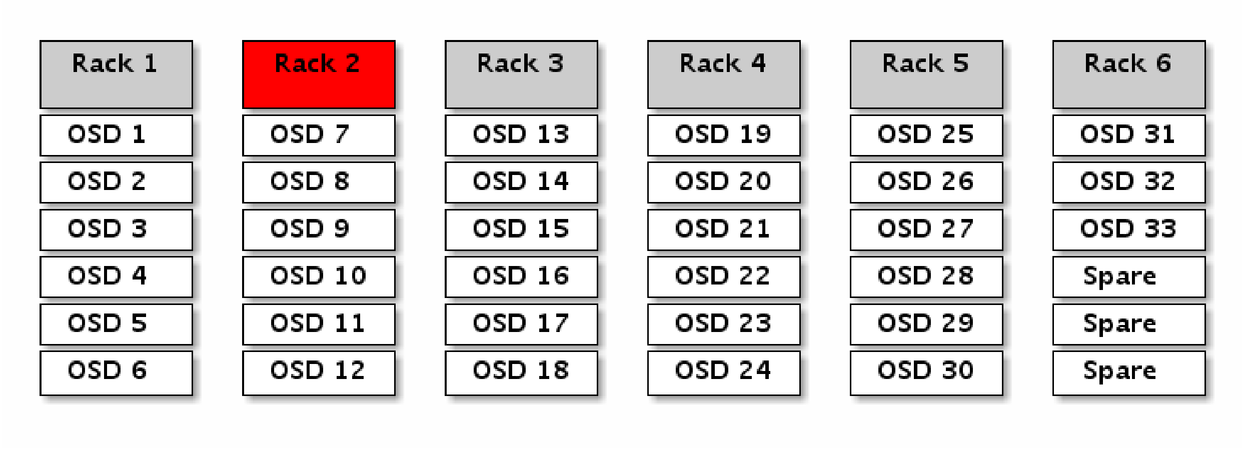
Storage Capacity

When a Ceph Storage Cluster gets close to its maximum capacity (i.e., `mon osd full ratio`), Ceph prevents you from writing to or reading from Ceph OSD Daemons as a safety measure to prevent data loss. Therefore, letting a production Ceph Storage Cluster approach its full ratio is not a good practice, because it sacrifices high availability. The default full ratio is `.95`, or 95% of capacity. This a very aggressive setting for a test cluster with a small number of OSDs.

Tip: When monitoring your cluster, be alert to warnings related to the `nearfull` ratio. This means that a failure of some OSDs could result in a temporary service disruption if one or more OSDs fails. Consider adding more OSDs to increase storage capacity.

A common scenario for test clusters involves a system administrator removing a Ceph OSD Daemon from the Ceph Storage Cluster to watch the cluster rebalance; then, removing another Ceph OSD Daemon, and so on until the Ceph Storage Cluster eventually reaches the full ratio and locks up. We recommend a bit of capacity planning even with a test cluster. Planning enables you to gauge how much spare capacity you will need in order to maintain high availability. Ideally, you want to plan for a series of Ceph OSD Daemon failures where the cluster can recover to an `active + clean` state without replacing those Ceph OSD Daemons immediately. You can run a cluster in an `active + degraded` state, but this is not ideal for normal operating conditions.

The following diagram depicts a simplistic Ceph Storage Cluster containing 33 Ceph Nodes with one Ceph OSD Daemon per host, each Ceph OSD Daemon reading from and writing to a 3TB drive. So this exemplary Ceph Storage Cluster has a maximum actual capacity of 99TB. With a `mon osd full ratio` of `0.95`, if the Ceph Storage Cluster falls to 5TB of remaining capacity, the cluster will not allow Ceph Clients to read and write data. So the Ceph Storage Cluster's operating capacity is 95TB, not 99TB.



It is normal in such a cluster for one or two OSDs to fail. A less frequent but reasonable scenario involves a rack's router or power supply failing, which brings down multiple OSDs simultaneously (e.g., OSDs 7-12). In such a scenario, you should still strive for a cluster that can remain operational and achieve an `active + clean` state–even if that means adding a few hosts with additional OSDs in short order. If your capacity utilization is too high, you may not lose data, but you could still sacrifice data availability while resolving an outage within a failure domain if capacity utilization of the cluster exceeds the full ratio. For this reason, we recommend at least some rough capacity planning.

Identify two numbers for your cluster:

1. The number of OSDs.
2. The total capacity of the cluster

If you divide the total capacity of your cluster by the number of OSDs in your cluster, you will find the mean average capacity of an OSD within your cluster. Consider multiplying that number by the number of OSDs you expect will fail simultaneously during normal operations (a relatively small number). Finally multiply the capacity of the cluster by the full ratio to arrive at a maximum operating capacity; then, subtract the number of amount of data from the OSDs you expect to fail to arrive at a reasonable full ratio. Repeat the foregoing process with a higher number of OSD failures (e.g., a rack of OSDs) to arrive at a reasonable number for a near full ratio.

```
[global]

        mon osd full ratio = .80
        mon osd nearfull ratio = .70
```

`mon osd full ratio`

Description The percentage of disk space used before an OSD is considered `full`.

Type Float

Default `.95`

`mon osd nearfull ratio`

Description The percentage of disk space used before an OSD is considered `nearfull`.

Type Float

Default `.85`

Tip: If some OSDs are nearfull, but others have plenty of capacity, you may have a problem with the CRUSH weight for the nearfull OSDs.

Heartbeat

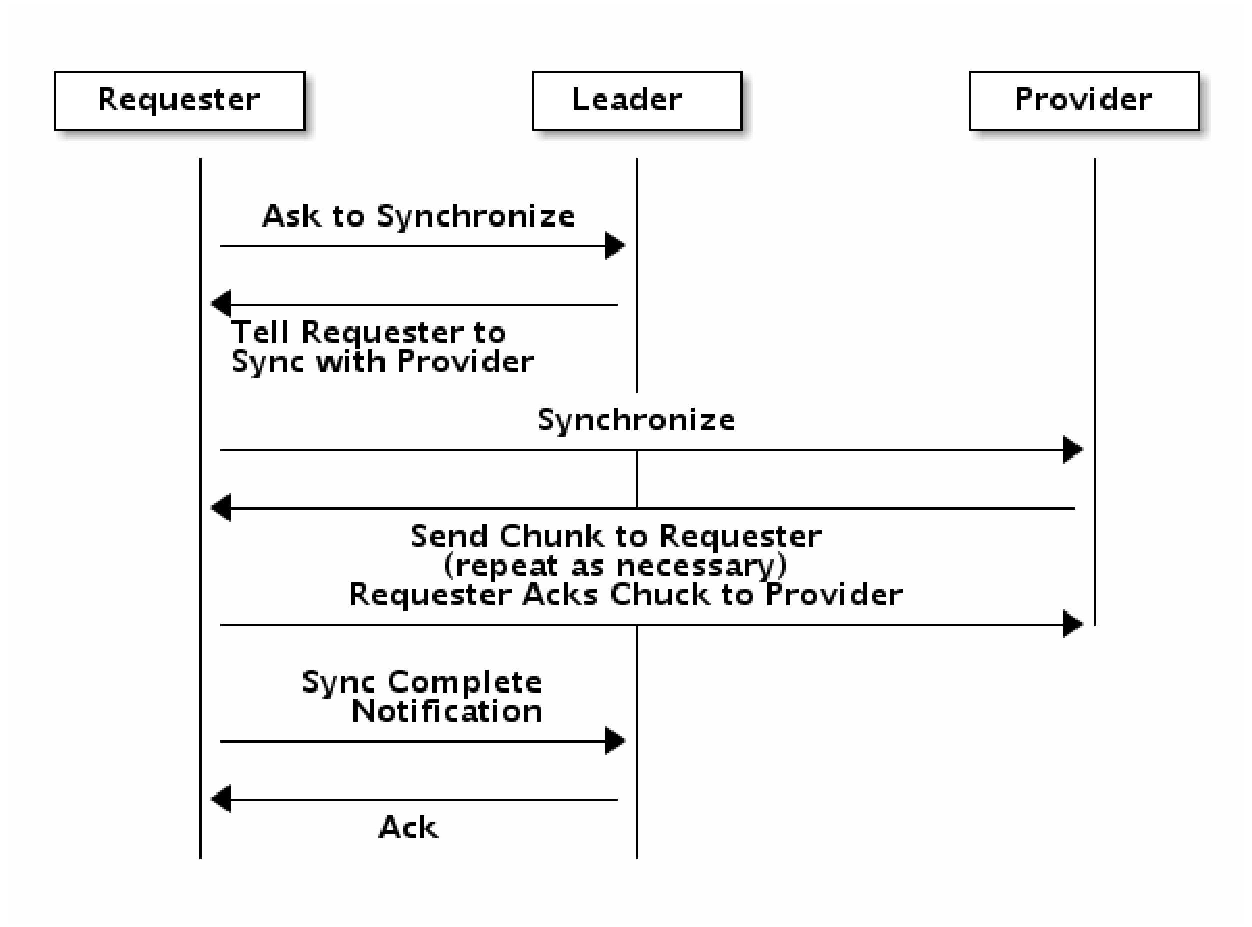
Ceph monitors know about the cluster by requiring reports from each OSD, and by receiving reports from OSDs about the status of their neighboring OSDs. Ceph provides reasonable default settings for monitor/OSD interaction; however, you may modify them as needed. See Monitor/OSD Interaction for details.

Monitor Store Synchronization

When you run a production cluster with multiple monitors (recommended), each monitor checks to see if a neighboring monitor has a more recent version of the cluster map (e.g., a map in a neighboring monitor with one or more epoch numbers higher than the most current epoch in the map of the instant monitor). Periodically, one monitor in the cluster may fall behind the other monitors to the point where it must leave the quorum, synchronize to retrieve the most current information about the cluster, and then rejoin the quorum. For the purposes of synchronization, monitors may assume one of three roles:

1. **Leader**: The *Leader* is the first monitor to achieve the most recent Paxos version of the cluster map.
2. **Provider**: The *Provider* is a monitor that has the most recent version of the cluster map, but wasn't the first to achieve the most recent version.
3. **Requester:** A *Requester* is a monitor that has fallen behind the leader and must synchronize in order to retrieve the most recent information about the cluster before it can rejoin the quorum.

These roles enable a leader to delegate synchronization duties to a provider, which prevents synchronization requests from overloading the leader–improving performance. In the following diagram, the requester has learned that it has fallen behind the other monitors. The requester asks the leader to synchronize, and the leader tells the requester to synchronize with a provider.


Requester
Leader
Provider
Ask to Synchronize
Tell Requester to Sync with Provider
Synchronize
Send Chunk to Requester (repeat as necessary) Requester Acks Chuck to Provider
Sync Complete Notification
Ack

Synchronization always occurs when a new monitor joins the cluster. During runtime operations, monitors may receive updates to the cluster map at different times. This means the leader and provider roles may migrate from one monitor to another. If this happens while synchronizing (e.g., a provider falls behind the leader), the provider can terminate synchronization with a requester.

Once synchronization is complete, Ceph requires trimming across the cluster. Trimming requires that the placement groups are `active + clean`.

`mon sync trim timeout`

Description

Type Double

Default `30.0`

`mon sync heartbeat timeout`

Description

Type Double

Default `30.0`

`mon sync heartbeat interval`

Description

Type Double

Default `5.0`

`mon sync backoff timeout`

Description

Type Double

Default `30.0`

`mon sync timeout`

Description

Type Double

Default `30.0`

`mon sync max retries`

Description

Type Integer

Default `5`

`mon sync max payload size`

Description The maximum size for a sync payload.

Type 32-bit Integer

Default `1045676`

`mon accept timeout`

Description Number of seconds the Leader will wait for the Requester(s) to accept a Paxos update. It is also used during the Paxos recovery phase for similar purposes.

Type Float

Default `10.0`

`paxos propose interval`

Description Gather updates for this time interval before proposing a map update.

Type Double

Default `1.0`

`paxos min wait`

Description The minimum amount of time to gather updates after a period of inactivity.

Type Double

Default `0.05`

`paxos trim tolerance`

Description The number of extra proposals tolerated before trimming.

Type Integer

Default `30`

`paxos trim disabled max versions`

Description The maximimum number of version allowed to pass without trimming.

Type Integer

Default `100`

`mon lease`

Description The length (in seconds) of the lease on the monitor's versions.

Type Float

Default `5`

`mon lease renew interval`

Description The interval (in seconds) for the Leader to renew the other monitor's leases.

Type Float

Default `3`

`mon lease ack timeout`

Description The number of seconds the Leader will wait for the Providers to acknowledge the lease extension.

Type Float

Default `10.0`

`mon min osdmap epochs`

Description Minimum number of OSD map epochs to keep at all times.

Type 32-bit Integer

Default `500`

`mon max pgmap epochs`

Description Maximum number of PG map epochs the monitor should keep.

Type 32-bit Integer

Default `500`

`mon max log epochs`

Description Maximum number of Log epochs the monitor should keep.

Type 32-bit Integer

Default `500`

Slurp

In Ceph version 0.58 and earlier, when a Paxos service drifts beyond a given number of versions, Ceph triggers the *slurp* mechanism, which establishes a connection with the quorum Leader and obtains every single version the Leader has for every service that has drifted. In Ceph versions 0.59 and later, slurp will not work, because there is a single Paxos instance for all services.

Deprecated since version 0.58.

`paxos max join drift`

Description The maximum Paxos iterations before we must first sync the monitor data stores.

Type Integer

Default `10`

`mon slurp timeout`

Description The number of seconds the monitor has to recover using slurp before the process is aborted and the monitor bootstraps.

Type Double

Default `10.0`

`mon slurp bytes`

Description Limits the slurp messages to the specified number of bytes.

Type 32-bit Integer

Default `256 * 1024`

Clock

Ceph daemons pass critical messages to each other, which must be processed before daemons reach a timeout threshold. If the clocks in Ceph monitors are not synchronized, it can lead to a number of anomalies. For example:

- Daemons ignoring received messages (e.g., timestamps outdated)
- Timeouts triggered too soon/late when a message wasn't received in time.

See *Monitor Store Synchronization* and *Slurp* for details.

Tip: You SHOULD install NTP on your Ceph monitor hosts to ensure that the monitor cluster operates with synchronized clocks.

Clock drift may still be noticeable with NTP even though the discrepancy isn't yet harmful. Ceph's clock drift / clock skew warnings may get triggered even though NTP maintains a reasonable level of synchronization. Increasing your clock drift may be tolerable under such circumstances; however, a number of factors such as workload, network latency, configuring overrides to default timeouts and the *Monitor Store Synchronization* settings may influence the level of acceptable clock drift without compromising Paxos guarantees.

Ceph provides the following tunable options to allow you to find acceptable values.

`clock offset`

Description How much to offset the system clock. See `Clock.cc` for details.

Type Double

Default `0`

Deprecated since version 0.58.

`mon tick interval`

Description A monitor's tick interval in seconds.

Type 32-bit Integer

Default `5`

`mon clock drift allowed`

Description The clock drift in seconds allowed between monitors.

Type Float

Default `.050`

`mon clock drift warn backoff`

Description Exponential backoff for clock drift warnings

Type Float

Default `5`

`mon timecheck interval`

Description The time check interval (clock drift check) in seconds for the leader.

Type Float

Default `300.0`

Client

`mon client hung interval`

Description The client will try a new monitor every `N` seconds until it establishes a connection.

Type Double

Default `3.0`

`mon client ping interval`

Description The client will ping the monitor every `N` seconds.

Type Double

Default `10.0`

`mon client max log entries per message`

Description The maximum number of log entries a monitor will generate per client message.

Type Integer

Default `1000`

`mon client bytes`

Description The amount of client message data allowed in memory (in bytes).

Type 64-bit Integer Unsigned

Default `100ul << 20`

Miscellaneous

`mon max osd`

Description The maximum number of OSDs allowed in the cluster.

Type 32-bit Integer

Default `10000`

`mon globalid prealloc`

Description The number of global IDs to pre-allocate for clients and daemons in the cluster.

Type 32-bit Integer

Default `100`

`mon sync fs threshold`

Description Synchronize with the filesystem when writing the specified number of objects. Set it to `0` to disable it.

Type 32-bit Integer

Default `5`

`mon subscribe interval`

Description The refresh interval (in seconds) for subscriptions. The subscription mechanism enables obtaining the cluster maps and log information.

Type Double

Default `300`

`mon stat smooth intervals`

Description Ceph will smooth statistics over the last `N` PG maps.

Type Integer

Default `2`

`mon probe timeout`

Description Number of seconds the monitor will wait to find peers before bootstrapping.

Type Double

Default `2.0`

`mon daemon bytes`

Description The message memory cap for metadata server and OSD messages (in bytes).

Type 64-bit Integer Unsigned

Default `400ul << 20`

`mon max log entries per event`

Description The maximum number of log entries per event.

Type Integer

Default `4096`

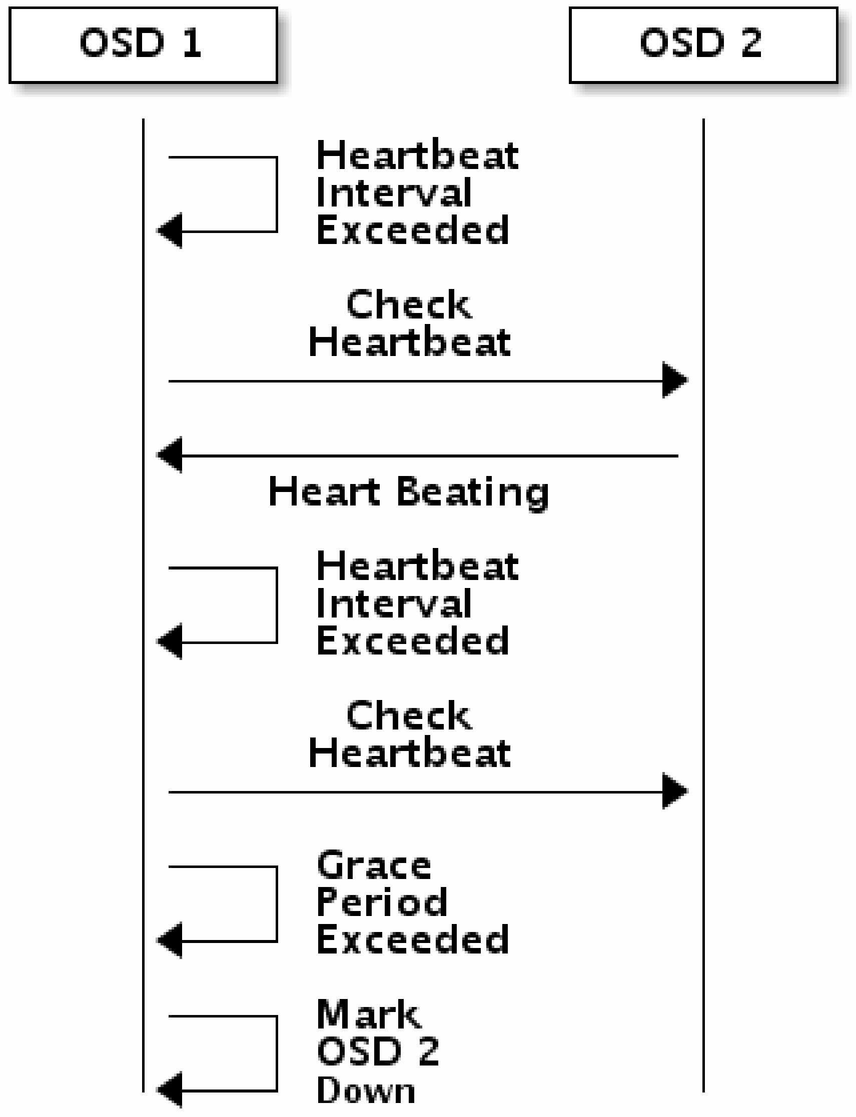
4.1.6 Configuring Monitor/OSD Interaction

After you have completed your initial Ceph configuration, you may deploy and run Ceph. When you execute a command such as `ceph health` or `ceph -s`, the *Ceph Monitor* reports on the current state of the *Ceph Storage Cluster*. The Ceph Monitor knows about the Ceph Storage Cluster by requiring reports from each *Ceph OSD Daemon*, and by receiving reports from Ceph OSD Daemons about the status of their neighboring Ceph OSD Daemons. If the Ceph Monitor doesn't receive reports, or if it receives reports of changes in the Ceph Storage Cluster, the Ceph Monitor updates the status of the *Ceph Cluster Map*.

Ceph provides reasonable default settings for Ceph Monitor/Ceph OSD Daemon interaction. However, you may override the defaults. The following sections describe how Ceph Monitors and Ceph OSD Daemons interact for the purposes of monitoring the Ceph Storage Cluster.

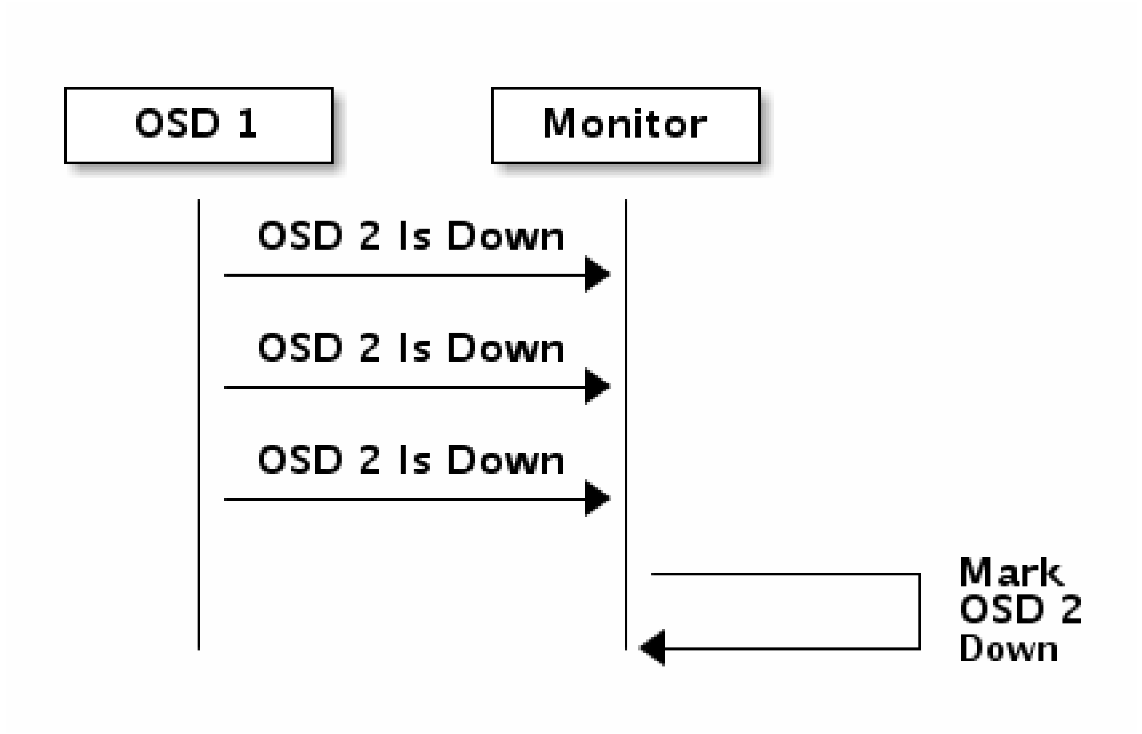
OSDs Check Heartbeats

Each Ceph OSD Daemon checks the heartbeat of other Ceph OSD Daemons every 6 seconds. You can change the heartbeat interval by adding an `osd heartbeat interval` setting under the `[osd]` section of your Ceph configuration file, or by setting the value at runtime. If a neighboring Ceph OSD Daemon doesn't show a heartbeat within a 20 second grace period, the Ceph OSD Daemon may consider the neighboring Ceph OSD Daemon `down` and report it back to a Ceph Monitor, which will update the Ceph Cluster Map. You may change this grace period by adding an `osd heartbeat grace` setting under the `[osd]` section of your Ceph configuration file, or by setting the value at runtime.



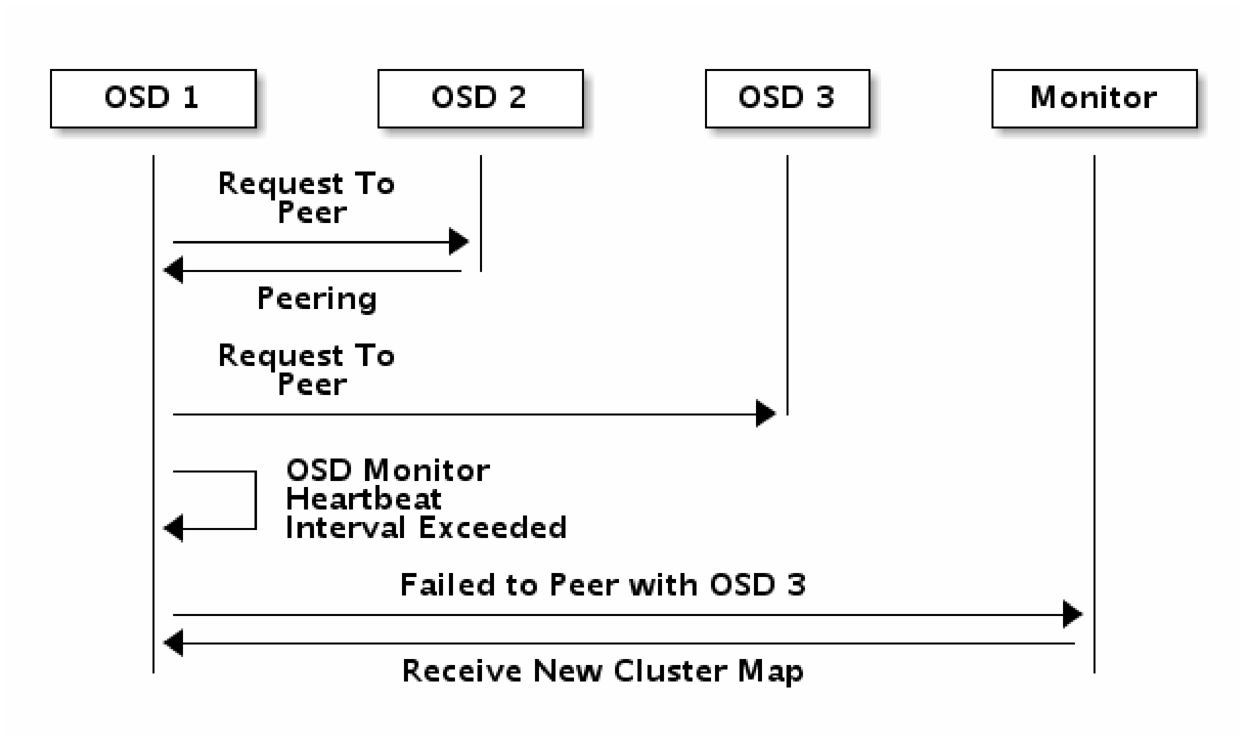
OSDs Report Down OSDs

By default, a Ceph OSD Daemon must report to the Ceph Monitors that another Ceph OSD Daemon is `down` three times before the Ceph Monitors acknowledge that the reported Ceph OSD Daemon is `down`. You can change the minimum number of `osd down` reports by adding an `mon osd min down reports` setting (`osd min down reports` prior to v0.62) under the `[mon]` section of your Ceph configuration file, or by setting the value at runtime. By default, only one Ceph OSD Daemon is required to report another Ceph OSD Daemon `down`. You can change the number of Ceph OSD Daemones required to report a Ceph OSD Daemon `down` to a Ceph Monitor by adding an `mon osd min down reporters` setting (`osd min down reporters` prior to v0.62) under the `[mon]` section of your Ceph configuration file, or by setting the value at runtime.



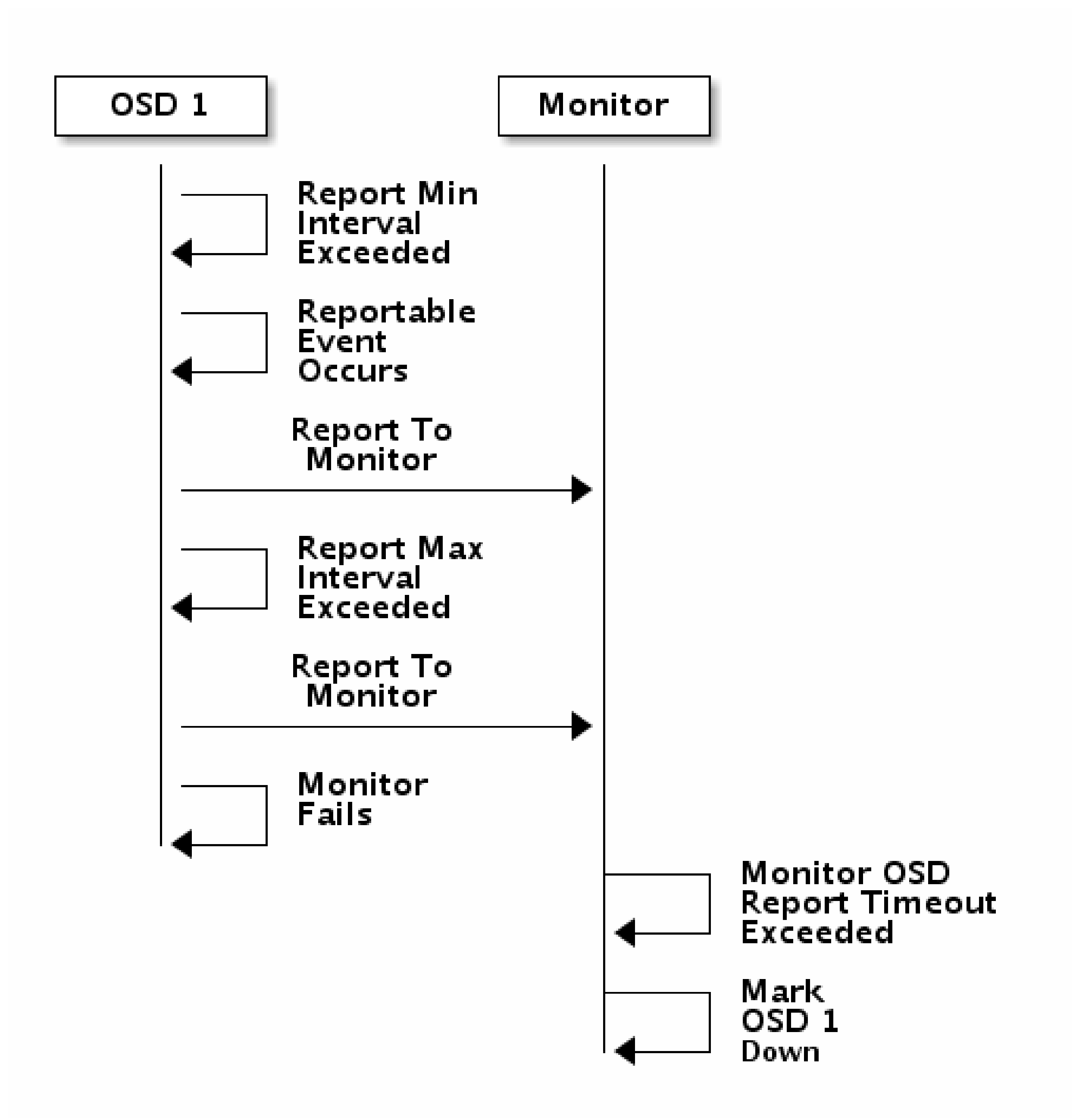
OSDs Report Peering Failure

If a Ceph OSD Daemon cannot peer with any of the Ceph OSD Daemons defined in its Ceph configuration file (or the cluster map), it will ping a Ceph Monitor for the most recent copy of the cluster map every 30 seconds. You can change the Ceph Monitor heartbeat interval by adding an `osd mon heartbeat interval` setting under the `[osd]` section of your Ceph configuration file, or by setting the value at runtime.



OSDs Report Their Status

If an Ceph OSD Daemon doesn't report to a Ceph Monitor, the Ceph Monitor will consider the Ceph OSD Daemon `down` after the `mon osd report timeout` elapses. A Ceph OSD Daemon sends a report to a Ceph Monitor when a reportable event such as a failure, a change in placement group stats, a change in `up_thru` or when it boots within 5 seconds. You can change the Ceph OSD Daemon minimum report interval by adding an `osd mon report interval min` setting under the `[osd]` section of your Ceph configuration file, or by setting the value at runtime. A Ceph OSD Daemon sends a report to a Ceph Monitor every 120 seconds irrespective of whether any notable changes occur. You can change the Ceph Monitor report interval by adding an `osd mon report interval max` setting under the `[osd]` section of your Ceph configuration file, or by setting the value at runtime.



Configuration Settings

When modifying heartbeat settings, you should include them in the `[global]` section of your configuration file.

Monitor Settings

`mon osd min up ratio`

Description The minimum ratio of `up` Ceph OSD Daemons before Ceph will mark Ceph OSD Daemons `down`.

Type Double

Default `.3`

`mon osd min in ratio`

Description The minimum ratio of `in` Ceph OSD Daemons before Ceph will mark Ceph OSD Daemons `out`.

Type Double

Default `.3`

`mon osd laggy halflife`

Description The number of seconds laggy estimates will decay.

Type Integer

Default `60*60`

`mon osd laggy weight`

Description The weight for new samples in laggy estimation decay.

Type Double

Default `0.3`

`mon osd adjust heartbeat grace`

Description If set to `true`, Ceph will scale based on laggy estimations.

Type Boolean

Default `true`

`mon osd adjust down out interval`

Description If set to `true`, Ceph will scaled based on laggy estimations.

Type Boolean

Default `true`

`mon osd auto mark in`

Description Ceph will mark any booting Ceph OSD Daemons as `in` the Ceph Storage Cluster.

Type Boolean

Default `false`

`mon osd auto mark auto out in`

Description Ceph will mark booting Ceph OSD Daemons auto marked `out` of the Ceph Storage Cluster as `in` the cluster.

Type Boolean

Default `true`

`mon osd auto mark new in`

Description Ceph will mark booting new Ceph OSD Daemons as `in` the Ceph Storage Cluster.

Type Boolean

Default `true`

`mon osd down out interval`

Description The number of seconds Ceph waits before marking a Ceph OSD Daemon `down` and `out` if it doesn't respond.

Type 32-bit Integer

Default `300`

`mon osd downout subtree limit`

Description The largest *CRUSH* unit type that Ceph will automatically mark `out`.

Type String

Default `rack`

`mon osd report timeout`

Description The grace period in seconds before declaring unresponsive Ceph OSD Daemons `down`.

Type 32-bit Integer

Default `900`

`mon osd min down reporters`

Description The minimum number of Ceph OSD Daemons required to report a `down` Ceph OSD Daemon.

Type 32-bit Integer

Default `1`

`mon osd min down reports`

Description The minimum number of times a Ceph OSD Daemon must report that another Ceph OSD Daemon is `down`.

Type 32-bit Integer

Default `3`

OSD Settings

`osd heartbeat address`

Description An Ceph OSD Daemon's network address for heartbeats.

Type Address

Default The host address.

`osd heartbeat interval`

Description How often an Ceph OSD Daemon pings its peers (in seconds).

Type 32-bit Integer

Default `6`

`osd heartbeat grace`

Description The elapsed time when a Ceph OSD Daemon hasn't shown a heartbeat that the Ceph Storage Cluster considers it `down`.

Type 32-bit Integer

Default `20`

`osd mon heartbeat interval`

Description How often the Ceph OSD Daemon pings a Ceph Monitor if it has no Ceph OSD Daemon peers.

Type 32-bit Integer

Default `30`

`osd mon report interval max`

Description The maximum time in seconds that a Ceph OSD Daemon can wait before it must report to a Ceph Monitor.

Type 32-bit Integer

Default `120`

`osd mon report interval min`

Description The minimum number of seconds a Ceph OSD Daemon may wait from startup or another reportable event before reporting to a Ceph Monitor.

Type 32-bit Integer

Default `5`

Valid Range Should be less than `osd mon report interval max`

`osd mon ack timeout`

Description The number of seconds to wait for a Ceph Monitor to acknowledge a request for statistics.

Type 32-bit Integer

Default `30`

4.1.7 OSD Config Reference

You can configure Ceph OSD Daemons in the Ceph configuration file, but Ceph OSD Daemons can use the default values and a very minimal configuration. A minimal Ceph OSD Daemon configuration sets `osd journal size` and `osd host`, and uses default values for nearly everything else.

Ceph OSD Daemons are numerically identified in incremental fashion, beginning with `0` using the following convention.

```
osd.0
osd.1
osd.2
```

In a configuration file, you may specify settings for all Ceph OSD Daemons in the cluster by adding configuration settings to the `[osd]` section of your configuration file. To add settings directly to a specific Ceph OSD Daemon (e.g., `osd host`), enter it in an OSD-specific section of your configuration file. For example:

```
[osd]
        osd journal size = 1024

[osd.0]
        osd host = osd-host-a

[osd.1]
        osd host = osd-host-b
```

General Settings

The following settings provide an Ceph OSD Daemon's ID, and determine paths to data and journals. Ceph deployment scripts typically generate the UUID automatically. We **DO NOT** recommend changing the default paths for data or journals, as it makes it more problematic to troubleshoot Ceph later.

The journal size should be at least twice the product of the expected drive speed multiplied by `filestore max sync interval`. However, the most common practice is to partition the journal drive (often an SSD), and mount it such that Ceph uses the entire partition for the journal.

`osd uuid`

Description The universally unique identifier (UUID) for the Ceph OSD Daemon.

Type UUID

Default The UUID.

Note The `osd uuid` applies to a single Ceph OSD Daemon. The `fsid` applies to the entire cluster.

`osd data`

Description The path to the OSDs data. You must create the directory when deploying Ceph. You should mount a drive for OSD data at this mount point. We do not recommend changing the default.

Type String

Default `/var/lib/ceph/osd/$cluster-$id`

`osd max write size`

Description The maximum size of a write in megabytes.

Type 32-bit Integer

Default `90`

`osd client message size cap`

Description The largest client data message allowed in memory.

Type 64-bit Integer Unsigned

Default 500MB default. `500*1024L*1024L`

`osd class dir`

Description The class path for RADOS class plug-ins.

Type String

Default `$libdir/rados-classes`

File System Settings

Ceph builds and mounts file systems which are used for Ceph OSDs.

`osd mkfs options {fs-type}`

Description Options used when creating a new Ceph OSD of type {fs-type}.

Type String

Default for xfs `-f -i 2048`

Default for other file systems {empty string}

For example:: `osd mkfs options xfs = -f -d agcount=24`

`osd mount options {fs-type}`

Description Options used when mounting a Ceph OSD of type {fs-type}.

Type String

Default for xfs `rw,noatime,inode64`

Default for other file systems `rw, noatime`

For example:: `osd mount options xfs = rw, noatime, inode64, logbufs=8`

Journal Settings

By default, Ceph expects that you will store an Ceph OSD Daemons journal with the following path:

```
/var/lib/ceph/osd/$cluster-$id/journal
```

Without performance optimization, Ceph stores the journal on the same disk as the Ceph OSD Daemons data. An Ceph OSD Daemon optimized for performance may use a separate disk to store journal data (e.g., a solid state drive delivers high performance journaling).

Ceph's default `osd journal size` is 0, so you will need to set this in your `ceph.conf` file. A journal size should find the product of the `filestore max sync interval` and the expected throughput, and multiply the product by two (2):

```
osd journal size = {2 * (expected throughput * filestore max sync interval)}
```

The expected throughput number should include the expected disk throughput (i.e., sustained data transfer rate), and network throughput. For example, a 7200 RPM disk will likely have approximately 100 MB/s. Taking the `min()` of the disk and network throughput should provide a reasonable expected throughput. Some users just start off with a 10GB journal size. For example:

```
osd journal size = 10000
```

`osd journal`

Description The path to the OSD's journal. This may be a path to a file or a block device (such as a partition of an SSD). If it is a file, you must create the directory to contain it. We recommend using a drive separate from the `osd data` drive.

Type String

Default `/var/lib/ceph/osd/$cluster-$id/journal`

`osd journal size`

Description The size of the journal in megabytes. If this is 0, and the journal is a block device, the entire block device is used. Since v0.54, this is ignored if the journal is a block device, and the entire block device is used.

Type 32-bit Integer

Default `5120`

Recommended Begin with 1GB. Should be at least twice the product of the expected speed multiplied by `filestore max sync interval`.

See Journal Config Reference for additional details.

Monitor OSD Interaction

Ceph OSD Daemons check each other's heartbeats and report to monitors periodically. Ceph can use default values in many cases. However, if your network has latency issues, you may need to adopt longer intervals. See Configuring Monitor/OSD Interaction for a detailed discussion of heartbeats.

Data Placement

See Pool & PG Config Reference for details.

Scrubbing

In addition to making multiple copies of objects, Ceph insures data integrity by scrubbing placement groups. Ceph scrubbing is analogous to `fsck` on the object storage layer. For each placement group, Ceph generates a catalog of all objects and compares each primary object and its replicas to ensure that no objects are missing or mismatched. Light scrubbing (daily) checks the object size and attributes. Deep scrubbing (weekly) reads the data and uses checksums to ensure data integrity.

Scrubbing is important for maintaining data integrity, but it can reduce performance. You can adjust the following settings to increase or decrease scrubbing operations.

`osd max scrubs`

Description The maximum number of simultaneous scrub operations for a Ceph OSD Daemon.

Type 32-bit Int

Default `1`

`osd scrub thread timeout`

Description The maximum time in seconds before timing out a scrub thread.

Type 32-bit Integer

Default `60`

`osd scrub finalize thread timeout`

Description The maximum time in seconds before timing out a scrub finalize thread.

Type 32-bit Integer

Default `60*10`

`osd scrub load threshold`

Description The maximum load. Ceph will not scrub when the system load (as defined by `getloadavg()`) is higher than this number. Default is `0.5`.

Type Float

Default `0.5`

`osd scrub min interval`

Description The maximum interval in seconds for scrubbing the Ceph OSD Daemon when the Ceph Storage Cluster load is low.

Type Float

Default Once per day. `60*60*24`

`osd scrub max interval`

Description The maximum interval in seconds for scrubbing the Ceph OSD Daemon irrespective of cluster load.

Type Float

Default Once per week. `7*60*60*24`

`osd deep scrub interval`

Description The interval for "deep" scrubbing (fully reading all data). The `osd scrub load threshold` does not affect this setting.

Type Float

Default Once per week. `60*60*24*7`

`osd deep scrub stride`

Description Read size when doing a deep scrub.

Type 32-bit Integer

Default 512 KB. `524288`

Operations

Operations settings allow you to configure the number of threads for servicing requests. If you set `osd op threads` to `0`, it disables multi-threading. By default, Ceph uses two threads with a 30 second timeout and a 30 second complaint time if an operation doesn't complete within those time parameters. You can set operations priority weights between client operations and recovery operations to ensure optimal performance during recovery.

`osd op threads`

Description The number of threads to service Ceph OSD Daemon operations. Set to `0` to disable it. Increasing the number may increase the request processing rate.

Type 32-bit Integer

Default `2`

`osd client op priority`

Description The priority set for client operations. It is relative to `osd recovery op priority`.

Type 32-bit Integer

Default `63`

Valid Range 1-63

`osd recovery op priority`

Description The priority set for recovery operations. It is relative to `osd client op priority`.

Type 32-bit Integer

Default `10`

Valid Range 1-63

`osd op thread timeout`

Description The Ceph OSD Daemon operation thread timeout in seconds.

Type 32-bit Integer

Default `30`

`osd op complaint time`

Description An operation becomes complaint worthy after the specified number of seconds have elapsed.

Type Float

Default `30`

`osd disk threads`

Description The number of disk threads, which are used to perform background disk intensive OSD operations such as scrubbing and snap trimming.

Type 32-bit Integer

Default `1`

`osd disk thread ioprio class`

Description Warning: it will only be used if both `osd disk thread ioprio class` and `osd disk thread ioprio priority` are set to a non default value. Sets the ioprio_set(2) I/O scheduling `class` for the disk thread. Acceptable values are `idle`, `be` or `rt`. The `idle` class means the disk thread will have lower priority than any other thread in the OSD. This is useful to slow down scrubbing on an OSD that is busy handling client operations. `be` is the default and is the same priority as all other threads in the OSD. `rt` means the disk thread will have precendence over all other threads in the OSD. This is useful if scrubbing is much needed and must make progress at the expense of client operations. Note: Only works with the Linux Kernel CFQ scheduler.

Type String

Default the empty string

`osd disk thread ioprio priority`

Description Warning: it will only be used if both `osd disk thread ioprio class` and `osd disk thread ioprio priority` are set to a non default value. It sets the ioprio_set(2) I/O scheduling `priority` of the disk thread ranging from 0 (highest) to 7 (lowest). If all OSDs on a given host were in class `idle` and compete for I/O (i.e. due to controller congestion), it can be used to lower the disk thread priority of one OSD to 7 so that another OSD with priority 0 can potentially scrub faster. Note: Only works with the Linux Kernel CFQ scheduler.

Type Integer in the range of 0 to 7 or -1 if not to be used.

Default `-1`

`osd op history size`

Description The maximum number of completed operations to track.

Type 32-bit Unsigned Integer

Default `20`

`osd op history duration`

Description The oldest completed operation to track.

Type 32-bit Unsigned Integer

Default `600`

`osd op log threshold`

Description How many operations logs to display at once.

Type 32-bit Integer

Default `5`

Backfilling

When you add or remove Ceph OSD Daemons to a cluster, the CRUSH algorithm will want to rebalance the cluster by moving placement groups to or from Ceph OSD Daemons to restore the balance. The process of migrating placement groups and the objects they contain can reduce the cluster's operational performance considerably. To maintain operational performance, Ceph performs this migration with 'backfilling', which allows Ceph to set backfill operations to a lower priority than requests to read or write data.

`osd max backfills`

Description The maximum number of backfills allowed to or from a single OSD.

Type 64-bit Unsigned Integer

Default `10`

`osd backfill scan min`

Description The minimum number of objects per backfill scan.

Type 32-bit Integer

Default `64`

`osd backfill scan max`

Description The maximum number of objects per backfill scan.

Type 32-bit Integer

Default `512`

`osd backfill full ratio`

Description Refuse to accept backfill requests when the Ceph OSD Daemon's full ratio is above this value.

Type Float

Default `0.85`

`osd backfill retry interval`

Description The number of seconds to wait before retrying backfill requests.

Type Double

Default `10.0`

OSD Map

OSD maps reflect the OSD daemons operating in the cluster. Over time, the number of map epochs increases. Ceph provides some settings to ensure that Ceph performs well as the OSD map grows larger.

`osd map dedup`

Description Enable removing duplicates in the OSD map.

Type Boolean

Default `true`

`osd map cache size`

Description The number of OSD maps to keep cached.

Type 32-bit Integer

Default `500`

`osd map cache bl size`

Description The size of the in-memory OSD map cache in OSD daemons.

Type 32-bit Integer

Default `50`

`osd map cache bl inc size`

Description The size of the in-memory OSD map cache incrementals in OSD daemons.

Type 32-bit Integer

Default `100`

`osd map message max`

Description The maximum map entries allowed per MOSDMap message.

Type 32-bit Integer

Default `100`

Recovery

When the cluster starts or when a Ceph OSD Daemon crashes and restarts, the OSD begins peering with other Ceph OSD Daemons before writes can occur. See Monitoring OSDs and PGs for details.

If a Ceph OSD Daemon crashes and comes back online, usually it will be out of sync with other Ceph OSD Daemons containing more recent versions of objects in the placement groups. When this happens, the Ceph OSD Daemon goes into recovery mode and seeks to get the latest copy of the data and bring its map back up to date. Depending upon how long the Ceph OSD Daemon was down, the OSD's objects and placement groups may be significantly out of date. Also, if a failure domain went down (e.g., a rack), more than one Ceph OSD Daemon may come back online at the same time. This can make the recovery process time consuming and resource intensive.

To maintain operational performance, Ceph performs recovery with limitations on the number recovery requests, threads and object chunk sizes which allows Ceph perform well in a degraded state.

`osd recovery delay start`

Description After peering completes, Ceph will delay for the specified number of seconds before starting to recover objects.

Type Float

Default `0`

`osd recovery max active`

Description The number of active recovery requests per OSD at one time. More requests will accelerate recovery, but the requests places an increased load on the cluster.

Type 32-bit Integer

Default `15`

`osd recovery max chunk`

Description The maximum size of a recovered chunk of data to push.

Type 64-bit Integer Unsigned

Default `8 << 20`

`osd recovery threads`

Description The number of threads for recovering data.

Type 32-bit Integer

Default `1`

`osd recovery thread timeout`

Description The maximum time in seconds before timing out a recovery thread.

Type 32-bit Integer

Default `30`

`osd recover clone overlap`

Description Preserves clone overlap during recovery. Should always be set to `true`.

Type Boolean

Default `true`

Miscellaneous

`osd snap trim thread timeout`

Description The maximum time in seconds before timing out a snap trim thread.

Type 32-bit Integer

Default `60*60*1`

`osd backlog thread timeout`

Description The maximum time in seconds before timing out a backlog thread.

Type 32-bit Integer

Default `60*60*1`

`osd default notify timeout`

Description The OSD default notification timeout (in seconds).

Type 32-bit Integer Unsigned

Default `30`

`osd check for log corruption`

Description Check log files for corruption. Can be computationally expensive.

Type Boolean

Default `false`

`osd remove thread timeout`

Description The maximum time in seconds before timing out a remove OSD thread.

Type 32-bit Integer

Default `60*60`

`osd command thread timeout`

Description The maximum time in seconds before timing out a command thread.

Type 32-bit Integer

Default `10*60`

`osd command max records`

Description Limits the number of lost objects to return.

Type 32-bit Integer

Default `256`

`osd auto upgrade tmap`

Description Uses `tmap` for `omap` on old objects.

Type Boolean

Default `true`

`osd tmapput sets users tmap`

Description Uses `tmap` for debugging only.

Type Boolean

Default `false`

`osd preserve trimmed log`

Description Preserves trimmed log files, but uses more disk space.

Type Boolean

Default `false`

4.1.8 Filestore Config Reference

`filestore debug omap check`

Description Debugging check on synchronization. Expensive. For debugging only.

Type Boolean

Required No

Default `0`

Extended Attributes

Extended Attributes (XATTRs) are an important aspect in your configuration. Some file systems have limits on the number of bytes stored in XATTRS. Additionally, in some cases, the filesystem may not be as fast as an alternative method of storing XATTRs. The following settings may help improve performance by using a method of storing XATTRs that is extrinsic to the underlying filesystem.

Ceph XATTRs are stored as `inline xattr`, using the XATTRs provided by the underlying file system, if it does not impose a size limit. If there is a size limit (4KB total on ext4, for instance), some Ceph XATTRs will be stored in an key/value database (aka `omap`) when the `filestore max inline xattr size` or `filestore max inline xattrs` threshold are reached.

`filestore xattr use omap`

Description Use object map for XATTRS. Set to `true` for `ext4` file systems.

Type Boolean

Required No

Default `false`

`filestore max inline xattr size`

Description The maximimum size of an XATTR stored in the filesystem (i.e., XFS, btrfs, ext4, etc.) per object. Should not be larger than the filesytem can handle.

Type Unsigned 32-bit Integer

Required No

Default `512`

`filestore max inline xattrs`

Description The maximum number of XATTRs stored in the fileystem per object.

Type 32-bit Integer

Required No

Default `2`

Synchronization Intervals

Periodically, the filestore needs to quiesce writes and synchronize the filesystem, which creates a consistent commit point. It can then free journal entries up to the commit point. Synchronizing more frequently tends to reduce the time required to perform synchronization, and reduces the amount of data that needs to remain in the journal. Less frequent synchronization allows the backing filesystem to coalesce small writes and metadata updates more optimally– potentially resulting in more efficient synchronization.

`filestore max sync interval`

Description The maximum interval in seconds for synchronizing the filestore.

Type Double

Required No

Default `5`

`filestore min sync interval`

Description The minimum interval in seconds for synchronizing the filestore.

Type Double

Required No

Default `.01`

Flusher

The filestore flusher forces data from large writes to be written out using `sync file range` before the sync in order to (hopefully) reduce the cost of the eventual sync. In practice, disabling 'filestore flusher' seems to improve performance in some cases.

`filestore flusher`

Description Enables the filestore flusher.

Type Boolean

Required No

Default `false`

Deprecated since version v.65.

`filestore flusher max fds`

Description Sets the maximum number of file descriptors for the flusher.

Type Integer

Required No

Default `512`

Deprecated since version v.65.

`filestore sync flush`

Description Enables the synchronization flusher.

Type Boolean

Required No

Default `false`

Deprecated since version v.65.

`filestore fsync flushes journal data`

Description Flush journal data during filesystem synchronization.

Type Boolean

Required No

Default `false`

Queue

The following settings provide limits on the size of filestore queue.

`filestore queue max ops`

Description Defines the maximum number of in progress operations the file store accepts before blocking on queuing new operations.

Type Integer

Required No. Minimal impact on performance.

Default `500`

`filestore queue max bytes`

Description The maximum number of bytes for an operation.

Type Integer

Required No

Default `100 << 20`

`filestore queue committing max ops`

Description The maximum number of operations the filestore can commit.

Type Integer

Required No

Default `500`

`filestore queue committing max bytes`

Description The maximum number of bytes the filestore can commit.

Type Integer

Required No

Default `100 << 20`

Timeouts

`filestore op threads`

Description The number of filesystem operation threads that execute in parallel.

Type Integer

Required No

Default `2`

`filestore op thread timeout`

Description The timeout for a filesystem operation thread (in seconds).

Type Integer

Required No

Default `60`

`filestore op thread suicide timeout`

Description The timeout for a commit operation before cancelling the commit (in seconds).

Type Integer

Required No

Default `180`

B-Tree Filesystem

`filestore btrfs snap`

Description Enable snapshots for a `btrfs` filestore.

Type Boolean

Required No. Only used for `btrfs`.

Default `true`

`filestore btrfs clone range`

Description Enable cloning ranges for a `btrfs` filestore.

Type Boolean

Required No. Only used for `btrfs`.

Default `true`

Journal

`filestore journal parallel`

Description Enables parallel journaling, default for btrfs.

Type Boolean

Required No

Default `false`

`filestore journal writeahead`

Description Enables writeahead journaling, default for xfs.

Type Boolean

Required No

Default `false`

`filestore journal trailing`

Description Deprecated, never use.

Type Boolean

Required No

Default `false`

Misc

`filestore merge threshold`

Description Min number of files in a subdir before merging into parent NOTE: A negative value means to disable subdir merging

Type Integer

Required No

Default `10`

`filestore split multiple`

Description `filestore_split_multiple * abs(filestore_merge_threshold) * 16` is the maximum number of files in a subdirectory before splitting into child directories.

Type Integer

Required No

Default `2`

`filestore update to`

Description Limits filestore auto upgrade to specified version.

Type Integer

Required No

Default `1000`

`filestore blackhole`

Description Drop any new transactions on the floor.

Type Boolean

Required No

Default `false`

`filestore dump file`

Description File onto which store transaction dumps.

Type Boolean

Required No

Default `false`

`filestore kill at`

Description inject a failure at the n'th opportunity

Type String

Required No

Default `false`

`filestore fail eio`

Description Fail/Crash on eio.

Type Boolean

Required No

Default `true`

4.1.9 KeyValueStore Config Reference

`KeyValueStore` is an alternative OSD backend compared to FileStore. Currently, it uses LevelDB as backend. `KeyValueStore` doesn't need journal device. Each operation will flush into the backend directly.

`keyvaluestore backend`

Description The backend used by `KeyValueStore`.

Type String

Required No

Default `leveldb`

Queue

The following settings provide limits on the size of the `KeyValueStore` queue.

`keyvaluestore queue max ops`

Description Defines the maximum number of operations in progress the `KeyValueStore` accepts before blocking on queuing new operations.

Type Integer

Required No. Minimal impact on performance.

Default `50`

`keyvaluestore queue max bytes`

Description The maximum number of bytes for an operation.

Type Integer

Required No

Default `100 << 20`

Thread

`keyvaluestore op threads`

Description The number of `KeyValueStore` operation threads that execute in parallel.

Type Integer

Required No

Default `2`

`keyvaluestore op thread timeout`

Description The timeout for a `KeyValueStore` operation thread (in seconds).

Type Integer

Required No

Default `60`

`keyvaluestore op thread suicide timeout`

Description The timeout for a commit operation before canceling the commit (in seconds).

Type Integer

Required No

Default `180`

Misc

`keyvaluestore default strip size`

Description Each object will be split into multiple key/value pairs and stored in the backend. **Note:** The size of the workload has a significant impact on performance.

Type Integer

Required No

Default `4096`

`keyvaluestore header cache size`

Description The size of the header cache (identical to `inode` in the local filesystem). A larger cache size enhances performance.

Type Integer

Required No

Default `4096`

4.1.10 Journal Config Reference

Ceph OSDs use a journal for two reasons: speed and consistency.

- **Speed:** The journal enables the Ceph OSD Daemon to commit small writes quickly. Ceph writes small, random i/o to the journal sequentially, which tends to speed up bursty workloads by allowing the backing filesystem more time to coalesce writes. The Ceph OSD Daemon's journal, however, can lead to spiky performance with short spurts of high-speed writes followed by periods without any write progress as the filesystem catches up to the journal.
- **Consistency:** Ceph OSD Daemons require a filesystem interface that guarantees atomic compound operations. Ceph OSD Daemons write a description of the operation to the journal and apply the operation to the filesystem. This enables atomic updates to an object (for example, placement group metadata). Every few seconds–between `filestore max sync interval` and `filestore min sync interval`–the Ceph OSD Daemon stops writes and synchronizes the journal with the filesystem, allowing Ceph OSD Daemons to trim operations from the journal and reuse the space. On failure, Ceph OSD Daemons replay the journal starting after the last synchronization operation.

Ceph OSD Daemons support the following journal settings:

`journal dio`

Description Enables direct i/o to the journal. Requires `journal block align` set to `true`.

Type Boolean

Required Yes when using `aio`.

Default `true`

`journal aio`

Changed in version 0.61: Cuttlefish

Description Enables using `libaio` for asynchronous writes to the journal. Requires `journal dio` set to `true`.

Type Boolean

Required No.

Default Version 0.61 and later, `true`. Version 0.60 and earlier, `false`.

`journal block align`

Description Block aligns write operations. Required for `dio` and `aio`.

Type Boolean

Required Yes when using `dio` and `aio`.

Default `true`

`journal max write bytes`

Description The maximum number of bytes the journal will write at any one time.

Type Integer

Required No

Default `10 << 20`

`journal max write entries`

Description The maximum number of entries the journal will write at any one time.

Type Integer

Required No

Default `100`

`journal queue max ops`

Description The maximum number of operations allowed in the queue at any one time.

Type Integer

Required No

Default `500`

`journal queue max bytes`

Description The maximum number of bytes allowed in the queue at any one time.

Type Integer

Required No

Default `10 << 20`

`journal align min size`

Description Align data payloads greater than the specified minimum.

Type Integer

Required No

Default `64 << 10`

`journal zero on create`

Description Causes the file store to overwrite the entire journal with `0`‘s during `mkfs`.

Type Boolean

Required No

Default `false`

4.1.11 Pool, PG and CRUSH Config Reference

When you create pools and set the number of placement groups for the pool, Ceph uses default values when you don’t specifically override the defaults. **We recommend** overridding some of the defaults. Specifically, we recommend setting a pool’s replica size and overriding the default number of placement groups. You can specifically set these values when running pool commands. You can also override the defaults by adding new ones in the [global] section of your Ceph configuration file.

```
[global]

        # By default, Ceph makes 3 replicas of objects. If you want to make four
        # copies of an object the default value--a primary copy and three replica
        # copies--reset the default values as shown in 'osd pool default size'.
        # If you want to allow Ceph to write a lesser number of copies in a degraded
        # state, set 'osd pool default min size' to a number less than the
        # 'osd pool default size' value.

        osd pool default size = 4  # Write an object 4 times.
        osd pool default min size = 1 # Allow writing one copy in a degraded state.

        # Ensure you have a realistic number of placement groups. We recommend
        # approximately 100 per OSD. E.g., total number of OSDs multiplied by 100
        # divided by the number of replicas (i.e., osd pool default size). So for
        # 10 OSDs and osd pool default size = 4, we'd recommend approximately
        # (100 * 10) / 4 = 250.

        osd pool default pg num = 250
        osd pool default pgp num = 250
```

`mon max pool pg num`

Description The maximum number of placement groups per pool.

Type Integer

Default `65536`

`mon pg create interval`

Description Number of seconds between PG creation in the same Ceph OSD Daemon.

Type Float

Default `30.0`

`mon pg stuck threshold`

Description Number of seconds after which PGs can be considered as being stuck.

Type 32-bit Integer

Default `300`

`osd pg bits`

Description Placement group bits per Ceph OSD Daemon.

Type 32-bit Integer

Default `6`

`osd pgp bits`

Description The number of bits per Ceph OSD Daemon for PGPs.

Type 32-bit Integer

Default `6`

`osd crush chooseleaf type`

Description The bucket type to use for `chooseleaf` in a CRUSH rule. Uses ordinal rank rather than name.

Type 32-bit Integer

Default `1`. Typically a host containing one or more Ceph OSD Daemons.

`osd pool default crush replicated ruleset`

Description The default CRUSH ruleset to use when creating a replicated pool.

Type 8-bit Integer

Default `0`

`osd pool erasure code stripe width`

Description Sets the desired size, in bytes, of an object stripe on every erasure coded pools. Every object if size S will be stored as N stripes and each stripe will be encoded/decoded individually.

Type Unsigned 32-bit Integer

Default `4096`

`osd pool default size`

Description Sets the number of replicas for objects in the pool. The default value is the same as `ceph osd pool set {pool-name} size {size}`.

Type 32-bit Integer

Default `3`

`osd pool default min size`

Description Sets the minimum number of written replicas for objects in the pool in order to acknowledge a write operation to the client. If minimum is not met, Ceph will not acknowledge the write to the client. This setting ensures a minimum number of replicas when operating in `degraded` mode.

Type 32-bit Integer

Default `0`, which means no particular minimum. If `0`, minimum is `size - (size / 2)`.

`osd pool default pg num`

Description The default number of placement groups for a pool. The default value is the same as `pg_num` with `mkpool`.

Type 32-bit Integer

Default `8`

`osd pool default pgp num`

Description The default number of placement groups for placement for a pool. The default value is the same as `pgp_num` with `mkpool`. PG and PGP should be equal (for now).

Type 32-bit Integer

Default `8`

`osd pool default flags`

Description The default flags for new pools.

Type 32-bit Integer

Default `0`

`osd max pgls`

Description The maximum number of placement groups to list. A client requesting a large number can tie up the Ceph OSD Daemon.

Type Unsigned 64-bit Integer

Default `1024`

Note Default should be fine.

`osd min pg log entries`

Description The minimum number of placement group logs to maintain when trimming log files.

Type 32-bit Int Unsigned

Default `1000`

`osd default data pool replay window`

Description The time (in seconds) for an OSD to wait for a client to replay a request.

Type 32-bit Integer

Default `45`

4.1.12 Messaging

`ms tcp nodelay`

Description Disables nagle's algorithm on messenger tcp sessions.

Type Boolean

Required No

Default `true`

`ms initial backoff`

Description The initial time to wait before reconnecting on a fault.

Type Double

Required No

Default `.2`

`ms max backoff`

Description The maximum time to wait before reconnecting on a fault.

Type Double

Required No

Default `15.0`

`ms nocrc`

Description Disables crc on network messages. May increase performance if cpu limited.

Type Boolean

Required No

Default `false`

`ms die on bad msg`

> **Description** Debug option; do not configure.
>
> **Type** Boolean
>
> **Required** No
>
> **Default** `false`

`ms dispatch throttle bytes`

> **Description** Throttles total size of messages waiting to be dispatched.
>
> **Type** 64-bit Unsigned Integer
>
> **Required** No
>
> **Default** `100 << 20`

`ms bind ipv6`

> **Description** Enable if you want your daemons to bind to IPv6 address instead of IPv4 ones. (Not required if you specify a daemon or cluster IP.)
>
> **Type** Boolean
>
> **Required** No
>
> **Default** `false`

`ms rwthread stack bytes`

> **Description** Debug option for stack size; do not configure.
>
> **Type** 64-bit Unsigned Integer
>
> **Required** No
>
> **Default** `1024 << 10`

`ms tcp read timeout`

> **Description** Controls how long (in seconds) the messenger will wait before closing an idle connection.
>
> **Type** 64-bit Unsigned Integer
>
> **Required** No
>
> **Default** `900`

`ms inject socket failures`

> **Description** Debug option; do not configure.
>
> **Type** 64-bit Unsigned Integer
>
> **Required** No
>
> **Default** `0`

4.1.13 General Config Reference

`fsid`

> **Description** The filesystem ID. One per cluster.
>
> **Type** UUID

Required No.

Default N/A. Usually generated by deployment tools.

`admin socket`

Description The socket for executing administrative commands on a daemon, irrespective of whether Ceph Monitors have established a quorum.

Type String

Required No

Default `/var/run/ceph/$cluster-$name.asok`

`pid file`

Description The file in which the mon, osd or mds will write its PID. For instance, `/var/run/$cluster/$type.$id.pid` will create /var/run/ceph/mon.a.pid for the `mon` with id `a` running in the `ceph` cluster. The `pid file` is removed when the daemon stops gracefully. If the process is not daemonized (i.e. runs with the `-f` or `-d` option), the `pid file` is not created.

Type String

Required No

Default No

`chdir`

Description The directory Ceph daemons change to once they are up and running. Default `/` directory recommended.

Type String

Required No

Default `/`

`max open files`

Description If set, when the *Ceph Storage Cluster* starts, Ceph sets the `max open fds` at the OS level (i.e., the max # of file descriptors). It helps prevents Ceph OSD Daemons from running out of file descriptors.

Type 64-bit Integer

Required No

Default `0`

`fatal signal handlers`

Description If set, we will install signal handlers for SEGV, ABRT, BUS, ILL, FPE, XCPU, XFSZ, SYS signals to generate a useful log message

Type Boolean

Default `true`

4.2 Ceph Deployment

The `ceph-deploy` tool is a way to deploy Ceph relying only upon SSH access to the servers, `sudo`, and some Python. It runs on your workstation, and does not require servers, databases, or any other tools. If you set up and tear down Ceph clusters a lot, and want minimal extra bureaucracy, `ceph-deploy` is an ideal tool. The `ceph-deploy` tool is not a generic deployment system. It was designed exclusively for Ceph users who want to get Ceph up and running quickly with sensible initial configuration settings without the overhead of installing Chef, Puppet or Juju. Users who want fine-control over security settings, partitions or directory locations should use a tool such as Juju, Puppet, Chef or Crowbar.

With `ceph-deploy`, you can develop scripts to install Ceph packages on remote hosts, create a cluster, add monitors, gather (or forget) keys, add OSDs and metadata servers, configure admin hosts, and tear down the clusters.

4.2.1 Preflight Checklist

New in version 0.60.

This **Preflight Checklist** will help you prepare an admin node for use with `ceph-deploy`, and server nodes for use with passwordless `ssh` and `sudo`.

Before you can deploy Ceph using `ceph-deploy`, you need to ensure that you have a few things set up first on your admin node and on nodes running Ceph daemons.

Install an Operating System

Install a recent release of Debian or Ubuntu (e.g., 12.04 LTS, 14.04 LTS) on your nodes. For additional details on operating systems or to use other operating systems other than Debian or Ubuntu, see OS Recommendations.

Install an SSH Server

The `ceph-deploy` utility requires `ssh`, so your server node(s) require an SSH server.

```
sudo apt-get install openssh-server
```

Create a User

Create a user on nodes running Ceph daemons.

Tip: We recommend a username that brute force attackers won't guess easily (e.g., something other than `root`, `ceph`, etc).

```
ssh user@ceph-server
sudo useradd -d /home/ceph -m ceph
sudo passwd ceph
```

`ceph-deploy` installs packages onto your nodes. This means that the user you create requires passwordless `sudo` privileges.

Note: We **DO NOT** recommend enabling the `root` password for security reasons.

To provide full privileges to the user, add the following to `/etc/sudoers.d/ceph`.

```
echo "ceph ALL = (root) NOPASSWD:ALL" | sudo tee /etc/sudoers.d/ceph
sudo chmod 0440 /etc/sudoers.d/ceph
```

Configure SSH

Configure your admin machine with password-less SSH access to each node running Ceph daemons (leave the passphrase empty).

```
ssh-keygen
Generating public/private key pair.
Enter file in which to save the key (/ceph-client/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /ceph-client/.ssh/id_rsa.
Your public key has been saved in /ceph-client/.ssh/id_rsa.pub.
```

Copy the key to each node running Ceph daemons:

```
ssh-copy-id ceph@ceph-server
```

Modify your ~/.ssh/config file of your admin node so that it defaults to logging in as the user you created when no username is specified.

```
Host ceph-server
        Hostname ceph-server.fqdn-or-ip-address.com
        User ceph
```

Install ceph-deploy

To install `ceph-deploy`, execute the following:

```
wget -q -O- 'https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc' | sudo apt-key add -
echo deb http://ceph.com/debian-dumpling/ $(lsb_release -sc) main | sudo tee /etc/apt/sources.list.d/
sudo apt-get update
sudo apt-get install ceph-deploy
```

Ensure Connectivity

Ensure that your Admin node has connectivity to the network and to your Server node (e.g., ensure `iptables`, `ufw` or other tools that may prevent connections, traffic forwarding, etc. to allow what you need).

Once you have completed this pre-flight checklist, you are ready to begin using `ceph-deploy`.

4.2.2 Package Management

Install

To install Ceph packages on your cluster hosts, open a command line on your client machine and type the following:

```
ceph-deploy install {hostname [hostname] ...}
```

Without additional arguments, `ceph-deploy` will install the most recent major release of Ceph to the cluster host(s). To specify a particular package, you may select from the following:

- `--release <code-name>`
- `--testing`
- `--dev <branch-or-tag>`

For example:

```
ceph-deploy install --release cuttlefish hostname1
ceph-deploy install --testing hostname2
ceph-deploy install --dev wip-some-branch hostname{1,2,3,4,5}
```

For additional usage, execute:

```
ceph-deploy install -h
```

Uninstall

To uninstall Ceph packages from your cluster hosts, open a terminal on your admin host and type the following:

```
ceph-deploy uninstall {hostname [hostname] ...}
```

On a Debian or Ubuntu system, you may also:

```
ceph-deploy purge {hostname [hostname] ...}
```

The tool will unininstall `ceph` packages from the specified hosts. Purge additionally removes configuration files.

4.2.3 Create a Cluster

The first step in using Ceph with `ceph-deploy` is to create a new Ceph cluster. A new Ceph cluster has:

- A Ceph configuration file, and
- A monitor keyring.

The Ceph configuration file consists of at least:

- Its own filesystem ID (`fsid`)
- The initial monitor(s) hostname(s), and
- The initial monitor(s) and IP address(es).

For additional details, see the Monitor Configuration Reference.

The `ceph-deploy` tool also creates a monitor keyring and populates it with a `[mon.]` key. For additional details, see the Cephx Guide.

Usage

To create a cluster with `ceph-deploy`, use the `new` command and specify the host(s) that will be initial members of the monitor quorum.

```
ceph-deploy new {host [host], ...}
```

For example:

```
ceph-deploy new mon1.foo.com
ceph-deploy new mon{1,2,3}
```

The `ceph-deploy` utility will use DNS to resolve hostnames to IP addresses. The monitors will be named using the first component of the name (e.g., `mon1` above). It will add the specified host names to the Ceph configuration file. For additional details, execute:

```
ceph-deploy new -h
```

Naming a Cluster

By default, Ceph clusters have a cluster name of `ceph`. You can specify a cluster name if you want to run multiple clusters on the same hardware. For example, if you want to optimize a cluster for use with block devices, and another for use with the gateway, you can run two different clusters on the same hardware if they have a different `fsid` and cluster name.

```
ceph-deploy --cluster {cluster-name} new {host [host], ...}
```

For example:

```
ceph-deploy --cluster rbdcluster new ceph-mon1
ceph-deploy --cluster rbdcluster new ceph-mon{1,2,3}
```

Note: If you run multiple clusters, ensure you adjust the default port settings and open ports for your additional cluster(s) so that the networks of the two different clusters don't conflict with each other.

4.2.4 Add/Remove Monitors

With `ceph-deploy`, adding and removing monitors is a simple task. You just add or remove one or more monitors on the command line with one command. Before `ceph-deploy`, the process of adding and removing monitors involved numerous manual steps. Using `ceph-deploy` imposes a restriction: **you may only install one monitor per host.**

Note: We do not recommend comingling monitors and OSDs on the same host.

For high availability, you should run a production Ceph cluster with **AT LEAST** three monitors. Ceph uses the Paxos algorithm, which requires a consensus among the majority of monitors in a quorum. With Paxos, the monitors cannot determine a majority for establishing a quorum with only two monitors. A majority of monitors must be counted as such: 1:1, 2:3, 3:4, 3:5, 4:6, etc.

See Monitor Config Reference for details on configuring monitors.

Add a Monitor

Once you create a cluster and install Ceph packages to the monitor host(s), you may deploy the monitor(s) to the monitor host(s). When using `ceph-deploy`, the tool enforces a single monitor per host.

```
ceph-deploy mon create {host-name [host-name]...}
```

Note: Ensure that you add monitors such that they may arrive at a consensus among a majority of monitors, otherwise other steps (like `ceph-deploy gatherkeys`) will fail.

Note: When adding a monitor on a host that was not in hosts initially defined with the `ceph-deploy new` command, a `public network` statement needs to be added to the ceph.conf file.

Remove a Monitor

If you have a monitor in your cluster that you'd like to remove, you may use the `destroy` option.

```
ceph-deploy mon destroy {host-name [host-name]...}
```

Note: Ensure that if you remove a monitor, the remaining monitors will be able to establish a consensus. If that is not possible, consider adding a monitor before removing the monitor you would like to take offline.

4.2.5 Keys Management

Gather Keys

Before you can provision a host to run OSDs or metadata servers, you must gather monitor keys and the OSD and MDS bootstrap keyrings. To gather keys, enter the following:

```
ceph-deploy gatherkeys {monitor-host}
```

Note: To retrieve the keys, you specify a host that has a Ceph monitor.

Note: If you have specified multiple monitors in the setup of the cluster, make sure, that all monitors are up and running. If the monitors haven't formed quorum, `ceph-create-keys` will not finish and the keys aren't generated.

Forget Keys

When you are no longer using `ceph-deploy` (or if you are recreating a cluster), you should delete the keys in the local directory of your admin host. To delete keys, enter the following:

```
ceph-deploy forgetkeys
```

4.2.6 Add/Remove OSDs

Adding and removing Ceph OSD Daemons to your cluster may involve a few more steps when compared to adding and removing other Ceph daemons. Ceph OSD Daemons write data to the disk and to journals. So you need to provide a disk for the OSD and a path to the journal partition (i.e., this is the most common configuration, but you may configure your system to your own needs).

In Ceph v0.60 and later releases, Ceph supports `dm-crypt` on disk encryption. You may specify the `--dmcrypt` argument when preparing an OSD to tell `ceph-deploy` that you want to use encryption. You may also specify the `--dmcrypt-key-dir` argument to specify the location of `dm-crypt` encryption keys.

You should test various drive configurations to gauge their throughput before before building out a large cluster. See Data Storage for additional details.

List Disks

To list the disks on a node, execute the following command:

```
ceph-deploy disk list {node-name [node-name]...}
```

Zap Disks

To zap a disk (delete its partition table) in preparation for use with Ceph, execute the following:

```
ceph-deploy disk zap {osd-server-name}:{disk-name}
ceph-deploy disk zap osdserver1:sdb
```

Important: This will delete all data.

Prepare OSDs

Once you create a cluster, install Ceph packages, and gather keys, you may prepare the OSDs and deploy them to the OSD node(s). If you need to identify a disk or zap it prior to preparing it for use as an OSD, see *List Disks* and *Zap Disks*.

```
ceph-deploy osd prepare {node-name}:{data-disk}[:{journal-disk}]
ceph-deploy osd prepare osdserver1:sdb:/dev/ssd
ceph-deploy osd prepare osdserver1:sdc:/dev/ssd
```

The `prepare` command only prepares the OSD. On most operating systems, the `activate` phase will automatically run when the partitions are created on the disk (using Ceph `udev` rules). If not use the `activate` command. See *Activate OSDs* for details.

The foregoing example assumes a disk dedicated to one Ceph OSD Daemon, and a path to an SSD journal partition. We recommend storing the journal on a separate drive to maximize throughput. You may dedicate a single drive for the journal too (which may be expensive) or place the journal on the same disk as the OSD (not recommended as it impairs performance). In the foregoing example we store the journal on a partitioned solid state drive.

Note: When running multiple Ceph OSD daemons on a single node, and sharing a partioned journal with each OSD daemon, you should consider the entire node the minimum failure domain for CRUSH purposes, because if the SSD drive fails, all of the Ceph OSD daemons that journal to it will fail too.

Activate OSDs

Once you prepare an OSD you may activate it with the following command.

```
ceph-deploy osd activate {node-name}:{data-disk-partition}[:{journal-disk-partition}]
ceph-deploy osd activate osdserver1:/dev/sdb1:/dev/ssd1
ceph-deploy osd activate osdserver1:/dev/sdc1:/dev/ssd2
```

The `activate` command will cause your OSD to come `up` and be placed `in` the cluster. The `activate` command uses the path to the partition created when running the `prepare` command.

Create OSDs

You may prepare OSDs, deploy them to the OSD node(s) and activate them in one step with the `create` command. The `create` command is a convenience method for executing the `prepare` and `activate` command sequentially.

```
ceph-deploy osd create {node-name}:{disk}[:{path/to/journal}]
ceph-deploy osd create osdserver1:sdb:/dev/ssd1
```

Destroy OSDs

Note: Coming soon. See Remove OSDs for manual procedures.

4.2.7 Add/Remove Metadata Server

With `ceph-deploy`, adding and removing metadata servers is a simple task. You just add or remove one or more metadata servers on the command line with one command.

Important: You must deploy at least one metadata server to use CephFS. There is experimental support for running multiple metadata servers. Do not run multiple metadata servers in production.

See MDS Config Reference for details on configuring metadata servers.

Add a Metadata Server

Once you deploy monitors and OSDs you may deploy the metadata server(s).

```
ceph-deploy mds create {host-name}[:{daemon-name}] [{host-name}[:{daemon-name}] ...]
```

You may specify a daemon instance a name (optional) if you would like to run multiple daemons on a single server.

Remove a Metadata Server

Coming soon...

4.2.8 Purge a Host

When you remove Ceph daemons and uninstall Ceph, there may still be extraneous data from the cluster on your server. The `purge` and `purgedata` commands provide a convenient means of cleaning up a host.

Purge Data

To remove all data from `/var/lib/ceph` (but leave Ceph packages intact), execute the `purgedata` command.

> ceph-deploy purgedata {hostname} [{hostname} ...]

Purge

To remove all data from `/var/lib/ceph` and uninstall Ceph packages, execute the `purge` command.

> ceph-deploy purge {hostname} [{hostname} ...]

4.2.9 Admin Tasks

Once you have set up a cluster with `ceph-deploy`, you may provide the client admin key and the Ceph configuration file to another host so that a user on the host may use the `ceph` command line as an administrative user.

Create an Admin Host

To enable a host to execute ceph commands with administrator privileges, use the `admin` command.

```
ceph-deploy admin {host-name [host-name]...}
```

Deploy Config File

To send an updated copy of the Ceph configuration file to hosts in your cluster, use the `config push` command.

```
ceph-deploy config push {host-name [host-name]...}
```

Tip: With a base name and increment host-naming convention, it is easy to deploy configuration files via simple scripts (e.g., `ceph-deploy config hostname{1,2,3,4,5}`).

Retrieve Config File

To retrieve a copy of the Ceph configuration file from a host in your cluster, use the `config pull` command.

```
ceph-deploy config pull {host-name [host-name]...}
```

Once you have a deployed a Ceph Storage Cluster, you may begin operating your cluster.

4.3 Cluster Operations

High-level cluster operations consist primarily of starting, stopping, and restarting a cluster with the `ceph` service; checking the cluster's health; and, monitoring an operating cluster.

4.3.1 Operating a Cluster

Running Ceph with Upstart

When deploying Ceph Cuttlefish and beyond with `ceph-deploy` on Ubuntu, you may start and stop Ceph daemons on a *Ceph Node* 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.

Starting all Daemons

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

```
sudo start ceph-all
```

Stopping all Daemons

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

```
sudo stop ceph-all
```

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
```

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
```

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
```

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
```

Running Ceph

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.
`--valgrind`	`N/A`	(Dev and QA only) Use Valgrind debugging.
`--allhosts`	`-a`	Execute on all nodes in `ceph.conf.` 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.
`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`
- `mds`

Running Ceph with sysvinit

Using traditional `sysvinit` is the recommended way to run Ceph with CentOS, Red Hat, Fedora, Debian and SLES distributions. You may also use it for older distributions of Ubuntu.

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.

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.

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
```

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
```

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
```

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
```

Running Ceph as a Service

When you deploy Ceph Argonaut or Bobtail with `ceph-deploy`, you may operate Ceph as a service (you may also use sysvinit).

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.

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.

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
```

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
```

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
```

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
```

4.3.2 Monitoring a Cluster

Once you have a running cluster, you may use the `ceph` tool to monitor your cluster. Monitoring a cluster typically involves checking OSD status, monitor status, placement group status and metadata server status.

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
```

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.

Watching a Cluster

To watch the cluster's ongoing events, 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
2014-06-02 15:45:05.718640 mon.0 [INF] pgmap v41340: 952 pgs: 1 active+clean+scrubbing+deep, 951 acti
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 acti
2014-06-02 15:45:15.722456 mon.0 [INF] pgmap v41342: 952 pgs: 952 active+clean; 17130 MB data, 115 GB
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 acti
```

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.

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. See Storage Capacity for additional details.

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 2MB or more depending on the number of replicas, clones and snapshots.

- **NAME:** The name of the pool.
- **ID:** The pool ID.
- **USED:** The notional amount of data stored in kilobytes, unless the number appends **M** for megabytes or **G** for gigabytes.
- **%USED:** The notional percentage of storage used per pool.
- **Objects:** The notional number of objects stored per pool.

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

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
```

Checking OSD Status

You can check OSDs to ensure they are `up` and `in` by executing:

```
ceph osd stat
```

Or:

```
ceph osd dump
```

You can also check view OSDs according to their position in the CRUSH map.

```
ceph osd tree
```

Ceph will print out a CRUSH tree with a host, its OSDs, whether they are up and their weight.

```
# 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
```

For a detailed discussion, refer to Monitoring OSDs and Placement Groups.

Checking Monitor Status

If your cluster has multiple monitors (likely), 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.

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"}
           ]
    }
}
```

Checking MDS Status

Metadata servers provide metadata services for Ceph FS. Metadata servers have two sets of states: `up | down` and `active | inactive`. To ensure your metadata servers are `up` and `active`, execute the following:

```
ceph mds stat
```

To display details of the metadata cluster, execute the following:

```
ceph mds dump
```

Checking Placement Group States

Placement groups map objects to OSDs. When you monitor your placement groups, you will want them to be `active` and `clean`. For a detailed discussion, refer to Monitoring OSDs and Placement Groups.

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. See Viewing a Configuration at Runtime for details.

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).

4.3.3 Monitoring OSDs and PGs

High availability and high reliability require a fault-tolerant approach to managing hardware and software issues. Ceph has no single point-of-failure, and can service requests for data in a "degraded" mode. Ceph's data placement introduces a layer of indirection to ensure that data doesn't bind directly to particular OSD addresses. This means that tracking down system faults requires finding the placement group and the underlying OSDs at root of the problem.

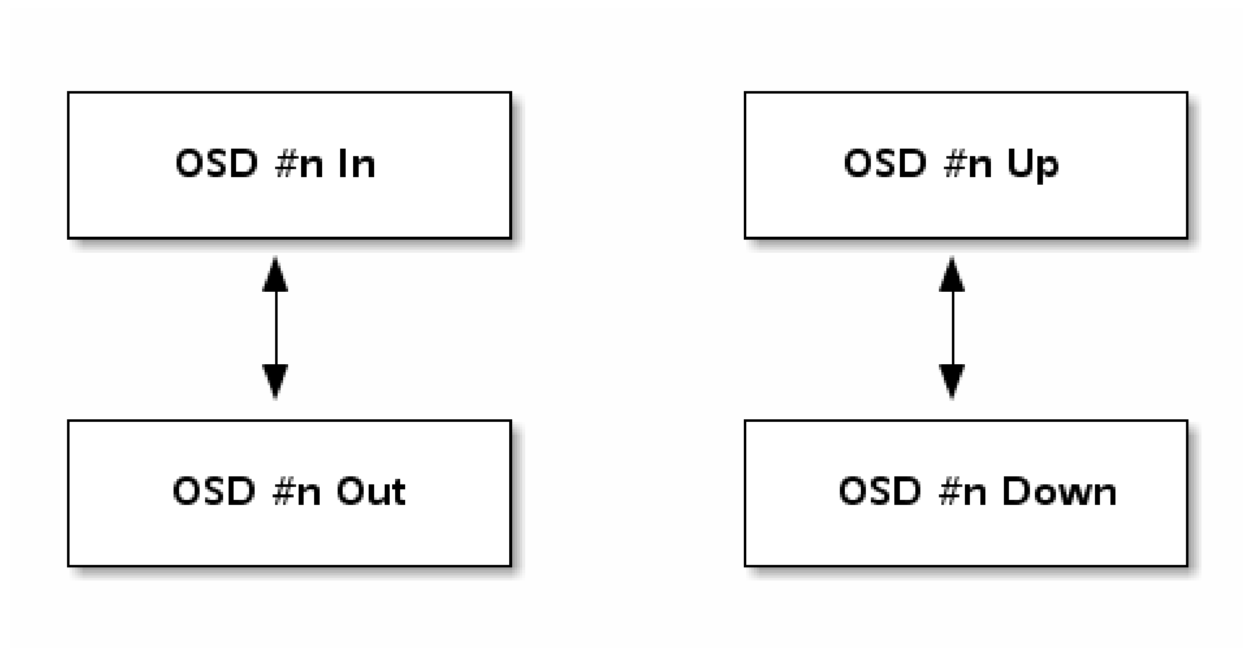
Tip: A fault in one part of the cluster may prevent you from accessing a particular object, but that doesn't mean that you can't access other objects. When you run into a fault, don't panic. Just follow the steps for monitoring your OSDs and placement groups. Then, begin troubleshooting.

Ceph is generally self-repairing. However, when problems persist, monitoring OSDs and placement groups will help you identify the problem.

Monitoring OSDs

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
```

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
```

```
dumped osdmap tree epoch 1
# id    weight  type name       up/down reweight
-1      2       pool openstack
```

```
-3      2               rack dell-2950-rack-A
-2      2                       host dell-2950-A1
0       1                               osd.0   up      1
1       1                               osd.1   down    1
```

Tip: The ability to search through a well-designed CRUSH hierarchy may help you troubleshoot your cluster by identifying the physcial locations faster.

If an OSD is `down`, start it:

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

See OSD Not Running for problems associated with OSDs that stopped, or won't restart.

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 restared, 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}
```

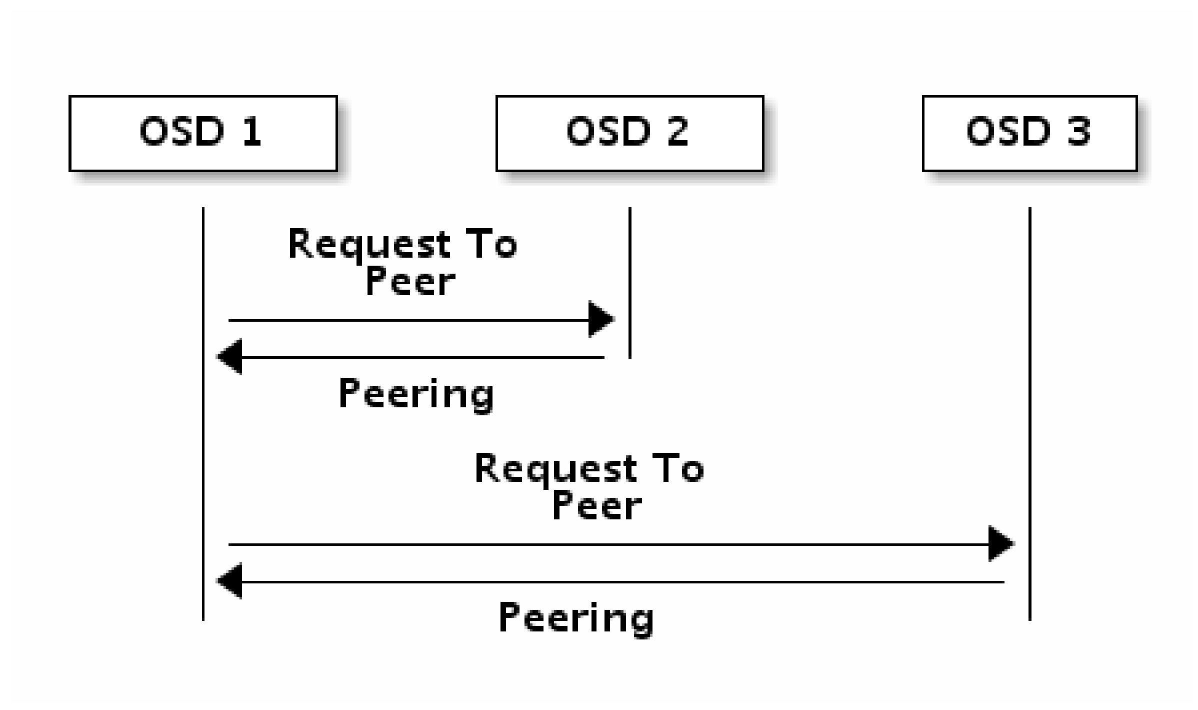
The result should tell you the osdmap epoch (eNNN), the placement group number ({pg-num}), the OSDs in the Up Set (up[]), and the OSDs in the acting set (acting[]).

```
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 rebalancing itself or of a potential problem with the cluster.

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).



The OSDs also report their status to the monitor. See Configuring Monitor/OSD Interaction for details. To troubleshoot peering issues, see Peering Failure.

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
```

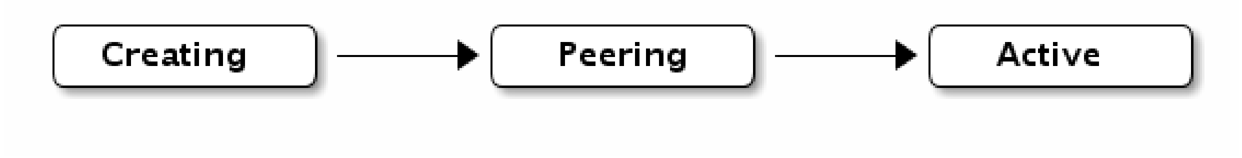
Ceph will output the query in JSON format.

```
{
  "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.

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.



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.

Authoratative 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.

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.

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.

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.

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.

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).

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.

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.

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]
```

See Placement Group Subsystem for additional details. To troubleshoot stuck placement groups, see Troubleshooting PG Errors.

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}
```

Exercise: Locate an Object

As an exercise, lets create an object. Specify an object name, a path to a test file containing some object data and a pool name using the `rados put` command on the command line. For example:

```
rados put {object-name} {file-path} --pool=data
rados put test-object-1 testfile.txt --pool=data
```

To verify that the Ceph Object Store stored the object, execute the following:

```
rados -p data ls
```

Now, identify the object location:

```
ceph osd map {pool-name} {object-name}
ceph osd map data test-object-1
```

Ceph should output the object's location. For example:

```
osdmap e537 pool 'data' (0) object 'test-object-1' -> pg 0.d1743484 (0.4) -> up [1,0] acting [1,0]
```

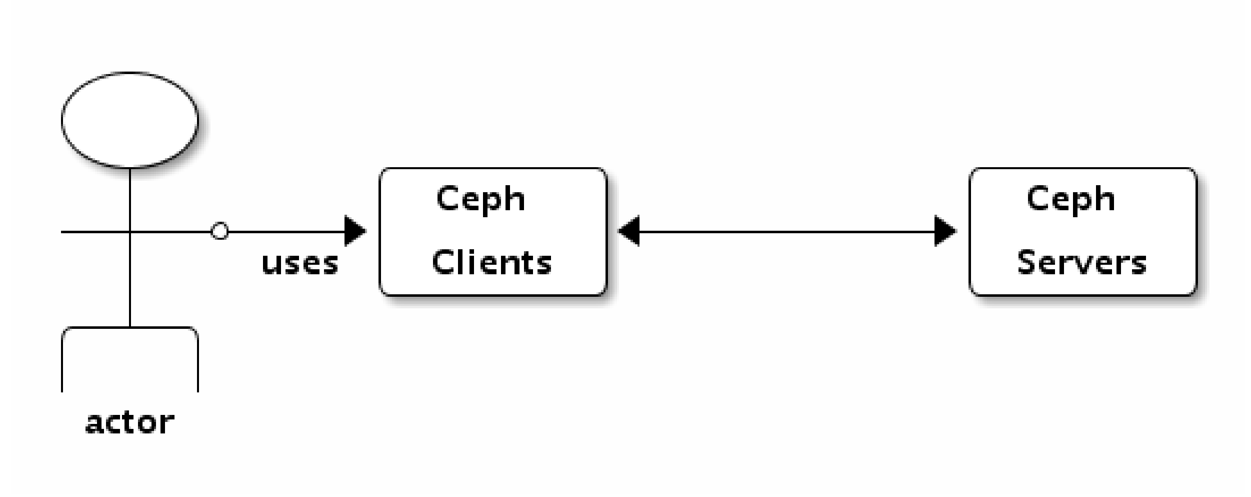
To remove the test object, simply delete it using the `rados rm` command. For example:

```
rados rm test-object-1 --pool=data
```

As the cluster evolves, the object location may change dynamically. One benefit of Ceph's dynamic rebalancing is that Ceph relieves you from having to perform the migration manually. See the Architecture section for details.

4.3.4 User Management

This document describes *Ceph Client* users, and their authentication and authorization with the *Ceph Storage Cluster*. Users are either individuals or system actors such as applications, which use Ceph clients to interact with the 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 `client.admin` 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 Ceph Storage Cluster to use authentication, see Cephx Config Reference. For details on the architecture of Cephx, see Architecture - High Availability Authentication.

Background

Irrespective of the type of Ceph client (e.g., Block Device, Object Storage, Filesystem, native API, etc.), Ceph stores all data as objects within pools. Ceph users must have access to pools in order to read and write data. Additionally, Ceph users must have execute permissions to use Ceph's administrative commands. The following concepts will help you understand Ceph user management.

User

A user is either an individual or a system actor such as an application. Creating users allows you to control who (or what) can access your Ceph Storage Cluster, its pools, and the data within pools.

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

Servers 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 Ceph Storage Cluster user is not the same as a Ceph Object Storage user or a Ceph Filesystem user. The Ceph Object Gateway uses a Ceph Storage Cluster user to communicate between the gateway daemon and the storage cluster, but the gateway has its own user management functionality for end users. The Ceph Filesystem uses POSIX semantics. The user space associated with the Ceph Filesystem is not the same as a Ceph Storage Cluster user.

Authorization (Capabilities)

Ceph uses the term "capabilities" (caps) to describe authorizing an authenticated user to exercise the functionality of the monitors, OSDs and metadata servers. 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}]
```

- **Metadata Server Caps:** Metadata server capability simply requires `allow`, or blank and does not parse anything further.

```
    mds 'allow'
```

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. Implies `rw` for MDS only.

`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 mds`

Description Gives a user permissions to connect as a MDS to other MDSs or monitors.

`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.

`profile bootstrap-mds`

Description Gives a user permissions to bootstrap a metadata server. Conferred on deployment tools such as `ceph-deploy`, etc. so they have permissions to add keys, etc. when bootstrapping a metadata server.

Pool

A pool is a logical partition where users store data. By default, a Ceph Storage Cluster has pools for `data`, `rbd` and `metadata` (metadata server). In Ceph deployments, it is common to create a pool as a logical partition for similar types of data. 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.

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 (i.e., `firefly`), namespaces are only useful for applications written on top of `librados`. Ceph clients such as block device, object storage and file system do not currently support this feature.

The rationale for namespaces is that pools can be a computationally expensive method of segregating data sets for the purposes of authorizing separate sets of users. For example, a pool should have ~100 placement groups per OSD. So an exemplary cluster with 1000 OSDs would have 100,000 placement groups for one pool. Each pool 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.

Managing Users

User management functionality provides Ceph Storage Cluster administrators with the ability to create, update and delete users directly in the Ceph Storage Cluster.

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

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

Note that the `TYPE.ID` notation for users applies such that `osd.0` is a user of type `osd` and its ID is `0`, `client.admin` is a user of type `client` and its ID is `admin` (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.

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 `auid`, which isn't relevant to end users.

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!

Modify User Capabilities

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

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

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.

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.

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`
```

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.

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.

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.

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
```

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 r
```

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
```

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.

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
```

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 it 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.

Once you have your cluster up and running, you may begin working with data placement. Ceph supports petabyte-scale data storage clusters, with storage pools and placement groups that distribute data across the cluster using Ceph's CRUSH algorithm.

4.3.5 Data Placement Overview

Ceph stores, replicates and rebalances data objects across a RADOS cluster dynamically. With many different users storing objects in different pools for different purposes on countless OSDs, Ceph operations require some data placement planning. The main data placement planning concepts in Ceph include:

- **Pools:** Ceph stores data within pools, which are logical groups for storing objects. Pools manage the number of placement groups, the number of replicas, and the ruleset for the pool. To store data in a pool, you must have an authenticated user with permissions for the pool. Ceph can snapshot pools. See Pools for additional details.
- **Placement Groups:** Ceph maps objects to placement groups (PGs). Placement groups (PGs) are shards or fragments of a logical object pool that place objects as a group into OSDs. Placement groups reduce the amount of per-object metadata when Ceph stores the data in OSDs. A larger number of placement groups (e.g., 100 per OSD) leads to better balancing. See Placement Groups for additional details.
- **CRUSH Maps:** CRUSH is a big part of what allows Ceph to scale without performance bottlenecks, without limitations to scalability, and without a single point of failure. CRUSH maps provide the physical topology of the cluster to the CRUSH algorithm to determine where the data for an object and its replicas should be stored, and how to do so across failure domains for added data safety among other things. See CRUSH Maps for additional details.

When you initially set up a test cluster, you can use the default values. Once you begin planning for a large Ceph cluster, refer to pools, placement groups and CRUSH for data placement operations. If you find some aspects challenging, Inktank provides excellent premium support for Ceph.

4.3.6 Pools

When you first deploy a cluster without creating a pool, Ceph uses the default pools for storing data. A pool provides you with:

- **Resilience**: You can set how many OSD are allowed to fail without losing data. For replicated pools, it is the desired number of copies/replicas of an object. A typical configuration stores an object and one additional copy (i.e., `size = 2`), but you can determine the number of copies/replicas. For erasure coded pools, it is the number of coding chunks (i.e. `m=2` in the **erasure code profile**)
- **Placement Groups**: You can set the number of placement groups for the pool. A typical configuration uses approximately 100 placement groups per OSD to provide optimal balancing without using up too many computing resources. When setting up multiple pools, be careful to ensure you set a reasonable number of placement groups for both the pool and the cluster as a whole.
- **CRUSH Rules**: When you store data in a pool, a CRUSH ruleset mapped to the pool enables CRUSH to identify a rule for the placement of the object and its replicas (or chunks for erasure coded pools) in your cluster. You can create a custom CRUSH rule for your pool.
- **Snapshots**: When you create snapshots with `ceph osd pool mksnap`, you effectively take a snapshot of a particular pool.
- **Set Ownership**: You can set a user ID as the owner of a pool.

To organize data into pools, you can list, create, and remove pools. You can also view the utilization statistics for each pool.

List Pools

To list your cluster's pools, execute:

```
ceph osd lspools
```

The default pools include:

- `data`
- `metadata`
- `rbd`

Create a Pool

Before creating pools, refer to the Pool, PG and CRUSH Config Reference. Ideally, you should override the default value for the number of placement groups in your Ceph configuration file, as the default is NOT ideal. For example:

```
osd pool default pg num = 100
osd pool default pgp num = 100
```

To create a pool, execute:

```
ceph osd pool create {pool-name} {pg-num} [{pgp-num}] [replicated] \
     [crush-ruleset-name]
ceph osd pool create {pool-name} {pg-num}  {pgp-num}   erasure \
     [erasure-code-profile] [crush-ruleset-name]
```

Where:

`{pool-name}`

Description The name of the pool. It must be unique.

Type String

Required Yes. Picks up default or Ceph configuration value if not specified.

`{pg-num}`

Description The total number of placement groups for the pool. See Placement Groups for details on calculating a suitable number. The default value `8` is NOT suitable for most systems.

Type Integer

Required Yes

Default 8

`{pgp-num}`

Description The total number of placement groups for placement purposes. This **should be equal to the total number of placement groups**, except for placement group splitting scenarios.

Type Integer

Required Yes. Picks up default or Ceph configuration value if not specified.

Default 8

`{replicated|erasure}`

Description The pool type which may either be **replicated** to recover from lost OSDs by keeping multiple copies of the objects or **erasure** to get a kind of generalized RAID5 capability. The **replicated** pools require more raw storage but implement all Ceph operations. The **erasure** pools require less raw storage but only implement a subset of the available operations.

Type String

Required No.

Default replicated

`[crush-ruleset-name]`

Description The name of the crush ruleset for this pool. If specified ruleset doesn't exist, the creation of **replicated** pool will fail with -ENOENT. But **replicated** pool will create a new erasure ruleset with specified name.

Type String

Required No.

Default "erasure-code" for **erasure pool**. Pick up Ceph configuraion variable **osd_pool_default_crush_replicated_ruleset** for **replicated** pool.

`[erasure-code-profile=profile]`

Description For **erasure** pools only. Use the erasure code profile. It must be an existing profile as defined by **osd erasure-code-profile set**.

Type String

Required No.

When you create a pool, set the number of placement groups to a reasonable value (e.g., `100`). Consider the total number of placement groups per OSD too. Placement groups are computationally expensive, so performance will degrade when you have many pools with many placement groups (e.g., 50 pools with 100 placement groups each). The point of diminishing returns depends upon the power of the OSD host.

See Placement Groups for details on calculating an appropriate number of placement groups for your pool.

Set Pool Quotas

You can set pool quotas for the maximum number of bytes and/or the maximum number of objects per pool.

```
ceph osd pool set-quota {pool-name} [max_objects {obj-count}] [max_bytes {bytes}]
```

For example:

```
ceph osd pool set-quota data max_objects 10000
```

To remove a quota, set its value to `0`.

Delete a Pool

To delete a pool, execute:

```
ceph osd pool delete {pool-name} [{pool-name} --yes-i-really-really-mean-it]
```

If you created your own rulesets and rules for a pool you created, you should consider removing them when you no longer need your pool. If you created users with permissions strictly for a pool that no longer exists, you should consider deleting those users too.

Rename a Pool

To rename a pool, execute:

```
ceph osd pool rename {current-pool-name} {new-pool-name}
```

If you rename a pool and you have per-pool capabilities for an authenticated user, you must update the user's capabilities (i.e., caps) with the new pool name.

Note: Version `0.48` Argonaut and above.

Show Pool Statistics

To show a pool's utilization statistics, execute:

```
rados df
```

Make a Snapshot of a Pool

To make a snapshot of a pool, execute:

```
ceph osd pool mksnap {pool-name} {snap-name}
```

Note: Version `0.48` Argonaut and above.

Remove a Snapshot of a Pool

To remove a snapshot of a pool, execute:

```
ceph osd pool rmsnap {pool-name} {snap-name}
```

Note: Version `0.48` Argonaut and above.

Set Pool Values

To set a value to a pool, execute the following:

```
ceph osd pool set {pool-name} {key} {value}
```

You may set values for the following keys:

`size`

Description Sets the number of replicas for objects in the pool. See *Set the Number of Object Replicas* for further details. Replicated pools only.

Type Integer

`min_size`

Description Sets the minimum number of replicas required for I/O. See *Set the Number of Object Replicas* for further details. Replicated pools only.

Type Integer

Version `0.54` and above

`crash_replay_interval`

Description The number of seconds to allow clients to replay acknowledged, but uncommitted requests.

Type Integer

`pgp_num`

Description The effective number of placement groups to use when calculating data placement.

Type Integer

Valid Range Equal to or less than `pg_num`.

`crush_ruleset`

Description The ruleset to use for mapping object placement in the cluster.

Type Integer

`hashpspool`

Description Set/Unset HASHPSPOOL flag on a given pool.

Type Integer

Valid Range 1 sets flag, 0 unsets flag

Version Version `0.48` Argonaut and above.

`nodelete`

Description Set/Unset NODELETE flag on a given pool.

Type Integer

Valid Range 1 sets flag, 0 unsets flag

Version Version `FIXME`

`nopgchange`

Description Set/Unset NOPGCHANGE flag on a given pool.

Type Integer

Valid Range 1 sets flag, 0 unsets flag

Version Version `FIXME`

`nosizechange`

Description Set/Unset NOSIZECHANGE flag on a given pool.

Type Integer

Valid Range 1 sets flag, 0 unsets flag

Version Version `FIXME`

`hit_set_type`

Description Enables hit set tracking for cache pools. See Bloom Filter for additional information.

Type String

Valid Settings `bloom`, `explicit_hash`, `explicit_object`

Default `bloom`. Other values are for testing.

`hit_set_count`

Description The number of hit sets to store for cache pools. The higher the number, the more RAM consumed by the `ceph-osd` daemon.

Type Integer

Valid Range `1`. Agent doesn't handle > 1 yet.

`hit_set_period`

Description The duration of a hit set period in seconds for cache pools. The higher the number, the more RAM consumed by the `ceph-osd` daemon.

Type Integer

Example `3600` 1hr

`hit_set_fpp`

Description The false positive probability for the `bloom` hit set type. See Bloom Filter for additional information.

Type Double

Valid Range 0.0 - 1.0

Default `0.05`

`cache_target_dirty_ratio`

Description The percentage of the cache pool containing modified (dirty) objects before the cache tiering agent will flush them to the backing storage pool.

Type Double

Default `.4`

`cache_target_full_ratio`

Description The percentage of the cache pool containing unmodified (clean) objects before the cache tiering agent will evict them from the cache pool.

Type Double

Default `.8`

`target_max_bytes`

Description Ceph will begin flushing or evicting objects when the `max_bytes` threshold is triggered.

Type Integer

Example `1000000000000` #1-TB

`target_max_objects`

Description Ceph will begin flushing or evicting objects when the `max_objects` threshold is triggered.

Type Integer

Example `1000000` #1M objects

`cache_min_flush_age`

Description The time (in seconds) before the cache tiering agent will flush an object from the cache pool to the storage pool.

Type Integer

Example `600` 10min

`cache_min_evict_age`

Description The time (in seconds) before the cache tiering agent will evict an object from the cache pool.

Type Integer

Example `1800` 30min

Get Pool Values

To get a value from a pool, execute the following:

```
ceph osd pool get {pool-name} {key}
```

You may get values for the following keys:

`size`

Description Gets the number of replicas for objects in the pool. See *Set the Number of Object Replicas* for further details. Replicated pools only.

Type Integer

`min_size`

Description Gets the minimum number of replicas required for I/O. See *Set the Number of Object Replicas* for further details. Replicated pools only.

Type Integer

Version `0.54` and above

`crash_replay_interval`

Description The number of seconds to allow clients to replay acknowledged, but uncommitted requests.

Type Integer

`pgp_num`

Description The effective number of placement groups to use when calculating data placement.

Type Integer

Valid Range Equal to or less than `pg_num`.

`crush_ruleset`

Description The ruleset to use for mapping object placement in the cluster.

Type Integer

`hit_set_type`

Description Enables hit set tracking for cache pools. See Bloom Filter for additional information.

Type String

Valid Settings `bloom`, `explicit_hash`, `explicit_object`

`hit_set_count`

Description The number of hit sets to store for cache pools. The higher the number, the more RAM consumed by the `ceph-osd` daemon.

Type Integer

`hit_set_period`

Description The duration of a hit set period in seconds for cache pools. The higher the number, the more RAM consumed by the `ceph-osd` daemon.

Type Integer

`hit_set_fpp`

Description The false positive probability for the `bloom` hit set type. See Bloom Filter for additional information.

Type Double

`cache_target_dirty_ratio`

Description The percentage of the cache pool containing modified (dirty) objects before the cache tiering agent will flush them to the backing storage pool.

Type Double

`cache_target_full_ratio`

Description The percentage of the cache pool containing unmodified (clean) objects before the cache tiering agent will evict them from the cache pool.

Type Double

`target_max_bytes`

Description Ceph will begin flushing or evicting objects when the `max_bytes` threshold is triggered.

Type Integer

`target_max_objects`

Description Ceph will begin flushing or evicting objects when the `max_objects` threshold is triggered.

Type Integer

`cache_min_flush_age`

Description The time (in seconds) before the cache tiering agent will flush an object from the cache pool to the storage pool.

Type Integer

`cache_min_evict_age`

Description The time (in seconds) before the cache tiering agent will evict an object from the cache pool.

Type Integer

Set the Number of Object Replicas

To set the number of object replicas on a replicated pool, execute the following:

```
ceph osd pool set {poolname} size {num-replicas}
```

Important: The `{num-replicas}` includes the object itself. If you want the object and two copies of the object for a total of three instances of the object, specify `3`.

For example:

```
ceph osd pool set data size 3
```

You may execute this command for each pool. **Note:** An object might accept I/Os in degraded mode with fewer than `pool size` replicas. To set a minimum number of required replicas for I/O, you should use the `min_size` setting. For example:

```
ceph osd pool set data min_size 2
```

This ensures that no object in the data pool will receive I/O with fewer than `min_size` replicas.

Get the Number of Object Replicas

To get the number of object replicas, execute the following:

```
ceph osd dump | grep 'replicated size'
```

Ceph will list the pools, with the `replicated size` attribute highlighted. By default, ceph Creates two replicas of an object (a total of three copies, or a size of 3).

4.3.7 Erasure code

A Ceph pool is associated to a type to sustain the loss of an OSD (i.e. a disk since most of the time there is one OSD per disk). The default choice when creating a pool is *replicated*, meaning every object is copied on multiple disks. The Erasure Code pool type can be used instead to save space.

Creating a sample erasure coded pool

The simplest erasure coded pool is equivalent to RAID5 and requires at least three hosts:

```
$ ceph osd pool create ecpool 12 12 erasure
pool 'ecpool' created
$ echo ABCDEFGHI | rados --pool ecpool put NYAN -
$ rados --pool ecpool get NYAN -
ABCDEFGHI
```

Note: the 12 in *pool create* stands for the number of placement groups.

Erasure code profiles

The default erasure code profile sustains the loss of a single OSD. It is equivalent to a replicated pool of size two but requires 1.5TB instead of 2TB to store 1TB of data. The default profile can be displayed with:

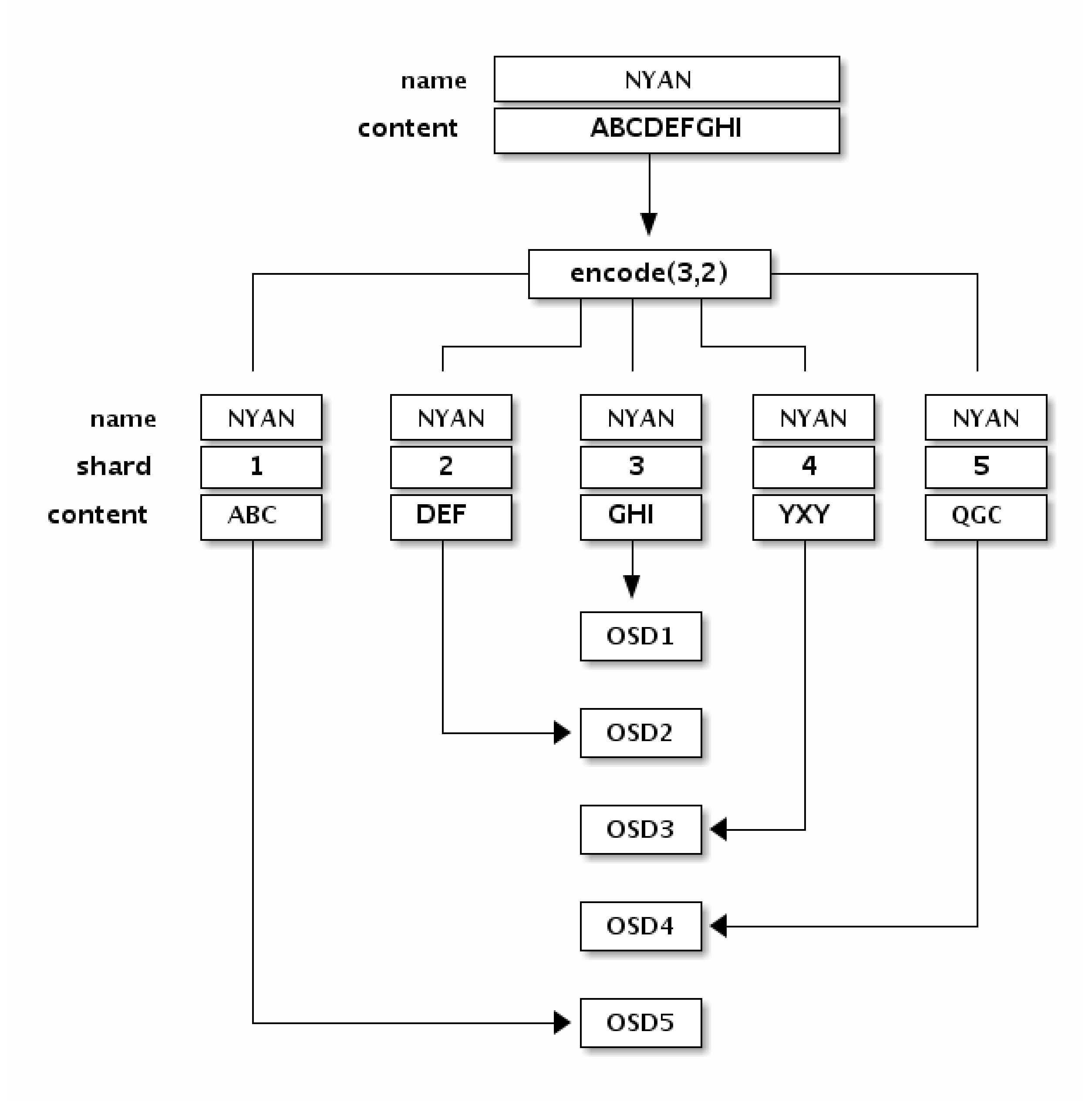
```
$ ceph osd erasure-code-profile get default
directory=.libs
k=2
m=1
plugin=jerasure
ruleset-failure-domain=host
technique=reed_sol_van
```

Choosing the right profile is important because it cannot be modified after the pool is created: a new pool with a different profile needs to be created and all objects from the previous pool moved to the new.

The most important parameters of the profile are *K*, *M* and *ruleset-failure-domain* because they define the storage overhead and the data durability. For instance, if the desired architecture must sustain the loss of two racks with a storage overhead of 40% overhead, the following profile can be defined:

```
$ ceph osd erasure-code-profile set myprofile \
   k=3 \
   m=2 \
   ruleset-failure-domain=rack
$ ceph osd pool create ecpool 12 12 erasure *myprofile*
$ echo ABCDEFGHI | rados --pool ecpool put NYAN -
$ rados --pool ecpool get NYAN -
ABCDEFGHI
```

The *NYAN* object will be divided in three (*K=3*) and two additional *chunks* will be created (*M=2*). The value of *M* defines how many OSD can be lost simultaneously without losing any data. The *ruleset-failure-domain=rack* will create a CRUSH ruleset that ensures no two *chunks* are stored in the same rack.



More information can be found in the erasure code profiles documentation.

Erasure coded pool and cache tiering

Erasure coded pools require more resources than replicated pools and lack some functionalities such as partial writes. To overcome these limitations, it is recommended to set a cache tier before the erasure coded pool.

For instance, if the pool *hot-storage* is made of fast storage:

```
$ ceph osd tier add ecpool hot-storage
$ ceph osd tier cache-mode hot-storage writeback
$ ceph osd tier set-overlay ecpool hot-storage
```

will place the *hot-storage* pool as tier of *ecpool* in *writeback* mode so that every write and read to the *ecpool* are actually using the *hot-storage* and benefit from its flexibility and speed.

It is not possible to create an RBD image on an erasure coded pool because it requires partial writes. It is however possible to create an RBD image on an erasure coded pools when a replicated pool tier set a cache tier:

```
$ rbd --pool ecpool create --size 10 myvolume
```

More information can be found in the cache tiering documentation.

Glossary

chunk when the encoding function is called, it returns chunks of the same size. Data chunks which can be concatenated to reconstruct the original object and coding chunks which can be used to rebuild a lost chunk.

K the number of data *chunks*, i.e. the number of *chunks* in which the original object is divided. For instance if $K = 2$ a 10KB object will be divided into K objects of 5KB each.

M the number of coding *chunks*, i.e. the number of additional *chunks* computed by the encoding functions. If there are 2 coding *chunks*, it means 2 OSDs can be out without losing data.

Table of content

Erasure code profiles

Erasure code is defined by a **profile** and is used when creating an erasure coded pool and the associated crush ruleset.

The **default** erasure code profile (which is created when the Ceph cluster is initialized) provides the same level of redundancy as two copies but requires 25% less disk space. It is described as a profile with **k=2** and **m=1**, meaning the information is spread over three OSD (k+m == 3) and one of them can be lost.

To improve redundancy without increasing raw storage requirements, a new profile can be created. For instance, a profile with **k=10** and **m=4** can sustain the loss of four (**m=4**) OSDs by distributing an object on fourteen (k+m=14) OSDs. The object is first divided in **10** chunks (if the object is 10MB, each chunk is 1MB) and **4** coding chunks are computed, for recovery (each coding chunk has the same size as the data chunk, i.e. 1MB). The raw space overhead is only 40% and the object will not be lost even if four OSDs break at the same time.

Jerasure erasure code plugin The *jerasure* plugin is the most generic and flexible plugin, it is also the default for Ceph erasure coded pools.

The *jerasure* plugin encapsulates the Jerasure library. It is recommended to read the *jerasure* documentation to get a better understanding of the parameters.

Create a jerasure profile To create a new *jerasure* erasure code profile:

```
ceph osd erasure-code-profile set {name} \
     plugin=jerasure \
     k={data-chunks} \
     m={coding-chunks} \
     technique={reed_sol_van|reed_sol_r6_op|cauchy_orig|cauchy_good|liberation|blaum_roth|liber8tion
     [ruleset-root={root}] \
     [ruleset-failure-domain={bucket-type}] \
     [directory={directory}] \
     [--force]
```

Where:

`k={data chunks}`

Description Each object is split in **data-chunks** parts, each stored on a different OSD.

Type Integer

Required Yes.

Example 4

`m={coding-chunks}`

Description Compute **coding chunks** for each object and store them on different OSDs. The number of coding chunks is also the number of OSDs that can be down without losing data.

Type Integer

Required Yes.

Example 2

`technique={reed_sol_van|reed_sol_r6_op|cauchy_orig|cauchy_good|liberation|blaum_roth|liber`

Description The more flexible technique is *reed_sol_van* : it is enough to set *k* and *m*. The *cauchy_good* technique can be faster but you need to chose the *packetsize* carefully. All of *reed_sol_r6_op*, *liberation*, *blaum_roth*, *liber8tion* are *RAID6* equivalents in the sense that they can only be configured with *m=2*.

Type String

Required No.

Default reed_sol_van

`packetsize={bytes}`

Description The encoding will be done on packets of *bytes* size at a time. Chosing the right packet size is difficult. The *jerasure* documentation contains extensive information on this topic.

Type Integer

Required No.

Default 2048

`ruleset-root={root}`

Description The name of the crush bucket used for the first step of the ruleset. For intance **step take default**.

Type String

Required No.

Default default

`ruleset-failure-domain={bucket-type}`

Description Ensure that no two chunks are in a bucket with the same failure domain. For instance, if the failure domain is **host** no two chunks will be stored on the same host. It is used to create a ruleset step such as **step chooseleaf host**.

Type String

Required No.

Default host

`directory={directory}`

Description Set the **directory** name from which the erasure code plugin is loaded.

Type String

Required No.

Default /usr/lib/ceph/erasure-code

`--force`

Description Override an existing profile by the same name.

Type String

Required No.

ISA erasure code plugin The *isa* plugin encapsulates the ISA library. It only runs on Intel processors.

Create an isa profile To create a new *isa* erasure code profile:

```
ceph osd erasure-code-profile set {name} \
     plugin=isa \
     technique={reed_sol_van|cauchy} \
     [k={data-chunks}] \
     [m={coding-chunks}] \
     [ruleset-root={root}] \
     [ruleset-failure-domain={bucket-type}] \
     [directory={directory}] \
     [--force]
```

Where:

`k={data chunks}`

Description Each object is split in **data-chunks** parts, each stored on a different OSD.

Type Integer

Required No.

Default 7

`m={coding-chunks}`

Description Compute **coding chunks** for each object and store them on different OSDs. The number of coding chunks is also the number of OSDs that can be down without losing data.

Type Integer

Required No.

Default 3

`technique={reed_sol_van|cauchy}`

Description The ISA plugin comes in two Reed Solomon forms. If *reed_sol_van* is set, it is Vandermonde, if *cauchy* is set, it is Cauchy.

Type String

Required No.

Default reed_sol_van

`ruleset-root={root}`

Description The name of the crush bucket used for the first step of the ruleset. For intance **step take default**.

Type String

Required No.

Default default

`ruleset-failure-domain={bucket-type}`

Description Ensure that no two chunks are in a bucket with the same failure domain. For instance, if the failure domain is **host** no two chunks will be stored on the same host. It is used to create a ruleset step such as **step chooseleaf host**.

Type String

Required No.

Default host

`directory={directory}`

Description Set the **directory** name from which the erasure code plugin is loaded.

Type String

Required No.

Default /usr/lib/ceph/erasure-code

`--force`

Description Override an existing profile by the same name.

Type String

Required No.

Locally repairable erasure code plugin With the *jerasure* plugin, when an erasure coded object is stored on multiple OSDs, recovering from the loss of one OSD requires reading from all the others. For instance if *jerasure* is configured with *k=8* and *m=4*, losing one OSD requires reading from the eleven others to repair.

The *lrc* erasure code plugin creates local parity chunks to be able to recover using less OSDs. For instance if *lrc* is configured with *k=8*, *m=4* and *l=4*, it will create an additional parity chunk for every four OSDs. When a single OSD is lost, it can be recovered with only four OSDs instead of eleven.

Erasure code profile examples

Reduce recovery bandwidth between hosts Although it is probably not an interesting use case when all hosts are connected to the same switch, reduced bandwidth usage can actually be observed.:

```
$ ceph osd erasure-code-profile set LRCprofile \
     plugin=lrc \
     k=4 m=2 l=3 \
     ruleset-failure-domain=host
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

Reduce recovery bandwidth between racks In Firefly the reduced bandwidth will only be observed if the primary OSD is in the same rack as the lost chunk.:

```
$ ceph osd erasure-code-profile set LRCprofile \
     plugin=lrc \
     k=4 m=2 l=3 \
     ruleset-locality=rack \
     ruleset-failure-domain=host
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

Create an lrc profile To create a new lrc erasure code profile:

```
ceph osd erasure-code-profile set {name} \
     plugin=lrc \
     k={data-chunks} \
     m={coding-chunks} \
     l={locality} \
     [ruleset-root={root}] \
     [ruleset-locality={bucket-type}] \
     [ruleset-failure-domain={bucket-type}] \
     [directory={directory}] \
     [--force]
```

Where:

`k={data chunks}`

Description Each object is split in **data-chunks** parts, each stored on a different OSD.

Type Integer

Required Yes.

Example 4

`m={coding-chunks}`

Description Compute **coding chunks** for each object and store them on different OSDs. The number of coding chunks is also the number of OSDs that can be down without losing data.

Type Integer

Required Yes.

Example 2

`l={locality}`

Description Group the coding and data chunks into sets of size **locality**. For instance, for **k=4** and **m=2**, when **locality=3** two groups of three are created. Each set can be recovered without reading chunks from another set.

Type Integer

Required Yes.

Example 3

`ruleset-root={root}`

Description The name of the crush bucket used for the first step of the ruleset. For intance **step take default**.

Type String

Required No.

Default default

`ruleset-locality={bucket-type}`

Description The type of the crush bucket in which each set of chunks defined by **l** will be stored. For instance, if it is set to **rack**, each group of **l** chunks will be placed in a different rack. It is used to create a ruleset step such as **step choose rack**. If it is not set, no such grouping is done.

Type String

Required No.

`ruleset-failure-domain={bucket-type}`

Description Ensure that no two chunks are in a bucket with the same failure domain. For instance, if the failure domain is **host** no two chunks will be stored on the same host. It is used to create a ruleset step such as **step chooseleaf host**.

Type String

Required No.

Default host

`directory={directory}`

Description Set the **directory** name from which the erasure code plugin is loaded.

Type String

Required No.

Default /usr/lib/ceph/erasure-code

`--force`

Description Override an existing profile by the same name.

Type String

Required No.

Low level plugin configuration The sum of **k** and **m** must be a multiple of the **l** parameter. The low level configuration parameters do not impose such a restriction and it may be more convienient to use it for specific purposes. It is for instance possible to define two groups, one with 4 chunks and another with 3 chunks. It is also possible to recursively define locality sets, for instance datacenters and racks into datacenters. The **k/m/l** are implemented by generating a low level configuration.

The *lrc* erasure code plugin recursively applies erasure code techniques so that recovering from the loss of some chunks only requires a subset of the available chunks, most of the time.

For instance, when three coding steps are described as:

```
chunk nr    01234567
step 1      _cDD_cDD
step 2      cDDD____
step 3      ____cDDD
```

where *c* are coding chunks calculated from the data chunks *D*, the loss of chunk *7* can be recovered with the last four chunks. And the loss of chun *2* chunk can be recovered with the first four chunks.

Erasure code profile examples using low level configuration

Minimal testing It is strictly equivalent to using the default erasure code profile. The *DD* implies *K=2*, the *c* implies *M=1* and the *jerasure* plugin is used by default.:

```
$ ceph osd erasure-code-profile set LRCprofile \
     plugin=lrc \
     mapping=DD_ \
     layers='[ [ "DDc", "" ] ]'
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

Reduce recovery bandwidth between hosts Although it is probably not an interesting use case when all hosts are connected to the same switch, reduced bandwidth usage can actually be observed. It is equivalent to **k=4**, **m=2** and **l=3** although the layout of the chunks is different:

```
$ ceph osd erasure-code-profile set LRCprofile \
     plugin=lrc \
     mapping=__DD__DD \
     layers='[
               [ "_cDD_cDD", "" ],
               [ "cDDD____", "" ],
               [ "____cDDD", "" ],
             ]'
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

Reduce recovery bandwidth between racks In Firefly the reduced bandwidth will only be observed if the primary OSD is in the same rack as the lost chunk.:

```
$ ceph osd erasure-code-profile set LRCprofile \
     plugin=lrc \
     mapping=__DD__DD \
     layers='[
               [ "_cDD_cDD", "" ],
               [ "cDDD____", "" ],
               [ "____cDDD", "" ],
             ]' \
     ruleset-steps='[
                     [ "choose", "rack", 2 ],
                     [ "chooseleaf", "host", 4 ],
                    ]'
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

Testing with different Erasure Code backends LRC now uses jerasure as the default EC backend. It is possible to specify the EC backend/algorithm on a per layer basis using the low level configuration. The second argument in layers='[["DDc", ""]]' is actually an erasure code profile to be used for this level. The example below specifies the ISA backend with the cauchy technique to be used in the lrcpool.:

```
$ ceph osd erasure-code-profile set LRCprofile \
     plugin=lrc \
     mapping=DD_ \
     layers='[ [ "DDc", "plugin=isa technique=cauchy" ] ]'
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

You could also use a different erasure code profile for for each layer.:

```
$ ceph osd erasure-code-profile set LRCprofile \
     plugin=lrc \
     mapping=__DD__DD \
     layers='[
               [ "_cDD_cDD", "plugin=isa technique=cauchy" ],
               [ "cDDD____", "plugin=isa" ],
               [ "____cDDD", "plugin=jerasure" ],
             ]'
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

Erasure coding and decoding algorithm The steps found in the layers description:

```
chunk nr    01234567

step 1      _cDD_cDD
step 2      cDDD____
step 3      ____cDDD
```

are applied in order. For instance, if a 4K object is encoded, it will first go thru *step 1* and be divided in four 1K chunks (the four uppercase D). They are stored in the chunks 2, 3, 6 and 7, in order. From these, two coding chunks are calculated (the two lowercase c). The coding chunks are stored in the chunks 1 and 4, respectively.

The *step 2* re-uses the content created by *step 1* in a similar fashion and stores a single coding chunk *c* at position 0. The last four chunks, marked with an underscore (_) for readability, are ignored.

The *step 3* stores a single coding chunk *c* at position 4. The three chunks created by *step 1* are used to compute this coding chunk, i.e. the coding chunk from *step 1* becomes a data chunk in *step 3*.

If chunk *2* is lost:

```
chunk nr    01234567

step 1      _c D_cDD
step 2      cD D____
step 3      __ _cDDD
```

decoding will attempt to recover it by walking the steps in reverse order: *step 3* then *step 2* and finally *step 1*.

The *step 3* knows nothing about chunk *2* (i.e. it is an underscore) and is skipped.

The coding chunk from *step 2*, stored in chunk *0*, allows it to recover the content of chunk *2*. There are no more chunks to recover and the process stops, without considering *step 1*.

Recovering chunk *2* required reading chunks *0, 1, 3* and writing back chunk *2*.

If chunk *2, 3, 6* are lost:

```
chunk nr    01234567

step 1      _c  _c D
step 2      cD  __ _
step 3      __  cD D
```

The *step 3* can recover the conten of chunk *6*:

```
chunk nr    01234567

step 1      _c  _cDD
step 2      cD  ____
step 3      __  cDDD
```

The *step 2* fails to recover and is skipped because there are two chunks missing (*2, 3*) and it can only recover from one missing chunk.

The coding chunk from *step 1*, stored in chunk *1, 5*, allows it to recover the content of chunk *2, 3*:

```
chunk nr    01234567

step 1      _cDD_cDD
step 2      cDDD____
step 3      ____cDDD
```

Controlling crush placement The default crush ruleset provides OSDs that are on different hosts. For instance:

```
chunk nr    01234567

step 1      _cDD_cDD
step 2      cDDD____
step 3      ____cDDD
```

needs exactly *8* OSDs, one for each chunk. If the hosts are in two adjacent racks, the first four chunks can be placed in the first rack and the last four in the second rack. So that recovering from the loss of a single OSD does not require using bandwidth between the two racks.

For instance:

```
ruleset-steps='[ [ "choose", "rack", 2 ], [ "chooseleaf", "host", 4 ] ]'
```

will create a ruleset that will select two crush buckets of type *rack* and for each of them choose four OSDs, each of them located in different bucket of type *host*.

The ruleset can also be manually crafted for finer control.

SHEC erasure code plugin The *shec* plugin encapsulates the multiple SHEC library. It allows ceph to recover data more efficiently than Reed Solomon codes.

Create an SHEC profile To create a new *shec* erasure code profile:

```
ceph osd erasure-code-profile set {name} \
     plugin=shec \
     [k={data-chunks}] \
     [m={coding-chunks}] \
     [c={durability-estimator}] \
     [ruleset-root={root}] \
     [ruleset-failure-domain={bucket-type}] \
     [directory={directory}] \
     [--force]
```

Where:

`k={data-chunks}`

Description Each object is split in **data-chunks** parts, each stored on a different OSD.

Type Integer

Required No.

Default 4

`m={coding-chunks}`

Description Compute **coding-chunks** for each object and store them on different OSDs. The number of **coding-chunks** does not necessarily equal the number of OSDs that can be down without losing data.

Type Integer

Required No.

Default 3

`c={durability-estimator}`

Description The number of parity chunks each of which includes each data chunk in its calculation range. The number is used as a **durability estimator**. For instance, if c=2, 2 OSDs can be down without losing data.

Type Integer

Required No.

Default 2

`ruleset-root={root}`

Description The name of the crush bucket used for the first step of the ruleset. For intance **step take default**.

Type String

Required No.

Default default

`ruleset-failure-domain={bucket-type}`

Description Ensure that no two chunks are in a bucket with the same failure domain. For instance, if the failure domain is **host** no two chunks will be stored on the same host. It is used to create a ruleset step such as **step chooseleaf host**.

Type String

Required No.

Default host

`directory={directory}`

Description Set the **directory** name from which the erasure code plugin is loaded.

Type String

Required No.

Default /usr/lib/ceph/erasure-code

`--force`

Description Override an existing profile by the same name.

Type String

Required No.

Brief description of SHEC's layouts

Space Efficiency Space efficiency is a ratio of data chunks to all ones in a object and represented as k/(k+m). In order to improve space efficiency, you should increase k or decrease m.

```
space efficiency of SHEC(4,3,2) = 4/(4+3) = 0.57
SHEC(5,3,2) or SHEC(4,2,2) improves SHEC(4,3,2)'s space efficiency
```

Durability The third parameter of SHEC (=c) is a durability estimator, which approximates the number of OSDs that can be down without losing data.

```
durability estimator of SHEC(4,3,2) = 2
```

Recovery Efficiency Describing calculation of recovery efficiency is beyond the scope of this document, but at least increasing m without increasing c achieves improvement of recovery efficiency. (However, we must pay attention to the sacrifice of space efficiency in this case.)

```
SHEC(4,2,2) -> SHEC(4,3,2) :  achieves improvement of recovery efficiency
```

Erasure code profile examples

```
$ ceph osd erasure-code-profile set SHECprofile \
     plugin=shec \
     k=8 m=4 c=3 \
     ruleset-failure-domain=host
$ ceph osd pool create shecpool 256 256 erasure SHECprofile
```

osd erasure-code-profile set To create a new erasure code profile:

```
ceph osd erasure-code-profile set {name} \
     [{directory=directory}] \
     [{plugin=plugin}] \
     [{key=value} ...] \
     [--force]
```

Where:

`{directory=directory}`

Description Set the **directory** name from which the erasure code plugin is loaded.

Type String

Required No.

Default /usr/lib/ceph/erasure-code

`{plugin=plugin}`

Description Use the erasure code **plugin** to compute coding chunks and recover missing chunks. See the *list of available plugins* for more information.

Type String

Required No.

Default jerasure

`{key=value}`

Description The semantic of the remaining key/value pairs is defined by the erasure code plugin.

Type String

Required No.

`--force`

Description Override an existing profile by the same name.

Type String

Required No.

osd erasure-code-profile rm To remove an erasure code profile:

```
ceph osd erasure-code-profile rm {name}
```

If the profile is referenced by a pool, the deletion will fail.

osd erasure-code-profile get To display an erasure code profile:

```
ceph osd erasure-code-profile get {name}
```

osd erasure-code-profile ls To list the names of all erasure code profiles:

```
ceph osd erasure-code-profile ls
```

Jerasure erasure code plugin

The *jerasure* plugin is the most generic and flexible plugin, it is also the default for Ceph erasure coded pools.

The *jerasure* plugin encapsulates the Jerasure library. It is recommended to read the *jerasure* documentation to get a better understanding of the parameters.

Create a jerasure profile To create a new *jerasure* erasure code profile:

```
ceph osd erasure-code-profile set {name} \
     plugin=jerasure \
     k={data-chunks} \
     m={coding-chunks} \
     technique={reed_sol_van|reed_sol_r6_op|cauchy_orig|cauchy_good|liberation|blaum_roth|liber8tion
     [ruleset-root={root}] \
     [ruleset-failure-domain={bucket-type}] \
     [directory={directory}] \
     [--force]
```

Where:

`k={data chunks}`

Description Each object is split in **data-chunks** parts, each stored on a different OSD.

Type Integer

Required Yes.

Example 4

`m={coding-chunks}`

Description Compute **coding chunks** for each object and store them on different OSDs. The number of coding chunks is also the number of OSDs that can be down without losing data.

Type Integer

Required Yes.

Example 2

`technique={reed_sol_van|reed_sol_r6_op|cauchy_orig|cauchy_good|liberation|blaum_roth|liber8`

Description The more flexible technique is *reed_sol_van* : it is enough to set *k* and *m*. The *cauchy_good* technique can be faster but you need to chose the *packetsize* carefully. All of *reed_sol_r6_op*, *liberation*, *blaum_roth*, *liber8tion* are *RAID6* equivalents in the sense that they can only be configured with *m=2*.

Type String

Required No.

Default reed_sol_van

`packetsize={bytes}`

Description The encoding will be done on packets of *bytes* size at a time. Chosing the right packet size is difficult. The *jerasure* documentation contains extensive information on this topic.

Type Integer

Required No.

Default 2048

`ruleset-root={root}`

Description The name of the crush bucket used for the first step of the ruleset. For intance **step take default**.

Type String

Required No.

Default default

`ruleset-failure-domain={bucket-type}`

Description Ensure that no two chunks are in a bucket with the same failure domain. For instance, if the failure domain is **host** no two chunks will be stored on the same host. It is used to create a ruleset step such as **step chooseleaf host**.

Type String

Required No.

Default host

`directory={directory}`

Description Set the **directory** name from which the erasure code plugin is loaded.

Type String

Required No.

Default /usr/lib/ceph/erasure-code

`--force`

Description Override an existing profile by the same name.

Type String

Required No.

ISA erasure code plugin

The *isa* plugin encapsulates the ISA library. It only runs on Intel processors.

Create an isa profile To create a new *isa* erasure code profile:

```
ceph osd erasure-code-profile set {name} \
     plugin=isa \
     technique={reed_sol_van|cauchy} \
     [k={data-chunks}] \
     [m={coding-chunks}] \
     [ruleset-root={root}] \
     [ruleset-failure-domain={bucket-type}] \
     [directory={directory}] \
     [--force]
```

Where:

`k={data chunks}`

Description Each object is split in **data-chunks** parts, each stored on a different OSD.

Type Integer

Required No.

Default 7

`m={coding-chunks}`

Description Compute **coding chunks** for each object and store them on different OSDs. The number of coding chunks is also the number of OSDs that can be down without losing data.

Type Integer

Required No.

Default 3

`technique={reed_sol_van|cauchy}`

Description The ISA plugin comes in two Reed Solomon forms. If *reed_sol_van* is set, it is Vandermonde, if *cauchy* is set, it is Cauchy.

Type String

Required No.

Default reed_sol_van

`ruleset-root={root}`

Description The name of the crush bucket used for the first step of the ruleset. For intance **step take default**.

Type String

Required No.

Default default

`ruleset-failure-domain={bucket-type}`

Description Ensure that no two chunks are in a bucket with the same failure domain. For instance, if the failure domain is **host** no two chunks will be stored on the same host. It is used to create a ruleset step such as **step chooseleaf host**.

Type String

Required No.

Default host

`directory={directory}`

Description Set the **directory** name from which the erasure code plugin is loaded.

Type String

Required No.

Default /usr/lib/ceph/erasure-code

`--force`

Description Override an existing profile by the same name.

Type String

Required No.

Locally repairable erasure code plugin

With the *jerasure* plugin, when an erasure coded object is stored on multiple OSDs, recovering from the loss of one OSD requires reading from all the others. For instance if *jerasure* is configured with *k=8* and *m=4*, losing one OSD requires reading from the eleven others to repair.

The *lrc* erasure code plugin creates local parity chunks to be able to recover using less OSDs. For instance if *lrc* is configured with *k=8*, *m=4* and *l=4*, it will create an additional parity chunk for every four OSDs. When a single OSD is lost, it can be recovered with only four OSDs instead of eleven.

Erasure code profile examples

Reduce recovery bandwidth between hosts Although it is probably not an interesting use case when all hosts are connected to the same switch, reduced bandwidth usage can actually be observed.:

```
$ ceph osd erasure-code-profile set LRCprofile \
     plugin=lrc \
     k=4 m=2 l=3 \
     ruleset-failure-domain=host
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

Reduce recovery bandwidth between racks In Firefly the reduced bandwidth will only be observed if the primary OSD is in the same rack as the lost chunk.:

```
$ ceph osd erasure-code-profile set LRCprofile \
     plugin=lrc \
     k=4 m=2 l=3 \
     ruleset-locality=rack \
     ruleset-failure-domain=host
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

Create an lrc profile To create a new lrc erasure code profile:

```
ceph osd erasure-code-profile set {name} \
     plugin=lrc \
     k={data-chunks} \
     m={coding-chunks} \
     l={locality} \
     [ruleset-root={root}] \
     [ruleset-locality={bucket-type}] \
     [ruleset-failure-domain={bucket-type}] \
     [directory={directory}] \
     [--force]
```

Where:

`k={data chunks}`

Description Each object is split in **data-chunks** parts, each stored on a different OSD.

Type Integer

Required Yes.

Example 4

`m={coding-chunks}`

Description Compute **coding chunks** for each object and store them on different OSDs. The number of coding chunks is also the number of OSDs that can be down without losing data.

Type Integer

Required Yes.

Example 2

`l={locality}`

Description Group the coding and data chunks into sets of size **locality**. For instance, for **k=4** and **m=2**, when **locality=3** two groups of three are created. Each set can be recovered without reading chunks from another set.

Type Integer

Required Yes.

Example 3

`ruleset-root={root}`

Description The name of the crush bucket used for the first step of the ruleset. For intance **step take default**.

Type String

Required No.

Default default

`ruleset-locality={bucket-type}`

Description The type of the crush bucket in which each set of chunks defined by **l** will be stored. For instance, if it is set to **rack**, each group of **l** chunks will be placed in a different rack. It is used to create a ruleset step such as **step choose rack**. If it is not set, no such grouping is done.

Type String

Required No.

`ruleset-failure-domain={bucket-type}`

Description Ensure that no two chunks are in a bucket with the same failure domain. For instance, if the failure domain is **host** no two chunks will be stored on the same host. It is used to create a ruleset step such as **step chooseleaf host**.

Type String

Required No.

Default host

`directory={directory}`

Description Set the **directory** name from which the erasure code plugin is loaded.

Type String

Required No.

Default /usr/lib/ceph/erasure-code

`--force`

Description Override an existing profile by the same name.

Type String

Required No.

Low level plugin configuration The sum of **k** and **m** must be a multiple of the **l** parameter. The low level configuration parameters do not impose such a restriction and it may be more convienient to use it for specific purposes. It is for instance possible to define two groups, one with 4 chunks and another with 3 chunks. It is also possible to recursively define locality sets, for instance datacenters and racks into datacenters. The **k/m/l** are implemented by generating a low level configuration.

The *lrc* erasure code plugin recursively applies erasure code techniques so that recovering from the loss of some chunks only requires a subset of the available chunks, most of the time.

For instance, when three coding steps are described as:

```
chunk nr    01234567
step 1      _cDD_cDD
step 2      cDDD____
step 3      ____cDDD
```

where *c* are coding chunks calculated from the data chunks *D*, the loss of chunk *7* can be recovered with the last four chunks. And the loss of chun *2* chunk can be recovered with the first four chunks.

Erasure code profile examples using low level configuration

Minimal testing It is strictly equivalent to using the default erasure code profile. The *DD* implies *K=2*, the *c* implies *M=1* and the *jerasure* plugin is used by default.:

```
$ ceph osd erasure-code-profile set LRCprofile \
     plugin=lrc \
     mapping=DD_ \
     layers='[ [ "DDc", "" ] ]'
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

Reduce recovery bandwidth between hosts Although it is probably not an interesting use case when all hosts are connected to the same switch, reduced bandwidth usage can actually be observed. It is equivalent to **k=4**, **m=2** and **l=3** although the layout of the chunks is different:

```
$ ceph osd erasure-code-profile set LRCprofile \
     plugin=lrc \
     mapping=__DD__DD \
     layers='[
               [ "_cDD_cDD", "" ],
               [ "cDDD____", "" ],
               [ "____cDDD", "" ],
             ]'
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

Reduce recovery bandwidth between racks In Firefly the reduced bandwidth will only be observed if the primary OSD is in the same rack as the lost chunk.:

```
$ ceph osd erasure-code-profile set LRCprofile \
     plugin=lrc \
     mapping=__DD__DD \
     layers='[
               [ "_cDD_cDD", "" ],
               [ "cDDD____", "" ],
               [ "____cDDD", "" ],
             ]' \
     ruleset-steps='[
                     [ "choose", "rack", 2 ],
                     [ "chooseleaf", "host", 4 ],
                    ]'
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

Testing with different Erasure Code backends LRC now uses jerasure as the default EC backend. It is possible to specify the EC backend/algorithm on a per layer basis using the low level configuration. The second argument in layers='[["DDc", ""]]' is actually an erasure code profile to be used for this level. The example below specifies the ISA backend with the cauchy technique to be used in the lrcpool.:

```
$ ceph osd erasure-code-profile set LRCprofile \
     plugin=lrc \
     mapping=DD_ \
     layers='[ [ "DDc", "plugin=isa technique=cauchy" ] ]'
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

You could also use a different erasure code profile for for each layer.:

```
$ ceph osd erasure-code-profile set LRCprofile \
     plugin=lrc \
     mapping=__DD__DD \
     layers='[
               [ "_cDD_cDD", "plugin=isa technique=cauchy" ],
               [ "cDDD____", "plugin=isa" ],
               [ "____cDDD", "plugin=jerasure" ],
            ]'
$ ceph osd pool create lrcpool 12 12 erasure LRCprofile
```

Erasure coding and decoding algorithm The steps found in the layers description:

```
chunk nr    01234567

step 1      _cDD_cDD
step 2      cDDD____
step 3      ____cDDD
```

are applied in order. For instance, if a 4K object is encoded, it will first go thru *step 1* and be divided in four 1K chunks (the four uppercase D). They are stored in the chunks 2, 3, 6 and 7, in order. From these, two coding chunks are calculated (the two lowercase c). The coding chunks are stored in the chunks 1 and 4, respectively.

The *step 2* re-uses the content created by *step 1* in a similar fashion and stores a single coding chunk *c* at position 0. The last four chunks, marked with an underscore (_) for readability, are ignored.

The *step 3* stores a single coding chunk *c* at position 4. The three chunks created by *step 1* are used to compute this coding chunk, i.e. the coding chunk from *step 1* becomes a data chunk in *step 3*.

If chunk *2* is lost:

```
chunk nr    01234567

step 1      _c D_cDD
step 2      cD D____
step 3      __ _cDDD
```

decoding will attempt to recover it by walking the steps in reverse order: *step 3* then *step 2* and finally *step 1*.

The *step 3* knows nothing about chunk *2* (i.e. it is an underscore) and is skipped.

The coding chunk from *step 2*, stored in chunk *0*, allows it to recover the content of chunk *2*. There are no more chunks to recover and the process stops, without considering *step 1*.

Recovering chunk *2* required reading chunks *0, 1, 3* and writing back chunk *2*.

If chunk *2, 3, 6* are lost:

```
chunk nr    01234567

step 1      _c  _c D
step 2      cD  __ _
step 3      __  cD D
```

The *step 3* can recover the conten of chunk *6*:

```
chunk nr    01234567

step 1      _c  _cDD
step 2      cD  ____
step 3      __  cDDD
```

The *step 2* fails to recover and is skipped because there are two chunks missing (*2, 3*) and it can only recover from one missing chunk.

The coding chunk from *step 1*, stored in chunk *1, 5*, allows it to recover the content of chunk *2, 3*:

```
chunk nr    01234567

step 1      _cDD_cDD
step 2      cDDD____
step 3      ____cDDD
```

Controlling crush placement The default crush ruleset provides OSDs that are on different hosts. For instance:

```
chunk nr    01234567

step 1      _cDD_cDD
step 2      cDDD____
step 3      ____cDDD
```

needs exactly *8* OSDs, one for each chunk. If the hosts are in two adjacent racks, the first four chunks can be placed in the first rack and the last four in the second rack. So that recovering from the loss of a single OSD does not require using bandwidth between the two racks.

For instance:

```
ruleset-steps='[ [ "choose", "rack", 2 ], [ "chooseleaf", "host", 4 ] ]'
```

will create a ruleset that will select two crush buckets of type *rack* and for each of them choose four OSDs, each of them located in different bucket of type *host*.

The ruleset can also be manually crafted for finer control.

SHEC erasure code plugin

The *shec* plugin encapsulates the multiple SHEC library. It allows ceph to recover data more efficiently than Reed Solomon codes.

Create an SHEC profile To create a new *shec* erasure code profile:

```
ceph osd erasure-code-profile set {name} \
     plugin=shec \
     [k={data-chunks}] \
     [m={coding-chunks}] \
     [c={durability-estimator}] \
     [ruleset-root={root}] \
     [ruleset-failure-domain={bucket-type}] \
     [directory={directory}] \
     [--force]
```

Where:

`k={data-chunks}`

Description Each object is split in **data-chunks** parts, each stored on a different OSD.

Type Integer

Required No.

Default 4

`m={coding-chunks}`

Description Compute **coding-chunks** for each object and store them on different OSDs. The number of **coding-chunks** does not necessarily equal the number of OSDs that can be down without losing data.

Type Integer

Required No.

Default 3

`c={durability-estimator}`

Description The number of parity chunks each of which includes each data chunk in its calculation range. The number is used as a **durability estimator**. For instance, if c=2, 2 OSDs can be down without losing data.

Type Integer

Required No.

Default 2

`ruleset-root={root}`

Description The name of the crush bucket used for the first step of the ruleset. For intance **step take default**.

Type String

Required No.

Default default

`ruleset-failure-domain={bucket-type}`

Description Ensure that no two chunks are in a bucket with the same failure domain. For instance, if the failure domain is **host** no two chunks will be stored on the same host. It is used to create a ruleset step such as **step chooseleaf host**.

Type String

Required No.

Default host

`directory={directory}`

Description Set the **directory** name from which the erasure code plugin is loaded.

Type String

Required No.

Default /usr/lib/ceph/erasure-code

`--force`

Description Override an existing profile by the same name.

Type String

Required No.

Brief description of SHEC's layouts

Space Efficiency Space efficiency is a ratio of data chunks to all ones in a object and represented as k/(k+m). In order to improve space efficiency, you should increase k or decrease m.

```
space efficiency of SHEC(4,3,2) = 4/(4+3) = 0.57
SHEC(5,3,2) or SHEC(4,2,2) improves SHEC(4,3,2)'s space efficiency
```

Durability The third parameter of SHEC (=c) is a durability estimator, which approximates the number of OSDs that can be down without losing data.

```
durability estimator of SHEC(4,3,2) = 2
```

Recovery Efficiency Describing calculation of recovery efficiency is beyond the scope of this document, but at least increasing m without increasing c achieves improvement of recovery efficiency. (However, we must pay attention to the sacrifice of space efficiency in this case.)

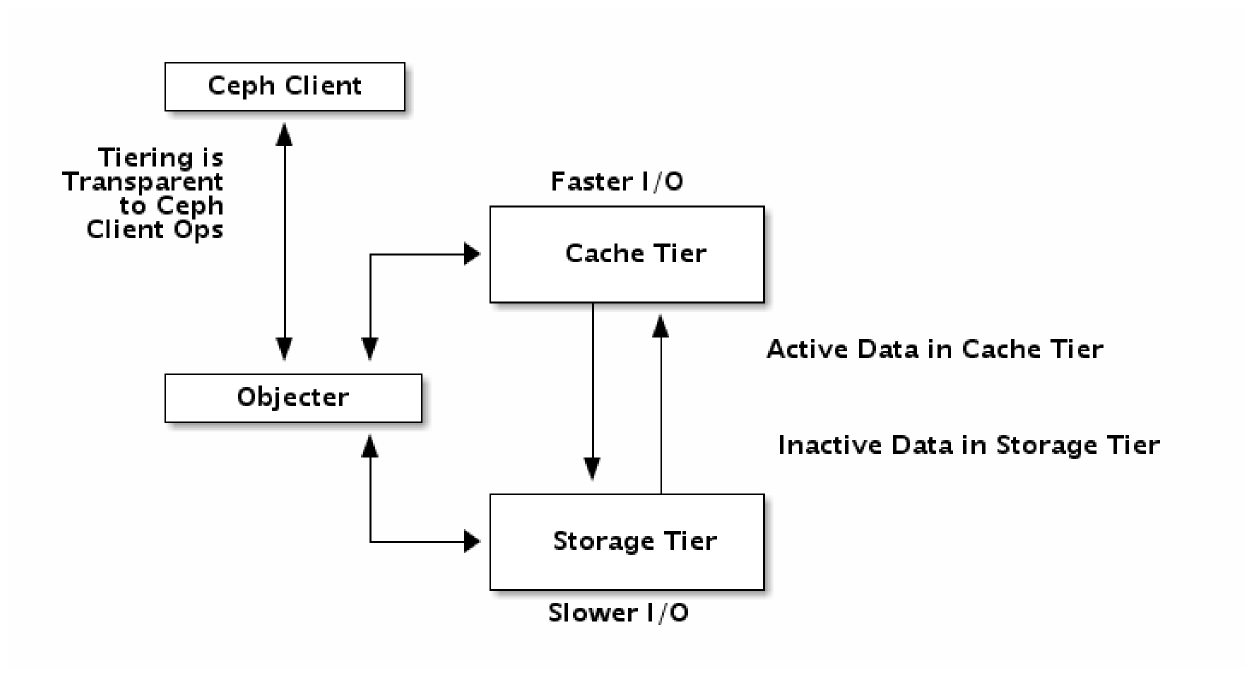
```
SHEC(4,2,2) -> SHEC(4,3,2) :  achieves improvement of recovery efficiency
```

Erasure code profile examples

```
$ ceph osd erasure-code-profile set SHECprofile \
     plugin=shec \
     k=8 m=4 c=3 \
     ruleset-failure-domain=host
$ ceph osd pool create shecpool 256 256 erasure SHECprofile
```

4.3.8 Cache Tiering

A cache tier provides Ceph Clients with better I/O performance for a subset of the data stored in a backing storage tier. Cache tiering involves creating a pool of relatively fast/expensive storage devices (e.g., solid state drives) configured to act as a cache tier, and a backing pool of either erasure-coded or relatively slower/cheaper devices configured to act as an economical storage tier. The Ceph objecter handles where to place the objects and the tiering agent determines when to flush objects from the cache to the backing storage tier. So the cache tier and the backing storage tier are completely transparent to Ceph clients.


Ceph Client
Tiering is Transparent to Ceph Client Ops
Faster I/O
Cache Tier
Active Data in Cache Tier
Objecter
Inactive Data in Storage Tier
Storage Tier
Slower I/O

The cache tiering agent handles the migration of data between the cache tier and the backing storage tier automatically. However, admins have the ability to configure how this migration takes place. There are two main scenarios:

- **Writeback Mode:** When admins configure tiers with `writeback` mode, Ceph clients write data to the cache tier and receive an ACK from the cache tier. In time, the data written to the cache tier migrates to the storage tier and gets flushed from the cache tier. Conceptually, the cache tier is overlaid "in front" of the backing storage tier. When a Ceph client needs data that resides in the storage tier, the cache tiering agent migrates the data to the cache tier on read, then it is sent to the Ceph client. Thereafter, the Ceph client can perform I/O using the cache tier, until the data becomes inactive. This is ideal for mutable data (e.g., photo/video editing, transactional data, etc.).
- **Read-only Mode:** When admins configure tiers with `readonly` mode, Ceph clients write data to the backing tier. On read, Ceph copies the requested object(s) from the backing tier to the cache tier. Stale objects get removed from the cache tier based on the defined policy. This approach is ideal for immutable data (e.g., presenting pictures/videos on a social network, DNA data, X-Ray imaging, etc.), because reading data from a cache pool that might contain out-of-date data provides weak consistency. Do not use `readonly` mode for mutable data.

Since all Ceph clients can use cache tiering, it has the potential to improve I/O performance for Ceph Block Devices, Ceph Object Storage, the Ceph Filesystem and native bindings.

Setting Up Pools

To set up cache tiering, you must have two pools. One will act as the backing storage and the other will act as the cache.

Setting Up a Backing Storage Pool

Setting up a backing storage pool typically involves one of two scenarios:

- **Standard Storage**: In this scenario, the pool stores multiple copies of an object in the Ceph Storage Cluster.

- **Erasure Coding:** In this scenario, the pool uses erasure coding to store data much more efficiently with a small performance tradeoff.

In the standard storage scenario, you can setup a CRUSH ruleset to establish the failure domain (e.g., osd, host, chassis, rack, row, etc.). Ceph OSD Daemons perform optimally when all storage drives in the ruleset are of the same size, speed (both RPMs and throughput) and type. See CRUSH Maps for details on creating a ruleset. Once you have created a ruleset, create a backing storage pool.

In the erasure coding scenario, the pool creation arguments will generate the appropriate ruleset automatically. See Create a Pool for details.

In subsequent examples, we will refer to the backing storage pool as `cold-storage`.

Setting Up a Cache Pool

Setting up a cache pool follows the same procedure as the standard storage scenario, but with this difference: the drives for the cache tier are typically high performance drives that reside in their own servers and have their own ruleset. When setting up a ruleset, it should take account of the hosts that have the high performance drives while omitting the hosts that don't. See Placing Different Pools on Different OSDs for details.

In subsequent examples, we will refer to the cache pool as `hot-storage` and the backing pool as `cold-storage`.

For cache tier configuration and default values, see Pools - Set Pool Values.

Creating a Cache Tier

Setting up a cache tier involves associating a backing storage pool with a cache pool

```
ceph osd tier add {storagepool} {cachepool}
```

For example

```
ceph osd tier add cold-storage hot-storage
```

To set the cache mode, execute the following:

```
ceph osd tier cache-mode {cachepool} {cache-mode}
```

For example:

```
ceph osd tier cache-mode hot-storage writeback
```

The cache tiers overlay the backing storage tier, so they require one additional step: you must direct all client traffic from the storage pool to the cache pool. To direct client traffic directly to the cache pool, execute the following:

```
ceph osd tier set-overlay {storagepool} {cachepool}
```

For example:

```
ceph osd tier set-overlay cold-storage hot-storage
```

Configuring a Cache Tier

Cache tiers have several configuration options. You may set cache tier configuration options with the following usage:

```
ceph osd pool set {cachepool} {key} {value}
```

See Pools - Set Pool Values for details.

Target Size and Type

Ceph's production cache tiers use a Bloom Filter for the `hit_set_type`:

```
ceph osd pool set {cachepool} hit_set_type bloom
```

For example:

```
ceph osd pool set hot-storage hit_set_type bloom
```

The `hit_set_count` and `hit_set_period` define how much time each HitSet should cover, and how many such HitSets to store. Currently there is minimal benefit for `hit_set_count` > 1 since the agent does not yet act intelligently on that information.

```
ceph osd pool set {cachepool} hit_set_count 1
ceph osd pool set {cachepool} hit_set_period 3600
ceph osd pool set {cachepool} target_max_bytes 1000000000000
```

Binning accesses over time allows Ceph to determine whether a Ceph client accessed an object at least once, or more than once over a time period ("age" vs "temperature").

Note: The longer the period and the higher the count, the more RAM the `ceph-osd` daemon consumes. In particular, when the agent is active to flush or evict cache objects, all `hit_set_count` HitSets are loaded into RAM.

Cache Sizing

The cache tiering agent performs two main functions:

- **Flushing:** The agent identifies modified (or dirty) objects and forwards them to the storage pool for long-term storage.
- **Evicting:** The agent identifies objects that haven't been modified (or clean) and evicts the least recently used among them from the cache.

Relative Sizing The cache tiering agent can flush or evict objects relative to the size of the cache pool. When the cache pool consists of a certain percentage of modified (or dirty) objects, the cache tiering agent will flush them to the storage pool. To set the `cache_target_dirty_ratio`, execute the following:

```
ceph osd pool set {cachepool} cache_target_dirty_ratio {0.0..1.0}
```

For example, setting the value to `0.4` will begin flushing modified (dirty) objects when they reach 40% of the cache pool's capacity:

```
ceph osd pool set hot-storage cache_target_dirty_ratio 0.4
```

When the cache pool reaches a certain percentage of its capacity, the cache tiering agent will evict objects to maintain free capacity. To set the `cache_target_full_ratio`, execute the following:

```
ceph osd pool set {cachepool} cache_target_full_ratio {0.0..1.0}
```

For example, setting the value to `0.8` will begin flushing unmodified (clean) objects when they reach 80% of the cache pool's capacity:

```
ceph osd pool set hot-storage cache_target_full_ratio 0.8
```

Absolute Sizing The cache tiering agent can flush or evict objects based upon the total number of bytes or the total number of objects. To specify a maximum number of bytes, execute the following:

```
ceph osd pool set {cachepool} target_max_bytes {#bytes}
```

For example, to flush or evict at 1 TB, execute the following:

```
ceph osd pool hot-storage target_max_bytes 1000000000000
```

To specify the maximum number of objects, execute the following:

```
ceph osd pool set {cachepool} target_max_objects {#objects}
```

For example, to flush or evict at 1M objects, execute the following:

```
ceph osd pool set hot-storage target_max_objects 1000000
```

Note: If you specify both limits, the cache tiering agent will begin flushing or evicting when either threshold is triggered.

Cache Age

You can specify the minimum age of an object before the cache tiering agent flushes a recently modified (or dirty) object to the backing storage pool:

```
ceph osd pool set {cachepool} cache_min_flush_age {#seconds}
```

For example, to flush modified (or dirty) objects after 10 minutes, execute the following:

```
ceph osd pool set hot-storage cache_min_flush_age 600
```

You can specify the minimum age of an object before it will be evicted from the cache tier:

```
ceph osd pool {cache-tier} cache_min_evict_age {#seconds}
```

For example, to evict objects after 30 minutes, execute the following:

```
ceph osd pool set hot-storage cache_min_evict_age 1800
```

Removing a Cache Tier

Removing a cache tier differs depending on whether it is a writeback cache or a read-only cache.

Removing a Read-Only Cache

Since a read-only cache does not have modified data, you can disable and remove it without losing any recent changes to objects in the cache.

1. Change the cache-mode to `none` to disable it.

   ```
   ceph osd tier cache-mode {cachepool} none
   ```

   For example:

   ```
   ceph osd tier cache-mode hot-storage none
   ```

2. Remove the cache pool from the backing pool.

```
ceph osd tier remove {storagepool} {cachepool}
```

   For example:

```
ceph osd tier remove cold-storage hot-storage
```

Removing a Writeback Cache

Since a writeback cache may have modified data, you must take steps to ensure that you do not lose any recent changes to objects in the cache before you disable and remove it.

1. Change the cache mode to `forward` so that new and modified objects will flush to the backing storage pool.

```
ceph osd tier cache-mode {cachepool} forward
```

   For example:

```
ceph osd tier cache-mode hot-storage forward
```

2. Ensure that the cache pool has been flushed. This may take a few minutes:

```
rados -p {cachepool} ls
```

   If the cache pool still has objects, you can flush them manually. For example:

```
rados -p {cachepool} cache-flush-evict-all
```

3. Remove the overlay so that clients will not direct traffic to the cache.

```
ceph osd tier remove-overlay {storagetier}
```

   For example:

```
ceph osd tier remove-overlay cold-storage
```

4. Finally, remove the cache tier pool from the backing storage pool.

```
ceph osd tier remove {storagepool} {cachepool}
```

   For example:

```
ceph osd tier remove cold-storage hot-storage
```

4.3.9 Placement Groups

A preselection of pg_num

When creating a new pool with:

```
ceph osd pool set {pool-name} pg_num
```

it is mandatory to choose the value of `pg_num` because it cannot be calculated automatically. Here are a few values commonly used:

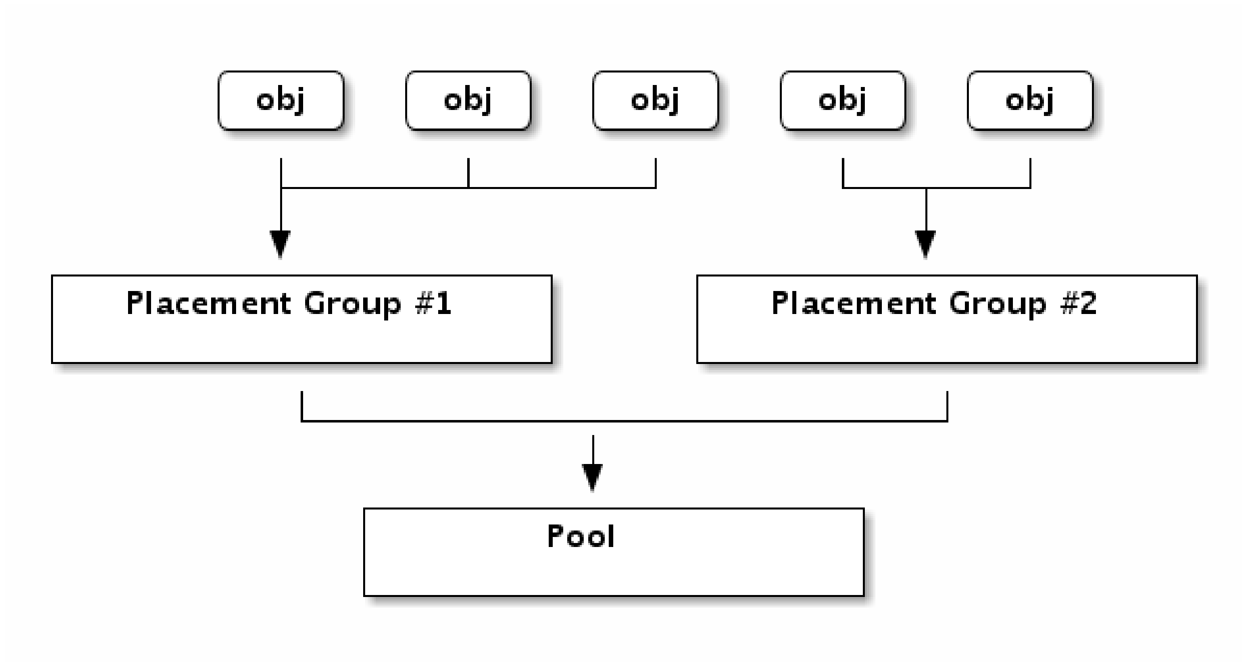
- Less than 5 OSDs set `pg_num` to 128
- Between 5 and 10 OSDs set `pg_num` to 512

- Between 10 and 50 OSDs set `pg_num` to 4096
- If you have more than 50 OSDs, you need to understand the tradeoffs and how to calculate the `pg_num` value by yourself

As the number of OSDs increases, chosing the right value for pg_num becomes more important because it has a significant influence on the behavior of the cluster as well as the durability of the data when something goes wrong (i.e. the probability that a catastrophic event leads to data loss).

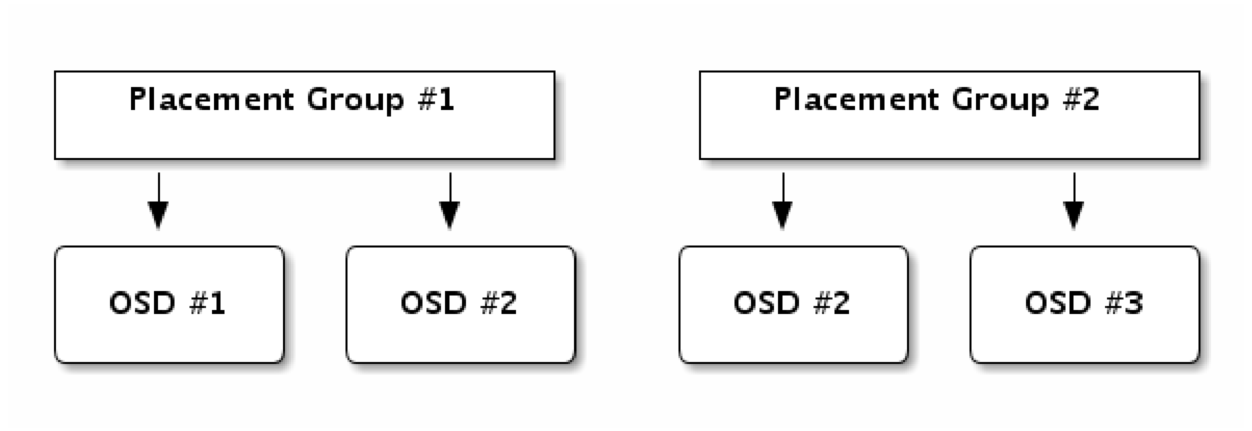
How are Placement Groups used ?

A placement group (PG) aggregates objects within a pool because tracking object placement and object metadata on a per-object basis is computationally expensive–i.e., a system with millions of objects cannot realistically track placement on a per-object basis.



Placement groups are invisible to the Ceph user: the CRUSH algorithm determines in which placement group the object will be placed. Although CRUSH is a deterministic function using the object name as a parameter, there is no way to force an object into a given placement group.

The object's contents within a placement group are stored in a set of OSDs. For instance, in a replicated pool of size two, each placement group will store objects on two OSDs, as shown below.



Should OSD #2 fail, another will be assigned to Placement Group #1 and will be filled with copies of all objects in OSD #1. If the pool size is changed from two to three, an additional OSD will be assigned to the placement group and will receive copies of all objects in the placement group.

Placement groups do not own the OSD, they share it with other placement groups from the same pool or even other pools. If OSD #2 fails, the Placement Group #2 will also have to restore copies of objects, using OSD #3.

When the number of placement groups increases, the new placement groups will be assigned OSDs. The result of the CRUSH function will also change and some objects from the former placement groups will be copied over to the new Placement Groups and removed from the old ones.

Placement Groups Tradeoffs

Data durability and even distribution among all OSDs call for more placement groups but their number should be reduced to the minimum to save CPU and memory.

Data durability

After an OSD fails, the risk of data loss increases until the data it contained is fully recovered. Let's imagine a scenario that causes permanent data loss in a single placement group:

- The OSD fails and all copies of the object it contains are lost. For all objects within the placement group the number of replica suddently drops from three to two.
- Ceph starts recovery for this placement group by chosing a new OSD to re-create the third copy of all objects.
- Another OSD, within the same placement group, fails before the new OSD is fully populated with the third copy. Some objects will then only have one surviving copies.
- Ceph picks yet another OSD and keeps copying objects to restore the desired number of copies.
- A third OSD, within the same placement group, fails before recovery is complete. If this OSD contained the only remaining copy of an object, it is permanently lost.

In a cluster containing 10 OSDs with 512 placement groups in a three replica pool, CRUSH will give each placement groups three OSDs. In the end, each OSDs will end up hosting (512 * 3) / 10 = ~150 Placement Groups. When the first OSD fails, the above scenario will therefore start recovery for all 150 placement groups at the same time.

The 150 placement groups being recovered are likely to be homogeneously spread over the 9 remaining OSDs. Each remaining OSD is therefore likely to send copies of objects to all others and also receive some new objects to be stored because they became part of a new placement group.

The amount of time it takes for this recovery to complete entirely depends on the architecture of the Ceph cluster. Let say each OSD is hosted by a 1TB SSD on a single machine and all of them are connected to a 10Gb/s switch and the recovery for a single OSD completes within M minutes. If there are two OSDs per machine using spinners with no SSD journal and a 1Gb/s switch, it will at least be an order of magnitude slower.

In a cluster of this size, the number of placement groups has almost no influence on data durability. It could be 128 or 8192 and the recovery would not be slower or faster.

However, growing the same Ceph cluster to 20 OSDs instead of 10 OSDs is likely to speed up recovery and therefore improve data durability significantly. Each OSD now participates in only ~75 placement groups instead of ~150 when there were only 10 OSDs and it will still require all 19 remaining OSDs to perform the same amount of object copies in order to recover. But where 10 OSDs had to copy approximately 100GB each, they now have to copy 50GB each instead. If the network was the bottleneck, recovery will happen twice as fast. In other words, recovery goes faster when the number of OSDs increases.

If this cluster grows to 40 OSDs, each of them will only host ~35 placement groups. If an OSD dies, recovery will keep going faster unless it is blocked by another bottleneck. However, if this cluster grows to 200 OSDs, each of them will only host ~7 placement groups. If an OSD dies, recovery will happen between at most of ~21 (7 * 3) OSDs in these placement groups: recovery will take longer than when there were 40 OSDs, meaning the number of placement groups should be increased.

No matter how short the recovery time is, there is a chance for a second OSD to fail while it is in progress. In the 10 OSDs cluster described above, if any of them fail, then ~17 placement groups (i.e. ~150 / 9 placement groups being recovered) will only have one surviving copy. And if any of the 8 remaining OSD fail, the last objects of two placement groups are likely to be lost (i.e. ~17 / 8 placement groups with only one remaining copy being recovered).

When the size of the cluster grows to 20 OSDs, the number of Placement Groups damaged by the loss of three OSDs drops. The second OSD lost will degrade ~4 (i.e. ~75 / 19 placement groups being recovered) instead of ~17 and the third OSD lost will only lose data if it is one of the four OSDs containing the surviving copy. In other words, if the probability of losing one OSD is 0.0001% during the recovery time frame, it goes from 17 * 10 * 0.0001% in the cluster with 10 OSDs to 4 * 20 * 0.0001% in the cluster with 20 OSDs.

In a nutshell, more OSDs mean faster recovery and a lower risk of cascading failures leading to the permanent loss of a Placement Group. Having 512 or 4096 Placement Groups is roughly equivalent in a cluster with less than 50 OSDs as far as data durability is concerned.

Note: It may take a long time for a new OSD added to the cluster to be populated with placement groups that were assigned to it. However there is no degradation of any object and it has no impact on the durability of the data contained in the Cluster.

Object distribution within a pool

Ideally objects are evenly distributed in each placement group. Since CRUSH computes the placement group for each object, but does not actually know how much data is stored in each OSD within this placement group, the ratio between the number of placement groups and the number of OSDs may influence the distribution of the data significantly.

For instance, if there was single a placement group for ten OSDs in a three replica pool, only three OSD would be used because CRUSH would have no other choice. When more placement groups are available, objects are more likely to be evenly spread among them. CRUSH also makes every effort to evenly spread OSDs among all existing Placement Groups.

As long as there are one or two orders of magnitude more Placement Groups than OSDs, the distribution should be even. For instance, 300 placement groups for 3 OSDs, 1000 placement groups for 10 OSDs etc.

Uneven data distribution can be caused by factors other than the ratio between OSDs and placement groups. Since CRUSH does not take into account the size of the objects, a few very large objects may create an imbalance. Let say one million 4K objects totaling 4GB are evenly spread among 1000 placement groups on 10 OSDs. They will use 4GB / 10 = 400MB on each OSD. If one 400MB object is added to the pool, the three OSDs supporting the placement group in which the object has been placed will be filled with 400MB + 400MB = 800MB while the seven others will remain occupied with only 400MB.

Memory, CPU and network usage

For each placement group, OSDs and MONs need memory, network and CPU at all times and even more during recovery. Sharing this overhead by clustering objects within a placement group is one of the main reasons they exist.

Minimizing the number of placement groups saves significant amounts of resources.

Choosing the number of Placement Groups

If you have more than 50 OSDs, we recommend approximately 50-100 placement groups per OSD to balance out resource usage, data durability and distribution. If you have less than 50 OSDs, chosing among the *preselection* above is best. For a single pool of objects, you can use the following formula to get a baseline:

```
              (OSDs * 100)
Total PGs =  ------------
               pool size
```

Where **pool size** is either the number of replicas for replicated pools or the K+M sum for erasure coded pools (as returned by **ceph osd erasure-code-profile get**).

You should then check if the result makes sense with the way you designed your Ceph cluster to maximize *data durability*, *object distribution* and minimize *resource usage*.

The result should be **rounded up to the nearest power of two.** Rounding up is optional, but recommended for CRUSH to evenly balance the number of objects among placement groups.

As an example, for a cluster with 200 OSDs and a pool size of 3 replicas, you would estimate your number of PGs as follows:

```
(200 * 100)
----------- = 6667. Nearest power of 2: 8192
     3
```

When using multiple data pools for storing objects, you need to ensure that you balance the number of placement groups per pool with the number of placement groups per OSD so that you arrive at a reasonable total number of placement groups that provides reasonably low variance per OSD without taxing system resources or making the peering process too slow.

For instance a cluster of 10 pools each with 512 placement groups on ten OSDs is a total of 5,120 placement groups spread over ten OSDs, that is 512 placement groups per OSD. That does not use too many resources. However, if 1,000 pools were created with 512 placement groups each, the OSDs will handle ~50,000 placement groups each and it would require significantly more resources and time for peering.

Set the Number of Placement Groups

To set the number of placement groups in a pool, you must specify the number of placement groups at the time you create the pool. See Create a Pool for details. Once you've set placement groups for a pool, you may increase the number of placement groups (but you cannot decrease the number of placement groups). To increase the number of placement groups, execute the following:

```
ceph osd pool set {pool-name} pg_num {pg_num}
```

Once you increase the number of placement groups, you must also increase the number of placement groups for placement (pgp_num) before your cluster will rebalance. The pgp_num should be equal to the pg_num. To increase the number of placement groups for placement, execute the following:

```
ceph osd pool set {pool-name} pgp_num {pgp_num}
```

Get the Number of Placement Groups

To get the number of placement groups in a pool, execute the following:

```
ceph osd pool get {pool-name} pg_num
```

Get a Cluster's PG Statistics

To get the statistics for the placement groups in your cluster, execute the following:

```
ceph pg dump [--format {format}]
```

Valid formats are plain (default) and json.

Get Statistics for Stuck PGs

To get the statistics for all placement groups stuck in a specified state, execute the following:

```
ceph pg dump_stuck inactive|unclean|stale|undersized|degraded [--format <format>] [-t|--threshold <se
```

Inactive Placement groups cannot process reads or writes because they are waiting for an OSD with the most up-to-date data to come up and in.

Unclean Placement groups contain objects that are not replicated the desired number of times. They should be recovering.

Stale Placement groups are in an unknown state - the OSDs that host them have not reported to the monitor cluster in a while (configured by mon_osd_report_timeout).

Valid formats are plain (default) and json. The threshold defines the minimum number of seconds the placement group is stuck before including it in the returned statistics (default 300 seconds).

Get a PG Map

To get the placement group map for a particular placement group, execute the following:

```
ceph pg map {pg-id}
```

For example:

```
ceph pg map 1.6c
```

Ceph will return the placement group map, the placement group, and the OSD status:

```
osdmap e13 pg 1.6c (1.6c) -> up [1,0] acting [1,0]
```

Get a PGs Statistics

To retrieve statistics for a particular placement group, execute the following:

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

Scrub a Placement Group

To scrub a placement group, execute the following:

```
ceph pg scrub {pg-id}
```

Ceph checks the primary and any replica nodes, generates a catalog of all objects in the placement group and compares them to ensure that no objects are missing or mismatched, and their contents are consistent. Assuming the replicas all match, a final semantic sweep ensures that all of the snapshot-related object metadata is consistent. Errors are reported via logs.

Revert Lost

If the cluster has lost one or more objects, and you have decided to abandon the search for the lost data, you must mark the unfound objects as `lost`.

If all possible locations have been queried and objects are still lost, you may have to give up on the lost objects. This is possible given unusual combinations of failures that allow the cluster to learn about writes that were performed before the writes themselves are recovered.

Currently the only supported option is "revert", which will either roll back to a previous version of the object or (if it was a new object) forget about it entirely. To mark the "unfound" objects as "lost", execute the following:

```
ceph pg {pg-id} mark_unfound_lost revert|delete
```

Important: Use this feature with caution, because it may confuse applications that expect the object(s) to exist.

Placement Group States

When checking a cluster's status (e.g., running `ceph -w` or `ceph -s`), Ceph will report on the status of the placement groups. A placement group has one or more states. The optimum state for placement groups in the placement group map is `active + clean`.

Creating Ceph is still creating the placement group.

Active Ceph will process requests to the placement group.

Clean Ceph replicated all objects in the placement group the correct number of times.

Down A replica with necessary data is down, so the placement group is offline.

Replay The placement group is waiting for clients to replay operations after an OSD crashed.

Splitting Ceph is splitting the placement group into multiple placement groups. (functional?)

Scrubbing Ceph is checking the placement group for inconsistencies.

Degraded Ceph has not replicated some objects in the placement group the correct number of times yet.

Inconsistent Ceph detects inconsistencies in the one or more replicas of an object in the placement group (e.g. objects are the wrong size, objects are missing from one replica *after* recovery finished, etc.).

Peering The placement group is undergoing the peering process

Repair Ceph is checking the placement group and repairing any inconsistencies it finds (if possible).

Recovering Ceph is migrating/synchronizing objects and their replicas.

Backfill Ceph is scanning and synchronizing the entire contents of a placement group instead of inferring what contents need to be synchronized from the logs of recent operations. *Backfill* is a special case of recovery.

Wait-backfill The placement group is waiting in line to start backfill.

Backfill-toofull A backfill operation is waiting because the destination OSD is over its full ratio.

Incomplete Ceph detects that a placement group is missing information about writes that may have occurred, or does not have any healthy copies. If you see this state, try to start any failed OSDs that may contain the needed information or temporarily adjust min_size to allow recovery.

Stale The placement group is in an unknown state - the monitors have not received an update for it since the placement group mapping changed.

Remapped The placement group is temporarily mapped to a different set of OSDs from what CRUSH specified.

Undersized The placement group fewer copies than the configured pool replication level.

Peered The placement group has peered, but cannot serve client IO due to not having enough copies to reach the pool's configured min_size parameter. Recovery may occur in this state, so the pg may heal up to min_size eventually.

Placement Group Concepts

When you execute commands like `ceph -w`, `ceph osd dump`, and other commands related to placement groups, Ceph may return values using some of the following terms:

Peering The process of bringing all of the OSDs that store a Placement Group (PG) into agreement about the state of all of the objects (and their metadata) in that PG. Note that agreeing on the state does not mean that they all have the latest contents.

Acting Set The ordered list of OSDs who are (or were as of some epoch) responsible for a particular placement group.

Up Set The ordered list of OSDs responsible for a particular placement group for a particular epoch according to CRUSH. Normally this is the same as the *Acting Set*, except when the *Acting Set* has been explicitly overridden via `pg_temp` in the OSD Map.

Current Interval* or *Past Interval A sequence of OSD map epochs during which the *Acting Set* and *Up Set* for particular placement group do not change.

Primary The member (and by convention first) of the *Acting Set*, that is responsible for coordination peering, and is the only OSD that will accept client-initiated writes to objects in a placement group.

Replica A non-primary OSD in the *Acting Set* for a placement group (and who has been recognized as such and *activated* by the primary).

Stray An OSD that is not a member of the current *Acting Set*, but has not yet been told that it can delete its copies of a particular placement group.

Recovery Ensuring that copies of all of the objects in a placement group are on all of the OSDs in the *Acting Set*. Once *Peering* has been performed, the *Primary* can start accepting write operations, and *Recovery* can proceed in the background.

PG Info Basic metadata about the placement group's creation epoch, the version for the most recent write to the placement group, *last epoch started*, *last epoch clean*, and the beginning of the *current interval*. Any inter-OSD communication about placement groups includes the *PG Info*, such that any OSD that knows a placement group exists (or once existed) also has a lower bound on *last epoch clean* or *last epoch started*.

PG Log A list of recent updates made to objects in a placement group. Note that these logs can be truncated after all OSDs in the *Acting Set* have acknowledged up to a certain point.

Missing Set Each OSD notes update log entries and if they imply updates to the contents of an object, adds that object to a list of needed updates. This list is called the *Missing Set* for that `<OSD,PG>`.

Authoritative History A complete, and fully ordered set of operations that, if performed, would bring an OSD's copy of a placement group up to date.

Epoch A (monotonically increasing) OSD map version number

Last Epoch Start The last epoch at which all nodes in the *Acting Set* for a particular placement group agreed on an *Authoritative History*. At this point, *Peering* is deemed to have been successful.

up_thru Before a *Primary* can successfully complete the *Peering* process, it must inform a monitor that is alive through the current OSD map *Epoch* by having the monitor set its *up_thru* in the osd map. This helps *Peering* ignore previous *Acting Sets* for which *Peering* never completed after certain sequences of failures, such as the second interval below:

- *acting set* = [A,B]
- *acting set* = [A]
- *acting set* = [] very shortly after (e.g., simultaneous failure, but staggered detection)
- *acting set* = [B] (B restarts, A does not)

Last Epoch Clean The last *Epoch* at which all nodes in the *Acting set* for a particular placement group were completely up to date (both placement group logs and object contents). At this point, *recovery* is deemed to have been completed.

4.3.10 CRUSH Maps

The CRUSH (Controlled Replication Under Scalable Hashing) algorithm determines how to store and retrieve data by computing data storage locations. CRUSH empowers Ceph clients to communicate with OSDs directly rather than through a centralized server or broker. With an algorithmically determined method of storing and retrieving data, Ceph avoids a single point of failure, a performance bottleneck, and a physical limit to its scalability.

CRUSH requires a map of your cluster, and uses the CRUSH map to pseudo-randomly store and retrieve data in OSDs with a uniform distribution of data across the cluster. For a detailed discussion of CRUSH, see CRUSH - Controlled, Scalable, Decentralized Placement of Replicated Data

CRUSH maps contain a list of OSDs (Object Storage Devices), a list of 'buckets' for aggregating the devices into physical locations, and a list of rules that tell CRUSH how it should replicate data in a Ceph cluster's pools. By reecting the underlying physical organization of the installation, CRUSH can model—and thereby address—potential sources of correlated device failures. Typical sources include physical proximity, a shared power source, and a shared network. By encoding this information into the cluster map, CRUSH placement policies can separate object replicas across different failure domains while still maintaining the desired distribution. For example, to address the possibility of concurrent failures, it may be desirable to ensure that data replicas are on devices using different shelves, racks, power supplies, controllers, and/or physical locations.

When you create a configuration file and deploy Ceph with `ceph-deploy`, Ceph generates a default CRUSH map for your configuration. The default CRUSH map is fine for your Ceph sandbox environment. However, when you deploy a large-scale data cluster, you should give significant consideration to developing a custom CRUSH map, because it will help you manage your Ceph cluster, improve performance and ensure data safety.

For example, if an OSD goes down, a CRUSH map can help you can locate the physical data center, room, row and rack of the host with the failed OSD in the event you need to use onsite support or replace hardware.

Similarly, CRUSH may help you identify faults more quickly. For example, if all OSDs in a particular rack go down simultaneously, the fault may lie with a network switch or power to the rack or the network switch rather than the OSDs themselves.

A custom CRUSH map can also help you identify the physical locations where Ceph stores redundant copies of data when the placement group(s) associated with a failed host are in a degraded state.

Note: Lines of code in example boxes may extend past the edge of the box. Please scroll when reading or copying longer examples.

CRUSH Location

The location of an OSD in terms of the CRUSH map's hierarchy is referred to as a 'crush location'. This location specifier takes the form of a list of key and value pairs describing a position. For example, if an OSD is in a particular row, rack, chassis and host, and is part of the 'default' CRUSH tree, its crush location could be described as:

```
root=default row=a rack=a2 chassis=a2a host=a2a1
```

Note:

1. Note that the order of the keys does not matter.
2. The key name (left of =) must be a valid CRUSH `type`. By default these include root, datacenter, room, row, pod, pdu, rack, chassis and host, but those types can be customized to be anything appropriate by modifying the CRUSH map.
3. Not all keys need to be specified. For example, by default, Ceph automatically sets a `ceph-osd` daemon's location to be `root=default host=HOSTNAME` (based on the output from `hostname -s`).

ceph-crush-location hook

By default, the `ceph-crush-location` utility will generate a CRUSH location string for a given daemon. The location is based on, in order of preference:

1. A `TYPE crush location` option in ceph.conf. For example, this is `osd crush location` for OSD daemons.
2. A `crush location` option in ceph.conf.
3. A default of `root=default host=HOSTNAME` where the hostname is generated with the `hostname -s` command.

In a typical deployment scenario, provisioning software (or the system administrator) can simply set the 'crush location' field in a host's ceph.conf to describe that machine's location within the datacenter or cluster. This will be provide location awareness to both Ceph daemons and clients alike.

It is possible to manage the CRUSH map entirely manually by toggling the hook off in the configuration:

```
osd crush update on start = false
```

Custom location hooks

A customize location hook can be used in place of the generic hook for OSD daemon placement in the hierarchy. (On startup, each OSD ensure its position is correct.):

```
osd crush location hook = /path/to/script
```

This hook is passed several arguments (below) and should output a single line to stdout with the CRUSH location description.:

```
$ ceph-crush-location --cluster CLUSTER --id ID --type TYPE
```

where the cluster name is typically 'ceph', the id is the daemon identifier (the OSD number), and the daemon type is typically `osd`.

Editing a CRUSH Map

To edit an existing CRUSH map:

1. *Get the CRUSH map*.
2. *Decompile* the CRUSH map.
3. Edit at least one of *Devices*, *Buckets* and *Rules*.
4. *Recompile* the CRUSH map.
5. *Set the CRUSH map*.

To activate CRUSH Map rules for a specific pool, identify the common ruleset number for those rules and specify that ruleset number for the pool. See Set Pool Values for details.

Get a CRUSH Map

To get the CRUSH map for your cluster, execute the following:

```
ceph osd getcrushmap -o {compiled-crushmap-filename}
```

Ceph will output (-o) a compiled CRUSH map to the filename you specified. Since the CRUSH map is in a compiled form, you must decompile it first before you can edit it.

Decompile a CRUSH Map

To decompile a CRUSH map, execute the following:

```
crushtool -d {compiled-crushmap-filename} -o {decompiled-crushmap-filename}
```

Ceph will decompile (-d) the compiled CRUSH map and output (-o) it to the filename you specified.

Compile a CRUSH Map

To compile a CRUSH map, execute the following:

```
crushtool -c {decompiled-crush-map-filename} -o {compiled-crush-map-filename}
```

Ceph will store a compiled CRUSH map to the filename you specified.

Set a CRUSH Map

To set the CRUSH map for your cluster, execute the following:

```
ceph osd setcrushmap -i  {compiled-crushmap-filename}
```

Ceph will input the compiled CRUSH map of the filename you specified as the CRUSH map for the cluster.

CRUSH Map Parameters

There are four main sections to a CRUSH Map.

1. **Devices:** Devices consist of any object storage device–i.e., the storage drive corresponding to a `ceph-osd` daemon. You should have a device for each OSD daemon in your Ceph configuration file.
2. **Bucket Types**: Bucket `types` define the types of buckets used in your CRUSH hierarchy. Buckets consist of a hierarchical aggregation of storage locations (e.g., rows, racks, chassis, hosts, etc.) and their assigned weights.
3. **Bucket Instances:** Once you define bucket types, you must declare bucket instances for your hosts, and any other failure domain partitioning you choose.
4. **Rules:** Rules consist of the manner of selecting buckets.

If you launched Ceph using one of our Quick Start guides, you'll notice that you didn't need to create a CRUSH map. Ceph's deployment tools generate a default CRUSH map that lists devices from the OSDs you defined in your Ceph configuration file, and it declares a bucket for each host you specified in the `[osd]` sections of your Ceph configuration file. You should create your own CRUSH maps with buckets that reflect your cluster's failure domains to better ensure data safety and availability.

Note: The generated CRUSH map doesn't take your larger grained failure domains into account. So you should modify your CRUSH map to account for larger grained failure domains such as chassis, racks, rows, data centers, etc.

CRUSH Map Devices

To map placement groups to OSDs, a CRUSH map requires a list of OSD devices (i.e., the names of the OSD daemons from the Ceph configuration file). The list of devices appears first in the CRUSH map. To declare a device in the CRUSH map, create a new line under your list of devices, enter `device` followed by a unique numeric ID, followed by the corresponding `ceph-osd` daemon instance.

```
#devices
device {num} {osd.name}
```

For example:

```
#devices
device 0 osd.0
device 1 osd.1
device 2 osd.2
device 3 osd.3
```

As a general rule, an OSD daemon maps to a single storage drive or to a RAID.

CRUSH Map Bucket Types

The second list in the CRUSH map defines 'bucket' types. Buckets facilitate a hierarchy of nodes and leaves. Node (or non-leaf) buckets typically represent physical locations in a hierarchy. Nodes aggregate other nodes or leaves. Leaf buckets represent ceph-osd daemons and their corresponding storage media.

Tip: The term "bucket" used in the context of CRUSH means a node in the hierarchy, i.e. a location or a piece of physical hardware. It is a different concept from the term "bucket" when used in the context of RADOS Gateway APIs.

To add a bucket type to the CRUSH map, create a new line under your list of bucket types. Enter type followed by a unique numeric ID and a bucket name. By convention, there is one leaf bucket and it is type 0; however, you may give it any name you like (e.g., osd, disk, drive, storage, etc.):

```
#types
type {num} {bucket-name}
```

For example:

```
# types
type 0 osd
type 1 host
type 2 chassis
type 3 rack
type 4 row
type 5 pdu
type 6 pod
type 7 room
type 8 datacenter
type 9 region
type 10 root
```

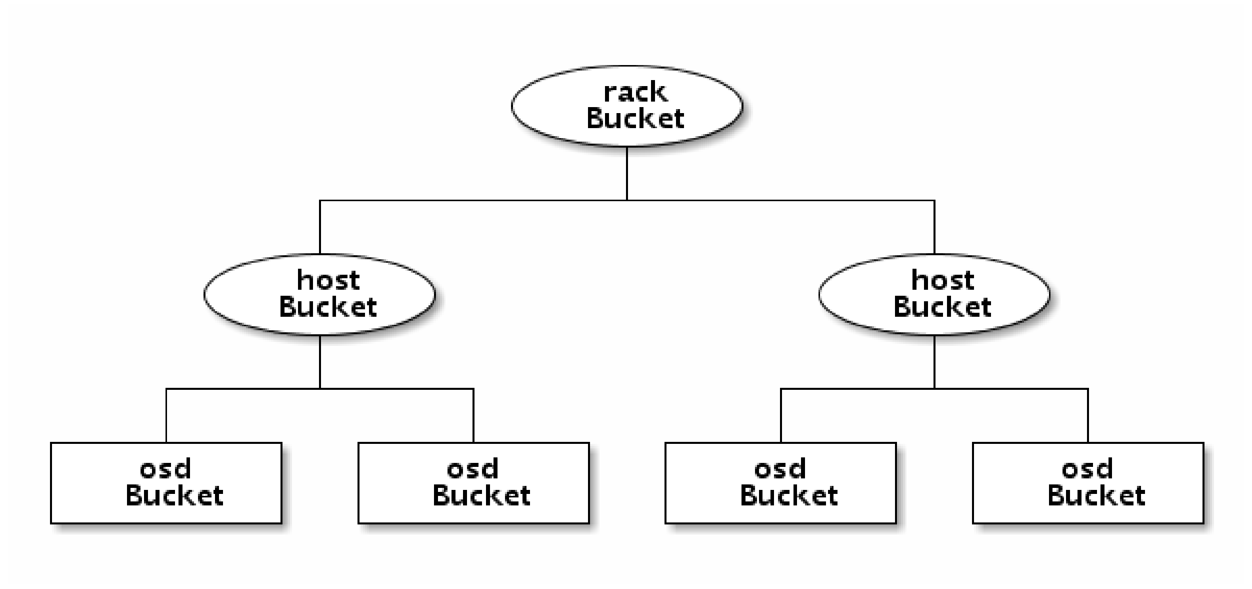
CRUSH Map Bucket Hierarchy

The CRUSH algorithm distributes data objects among storage devices according to a per-device weight value, approximating a uniform probability distribution. CRUSH distributes objects and their replicas according to the hierarchical cluster map you define. Your CRUSH map represents the available storage devices and the logical elements that contain them.

To map placement groups to OSDs across failure domains, a CRUSH map defines a hierarchical list of bucket types (i.e., under #types in the generated CRUSH map). The purpose of creating a bucket hierarchy is to segregate the leaf nodes by their failure domains, such as hosts, chassis, racks, power distribution units, pods, rows, rooms, and data centers. With the exception of the leaf nodes representing OSDs, the rest of the hierarchy is arbitrary, and you may define it according to your own needs.

We recommend adapting your CRUSH map to your firms's hardware naming conventions and using instances names that reflect the physical hardware. Your naming practice can make it easier to administer the cluster and troubleshoot problems when an OSD and/or other hardware malfunctions and the administrator need access to physical hardware.

In the following example, the bucket hierarchy has a leaf bucket named osd, and two node buckets named host and rack respectively.



Note: The higher numbered `rack` bucket type aggregates the lower numbered `host` bucket type.

Since leaf nodes reflect storage devices declared under the `#devices` list at the beginning of the CRUSH map, you do not need to declare them as bucket instances. The second lowest bucket type in your hierarchy usually aggregates the devices (i.e., it's usually the computer containing the storage media, and uses whatever term you prefer to describe it, such as "node", "computer", "server," "host", "machine", etc.). In high density environments, it is increasingly common to see multiple hosts/nodes per chassis. You should account for chassis failure too–e.g., the need to pull a chassis if a node fails may result in bringing down numerous hosts/nodes and their OSDs.

When declaring a bucket instance, you must specify its type, give it a unique name (string), assign it a unique ID expressed as a negative integer (optional), specify a weight relative to the total capacity/capability of its item(s), specify the bucket algorithm (usually `straw`), and the hash (usually `0`, reflecting hash algorithm `rjenkins1`). A bucket may have one or more items. The items may consist of node buckets or leaves. Items may have a weight that reflects the relative weight of the item.

You may declare a node bucket with the following syntax:

```
[bucket-type] [bucket-name] {
        id [a unique negative numeric ID]
        weight [the relative capacity/capability of the item(s)]
        alg [the bucket type: uniform | list | tree | straw ]
        hash [the hash type: 0 by default]
        item [item-name] weight [weight]
}
```

For example, using the diagram above, we would define two host buckets and one rack bucket. The OSDs are declared as items within the host buckets:

```
host node1 {
        id -1
        alg straw
        hash 0
        item osd.0 weight 1.00
        item osd.1 weight 1.00
}

host node2 {
        id -2
        alg straw
        hash 0
        item osd.2 weight 1.00
        item osd.3 weight 1.00
}

rack rack1 {
        id -3
        alg straw
        hash 0
        item node1 weight 2.00
        item node2 weight 2.00
}
```

Note: In the foregoing example, note that the rack bucket does not contain any OSDs. Rather it contains lower level host buckets, and includes the sum total of their weight in the item entry.

Bucket Types

Ceph supports four bucket types, each representing a tradeoff between performance and reorganization efficiency. If you are unsure of which bucket type to use, we recommend using a `straw` bucket. For a detailed discussion of bucket types, refer to CRUSH - Controlled, Scalable, Decentralized Placement of Replicated Data, and more specifically to **Section 3.4**. The bucket types are:

1. **Uniform:** Uniform buckets aggregate devices with **exactly** the same weight. For example, when firms commission or decommission hardware, they typically do so with many machines that have exactly the same physical configuration (e.g., bulk purchases). When storage devices have exactly the same weight, you may use the `uniform` bucket type, which allows CRUSH to map replicas into uniform buckets in constant time. With non-uniform weights, you should use another bucket algorithm.
2. **List**: List buckets aggregate their content as linked lists. Based on the RUSH (Replication Under Scalable Hashing) $_P$ algorithm, a list is a natural and intuitive choice for an **expanding cluster**: either an object is relocated to the newest device with some appropriate probability, or it remains on the older devices as before. The result is optimal data migration when items are added to the bucket. Items removed from the middle or tail of the list, however, can result in a signicant amount of unnecessary movement, making list buckets most suitable for circumstances in which they **never (or very rarely) shrink**.
3. **Tree**: Tree buckets use a binary search tree. They are more efficient than list buckets when a bucket contains a larger set of items. Based on the RUSH $_R$ algorithm, tree buckets reduce the placement time to O(log $_n$), making them suitable for managing much larger sets of devices or nested buckets.
4. **Straw:** List and Tree buckets use a divide and conquer strategy in a way that either gives certain items precedence (e.g., those at the beginning of a list) or obviates the need to consider entire subtrees of items at all. That improves the performance of the replica placement process, but can also introduce suboptimal reorganization behavior when the contents of a bucket change due an addition, removal, or re-weighting of an item. The straw bucket type allows all items to fairly "compete" against each other for replica placement through a process analogous to a draw of straws.

Hash

Each bucket uses a hash algorithm. Currently, Ceph supports `rjenkins1`. Enter `0` as your hash setting to select `rjenkins1`.

Weighting Bucket Items

Ceph expresses bucket weights as doubles, which allows for fine weighting. A weight is the relative difference between device capacities. We recommend using `1.00` as the relative weight for a 1TB storage device. In such a scenario, a weight of `0.5` would represent approximately 500GB, and a weight of `3.00` would represent approximately 3TB. Higher level buckets have a weight that is the sum total of the leaf items aggregated by the bucket.
A bucket item weight is one dimensional, but you may also calculate your item weights to reflect the performance of the storage drive. For example, if you have many 1TB drives where some have relatively low data transfer rate and the others have a relatively high data transfer rate, you may weight them differently, even though they have the same capacity (e.g., a weight of 0.80 for the first set of drives with lower total throughput, and 1.20 for the second set of drives with higher total throughput).

CRUSH Map Rules

CRUSH maps support the notion of 'CRUSH rules', which are the rules that determine data placement for a pool. For large clusters, you will likely create many pools where each pool may have its own CRUSH ruleset and rules. The default CRUSH map has a rule for each pool, and one ruleset assigned to each of the default pools, which include:

- `data`
- `metadata`
- `rbd`

Note: In most cases, you will not need to modify the default rules. When you create a new pool, its default ruleset is `0`.

CRUSH rules denes placement and replication strategies or distribution policies that allow you to specify exactly how CRUSH places object replicas. For example, you might create a rule selecting a pair of targets for 2-way mirroring, another rule for selecting three targets in two different data centers for 3-way mirroring, and yet another rule for erasure coding over six storage devices. For a detailed discussion of CRUSH rules, refer to CRUSH - Controlled, Scalable, Decentralized Placement of Replicated Data, and more specifically to **Section 3.2**.

A rule takes the following form:

```
rule <rulename> {

        ruleset <ruleset>
        type [ replicated | erasure ]
        min_size <min-size>
        max_size <max-size>
        step take <bucket-type>
        step [choose|chooseleaf] [firstn|indep] <N> <bucket-type>
        step emit
}
```

`ruleset`

Description A means of classifying a rule as belonging to a set of rules. Activated by setting the ruleset in a pool.

Purpose A component of the rule mask.

Type Integer

Required Yes

Default 0

`type`

Description Describes a rule for either a storage drive (replicated) or a RAID.

Purpose A component of the rule mask.

Type String

Required Yes

Default `replicated`

Valid Values Currently only `replicated` and `erasure`

`min_size`

Description If a pool makes fewer replicas than this number, CRUSH will **NOT** select this rule.

Type Integer

Purpose A component of the rule mask.

Required Yes

Default `1`

`max_size`

Description If a pool makes more replicas than this number, CRUSH will **NOT** select this rule.

Type Integer

Purpose A component of the rule mask.

Required Yes

Default 10

`step take <bucket-name>`

Description Takes a bucket name, and begins iterating down the tree.

Purpose A component of the rule.

Required Yes

Example `step take data`

`step choose firstn {num} type {bucket-type}`

Description Selects the number of buckets of the given type. The number is usually the number of replicas in the pool (i.e., pool size).

- If `{num} == 0`, choose `pool-num-replicas` buckets (all available).
- If `{num} > 0 && < pool-num-replicas`, choose that many buckets.
- If `{num} < 0`, it means `pool-num-replicas - {num}`.

Purpose A component of the rule.

Prerequisite Follows `step take` or `step choose`.

Example `step choose firstn 1 type row`

`step chooseleaf firstn {num} type {bucket-type}`

Description Selects a set of buckets of `{bucket-type}` and chooses a leaf node from the subtree of each bucket in the set of buckets. The number of buckets in the set is usually the number of replicas in the pool (i.e., pool size).

- If `{num} == 0`, choose `pool-num-replicas` buckets (all available).
- If `{num} > 0 && < pool-num-replicas`, choose that many buckets.
- If `{num} < 0`, it means `pool-num-replicas - {num}`.

Purpose A component of the rule. Usage removes the need to select a device using two steps.

Prerequisite Follows `step take` or `step choose`.

Example `step chooseleaf firstn 0 type row`

`step emit`

Description Outputs the current value and empties the stack. Typically used at the end of a rule, but may also be used to pick from different trees in the same rule.

Purpose A component of the rule.

Prerequisite Follows `step choose`.

Example `step emit`

Important: To activate one or more rules with a common ruleset number to a pool, set the ruleset number of the pool.

Primary Affinity

When a Ceph Client reads or writes data, it always contacts the primary OSD in the acting set. For set `[2, 3, 4]`, `osd.2` is the primary. Sometimes an OSD isn't well suited to act as a primary compared to other OSDs (e.g., it has a slow disk or a slow controller). To prevent performance bottlenecks (especially on read operations) while maximizing utilization of your hardware, you can set a Ceph OSD's primary affinity so that CRUSH is less likely to use the OSD as a primary in an acting set.

```
ceph osd primary-affinity <osd-id> <weight>
```

Primary affinity is `1` by default (*i.e.,* an OSD may act as a primary). You may set the OSD primary range from `0-1`, where `0` means that the OSD may **NOT** be used as a primary and `1` means that an OSD may be used as a primary. When the weight is `< 1`, it is less likely that CRUSH will select the Ceph OSD Daemon to act as a primary.

Placing Different Pools on Different OSDS:

Suppose you want to have most pools default to OSDs backed by large hard drives, but have some pools mapped to OSDs backed by fast solid-state drives (SSDs). It's possible to have multiple independent CRUSH heirarchies within the same CRUSH map. Define two hierarchies with two different root nodes–one for hard disks (e.g., “root platter”) and one for SSDs (e.g., “root ssd”) as shown below:

```
device 0 osd.0
device 1 osd.1
device 2 osd.2
device 3 osd.3
device 4 osd.4
device 5 osd.5
device 6 osd.6
device 7 osd.7
```

```
host ceph-osd-ssd-server-1 {
        id -1
        alg straw
        hash 0
        item osd.0 weight 1.00
        item osd.1 weight 1.00
}

host ceph-osd-ssd-server-2 {
        id -2
        alg straw
        hash 0
        item osd.2 weight 1.00
        item osd.3 weight 1.00
}

host ceph-osd-platter-server-1 {
        id -3
        alg straw
        hash 0
        item osd.4 weight 1.00
        item osd.5 weight 1.00
}

host ceph-osd-platter-server-2 {
        id -4
        alg straw
        hash 0
        item osd.6 weight 1.00
        item osd.7 weight 1.00
}

root platter {
        id -5
        alg straw
        hash 0
        item ceph-osd-platter-server-1 weight 2.00
        item ceph-osd-platter-server-2 weight 2.00
}

root ssd {
        id -6
        alg straw
        hash 0
        item ceph-osd-ssd-server-1 weight 2.00
        item ceph-osd-ssd-server-2 weight 2.00
}

rule data {
        ruleset 0
        type replicated
        min_size 2
        max_size 2
        step take platter
        step chooseleaf firstn 0 type host
        step emit
}
```

```
      rule metadata {
              ruleset 1
              type replicated
              min_size 0
              max_size 10
              step take platter
              step chooseleaf firstn 0 type host
              step emit
      }

      rule rbd {
              ruleset 2
              type replicated
              min_size 0
              max_size 10
              step take platter
              step chooseleaf firstn 0 type host
              step emit
      }

      rule platter {
              ruleset 3
              type replicated
              min_size 0
              max_size 10
              step take platter
              step chooseleaf firstn 0 type host
              step emit
      }

      rule ssd {
              ruleset 4
              type replicated
              min_size 0
              max_size 4
              step take ssd
              step chooseleaf firstn 0 type host
              step emit
      }

      rule ssd-primary {
              ruleset 5
              type replicated
              min_size 5
              max_size 10
              step take ssd
              step chooseleaf firstn 1 type host
              step emit
              step take platter
              step chooseleaf firstn -1 type host
              step emit
      }
```

You can then set a pool to use the SSD rule by:

```
ceph osd pool set <poolname> crush_ruleset 4
```

Similarly, using the `ssd-primary` rule will cause each placement group in the pool to be placed with an SSD as the primary and platters as the replicas.

Add/Move an OSD

To add or move an OSD in the CRUSH map of a running cluster, execute the `ceph osd crush set`. For Argonaut (v 0.48), execute the following:

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

For Bobtail (v 0.56), execute the following:

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

Where:

`id`

Description The numeric ID of the OSD.

Type Integer

Required Yes

Example `0`

`name`

Description The full name of the OSD.

Type String

Required Yes

Example `osd.0`

`weight`

Description The CRUSH weight for the OSD.

Type Double

Required Yes

Example `2.0`

`root`

Description The root of the tree in which the OSD resides.

Type Key/value pair.

Required Yes

Example `root=default`

`bucket-type`

Description You may specify the OSD's location in the CRUSH hierarchy.

Type Key/value pairs.

Required No

Example `datacenter=dc1 room=room1 row=foo rack=bar host=foo-bar-1`

The following example adds `osd.0` to the hierarchy, or moves the OSD from a previous location.

```
ceph osd crush set osd.0 1.0 root=default datacenter=dc1 room=room1 row=foo rack=bar host=foo-bar-1
```

Adjust an OSD's CRUSH Weight

To adjust an OSD's crush weight in the CRUSH map of a running cluster, execute the following:

```
ceph osd crush reweight {name} {weight}
```

Where:

`name`

Description The full name of the OSD.

Type String

Required Yes

Example `osd.0`

`weight`

Description The CRUSH weight for the OSD.

Type Double

Required Yes

Example `2.0`

Remove an OSD

To remove an OSD from the CRUSH map of a running cluster, execute the following:

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

Where:

`name`

Description The full name of the OSD.

Type String

Required Yes

Example `osd.0`

Add a Bucket

To add a bucket in the CRUSH map of a running cluster, execute the `ceph osd crush add-bucket` command:

```
ceph osd crush add-bucket {bucket-name} {bucket-type}
```

Where:

`bucket-name`

Description The full name of the bucket.

Type String

Required Yes

Example `rack12`

`bucket-type`

Description The type of the bucket. The type must already exist in the hierarchy.

Type String

Required Yes

Example `rack`

The following example adds the `rack12` bucket to the hierarchy:

```
ceph osd crush add-bucket rack12 rack
```

Move a Bucket

To move a bucket to a different location or position in the CRUSH map hierarchy, execute the following:

```
ceph osd crush move {bucket-name} {bucket-type}={bucket-name}, [...]
```

Where:

`bucket-name`

Description The name of the bucket to move/reposition.

Type String

Required Yes

Example `foo-bar-1`

`bucket-type`

Description You may specify the bucket's location in the CRUSH hierarchy.

Type Key/value pairs.

Required No

Example `datacenter=dc1 room=room1 row=foo rack=bar host=foo-bar-1`

Remove a Bucket

To remove a bucket from the CRUSH map hierarchy, execute the following:

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

Note: A bucket must be empty before removing it from the CRUSH hierarchy.

Where:

`bucket-name`

Description The name of the bucket that you'd like to remove.

Type String

Required Yes

Example `rack12`

The following example removes the `rack12` bucket from the hierarchy:

```
ceph osd crush remove rack12
```

Tunables

New in version 0.48.

There are several magic numbers that were used in the original CRUSH implementation that have proven to be poor choices. To support the transition away from them, newer versions of CRUSH (starting with the v0.48 argonaut series) allow the values to be adjusted or tuned.

Clusters running recent Ceph releases support using the tunable values in the CRUSH maps. However, older clients and daemons will not correctly interact with clusters using the “tuned” CRUSH maps. To detect this situation, there are now features bits `CRUSH_TUNABLES` (value 0x40000) and `CRUSH_TUNABLES2` to reflect support for tunables.

If the OSDMap currently used by the `ceph-mon` or `ceph-osd` daemon has non-legacy values, it will require the `CRUSH_TUNABLES` or `CRUSH_TUNABLES2` feature bits from clients and daemons who connect to it. This means that old clients will not be able to connect.

At some future point in time, newly created clusters will have improved default values for the tunables. This is a matter of waiting until the support has been present in the Linux kernel clients long enough to make this a painless transition for most users.

Impact of Legacy Values

The legacy values result in several misbehaviors:

- For hiearchies with a small number of devices in the leaf buckets, some PGs map to fewer than the desired number of replicas. This commonly happens for hiearchies with “host” nodes with a small number (1-3) of OSDs nested beneath each one.
- For large clusters, some small percentages of PGs map to less than the desired number of OSDs. This is more prevalent when there are several layers of the hierarchy (e.g., row, rack, host, osd).
- When some OSDs are marked out, the data tends to get redistributed to nearby OSDs instead of across the entire hierarchy.

CRUSH_TUNABLES

- `choose_local_tries`: Number of local retries. Legacy value is 2, optimal value is 0.
- `choose_local_fallback_tries`: Legacy value is 5, optimal value is 0.
- `choose_total_tries`: Total number of attempts to choose an item. Legacy value was 19, subsequent testing indicates that a value of 50 is more appropriate for typical clusters. For extremely large clusters, a larger value might be necessary.

CRUSH_TUNABLES2

- `chooseleaf_descend_once`: Whether a recursive chooseleaf attempt will retry, or only try once and allow the original placement to retry. Legacy default is 0, optimal value is 1.

CRUSH_TUNABLES3

- `chooseleaf_vary_r`: Whether a recursive chooseleaf attempt will start with a non-zero value of r, based on how many attempts the parent has already made. Legacy default is 0, but with this value CRUSH is sometimes unable to find a mapping. The optimal value (in terms of computational cost and correctness) is 1. However, for legacy clusters that have lots of existing data, changing from 0 to 1 will cause a lot of data to move; a value of 4 or 5 will allow CRUSH to find a valid mapping but will make less data move.

Which client versions support CRUSH_TUNABLES

- argonaut series, v0.48.1 or later
- v0.49 or later
- Linux kernel version v3.6 or later (for the file system and RBD kernel clients)

Which client versions support CRUSH_TUNABLES2

- v0.55 or later, including bobtail series (v0.56.x)
- Linux kernel version v3.9 or later (for the file system and RBD kernel clients)

Which client versions support CRUSH_TUNABLES3

- v0.78 (firefly) or later
- Linux kernel version v3.15 or later (for the file system and RBD kernel clients)

Warning when tunables are non-optimal

Starting with version v0.74, Ceph will issue a health warning if the CRUSH tunables are not set to their optimal values (the optimal values are the default as of v0.73). To make this warning go away, you have two options:

1. Adjust the tunables on the existing cluster. Note that this will result in some data movement (possibly as much as 10%). This is the preferred route, but should be taken with care on a production cluster where the data movement may affect performance. You can enable optimal tunables with:

```
ceph osd crush tunables optimal
```

   If things go poorly (e.g., too much load) and not very much progress has been made, or there is a client compatibility problem (old kernel cephfs or rbd clients, or pre-bobtail librados clients), you can switch back with:

```
ceph osd crush tunables legacy
```

2. You can make the warning go away without making any changes to CRUSH by adding the following option to your ceph.conf `[mon]` section:

```
mon warn on legacy crush tunables = false
```

   For the change to take effect, you will need to restart the monitors, or apply the option to running monitors with:

```
ceph tell mon.\* injectargs --no-mon-warn-on-legacy-crush-tunables
```

A few important points

- Adjusting these values will result in the shift of some PGs between storage nodes. If the Ceph cluster is already storing a lot of data, be prepared for some fraction of the data to move.
- The `ceph-osd` and `ceph-mon` daemons will start requiring the feature bits of new connections as soon as they get the updated map. However, already-connected clients are effectively grandfathered in, and will misbehave if they do not support the new feature.
- If the CRUSH tunables are set to non-legacy values and then later changed back to the defult values, `ceph-osd` daemons will not be required to support the feature. However, the OSD peering process requires examining and understanding old maps. Therefore, you should not run old versions of the `ceph-osd` daemon if the cluster has previously used non-legacy CRUSH values, even if the latest version of the map has been switched back to using the legacy defaults.

Tuning CRUSH

The simplest way to adjust the crush tunables is by changing to a known profile. Those are:

- `legacy`: the legacy behavior from argonaut and earlier.
- `argonaut`: the legacy values supported by the original argonaut release
- `bobtail`: the values supported by the bobtail release
- `firefly`: the values supported by the firefly release
- `optimal`: the current best values
- `default`: the current default values for a new cluster

You can select a profile on a running cluster with the command:

```
ceph osd crush tunables {PROFILE}
```

Note that this may result in some data movement.

Tuning CRUSH, the hard way

If you can ensure that all clients are running recent code, you can adjust the tunables by extracting the CRUSH map, modifying the values, and reinjecting it into the cluster.

- Extract the latest CRUSH map:

```
    ceph osd getcrushmap -o /tmp/crush
```

- Adjust tunables. These values appear to offer the best behavior for both large and small clusters we tested with. You will need to additionally specify the `--enable-unsafe-tunables` argument to `crushtool` for this to work. Please use this option with extreme care.:

```
    crushtool -i /tmp/crush --set-choose-local-tries 0 --set-choose-local-fallback-tries 0 --set-cho
```

- Reinject modified map:

```
    ceph osd setcrushmap -i /tmp/crush.new
```

Legacy values

For reference, the legacy values for the CRUSH tunables can be set with:

```
crushtool -i /tmp/crush --set-choose-local-tries 2 --set-choose-local-fallback-tries 5 --set-choose-t
```

Again, the special `--enable-unsafe-tunables` option is required. Further, as noted above, be careful running old versions of the `ceph-osd` daemon after reverting to legacy values as the feature bit is not perfectly enforced.

Low-level cluster operations consist of starting, stopping, and restarting a particular daemon within a cluster; changing the settings of a particular daemon or subsystem; and, adding a daemon to the cluster or removing a daemon from the cluster. The most common use cases for low-level operations include growing or shrinking the Ceph cluster and replacing legacy or failed hardware with new hardware.

4.3.11 Adding/Removing OSDs

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

Adding OSDs

When you want to expand a cluster, you may add 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 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.

Warning: 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`.

Deploy your Hardware

If you are adding a new host when adding a new OSD, see Hardware Recommendations for details on minimum recommendations for OSD hardware. To add an OSD host to your cluster, first make sure you have an up-to-date version of Linux installed, and you have made some initial preparations for your storage drives. See Filesystem Recommendations for details.

Add your OSD host to a rack in your cluster, connect it to the network and ensure that it has network connectivity. See the Network Configuration Reference for details.

Install the Required Software

For manually deployed clusters, you must install Ceph packages manually. See Installing Ceph (Manual) for details. You should configure SSH to a user with password-less authentication and root permissions.

Adding an OSD (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.

When you add the OSD to the CRUSH map, consider the weight you give to the new OSD. Hard drive capacity grows 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).

Tip: Ceph prefers uniform hardware across pools. If you are adding drives of dissimilar size, you can adjust their weights. However, for best performance, consider a CRUSH hierarchy with drives of the same type/size.

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}/ke
```

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.

   For Argonaut (v 0.48), execute the following:

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

   For Bobtail (v 0.56) and later releases, 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. See Add/Move an OSD for details.

Argonaut (v0.48) Best Practices

To limit impact on user I/O performance, add an OSD to the CRUSH map with an initial weight of `0`. Then, ramp up the CRUSH weight a little bit at a time. For example, to ramp by increments of `0.2`, start with:

```
ceph osd crush reweight {osd-id} .2
```

and allow migration to complete before reweighting to `0.4`, `0.6`, and so on until the desired CRUSH weight is reached.
To limit the impact of OSD failures, you can set:

```
mon osd down out interval = 0
```

which prevents down OSDs from automatically being marked out, and then ramp them down manually with:

```
ceph osd reweight {osd-num} .8
```

Again, wait for the cluster to finish migrating data, and then adjust the weight further until you reach a weight of 0. Note that this problem prevents the cluster to automatically re-replicate data after a failure, so please ensure that sufficient monitoring is in place for an administrator to intervene promptly.
Note that this practice will no longer be necessary in Bobtail and subsequent releases.

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. You may use `service ceph` from your admin host or start the OSD from its host machine.

For Debian/Ubuntu use Upstart.

```
sudo start ceph-osd id={osd-num}
```

For CentOS/RHEL, use sysvinit.

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

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

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.)

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`.

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}
```

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 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.)

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`.

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 Remove an OSD 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.

4.3.12 Adding/Removing Monitors

When you have a cluster up and running, you may add or remove monitors from the cluster at runtime. To bootstrap a monitor, see Manual Deployment or Monitor Bootstrap.

Adding Monitors

Ceph monitors are light-weight processes that maintain a master copy of the cluster map. 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).

It is advisable to run an odd-number of monitors but not mandatory. An odd-number of monitors has a higher resiliency to failures than an even-number of monitors. For instance, on a 2 monitor deployment, no failures can be tolerated in order to maintain a quorum; with 3 monitors, one failure can be tolerated; in a 4 monitor deployment, one failure can be tolerated; with 5 monitors, two failures can be tolerated. 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, it is advisable to deploy 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.

Deploy your Hardware

If you are adding a new host when adding a new monitor, see Hardware Recommendations for details on minimum recommendations for monitor hardware. To add a monitor host to your cluster, first make sure you have an up-to-date version of Linux installed (typically Ubuntu 14.04 or RHEL 7).

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

Install the Required Software

For manually deployed clusters, you must install Ceph packages manually. See Installing Packages for details. You should configure SSH to a user with password-less authentication and root permissions.

Adding a Monitor (Manual)

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}
```

Removing Monitors

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.

Removing a Monitor (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`.

Removing Monitors from an Unhealthy Cluster

This procedure removes a `ceph-mon` daemon from an unhealthy cluster, for example a cluster where the monitors cannot form a quorum.

1. Stop all `ceph-mon` daemons on all monitor hosts.

```
ssh {mon-host}
service ceph stop mon || stop ceph-mon-all
# and repeat for all mons
```

2. Identify a surviving monitor and log in to that host.

```
ssh {mon-host}
```

3. Extract a copy of the monmap file.

```
ceph-mon -i {mon-id} --extract-monmap {map-path}
# in most cases, that's
ceph-mon -i `hostname` --extract-monmap /tmp/monmap
```

4. Remove the non-surviving or problematic 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
```

5. Inject the surviving map with the removed monitors into the surviving monitor(s). 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
```

6. Start only the surviving monitors.

7. Verify the monitors form a quorum (`ceph -s`).

8. You may wish to archive the removed monitors' data directory in `/var/lib/ceph/mon` in a safe location, or delete it if you are confident the remaining monitors are healthy and are sufficiently redundant.

Changing a Monitor's IP Address

Important: Existing monitors are not supposed to change their IP addresses.

Monitors are critical components of a Ceph cluster, and they need to maintain a quorum for the whole system to work properly. To establish a quorum, the monitors need to discover each other. Ceph has strict requirements for discovering monitors.

Ceph clients and other Ceph daemons use `ceph.conf` to discover monitors. However, monitors discover each other using the monitor map, not `ceph.conf`. For example, if you refer to *Adding a Monitor (Manual)* you will see that you need to obtain the current monmap for the cluster when creating a new monitor, as it is one of the required arguments of `ceph-mon -i {mon-id} --mkfs`. The following sections explain the consistency requirements for Ceph monitors, and a few safe ways to change a monitor's IP address.

Consistency Requirements

A monitor always refers to the local copy of the monmap when discovering other monitors in the cluster. Using the monmap instead of `ceph.conf` avoids errors that could break the cluster (e.g., typos in `ceph.conf` when specifying a monitor address or port). Since monitors use monmaps for discovery and they share monmaps with clients and other Ceph daemons, the monmap provides monitors with a strict guarantee that their consensus is valid.

Strict consistency also applies to updates to the monmap. As with any other updates on the monitor, changes to the monmap always run through a distributed consensus algorithm called Paxos. The monitors must agree on each update to the monmap, such as adding or removing a monitor, to ensure that each monitor in the quorum has the same version of the monmap. Updates to the monmap are incremental so that monitors have the latest agreed upon version, and a set of previous versions, allowing a monitor that has an older version of the monmap to catch up with the current state of the cluster.

If monitors discovered each other through the Ceph configuration file instead of through the monmap, it would introduce additional risks because the Ceph configuration files aren't updated and distributed automatically. Monitors might inadvertently use an older `ceph.conf` file, fail to recognize a monitor, fall out of a quorum, or develop a situation where Paxos isn't able to determine the current state of the system accurately. Consequently, making changes to an existing monitor's IP address must be done with great care.

Changing a Monitor's IP address (The Right Way)

Changing a monitor's IP address in `ceph.conf` only is not sufficient to ensure that other monitors in the cluster will receive the update. To change a monitor's IP address, you must add a new monitor with the IP address you want to use (as described in *Adding a Monitor (Manual)*), ensure that the new monitor successfully joins the quorum; then, remove the monitor that uses the old IP address. Then, update the `ceph.conf` file to ensure that clients and other daemons know the IP address of the new monitor.

For example, lets assume there are three monitors in place, such as

```
[mon.a]
        host = host01
        addr = 10.0.0.1:6789
[mon.b]
        host = host02
        addr = 10.0.0.2:6789
```

```
[mon.c]
        host = host03
        addr = 10.0.0.3:6789
```

To change `mon.c` to `host04` with the IP address `10.0.0.4`, follow the steps in *Adding a Monitor (Manual)* by adding a new monitor `mon.d`. Ensure that `mon.d` is running before removing `mon.c`, or it will break the quorum. Remove `mon.c` as described on *Removing a Monitor (Manual)*. Moving all three monitors would thus require repeating this process as many times as needed.

Changing a Monitor's IP address (The Messy Way)

There may come a time when the monitors must be moved to a different network, a different part of the datacenter or a different datacenter altogether. While it is possible to do it, the process becomes a bit more hazardous.

In such a case, the solution is to generate a new monmap with updated IP addresses for all the monitors in the cluster, and inject the new map on each individual monitor. This is not the most user-friendly approach, but we do not expect this to be something that needs to be done every other week. As it is clearly stated on the top of this section, monitors are not supposed to change IP addresses.

Using the previous monitor configuration as an example, assume you want to move all the monitors from the `10.0.0.x` range to `10.1.0.x`, and these networks are unable to communicate. Use the following procedure:

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

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

2. The following example demonstrates the contents of the monmap.

```
    $ monmaptool --print {tmp}/{filename}

    monmaptool: monmap file {tmp}/{filename}
    epoch 1
    fsid 224e376d-c5fe-4504-96bb-ea6332a19e61
    last_changed 2012-12-17 02:46:41.591248
    created 2012-12-17 02:46:41.591248
    0: 10.0.0.1:6789/0 mon.a
    1: 10.0.0.2:6789/0 mon.b
    2: 10.0.0.3:6789/0 mon.c
```

3. Remove the existing monitors.

```
    $ monmaptool --rm a --rm b --rm c {tmp}/{filename}

    monmaptool: monmap file {tmp}/{filename}
    monmaptool: removing a
    monmaptool: removing b
    monmaptool: removing c
    monmaptool: writing epoch 1 to {tmp}/{filename} (0 monitors)
```

4. Add the new monitor locations.

```
    $ monmaptool --add a 10.1.0.1:6789 --add b 10.1.0.2:6789 --add c 10.1.0.3:6789 {tmp}/{filename}

    monmaptool: monmap file {tmp}/{filename}
    monmaptool: writing epoch 1 to {tmp}/{filename} (3 monitors)
```

5. Check new contents.

```
    $ monmaptool --print {tmp}/{filename}

    monmaptool: monmap file {tmp}/{filename}
    epoch 1
    fsid 224e376d-c5fe-4504-96bb-ea6332a19e61
    last_changed 2012-12-17 02:46:41.591248
    created 2012-12-17 02:46:41.591248
    0: 10.1.0.1:6789/0 mon.a
    1: 10.1.0.2:6789/0 mon.b
    2: 10.1.0.3:6789/0 mon.c
```

At this point, we assume the monitors (and stores) are installed at the new location. The next step is to propagate the modified monmap to the new monitors, and inject the modified monmap into each new monitor.

1. First, make sure to stop all your monitors. Injection must be done while the daemon is not running.
2. Inject the monmap.

```
    ceph-mon -i {mon-id} --inject-monmap {tmp}/{filename}
```

3. Restart the monitors.

After this step, migration to the new location is complete and the monitors should operate successfully.

4.3.13 Control Commands

Monitor Commands

Monitor commands are issued using the ceph utility:

```
ceph [-m monhost] {command}
```

The command is usually (though not always) of the form:

```
ceph {subsystem} {command}
```

System Commands

Execute the following to display the current status of the cluster.

```
ceph -s
ceph status
```

Execute the following to display a running summary of the status of the cluster, and major events.

```
ceph -w
```

Execute the following to show the monitor quorum, including which monitors are participating and which one is the leader.

```
ceph quorum_status
```

Execute the following to query the status of a single monitor, including whether or not it is in the quorum.

```
ceph [-m monhost] mon_status
```

Authentication Subsystem

To add a keyring for an OSD, execute the following:

```
ceph auth add {osd} {--in-file|-i} {path-to-osd-keyring}
```

To list the cluster's keys and their capabilities, execute the following:

```
ceph auth list
```

Placement Group Subsystem

To display the statistics for all placement groups, execute the following:

```
ceph pg dump [--format {format}]
```

The valid formats are `plain` (default) and `json`.

To display the statistics for all placement groups stuck in a specified state, execute the following:

```
ceph pg dump_stuck inactive|unclean|stale|undersized|degraded [--format {format}] [-t|--threshold {se
```

`--format` may be `plain` (default) or `json`

`--threshold` defines how many seconds "stuck" is (default: 300)

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.

Unclean Placement groups contain objects that are not replicated the desired number of times. They should be recovering.

Stale Placement groups are in an unknown state - the OSDs that host them have not reported to the monitor cluster in a while (configured by `mon_osd_report_timeout`).

Delete "lost" objects or revert them to their prior state, either a previous version or delete them if they were just created.

```
ceph pg {pgid} mark_unfound_lost revert|delete
```

OSD Subsystem

Query OSD subsystem status.

```
ceph osd stat
```

Write a copy of the most recent OSD map to a file. See osdmaptool.

```
ceph osd getmap -o file
```

Write a copy of the crush map from the most recent OSD map to file.

```
ceph osd getcrushmap -o file
```

The foregoing functionally equivalent to

```
ceph osd getmap -o /tmp/osdmap
osdmaptool /tmp/osdmap --export-crush file
```

Dump the OSD map. Valid formats for `-f` are `plain` and `json`. If no `--format` option is given, the OSD map is dumped as plain text.

```
ceph osd dump [--format {format}]
```

Dump the OSD map as a tree with one line per OSD containing weight and state.

```
ceph osd tree [--format {format}]
```

Find out where a specific object is or would be stored in the system:

```
ceph osd map <pool-name> <object-name>
```

Add or move a new item (OSD) with the given id/name/weight at the specified location.

```
ceph osd crush set {id} {weight} [{loc1} [{loc2} ...]]
```

Remove an existing item (OSD) from the CRUSH map.

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

Remove an existing bucket from the CRUSH map.

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

Move an existing bucket from one position in the hierarchy to another.

```
ceph osd crush move {id} {loc1} [{loc2} ...]
```

Set the weight of the item given by `{name}` to `{weight}`.

```
ceph osd crush reweight {name} {weight}
```

Create a cluster snapshot.

```
ceph osd cluster_snap {name}
```

Mark an OSD as lost. This may result in permanent data loss. Use with caution.

```
ceph osd lost {id} [--yes-i-really-mean-it]
```

Create a new OSD. If no UUID is given, it will be set automatically when the OSD starts up.

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

Remove the given OSD(s).

```
ceph osd rm [{id}...]
```

Query the current max_osd parameter in the OSD map.

```
ceph osd getmaxosd
```

Import the given crush map.

```
ceph osd setcrushmap -i file
```

Set the `max_osd` parameter in the OSD map. This is necessary when expanding the storage cluster.

```
ceph osd setmaxosd
```

Mark OSD `{osd-num}` down.

```
ceph osd down {osd-num}
```

Mark OSD `{osd-num}` out of the distribution (i.e. allocated no data).

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

Mark `{osd-num}` in the distribution (i.e. allocated data).

```
ceph osd in {osd-num}
```

List classes that are loaded in the ceph cluster.

```
ceph class list
```

Set or clear the pause flags in the OSD map. If set, no IO requests will be sent to any OSD. Clearing the flags via unpause results in resending pending requests.

```
ceph osd pause
ceph osd unpause
```

Set the weight of `{osd-num}` to `{weight}`. Two OSDs with the same weight will receive roughly the same number of I/O requests and store approximately the same amount of data. `ceph osd reweight` sets an override weight on the OSD. This value is in the range 0 to 1, and forces CRUSH to re-place (1-weight) of the data that would otherwise live on this drive. It does not change the weights assigned to the buckets above the OSD in the crush map, and is a corrective measure in case the normal CRUSH distribution isn't working out quite right. For instance, if one of your OSDs is at 90% and the others are at 50%, you could reduce this weight to try and compensate for it.

```
ceph osd reweight {osd-num} {weight}
```

Reweights all the OSDs by reducing the weight of OSDs which are heavily overused. By default it will adjust the weights downward on OSDs which have 120% of the average utilization, but if you include threshold it will use that percentage instead.

```
ceph osd reweight-by-utilization [threshold]
```

Adds/removes the address to/from the blacklist. When adding an address, you can specify how long it should be blacklisted in seconds; otherwise, it will default to 1 hour. A blacklisted address is prevented from connecting to any OSD. Blacklisting is most often used to prevent a lagging metadata server from making bad changes to data on the OSDs.

These commands are mostly only useful for failure testing, as blacklists are normally maintained automatically and shouldn't need manual intervention.

```
ceph osd blacklist add ADDRESS[:source_port] [TIME]
ceph osd blacklist rm ADDRESS[:source_port]
```

Creates/deletes a snapshot of a pool.

```
ceph osd pool mksnap {pool-name} {snap-name}
ceph osd pool rmsnap {pool-name} {snap-name}
```

Creates/deletes/renames a storage pool.

```
ceph osd pool create {pool-name} pg_num [pgp_num]
ceph osd pool delete {pool-name} [{pool-name} --yes-i-really-really-mean-it]
ceph osd pool rename {old-name} {new-name}
```

Changes a pool setting.

```
ceph osd pool set {pool-name} {field} {value}
```

Valid fields are:

- `size`: Sets the number of copies of data in the pool.
- `crash_replay_interval`: The number of seconds to allow clients to replay acknowledged but uncommited requests.
- `pg_num`: The placement group number.
- `pgp_num`: Effective number when calculating pg placement.
- `crush_ruleset`: rule number for mapping placement.

Get the value of a pool setting.

```
ceph osd pool get {pool-name} {field}
```

Valid fields are:

- `pg_num`: The placement group number.
- `pgp_num`: Effective number of placement groups when calculating placement.
- `lpg_num`: The number of local placement groups.
- `lpgp_num`: The number used for placing the local placement groups.

Sends a scrub command to OSD `{osd-num}`. To send the command to all OSDs, use `*`.

```
ceph osd scrub {osd-num}
```

Sends a repair command to OSD.N. To send the command to all OSDs, use `*`.

```
ceph osd repair N
```

Runs a simple throughput benchmark against OSD.N, writing `NUMBER_OF_OBJECTS` in write requests of `BYTES_PER_WRITE` each. By default, the test writes 1 GB in total in 4-MB increments. The benchmark is non-destructive and will not overwrite existing live OSD data, but might temporarily affect the performance of clients concurrently accessing the OSD.

```
ceph tell osd.N bench [NUMER_OF_OBJECTS] [BYTES_PER_WRITE]
```

MDS Subsystem

Change configuration parameters on a running mds.

```
ceph tell mds.{mds-id} injectargs --{switch} {value} [--{switch} {value}]
```

Example:

```
ceph tell mds.0 injectargs --debug_ms 1 --debug_mds 10
```

Enables debug messages.

```
ceph mds stat
```

Displays the status of all metadata servers.

ceph mds fail 0

Marks the active MDS as failed, triggering failover to a standby if present.

Todo

`ceph mds` subcommands missing docs: set, dump, getmap, stop, setmap

Mon Subsystem

Show monitor stats:

```
ceph mon stat

2011-12-14 10:40:59.044395 mon {- [mon,stat]
2011-12-14 10:40:59.057111 mon.1 -} 'e3: 5 mons at {a=10.1.2.3:6789/0,b=10.1.2.4:6789/0,c=10.1.2.5:6
```

The `quorum` list at the end lists monitor nodes that are part of the current quorum.

This is also available more directly:

```
$ ./ceph quorum_status

2011-12-14 10:44:20.417705 mon {- [quorum_status]
2011-12-14 10:44:20.431890 mon.0 -}
```

```
'{ "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"}]}}' (0)
```

The above will block until a quorum is reached.

For a status of just the monitor you connect to (use `-m HOST:PORT` to select):

```
ceph mon_status


2011-12-14 10:45:30.644414 mon {- [mon_status]
2011-12-14 10:45:30.644632 mon.0 -}
```

```
'{ "name": "a",
  "rank": 0,
  "state": "leader",
  "election_epoch": 10,
  "quorum": [
        0,
        1,
        2],
  "outside_quorum": [],
  "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"}]}}' (0)
```

A dump of the monitor state:

```
ceph mon dump

2011-12-14 10:43:08.015333 mon {- [mon,dump]
2011-12-14 10:43:08.015567 mon.0 -} 'dumped monmap epoch 1' (0)
epoch 1
fsid 444b489c-4f16-4b75-83f0-cb8097468898
last_changed 2011-12-12 13:28:27.505520
created 2011-12-12 13:28:27.505520
0: 127.0.0.1:6789/0 mon.a
1: 127.0.0.1:6790/0 mon.b
2: 127.0.0.1:6791/0 mon.c
```

Ceph is still on the leading edge, so you may encounter situations that require you to evaluate your Ceph configuration and modify your logging and debugging settings to identify and remedy issues you are encountering with your cluster.

4.3.14 The Ceph Community

The Ceph community is an excellent source of information and help. For operational issues with Ceph releases we recommend you subscribe to the ceph-users email list. When you no longer want to receive emails, you can unsubscribe from the ceph-users email list.

You may also subscribe to the ceph-devel email list. You should do so if your issue is:

- Likely related to a bug
- Related to a development release package
- Related to a development testing package
- Related to your own builds

If you no longer want to receive emails from the `ceph-devel` email list, you may unsubscribe from the ceph-devel email list.

Tip: The Ceph community is growing rapidly, and community members can help you if you provide them with detailed information about your problem. You can attach your ceph configuration file, log files, CRUSH map, and other details (e.g., `ceph osd tree`) to help people understand your issues.

4.3.15 Troubleshooting Monitors

When a cluster encounters monitor-related troubles there's a tendency to panic, and some times with good reason. You should keep in mind that losing a monitor, or a bunch of them, don't necessarily mean that your cluster is down, as long as a majority is up, running and with a formed quorum. Regardless of how bad the situation is, the first thing you should do is to calm down, take a breath and try answering our initial troubleshooting script.

Initial Troubleshooting

Are the monitors running?

First of all, we need to make sure the monitors are running. You would be amazed by how often people forget to run the monitors, or restart them after an upgrade. There's no shame in that, but let's try not losing a couple of hours chasing an issue that is not there.

Are you able to connect to the monitor's servers?

Doesn't happen often, but sometimes people do have `iptables` rules that block accesses to monitor servers or monitor ports. Usually leftovers from monitor stress-testing that were forgotten at some point. Try ssh'ing into the server and, if that succeeds, try connecting to the monitor's port using you tool of choice (telnet, nc,...).

Does ‘‘ceph -s‘‘ run and obtain a reply from the cluster?

If the answer is yes then your cluster is up and running. One thing you can take for granted is that the monitors will only answer to a `status` request if there is a formed quorum.

If `ceph -s` blocked however, without obtaining a reply from the cluster or showing a lot of `fault` messages, then it is likely that your monitors are either down completely or just a portion is up – a portion that is not enough to form a quorum (keep in mind that a quorum if formed by a majority of monitors).

What if ‘‘ceph -s‘‘ doesn't finish?

If you haven't gone through all the steps so far, please go back and do.

For those running on Emperor 0.72-rc1 and forward, you will be able to contact each monitor individually asking them for their status, regardless of a quorum being formed. This an be achieved using `ceph ping mon.ID`, ID being the monitor's identifier. You should perform this for each monitor in the cluster. In section *Understanding mon_status* we will explain how to interpret the output of this command.

For the rest of you who don't tread on the bleeding edge, you will need to ssh into the server and use the monitor's admin socket. Please jump to *Using the monitor's admin socket*.

For other specific issues, keep on reading.

Using the monitor's admin socket

The admin socket allows you to interact with a given daemon directly using a Unix socket file. This file can be found in your monitor's `run` directory. By default, the admin socket will be kept in `/var/run/ceph/ceph-mon.ID.asok` but this can vary if you defined it otherwise. If you don't find it there, please check your `ceph.conf` for an alternative path or run:

```
ceph-conf --name mon.ID --show-config-value admin_socket
```

Please bear in mind that the admin socket will only be available while the monitor is running. When the monitor is properly shutdown, the admin socket will be removed. If however the monitor is not running and the admin socket still persists, it is likely that the monitor was improperly shutdown. Regardless, if the monitor is not running, you will not be able to use the admin socket, with `ceph` likely returning `Error 111: Connection Refused`.

Accessing the admin socket is as simple as telling the `ceph` tool to use the `asok` file. In pre-Dumpling Ceph, this can be achieved by:

```
ceph --admin-daemon /var/run/ceph/ceph-mon.ID.asok <command>
```

while in Dumpling and beyond you can use the alternate (and recommended) format:

```
ceph daemon mon.ID <command>
```

Using `help` as the command to the `ceph` tool will show you the supported commands available through the admin socket. Please take a look at `config get`, `config show`, `mon_status` and `quorum_status`, as those can be enlightening when troubleshooting a monitor.

Understanding mon_status

`mon_status` can be obtained through the `ceph` tool when you have a formed quorum, or via the admin socket if you don't. This command will output a multitude of information about the monitor, including the same output you would get with `quorum_status`.

Take the following example of `mon_status`:

```
{ "name": "c",
  "rank": 2,
  "state": "peon",
  "election_epoch": 38,
  "quorum": [
        1,
        2],
  "outside_quorum": [],
  "extra_probe_peers": [],
  "sync_provider": [],
  "monmap": { "epoch": 3,
      "fsid": "5c4e9d53-e2e1-478a-8061-f543f8be4cf8",
      "modified": "2013-10-30 04:12:01.945629",
      "created": "2013-10-29 14:14:41.914786",
      "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": "127.0.0.1:6795\/0"}]}}
```

A couple of things are obvious: we have three monitors in the monmap (*a*, *b* and *c*), the quorum is formed by only two monitors, and *c* is in the quorum as a *peon*.

Which monitor is out of the quorum?

The answer would be **a**.

Why?

Take a look at the `quorum` set. We have two monitors in this set: *1* and *2*. These are not monitor names. These are monitor ranks, as established in the current monmap. We are missing the monitor with rank 0, and according to the monmap that would be `mon.a`.

By the way, how are ranks established?

Ranks are (re)calculated whenever you add or remove monitors and follow a simple rule: the **greater** the `IP:PORT` combination, the **lower** the rank is. In this case, considering that `127.0.0.1:6789` is lower than all the remaining `IP:PORT` combinations, `mon.a` has rank 0.

Most Common Monitor Issues

Have Quorum but at least one Monitor is down

When this happens, depending on the version of Ceph you are running, you should be seeing something similar to:

```
$ ceph health detail
[snip]
mon.a (rank 0) addr 127.0.0.1:6789/0 is down (out of quorum)
```

How to troubleshoot this?

First, make sure `mon.a` is running.

Second, make sure you are able to connect to `mon.a`'s server from the other monitors' servers. Check the ports as well. Check `iptables` on all your monitor nodes and make sure you're not dropping/rejecting connections.

If this initial troubleshooting doesn't solve your problems, then it's time to go deeper.

First, check the problematic monitor's `mon_status` via the admin socket as explained in *Using the monitor's admin socket* and *Understanding mon_status*.

Considering the monitor is out of the quorum, its state should be one of `probing`, `electing` or `synchronizing`. If it happens to be either `leader` or `peon`, then the monitor believes to be in quorum, while the remaining cluster is sure it is not; or maybe it got into the quorum while we were troubleshooting the monitor, so check you `ceph -s` again just to make sure. Proceed if the monitor is not yet in the quorum.

What if the state is `probing`?

This means the monitor is still looking for the other monitors. Every time you start a monitor, the monitor will stay in this state for some time while trying to find the rest of the monitors specified in the `monmap`. The time a monitor will spend in this state can vary. For instance, when on a single-monitor cluster, the monitor will pass through the probing state almost instantaneously, since there are no other monitors around. On a multi-monitor cluster, the monitors will stay in this state until they find enough monitors to form a quorum – this means that if you have 2 out of 3 monitors down, the one remaining monitor will stay in this state indefinitively until you bring one of the other monitors up.

If you have a quorum, however, the monitor should be able to find the remaining monitors pretty fast, as long as they can be reached. If your monitor is stuck probing and you've gone through with all the communication troubleshooting, then there is a fair chance that the monitor is trying to reach the other monitors on a wrong address. `mon_status` outputs the `monmap` known to the monitor: check if the other monitor's locations match reality. If they don't, jump to *Recovering a Monitor's Broken monmap*; if they do, then it may be related to severe clock skews amongst the monitor nodes and you should refer to *Clock Skews* first, but if that doesn't solve your problem then it is the time to prepare some logs and reach out to the community (please refer to *Preparing your logs* on how to best prepare your logs).

What if state is `electing`?

This means the monitor is in the middle of an election. These should be fast to complete, but at times the monitors can get stuck electing. This is usually a sign of a clock skew among the monitor nodes; jump to *Clock Skews* for more infos on that. If all your clocks are properly synchronized, it is best if you prepare some logs and reach out to the community. This is not a state that is likely to persist and aside from (*really*) old bugs there isn't an obvious reason besides clock skews on why this would happen.

What if state is `synchronizing`?

This means the monitor is synchronizing with the rest of the cluster in order to join the quorum. The synchronization process is as faster as smaller your monitor store is, so if you have a big store it may take a while. Don't worry, it should be finished soon enough.

However, if you notice that the monitor jumps from `synchronizing` to `electing` and then back to `synchronizing`, then you do have a problem: the cluster state is advancing (i.e., generating new maps) way too fast for the synchronization process to keep up. This used to be a thing in early Cuttlefish, but since then the synchronization process was quite refactored and enhanced to avoid just this sort of behavior. If this happens in later versions let us know. And bring some logs (see *Preparing your logs*).

What if state is `leader` or `peon`?

This should not happen. There is a chance this might happen however, and it has a lot to do with clock skews – see *Clock Skews*. If you're not suffering from clock skews, then please prepare your logs (see *Preparing your logs*) and reach out to us.

Recovering a Monitor's Broken monmap

This is how a `monmap` usually looks like, depending on the number of monitors:

```
epoch 3
fsid 5c4e9d53-e2e1-478a-8061-f543f8be4cf8
last_changed 2013-10-30 04:12:01.945629
created 2013-10-29 14:14:41.914786
0: 127.0.0.1:6789/0 mon.a
1: 127.0.0.1:6790/0 mon.b
2: 127.0.0.1:6795/0 mon.c
```

This may not be what you have however. For instance, in some versions of early Cuttlefish there was this one bug that could cause your `monmap` to be nullified. Completely filled with zeros. This means that not even `monmaptool` would be able to read it because it would find it hard to make sense of only-zeros. Some other times, you may end up with a monitor with a severely outdated monmap, thus being unable to find the remaining monitors (e.g., say `mon.c` is down; you add a new monitor `mon.d`, then remove `mon.a`, then add a new monitor `mon.e` and remove `mon.b`; you will end up with a totally different monmap from the one `mon.c` knows).

In this sort of situations, you have two possible solutions:

Scrap the monitor and create a new one

You should only take this route if you are positive that you won't lose the information kept by that monitor; that you have other monitors and that they are running just fine so that your new monitor is able to synchronize from the remaining monitors. Keep in mind that destroying a monitor, if there are no other copies of its contents, may lead to loss of data.

Inject a monmap into the monitor

Usually the safest path. You should grab the monmap from the remaining monitors and inject it into the monitor with the corrupted/lost monmap.

These are the basic steps:

1. Is there a formed quorum? If so, grab the monmap from the quorum:

```
$ ceph mon getmap -o /tmp/monmap
```

2. No quorum? Grab the monmap directly from another monitor (this assumes the monitor you're grabbing the monmap from has id ID-FOO and has been stopped):

```
$ ceph-mon -i ID-FOO --extract-monmap /tmp/monmap
```

3. Stop the monitor you're going to inject the monmap into.
4. Inject the monmap:

```
$ ceph-mon -i ID --inject-monmap /tmp/monmap
```

5. Start the monitor

Please keep in mind that the ability to inject monmaps is a powerful feature that can cause havoc with your monitors if misused as it will overwrite the latest, existing monmap kept by the monitor.

Clock Skews

Monitors can be severely affected by significant clock skews across the monitor nodes. This usually translates into weird behavior with no obvious cause. To avoid such issues, you should run a clock synchronization tool on your monitor nodes.

What's the maximum tolerated clock skew?

By default the monitors will allow clocks to drift up to `0.05 seconds`.

Can I increase the maximum tolerated clock skew?

This value is configurable via the `mon-clock-drift-allowed` option, and although you *CAN* it doesn't mean you *SHOULD*. The clock skew mechanism is in place because clock skewed monitor may not properly behave. We, as developers and QA afficcionados, are comfortable with the current default value, as it will alert the user before the monitors get out hand. Changing this value without testing it first may cause unforeseen effects on the stability of the monitors and overall cluster healthiness, although there is no risk of dataloss.

How do I know there's a clock skew?

The monitors will warn you in the form of a `HEALTH_WARN`. `ceph health detail` should show something in the form of:

```
mon.c addr 10.10.0.1:6789/0 clock skew 0.08235s > max 0.05s (latency 0.0045s)
```

That means that `mon.c` has been flagged as suffering from a clock skew.

What should I do if there's a clock skew?

Synchronize your clocks. Running an NTP client may help. If you are already using one and you hit this sort of issues, check if you are using some NTP server remote to your network and consider hosting your own NTP server on your network. This last option tends to reduce the amount of issues with monitor clock skews.

Client Can't Connect or Mount

Check your IP tables. Some OS install utilities add a `REJECT` rule to `iptables`. The rule rejects all clients trying to connect to the host except for `ssh`. If your monitor host's IP tables have such a `REJECT` rule in place, clients connecting from a separate node will fail to mount with a timeout error. You need to address `iptables` rules that reject clients trying to connect to Ceph daemons. For example, you would need to address rules that look like this appropriately:

```
REJECT all -- anywhere anywhere reject-with icmp-host-prohibited
```

You may also need to add rules to IP tables on your Ceph hosts to ensure that clients can access the ports associated with your Ceph monitors (i.e., port 6789 by default) and Ceph OSDs (i.e., 6800 et. seq. by default). For example:

```
iptables -A INPUT -m multiport -p tcp -s {ip-address}/{netmask} --dports 6789,6800:6810 -j ACCEPT
```

Everything Failed! Now What?

Reaching out for help

You can find us on IRC at #ceph and #ceph-devel at OFTC (server irc.oftc.net) and on `ceph-devel@vger.kernel.org` and `ceph-users@lists.ceph.com`. Make sure you have grabbed your logs and have them ready if someone asks: the faster the interaction and lower the latency in response, the better chances everyone's time is optimized.

Preparing your logs

Monitor logs are, by default, kept in `/var/log/ceph/ceph-mon.FOO.log*`. We may want them. However, your logs may not have the necessary information. If you don't find your monitor logs at their default location, you can check where they should be by running:

```
ceph-conf --name mon.FOO --show-config-value log_file
```

The amount of information in the logs are subject to the debug levels being enforced by your configuration files. If you have not enforced a specific debug level then Ceph is using the default levels and your logs may not contain important information to track down you issue. A first step in getting relevant information into your logs will be to raise debug levels. In this case we will be interested in the information from the monitor. Similarly to what happens on other components, different parts of the monitor will output their debug information on different subsystems.

You will have to raise the debug levels of those subsystems more closely related to your issue. This may not be an easy task for someone unfamiliar with troubleshooting Ceph. For most situations, setting the following options on your monitors will be enough to pinpoint a potential source of the issue:

```
debug mon = 10
debug ms = 1
```

If we find that these debug levels are not enough, there's a chance we may ask you to raise them or even define other debug subsystems to obtain infos from – but at least we started off with some useful information, instead of a massively empty log without much to go on with.

Do I need to restart a monitor to adjust debug levels?

No. You may do it in one of two ways:

You have quorum

Either inject the debug option into the monitor you want to debug:

```
ceph tell mon.FOO injectargs --debug_mon 10/10
```

or into all monitors at once:

```
    ceph tell mon.* injectargs --debug_mon 10/10
```

No quourm

Use the monitor's admin socket and directly adjust the configuration options:

```
    ceph daemon mon.FOO config set debug_mon 10/10
```

Going back to default values is as easy as rerunning the above commands using the debug level `1/10` instead. You can check your current values using the admin socket and the following commands:

```
ceph daemon mon.FOO config show
```

or:

```
ceph daemon mon.FOO config get 'OPTION_NAME'
```

Reproduced the problem with appropriate debug levels. Now what?

Ideally you would send us only the relevant portions of your logs. We realise that figuring out the corresponding portion may not be the easiest of tasks. Therefore, we won't hold it to you if you provide the full log, but common sense should be employed. If your log has hundreds of thousands of lines, it may get tricky to go through the whole thing, specially if we are not aware at which point, whatever your issue is, happened. For instance, when reproducing, keep in mind to write down current time and date and to extract the relevant portions of your logs based on that.

Finally, you should reach out to us on the mailing lists, on IRC or file a new issue on the tracker.

4.3.16 Troubleshooting OSDs

Before troubleshooting your OSDs, check your monitors and network first. If you execute `ceph health` or `ceph -s` on the command line and Ceph returns a health status, the return of a status means that the monitors have a quorum. If you don't have a monitor quorum or if there are errors with the monitor status, address the monitor issues first. Check your networks to ensure they are running properly, because networks may have a significant impact on OSD operation and performance.

Obtaining Data About OSDs

A good first step in troubleshooting your OSDs is to obtain information in addition to the information you collected while monitoring your OSDs (e.g., `ceph osd tree`).

Ceph Logs

If you haven't changed the default path, you can find Ceph log files at `/var/log/ceph`:

```
ls /var/log/ceph
```

If you don't get enough log detail, you can change your logging level. See Logging and Debugging for details to ensure that Ceph performs adequately under high logging volume.

Admin Socket

Use the admin socket tool to retrieve runtime information. For details, list the sockets for your Ceph processes:

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

Then, execute the following, replacing `{socket-name}` with an actual socket name to show the list of available options:

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

The admin socket, among other things, allows you to:

- List your configuration at runtime
- Dump historic operations
- Dump the operation priority queue state
- Dump operations in flight
- Dump perfcounters

Display Freespace

Filesystem issues may arise. To display your filesystem's free space, execute `df`.

```
df -h
```

Execute `df --help` for additional usage.

I/O Statistics

Use iostat to identify I/O-related issues.

```
iostat -x
```

Diagnostic Messages

To retrieve diagnostic messages, use `dmesg` with `less`, `more`, `grep` or `tail`. For example:

```
dmesg | grep scsi
```

Stopping w/out Rebalancing

Periodically, you may need to perform maintenance on a subset of your cluster, or resolve a problem that affects a failure domain (e.g., a rack). If you do not want CRUSH to automatically rebalance the cluster as you stop OSDs for maintenance, set the cluster to `noout` first:

```
ceph osd set noout
```

Once the cluster is set to `noout`, you can begin stopping the OSDs within the failure domain that requires maintenance work.

```
stop ceph-osd id={num}
```

Note: Placement groups within the OSDs you stop will become `degraded` while you are addressing issues with within the failure domain.

Once you have completed your maintenance, restart the OSDs.

```
start ceph-osd id={num}
```

Finally, you must unset the cluster from `noout`.

```
ceph osd unset noout
```

OSD Not Running

Under normal circumstances, simply restarting the `ceph-osd` daemon will allow it to rejoin the cluster and recover.

An OSD Won't Start

If you start your cluster and an OSD won't start, check the following:

- **Configuration File:** If you were not able to get OSDs running from a new installation, check your configuration file to ensure it conforms (e.g., `host` not `hostname`, etc.).
- **Check Paths:** Check the paths in your configuration, and the actual paths themselves for data and journals. If you separate the OSD data from the journal data and there are errors in your configuration file or in the actual mounts, you may have trouble starting OSDs. If you want to store the journal on a block device, you should partition your journal disk and assign one partition per OSD.
- **Check Max Threadcount:** If you have a node with a lot of OSDs, you may be hitting the default maximum number of threads (e.g., usually 32k), especially during recovery. You can increase the number of threads using `sysctl` to see if increasing the maximum number of threads to the maximum possible number of threads allowed (i.e., 4194303) will help. For example:

```
sysctl -w kernel.pid_max=4194303
```

  If increasing the maximum thread count resolves the issue, you can make it permanent by including a `kernel.pid_max` setting in the `/etc/sysctl.conf` file. For example:

```
kernel.pid_max = 4194303
```

- **Kernel Version:** Identify the kernel version and distribution you are using. Ceph uses some third party tools by default, which may be buggy or may conflict with certain distributions and/or kernel versions (e.g., Google perftools). Check the OS recommendations to ensure you have addressed any issues related to your kernel.
- **Segment Fault:** If there is a segment fault, turn your logging up (if it isn't already), and try again. If it segment faults again, contact the ceph-devel email list and provide your Ceph configuration file, your monitor output and the contents of your log file(s).

If you cannot resolve the issue and the email list isn't helpful, you may contact Inktank for support.

An OSD Failed

When a `ceph-osd` process dies, the monitor will learn about the failure from surviving `ceph-osd` daemons and report it via the `ceph health` command:

```
ceph health
HEALTH_WARN 1/3 in osds are down
```

Specifically, you will get a warning whenever there are `ceph-osd` processes that are marked `in` and `down`. You can identify which `ceph-osds` are `down` with:

```
ceph health detail
HEALTH_WARN 1/3 in osds are down
osd.0 is down since epoch 23, last address 192.168.106.220:6800/11080
```

If there is a disk failure or other fault preventing `ceph-osd` from functioning or restarting, an error message should be present in its log file in `/var/log/ceph`.

If the daemon stopped because of a heartbeat failure, the underlying kernel file system may be unresponsive. Check `dmesg` output for disk or other kernel errors.

If the problem is a software error (failed assertion or other unexpected error), it should be reported to the ceph-devel email list.

No Free Drive Space

Ceph prevents you from writing to a full OSD so that you don't lose data. In an operational cluster, you should receive a warning when your cluster is getting near its full ratio. The `mon osd full ratio` defaults to `0.95`, or 95% of capacity before it stops clients from writing data. The `mon osd nearfull ratio` defaults to `0.85`, or 85% of capacity when it generates a health warning.

Full cluster issues usually arise when testing how Ceph handles an OSD failure on a small cluster. When one node has a high percentage of the cluster's data, the cluster can easily eclipse its nearfull and full ratio immediately. If you are testing how Ceph reacts to OSD failures on a small cluster, you should leave ample free disk space and consider temporarily lowering the `mon osd full ratio` and `mon osd nearfull ratio`.

Full `ceph-osds` will be reported by `ceph health`:

```
ceph health
HEALTH_WARN 1 nearfull osds
osd.2 is near full at 85%
```

Or:

```
ceph health
HEALTH_ERR 1 nearfull osds, 1 full osds
osd.2 is near full at 85%
osd.3 is full at 97%
```

The best way to deal with a full cluster is to add new `ceph-osds`, allowing the cluster to redistribute data to the newly available storage.

If you cannot start an OSD because it is full, you may delete some data by deleting some placement group directories in the full OSD.

Important: If you choose to delete a placement group directory on a full OSD, **DO NOT** delete the same placement group directory on another full OSD, or **YOU MAY LOSE DATA**. You **MUST** maintain at least one copy of your data on at least one OSD.

See Monitor Config Reference for additional details.

OSDs are Slow/Unresponsive

A commonly recurring issue involves slow or unresponsive OSDs. Ensure that you have eliminated other troubleshooting possibilities before delving into OSD performance issues. For example, ensure that your network(s) is working properly and your OSDs are running. Check to see if OSDs are throttling recovery traffic.

Tip: Newer versions of Ceph provide better recovery handling by preventing recovering OSDs from using up system resources so that `up` and `in` OSDs aren't available or are otherwise slow.

Networking Issues

Ceph is a distributed storage system, so it depends upon networks to peer with OSDs, replicate objects, recover from faults and check heartbeats. Networking issues can cause OSD latency and flapping OSDs. See *Flapping OSDs* for details.

Ensure that Ceph processes and Ceph-dependent processes are connected and/or listening.

```
netstat -a | grep ceph
netstat -l | grep ceph
sudo netstat -p | grep ceph
```

Check network statistics.

```
netstat -s
```

Drive Configuration

A storage drive should only support one OSD. Sequential read and sequential write throughput can bottleneck if other processes share the drive, including journals, operating systems, monitors, other OSDs and non-Ceph processes.

Ceph acknowledges writes *after* journaling, so fast SSDs are an attractive option to accelerate the response time–particularly when using the `ext4` or XFS filesystems. By contrast, the `btrfs` filesystem can write and journal simultaneously.

Note: Partitioning a drive does not change its total throughput or sequential read/write limits. Running a journal in a separate partition may help, but you should prefer a separate physical drive.

Bad Sectors / Fragmented Disk

Check your disks for bad sectors and fragmentation. This can cause total throughput to drop substantially.

Co-resident Monitors/OSDs

Monitors are generally light-weight processes, but they do lots of `fsync()`, which can interfere with other workloads, particularly if monitors run on the same drive as your OSDs. Additionally, if you run monitors on the same host as the OSDs, you may incur performance issues related to:

- Running an older kernel (pre-3.0)
- Running Argonaut with an old `glibc`
- Running a kernel with no syncfs(2) syscall.

In these cases, multiple OSDs running on the same host can drag each other down by doing lots of commits. That often leads to the bursty writes.

Co-resident Processes

Spinning up co-resident processes such as a cloud-based solution, virtual machines and other applications that write data to Ceph while operating on the same hardware as OSDs can introduce significant OSD latency. Generally, we recommend optimizing a host for use with Ceph and using other hosts for other processes. The practice of separating Ceph operations from other applications may help improve performance and may streamline troubleshooting and maintenance.

Logging Levels

If you turned logging levels up to track an issue and then forgot to turn logging levels back down, the OSD may be putting a lot of logs onto the disk. If you intend to keep logging levels high, you may consider mounting a drive to the default path for logging (i.e., `/var/log/ceph/$cluster-$name.log`).

Recovery Throttling

Depending upon your configuration, Ceph may reduce recovery rates to maintain performance or it may increase recovery rates to the point that recovery impacts OSD performance. Check to see if the OSD is recovering.

Kernel Version

Check the kernel version you are running. Older kernels may not receive new backports that Ceph depends upon for better performance.

Kernel Issues with SyncFS

Try running one OSD per host to see if performance improves. Old kernels might not have a recent enough version of `glibc` to support `syncfs(2)`.

Filesystem Issues

Currently, we recommend deploying clusters with XFS or ext4. The btrfs filesystem has many attractive features, but bugs in the filesystem may lead to performance issues.

Insufficient RAM

We recommend 1GB of RAM per OSD daemon. You may notice that during normal operations, the OSD only uses a fraction of that amount (e.g., 100-200MB). Unused RAM makes it tempting to use the excess RAM for co-resident applications, VMs and so forth. However, when OSDs go into recovery mode, their memory utilization spikes. If there is no RAM available, the OSD performance will slow considerably.

Old Requests or Slow Requests

If a `ceph-osd` daemon is slow to respond to a request, it will generate log messages complaining about requests that are taking too long. The warning threshold defaults to 30 seconds, and is configurable via the `osd op complaint time` option. When this happens, the cluster log will receive messages.

Legacy versions of Ceph complain about 'old requests':

```
osd.0 192.168.106.220:6800/18813 312 : [WRN] old request osd_op(client.5099.0:790 fatty_26485_object
```

New versions of Ceph complain about 'slow requests':

```
{date} {osd.num} [WRN] 1 slow requests, 1 included below; oldest blocked for > 30.005692 secs
{date} {osd.num}  [WRN] slow request 30.005692 seconds old, received at {date-time}: osd_op(client.4
```

Possible causes include:

- A bad drive (check `dmesg` output)
- A bug in the kernel file system bug (check `dmesg` output)
- An overloaded cluster (check system load, iostat, etc.)
- A bug in the `ceph-osd` daemon.

Possible solutions

- Remove VMs Cloud Solutions from Ceph Hosts
- Upgrade Kernel
- Upgrade Ceph
- Restart OSDs

Flapping OSDs

We recommend using both a public (front-end) network and a cluster (back-end) network so that you can better meet the capacity requirements of object replication. Another advantage is that you can run a cluster network such that it isn't connected to the internet, thereby preventing some denial of service attacks. When OSDs peer and check heartbeats, they use the cluster (back-end) network when it's available. See Monitor/OSD Interaction for details.

However, if the cluster (back-end) network fails or develops significant latency while the public (front-end) network operates optimally, OSDs currently do not handle this situation well. What happens is that OSDs mark each other `down` on the monitor, while marking themselves `up`. We call this scenario 'flapping'.

If something is causing OSDs to 'flap' (repeatedly getting marked `down` and then `up` again), you can force the monitors to stop the flapping with:

```
ceph osd set noup      # prevent OSDs from getting marked up
ceph osd set nodown    # prevent OSDs from getting marked down
```

These flags are recorded in the osdmap structure:

```
ceph osd dump | grep flags
flags no-up,no-down
```

You can clear the flags with:

```
ceph osd unset noup
ceph osd unset nodown
```

Two other flags are supported, `noin` and `noout`, which prevent booting OSDs from being marked `in` (allocated data) or protect OSDs from eventually being marked `out` (regardless of what the current value for `mon osd down out interval` is).

Note: `noup`, `noout`, and `nodown` are temporary in the sense that once the flags are cleared, the action they were blocking should occur shortly after. The `noin` flag, on the other hand, prevents OSDs from being marked `in` on boot, and any daemons that started while the flag was set will remain that way.

4.3.17 Troubleshooting PGs

Placement Groups Never Get Clean

When you create a cluster and your cluster remains in `active`, `active+remapped` or `active+degraded` status and never achieve an `active+clean` status, you likely have a problem with your configuration.

You may need to review settings in the Pool, PG and CRUSH Config Reference and make appropriate adjustments.

As a general rule, you should run your cluster with more than one OSD and a pool size greater than 1 object replica.

One Node Cluster

Ceph no longer provides documentation for operating on a single node, because you would never deploy a system designed for distributed computing on a single node. Additionally, mounting client kernel modules on a single node containing a Ceph daemon may cause a deadlock due to issues with the Linux kernel itself (unless you use VMs for the clients). You can experiment with Ceph in a 1-node configuration, in spite of the limitations as described herein.

If you are trying to create a cluster on a single node, you must change the default of the `osd crush chooseleaf type` setting from `1` (meaning `host` or `node`) to `0` (meaning `osd`) in your Ceph configuration file before you create your monitors and OSDs. This tells Ceph that an OSD can peer with another OSD on the same host. If you are trying to set up a 1-node cluster and `osd crush chooseleaf type` is greater than `0`, Ceph will try to peer the PGs of one OSD with the PGs of another OSD on another node, chassis, rack, row, or even datacenter depending on the setting.

Tip: DO NOT mount kernel clients directly on the same node as your Ceph Storage Cluster, because kernel conflicts can arise. However, you can mount kernel clients within virtual machines (VMs) on a single node.

If you are creating OSDs using a single disk, you must create directories for the data manually first. For example:

```
mkdir /var/local/osd0 /var/local/osd1
ceph-deploy osd prepare {localhost-name}:/var/local/osd0 {localhost-name}:/var/local/osd1
ceph-deploy osd activate {localhost-name}:/var/local/osd0 {localhost-name}:/var/local/osd1
```

Fewer OSDs than Replicas

If you've brought up two OSDs to an `up` and `in` state, but you still don't see `active + clean` placement groups, you may have an `osd pool default size` set to greater than `2`.

There are a few ways to address this situation. If you want to operate your cluster in an `active + degraded` state with two replicas, you can set the `osd pool default min size` to `2` so that you can write objects in an `active + degraded` state. You may also set the `osd pool default size` setting to `2` so that you only have two stored replicas (the original and one replica), in which case the cluster should achieve an `active + clean` state.

Note: You can make the changes at runtime. If you make the changes in your Ceph configuration file, you may need to restart your cluster.

Pool Size = 1

If you have the `osd pool default size` set to `1`, you will only have one copy of the object. OSDs rely on other OSDs to tell them which objects they should have. If a first OSD has a copy of an object and there is no second copy, then no second OSD can tell the first OSD that it should have that copy. For each placement group mapped to the first OSD (see `ceph pg dump`), you can force the first OSD to notice the placement groups it needs by running:

```
ceph pg force_create_pg <pgid>
```

CRUSH Map Errors

Another candidate for placement groups remaining unclean involves errors in your CRUSH map.

Stuck Placement Groups

It is normal for placement groups to enter states like "degraded" or "peering" following a failure. Normally these states indicate the normal progression through the failure recovery process. However, if a placement group stays in one of these states for a long time this may be an indication of a larger problem. For this reason, the monitor will warn when placement groups get "stuck" in a non-optimal state. Specifically, we check for:

- `inactive` - The placement group has not been `active` for too long (i.e., it hasn't been able to service read/write requests).
- `unclean` - The placement group has not been `clean` for too long (i.e., it hasn't been able to completely recover from a previous failure).
- `stale` - The placement group status has not been updated by a `ceph-osd`, indicating that all nodes storing this placement group may be `down`.

You can explicitly list stuck placement groups with one of:

```
ceph pg dump_stuck stale
ceph pg dump_stuck inactive
ceph pg dump_stuck unclean
```

For stuck `stale` placement groups, it is normally a matter of getting the right `ceph-osd` daemons running again. For stuck `inactive` placement groups, it is usually a peering problem (see *Placement Group Down - Peering Failure*). For stuck `unclean` placement groups, there is usually something preventing recovery from completing, like unfound objects (see *Unfound Objects*);

Placement Group Down - Peering Failure

In certain cases, the `ceph-osd` *Peering* process can run into problems, preventing a PG from becoming active and usable. For example, `ceph health` might report:

```
ceph health detail
HEALTH_ERR 7 pgs degraded; 12 pgs down; 12 pgs peering; 1 pgs recovering; 6 pgs stuck unclean; 114/3
...
pg 0.5 is down+peering
pg 1.4 is down+peering
```

```
...
osd.1 is down since epoch 69, last address 192.168.106.220:6801/8651
```

We can query the cluster to determine exactly why the PG is marked `down` with:

```
ceph pg 0.5 query
```

```
{ "state": "down+peering",
  ...
  "recovery_state": [
       { "name": "Started\/Primary\/Peering\/GetInfo",
         "enter_time": "2012-03-06 14:40:16.169679",
         "requested_info_from": []},
       { "name": "Started\/Primary\/Peering",
         "enter_time": "2012-03-06 14:40:16.169659",
         "probing_osds": [
               0,
               1],
         "blocked": "peering is blocked due to down osds",
         "down_osds_we_would_probe": [
               1],
         "peering_blocked_by": [
               { "osd": 1,
                 "current_lost_at": 0,
                 "comment": "starting or marking this osd lost may let us proceed"}]},
       { "name": "Started",
         "enter_time": "2012-03-06 14:40:16.169513"}
   ]
}
```

The `recovery_state` section tells us that peering is blocked due to down `ceph-osd` daemons, specifically `osd.1`. In this case, we can start that `ceph-osd` and things will recover.

Alternatively, if there is a catastrophic failure of `osd.1` (e.g., disk failure), we can tell the cluster that it is `lost` and to cope as best it can.

Important: This is dangerous in that the cluster cannot guarantee that the other copies of the data are consistent and up to date.

To instruct Ceph to continue anyway:

```
ceph osd lost 1
```

Recovery will proceed.

Unfound Objects

Under certain combinations of failures Ceph may complain about `unfound` objects:

```
ceph health detail
HEALTH_WARN 1 pgs degraded; 78/3778 unfound (2.065%)
pg 2.4 is active+degraded, 78 unfound
```

This means that the storage cluster knows that some objects (or newer copies of existing objects) exist, but it hasn't found copies of them. One example of how this might come about for a PG whose data is on ceph-osds 1 and 2:

- 1 goes down
- 2 handles some writes, alone

- 1 comes up
- 1 and 2 repeer, and the objects missing on 1 are queued for recovery.
- Before the new objects are copied, 2 goes down.

Now 1 knows that these object exist, but there is no live `ceph-osd` who has a copy. In this case, IO to those objects will block, and the cluster will hope that the failed node comes back soon; this is assumed to be preferable to returning an IO error to the user.

First, you can identify which objects are unfound with:

```
ceph pg 2.4 list_missing [starting offset, in json]
```

```
{ "offset": { "oid": "",
     "key": "",
     "snapid": 0,
     "hash": 0,
     "max": 0},
 "num_missing": 0,
 "num_unfound": 0,
 "objects": [
    { "oid": "object 1",
      "key": "",
      "hash": 0,
      "max": 0 },
    ...
 ],
 "more": 0}
```

If there are too many objects to list in a single result, the `more` field will be true and you can query for more. (Eventually the command line tool will hide this from you, but not yet.)

Second, you can identify which OSDs have been probed or might contain data:

```
ceph pg 2.4 query
```

```
"recovery_state": [
     { "name": "Started\/Primary\/Active",
       "enter_time": "2012-03-06 15:15:46.713212",
       "might_have_unfound": [
             { "osd": 1,
               "status": "osd is down"}]},
```

In this case, for example, the cluster knows that `osd.1` might have data, but it is `down`. The full range of possible states include:

```
* already probed
* querying
* OSD is down
* not queried (yet)
```

Sometimes it simply takes some time for the cluster to query possible locations.

It is possible that there are other locations where the object can exist that are not listed. For example, if a ceph-osd is stopped and taken out of the cluster, the cluster fully recovers, and due to some future set of failures ends up with an unfound object, it won't consider the long-departed ceph-osd as a potential location to consider. (This scenario, however, is unlikely.)

If all possible locations have been queried and objects are still lost, you may have to give up on the lost objects. This, again, is possible given unusual combinations of failures that allow the cluster to learn about writes that were performed before the writes themselves are recovered. To mark the "unfound" objects as "lost":

```
ceph pg 2.5 mark_unfound_lost revert|delete
```

This the final argument specifies how the cluster should deal with lost objects.

The “delete” option will forget about them entirely.

The “revert” option (not available for erasure coded pools) will either roll back to a previous version of the object or (if it was a new object) forget about it entirely. Use this with caution, as it may confuse applications that expected the object to exist.

Homeless Placement Groups

It is possible for all OSDs that had copies of a given placement groups to fail. If that’s the case, that subset of the object store is unavailable, and the monitor will receive no status updates for those placement groups. To detect this situation, the monitor marks any placement group whose primary OSD has failed as `stale`. For example:

```
ceph health
HEALTH_WARN 24 pgs stale; 3/300 in osds are down
```

You can identify which placement groups are `stale`, and what the last OSDs to store them were, with:

```
ceph health detail
HEALTH_WARN 24 pgs stale; 3/300 in osds are down
...
pg 2.5 is stuck stale+active+remapped, last acting [2,0]
...
osd.10 is down since epoch 23, last address 192.168.106.220:6800/11080
osd.11 is down since epoch 13, last address 192.168.106.220:6803/11539
osd.12 is down since epoch 24, last address 192.168.106.220:6806/11861
```

If we want to get placement group 2.5 back online, for example, this tells us that it was last managed by `osd.0` and `osd.2`. Restarting those `ceph-osd` daemons will allow the cluster to recover that placement group (and, presumably, many others).

Only a Few OSDs Receive Data

If you have many nodes in your cluster and only a few of them receive data, check the number of placement groups in your pool. Since placement groups get mapped to OSDs, a small number of placement groups will not distribute across your cluster. Try creating a pool with a placement group count that is a multiple of the number of OSDs. See Placement Groups for details. The default placement group count for pools isn’t useful, but you can change it here.

Can’t Write Data

If your cluster is up, but some OSDs are down and you cannot write data, check to ensure that you have the minimum number of OSDs running for the placement group. If you don’t have the minimum number of OSDs running, Ceph will not allow you to write data because there is no guarantee that Ceph can replicate your data. See `osd pool default min size` in the Pool, PG and CRUSH Config Reference for details.

PGs Inconsistent

If you receive an `active + clean + inconsistent` state, this may happen due to an error during scrubbing. If the inconsistency is due to disk errors, check your disks.

You can repair the inconsistent placement group by executing:

```
ceph pg repair {placement-group-ID}
```

If you receive `active + clean + inconsistent` states periodically due to clock skew, you may consider configuring your NTP daemons on your monitor hosts to act as peers. See The Network Time Protocol and Ceph Clock Settings for additional details.

Erasure Coded PGs are not active+clean

When CRUSH fails to find enough OSDs to map to a PG, it will show as a `2147483647` which is ITEM_NONE or `no OSD found`. For instance:

```
[2,1,6,0,5,8,2147483647,7,4]
```

Not enough OSDs

If the Ceph cluster only has 8 OSDs and the erasure coded pool needs 9, that is what it will show. You can either create another erasure coded pool that requires less OSDs:

```
ceph osd erasure-code-profile set myprofile k=5 m=3
ceph osd pool create erasurepool 16 16 erasure myprofile
```

or add a new OSDs and the PG will automatically use them.

CRUSH constraints cannot be satisfied

If the cluster has enough OSDs, it is possible that the CRUSH ruleset imposes constraints that cannot be satisfied. If there are 10 OSDs on two hosts and the CRUSH rulesets require that no two OSDs from the same host are used in the same PG, the mapping may fail because only two OSD will be found. You can check the constraint by displaying the ruleset:

```
$ ceph osd crush rule ls
[
    "replicated_ruleset",
    "erasurepool"]
$ ceph osd crush rule dump erasurepool
{ "rule_id": 1,
  "rule_name": "erasurepool",
  "ruleset": 1,
  "type": 3,
  "min_size": 3,
  "max_size": 20,
  "steps": [
        { "op": "take",
          "item": -1,
          "item_name": "default"},
        { "op": "chooseleaf_indep",
          "num": 0,
          "type": "host"},
        { "op": "emit"}]}
```

You can resolve the problem by creating a new pool in which PGs are allowed to have OSDs residing on the same host with:

```
ceph osd erasure-code-profile set myprofile ruleset-failure-domain=osd
ceph osd pool create erasurepool 16 16 erasure myprofile
```

CRUSH gives up too soon

If the Ceph cluster has just enough OSDs to map the PG (for instance a cluster with a total of 9 OSDs and an erasure coded pool that requires 9 OSDs per PG), it is possible that CRUSH gives up before finding a mapping. It can be resolved by:

- lowering the erasure coded pool requirements to use less OSDs per PG (that requires the creation of another pool as erasure code profiles cannot be dynamically modified).
- adding more OSDs to the cluster (that does not require the erasure coded pool to be modified, it will become clean automatically)
- use a hand made CRUSH ruleset that tries more times to find a good mapping. It can be done by setting `set_choose_tries` to a value greater than the default.

You should first verify the problem with `crushtool` after extracting the crushmap from the cluster so your experiments do not modify the Ceph cluster and only work on a local files:

```
$ ceph osd crush rule dump erasurepool
{ "rule_name": "erasurepool",
  "ruleset": 1,
  "type": 3,
  "min_size": 3,
  "max_size": 20,
  "steps": [
        { "op": "take",
          "item": -1,
          "item_name": "default"},
        { "op": "chooseleaf_indep",
          "num": 0,
          "type": "host"},
        { "op": "emit"}]}
$ ceph osd getcrushmap > crush.map
got crush map from osdmap epoch 13
$ crushtool -i crush.map --test --show-bad-mappings \
   --rule 1 \
   --num-rep 9 \
   --min-x 1 --max-x $((1024 * 1024))
bad mapping rule 8 x 43 num_rep 9 result [3,2,7,1,2147483647,8,5,6,0]
bad mapping rule 8 x 79 num_rep 9 result [6,0,2,1,4,7,2147483647,5,8]
bad mapping rule 8 x 173 num_rep 9 result [0,4,6,8,2,1,3,7,2147483647]
```

Where `--num-rep` is the number of OSDs the erasure code crush ruleset needs, `--rule` is the value of the `ruleset` field displayed by `ceph osd crush rule dump`. The test will try mapping one million values (i.e. the range defined by `[--min-x,--max-x]`) and must display at least one bad mapping. If it outputs nothing it means all mappings are successfull and you can stop right there: the problem is elsewhere.

The crush ruleset can be edited by decompiling the crush map:

```
$ crushtool --decompile crush.map > crush.txt
```

and adding the following line to the ruleset:

```
step set_choose_tries 100
```

The relevant part of of the `crush.txt` file should look something like:

```
rule erasurepool {
        ruleset 1
        type erasure
        min_size 3
        max_size 20
        step set_chooseleaf_tries 5
        step set_choose_tries 100
        step take default
        step chooseleaf indep 0 type host
        step emit
}
```

It can then be compiled and tested again:

```
$ crushtool --compile crush.txt -o better-crush.map
```

When all mappings succeed, an histogram of the number of tries that were necessary to find all of them can be displayed with the `--show-choose-tries` option of `crushtool`:

```
$ crushtool -i better-crush.map --test --show-bad-mappings \
   --show-choose-tries \
   --rule 1 \
   --num-rep 9 \
   --min-x 1 --max-x $((1024 * 1024))
...
11:        42
12:        44
13:        54
14:        45
15:        35
16:        34
17:        30
18:        25
19:        19
20:        22
21:        20
22:        17
23:        13
24:        16
25:        13
26:        11
27:        11
28:        13
29:        11
30:        10
31:         6
32:         5
33:        10
34:         3
35:         7
36:         5
37:         2
38:         5
39:         5
40:         2
41:         5
42:         4
```

```
43:          1
44:          2
45:          2
46:          3
47:          1
48:          0
...
102:          0
103:          1
104:          0
...
```

It took 11 tries to map 42 PGs, 12 tries to map 44 PGs etc. The highest number of tries is the minimum value of `set_choose_tries` that prevents bad mappings (i.e. 103 in the above output because it did not take more than 103 tries for any PG to be mapped).

4.3.18 Logging and Debugging

Typically, when you add debugging to your Ceph configuration, you do so at runtime. You can also add Ceph debug logging to your Ceph configuration file if you are encountering issues when starting your cluster. You may view Ceph log files under `/var/log/ceph` (the default location).

Tip: When debug output slows down your system, the latency can hide race conditions.

Logging is resource intensive. If you are encountering a problem in a specific area of your cluster, enable logging for that area of the cluster. For example, if your OSDs are running fine, but your metadata servers are not, you should start by enabling debug logging for the specific metadata server instance(s) giving you trouble. Enable logging for each subsystem as needed.

Important: Verbose logging can generate over 1GB of data per hour. If your OS disk reaches its capacity, the node will stop working.

If you enable or increase the rate of Ceph logging, ensure that you have sufficient disk space on your OS disk. See *Accelerating Log Rotation* for details on rotating log files. When your system is running well, remove unnecessary debugging settings to ensure your cluster runs optimally. Logging debug output messages is relatively slow, and a waste of resources when operating your cluster.

See *Subsystem, Log and Debug Settings* for details on available settings.

Runtime

If you would like to see the configuration settings at runtime, you must log in to a host with a running daemon and execute the following:

```
ceph --admin-daemon {/path/to/admin/socket} config show | less
ceph --admin-daemon /var/run/ceph/ceph-osd.0.asok config show | less
```

To activate Ceph's debugging output (*i.e.*, `dout()`) at runtime, use the `ceph tell` command to inject arguments into the runtime configuration:

```
ceph tell {daemon-type}.{daemon id or *} injectargs --{name} {value} [--{name} {value}]
```

Replace `{daemon-type}` with one of `osd`, `mon` or `mds`. You may apply the runtime setting to all daemons of a particular type with `*`, or specify a specific daemon's ID (i.e., its number or letter). For example, to increase debug logging for a `ceph-osd` daemon named `osd.0`, execute the following:

```
ceph tell osd.0 injectargs --debug-osd 0/5
```

The `ceph tell` command goes through the monitors. If you cannot bind to the monitor, you can still make the change by logging into the host of the daemon whose configuration you'd like to change using `ceph --admin-daemon`. For example:

```
sudo ceph --admin-daemon /var/run/ceph/ceph-osd.0.asok config set debug_osd 0/5
```

See *Subsystem, Log and Debug Settings* for details on available settings.

Boot Time

To activate Ceph's debugging output (*i.e.*, `dout()`) at boot time, you must add settings to your Ceph configuration file. Subsystems common to each daemon may be set under `[global]` in your configuration file. Subsystems for particular daemons are set under the daemon section in your configuration file (*e.g.*, `[mon]`, `[osd]`, `[mds]`). For example:

```
[global]
        debug ms = 1/5

[mon]
        debug mon = 20
        debug paxos = 1/5
        debug auth = 2

[osd]
        debug osd = 1/5
        debug filestore = 1/5
        debug journal = 1
        debug monc = 5/20

[mds]
        debug mds = 1
        debug mds balancer = 1
        debug mds log = 1
        debug mds migrator = 1
```

See *Subsystem, Log and Debug Settings* for details.

Accelerating Log Rotation

If your OS disk is relatively full, you can accelerate log rotation by modifying the Ceph log rotation file at `/etc/logrotate.d/ceph`. Add a size setting after the rotation frequency to accelerate log rotation (via cronjob) if your logs exceed the size setting. For example, the default setting looks like this:

```
rotate 7
weekly
compress
sharedscripts
```

Modify it by adding a `size` setting.

```
rotate 7
weekly
size 500M
compress
sharedscripts
```

Then, start the crontab editor for your user space.

```
crontab -e
```

Finally, add an entry to check the `etc/logrotate.d/ceph` file.

```
30 * * * * /usr/sbin/logrotate /etc/logrotate.d/ceph >/dev/null 2>&1
```

The preceding example checks the `etc/logrotate.d/ceph` file every 30 minutes.

Valgrind

Debugging may also require you to track down memory and threading issues. You can run a single daemon, a type of daemon, or the whole cluster with Valgrind. You should only use Valgrind when developing or debugging Ceph. Valgrind is computationally expensive, and will slow down your system otherwise. Valgrind messages are logged to `stderr`.

Subsystem, Log and Debug Settings

In most cases, you will enable debug logging output via subsystems.

Ceph Subsystems

Each subsystem has a logging level for its output logs, and for its logs in-memory. You may set different values for each of these subsystems by setting a log file level and a memory level for debug logging. Ceph's logging levels operate on a scale of `1` to `20`, where `1` is terse and `20` is verbose. In general, the logs in-memory are not sent to the output log unless:

- a fatal signal is raised or
- an `assert` in source code is triggered or
- upon requested. Please consult document on admin socket for more details.

A debug logging setting can take a single value for the log level and the memory level, which sets them both as the same value. For example, if you specify `debug ms = 5`, Ceph will treat it as a log level and a memory level of `5`. You may also specify them separately. The first setting is the log level, and the second setting is the memory level. You must separate them with a forward slash (/). For example, if you want to set the `ms` subsystem's debug logging level to `1` and its memory level to `5`, you would specify it as `debug ms = 1/5`. For example:

```
debug {subsystem} = {log-level}/{memory-level}
#for example
debug mds log = 1/20
```

The following table provides a list of Ceph subsystems and their default log and memory levels. Once you complete your logging efforts, restore the subsystems to their default level or to a level suitable for normal operations.

Subsystem	Log Level	Memory Level
`default`	0	5
`lockdep`	0	5
`context`	0	5
`crush`	1	5
`mds`	1	5
`mds balancer`	1	5

Continued on next page

Table 4.1 – continued from previous page

Subsystem	Log Level	Memory Level
`mds locker`	1	5
`mds log`	1	5
`mds log expire`	1	5
`mds migrator`	1	5
`buffer`	0	0
`timer`	0	5
`filer`	0	5
`objecter`	0	0
`rados`	0	5
`rbd`	0	5
`journaler`	0	5
`objectcacher`	0	5
`client`	0	5
`osd`	0	5
`optracker`	0	5
`objclass`	0	5
`filestore`	1	5
`journal`	1	5
`ms`	0	5
`mon`	1	5
`monc`	0	5
`paxos`	0	5
`tp`	0	5
`auth`	1	5
`finisher`	1	5
`heartbeatmap`	1	5
`perfcounter`	1	5
`rgw`	1	5
`javaclient`	1	5
`asok`	1	5
`throttle`	1	5

Logging Settings

Logging and debugging settings are not required in a Ceph configuration file, but you may override default settings as needed. Ceph supports the following settings:

`log file`

Description The location of the logging file for your cluster.

Type String

Required No

Default `/var/log/ceph/$cluster-$name.log`

`log max new`

Description The maximum number of new log files.

Type Integer

Required No

Default `1000`

`log max recent`

Description The maximum number of recent events to include in a log file.

Type Integer

Required No

Default `1000000`

`log to stderr`

Description Determines if logging messages should appear in `stderr`.

Type Boolean

Required No

Default `true`

`err to stderr`

Description Determines if error messages should appear in `stderr`.

Type Boolean

Required No

Default `true`

`log to syslog`

Description Determines if logging messages should appear in `syslog`.

Type Boolean

Required No

Default `false`

`err to syslog`

Description Determines if error messages should appear in `syslog`.

Type Boolean

Required No

Default `false`

`log flush on exit`

Description Determines if Ceph should flush the log files after exit.

Type Boolean

Required No

Default `true`

`clog to monitors`

Description Determines if `clog` messages should be sent to monitors.

Type Boolean

Required No

Default `true`

`clog to syslog`

Description Determines if `clog` messages should be sent to syslog.

Type Boolean

Required No

Default `false`

`mon cluster log to syslog`

Description Determines if the cluster log should be output to the syslog.

Type Boolean

Required No

Default `false`

`mon cluster log file`

Description The location of the cluster's log file.

Type String

Required No

Default `/var/log/ceph/$cluster.log`

OSD

`osd debug drop ping probability`

Description ?

Type Double

Required No

Default 0

`osd debug drop ping duration`

Description

Type Integer

Required No

Default 0

`osd debug drop pg create probability`

Description

Type Integer

Required No

Default 0

`osd debug drop pg create duration`

Description ?

Type Double

Required No

Default 1

`osd preserve trimmed log`

Description Preserves trimmed logs after trimming.

Type Boolean

Required No

Default `false`

`osd tmapput sets uses tmap`

Description Uses `tmap`. For debug only.

Type Boolean

Required No

Default `false`

`osd min pg log entries`

Description The minimum number of log entries for placement groups.

Type 32-bit Unsigned Integer

Required No

Default 1000

`osd op log threshold`

Description How many op log messages to show up in one pass.

Type Integer

Required No

Default 5

Filestore

`filestore debug omap check`

Description Debugging check on synchronization. This is an expensive operation.

Type Boolean

Required No

Default 0

MDS

`mds debug scatterstat`

Description Ceph will assert that various recursive stat invariants are true (for developers only).

Type Boolean

Required No

Default `false`

`mds debug frag`

Description Ceph will verify directory fragmentation invariants when convenient (developers only).

Type Boolean

Required No

Default `false`

`mds debug auth pins`

Description The debug auth pin invariants (for developers only).

Type Boolean

Required No

Default `false`

`mds debug subtrees`

Description The debug subtree invariants (for developers only).

Type Boolean

Required No

Default `false`

RADOS Gateway

`rgw log nonexistent bucket`

Description Should we log a non-existent buckets?

Type Boolean

Required No

Default `false`

`rgw log object name`

Description Should an object's name be logged. // man date to see codes (a subset are supported)

Type String

Required No

Default `%Y-%m-%d-%H-%i-%n`

`rgw log object name utc`

Description Object log name contains UTC?

Type Boolean

Required No

Default `false`

`rgw enable ops log`

Description Enables logging of every RGW operation.

Type Boolean

Required No

Default `true`

`rgw enable usage log`

Description Enable logging of RGW's bandwidth usage.

Type Boolean

Required No

Default `true`

`rgw usage log flush threshold`

Description Threshold to flush pending log data.

Type Integer

Required No

Default `1024`

`rgw usage log tick interval`

Description Flush pending log data every `s` seconds.

Type Integer

Required No

Default 30

`rgw intent log object name`

Description

Type String

Required No

Default `%Y-%m-%d-%i-%n`

`rgw intent log object name utc`

Description Include a UTC timestamp in the intent log object name.

Type Boolean

Required No

Default `false`

4.3.19 CPU Profiling

If you built Ceph from source and compiled Ceph for use with oprofile you can profile Ceph's CPU usage. See Installing Oprofile for details.

Initializing oprofile

The first time you use `oprofile` you need to initialize it. Locate the `vmlinux` image corresponding to the kernel you are now running.

```
ls /boot
sudo opcontrol --init
sudo opcontrol --setup --vmlinux={path-to-image} --separate=library --callgraph=6
```

Starting oprofile

To start `oprofile` execute the following command:

```
opcontrol --start
```

Once you start `oprofile`, you may run some tests with Ceph.

Stopping oprofile

To stop `oprofile` execute the following command:

```
opcontrol --stop
```

Retrieving oprofile Results

To retrieve the top `cmon` results, execute the following command:

```
opreport -gal ./cmon | less
```

To retrieve the top `cmon` results with call graphs attached, execute the following command:

```
opreport -cal ./cmon | less
```

Important: After reviewing results, you should reset `oprofile` before running it again. Resetting `oprofile` removes data from the session directory.

Resetting oprofile

To reset `oprofile`, execute the following command:

```
sudo opcontrol --reset
```

Important: You should reset `oprofile` after analyzing data so that you do not commingle results from different tests.

4.3.20 Memory Profiling

Ceph MON, OSD and MDS can generate heap profiles using `tcmalloc`. To generate heap profiles, ensure you have `google-perftools` installed:

```
sudo apt-get google-perftools
```

The profiler dumps output to your `log file` directory (i.e., `/var/log/ceph`). See Logging and Debugging for details. To view the profiler logs with Google's performance tools, execute the following:

```
google-pprof --text {path-to-daemon}  {log-path/filename}
```

For example:

```
$ ceph tell osd.0 heap start_profiler
$ ceph tell osd.0 heap dump
osd.0 tcmalloc heap stats:------------------------------------------------
MALLOC:        2632288 (    2.5 MiB) Bytes in use by application
MALLOC: +       499712 (    0.5 MiB) Bytes in page heap freelist
MALLOC: +       543800 (    0.5 MiB) Bytes in central cache freelist
MALLOC: +       327680 (    0.3 MiB) Bytes in transfer cache freelist
MALLOC: +      1239400 (    1.2 MiB) Bytes in thread cache freelists
MALLOC: +      1142936 (    1.1 MiB) Bytes in malloc metadata
MALLOC:   ------------
MALLOC: =      6385816 (    6.1 MiB) Actual memory used (physical + swap)
MALLOC: +            0 (    0.0 MiB) Bytes released to OS (aka unmapped)
MALLOC:   ------------
MALLOC: =      6385816 (    6.1 MiB) Virtual address space used
MALLOC:
MALLOC:            231              Spans in use
MALLOC:             56              Thread heaps in use
MALLOC:           8192              Tcmalloc page size
------------------------------------------------
Call ReleaseFreeMemory() to release freelist memory to the OS (via madvise()).
Bytes released to the OS take up virtual address space but no physical memory.
$ google-pprof --text \
               /usr/bin/ceph-osd  \
               /var/log/ceph/ceph-osd.0.profile.0001.heap
 Total: 3.7 MB
 1.9  51.1%  51.1%      1.9  51.1% ceph::log::Log::create_entry
 1.8  47.3%  98.4%      1.8  47.3% std::string::_Rep::_S_create
 0.0   0.4%  98.9%      0.0   0.6% SimpleMessenger::add_accept_pipe
 0.0   0.4%  99.2%      0.0   0.6% decode_message
 ...
```

Another heap dump on the same daemon will add another file. It is convenient to compare to a previous heap dump to show what has grown in the interval. For instance:

```
$ google-pprof --text --base out/osd.0.profile.0001.heap \
      ceph-osd out/osd.0.profile.0003.heap
 Total: 0.2 MB
 0.1  50.3%  50.3%      0.1  50.3% ceph::log::Log::create_entry
 0.1  46.6%  96.8%      0.1  46.6% std::string::_Rep::_S_create
 0.0   0.9%  97.7%      0.0  26.1% ReplicatedPG::do_op
 0.0   0.8%  98.5%      0.0   0.8% __gnu_cxx::new_allocator::allocate
```

Refer to Google Heap Profiler for additional details.

Once you have the heap profiler installed, start your cluster and begin using the heap profiler. You may enable or disable the heap profiler at runtime, or ensure that it runs continuously. For the following commandline usage, replace `{daemon-type}` with `mon`, `osd` or `mds`, and replace `{daemon-id}` with the OSD number or the MON or MDS id.

Starting the Profiler

To start the heap profiler, execute the following:

```
ceph tell {daemon-type}.{daemon-id} heap start_profiler
```

For example:

```
ceph tell osd.1 heap start_profiler
```

Alternatively the profile can be started when the daemon starts running if the `CEPH_HEAP_PROFILER_INIT=true` variable is found in the environment.

Printing Stats

To print out statistics, execute the following:

```
ceph  tell {daemon-type}.{daemon-id} heap stats
```

For example:

```
ceph tell osd.0 heap stats
```

Note: Printing stats does not require the profiler to be running and does not dump the heap allocation information to a file.

Dumping Heap Information

To dump heap information, execute the following:

```
ceph tell {daemon-type}.{daemon-id} heap dump
```

For example:

```
ceph tell mds.a heap dump
```

Note: Dumping heap information only works when the profiler is running.

Releasing Memory

To release memory that `tcmalloc` has allocated but which is not being used by the Ceph daemon itself, execute the following:

```
ceph tell {daemon-type}{daemon-id} heap release
```

For example:

```
ceph tell osd.2 heap release
```

Stopping the Profiler

To stop the heap profiler, execute the following:

```
ceph tell {daemon-type}.{daemon-id} heap stop_profiler
```

For example:

```
ceph tell osd.0 heap stop_profiler
```

4.4 Object Store Manpages

4.4.1 ceph-disk – Ceph disk preparation and activation utility for OSD

Synopsis

ceph-disk prepare [–cluster *clustername*] [–cluster-uuid *uuid*] [–fs-type *xfs|ext4|btrfs*] [*data-path*] [*journal-path*]

ceph-disk activate [*data-path*] [–activate-key *path*]

ceph-disk activate-all

ceph-disk list

Description

`ceph-disk` is a utility that can prepare and activate a disk, partition or directory as a Ceph OSD. It is run directly or triggered by **`ceph-deploy`** or `udev`. It can also be triggered by other deployment utilities like `Chef`, `Juju`, `Puppet` etc.

It actually automates the multiple steps involved in manual creation and start of an OSD into two steps of preparing and activating the OSD by using the subcommands `prepare` and `activate`.

Subcommands

prepare

Prepare a directory, disk or drive for a Ceph OSD. It creates a GPT partition, marks the partition with Ceph type `uuid`, creates a file system, marks the file system as ready for Ceph consumption, uses entire partition and adds a new partition to the journal disk. It is run directly or triggered by **`ceph-deploy`**.

Usage:

```
ceph-disk prepare --cluster [cluster-name] --cluster-uuid [uuid] --fs-type
[ext4|xfs|btrfs] [data-path] [journal-path]
```

Other options like *`--osd-uuid`*, *`--journal-uuid`*, *`--zap-disk`*, *`--data-dir`*, *`--data-dev`*, *`--journal-file`*, *`--journal-dev`*, *`--dmcrypt`* and *`--dmcrypt-key-dir`* can also be used with the subcommand.

activate

Activate the Ceph OSD. It mounts the volume in a temporary location, allocates an OSD id (if needed), remounts in the correct location `/var/lib/ceph/osd/$cluster-$id` and starts ceph-osd. It is triggered by `udev` when it sees the OSD GPT partition type or on ceph service start with `ceph disk activate-all`. It is also run directly or triggered by **`ceph-deploy`**.

Usage:

```
ceph-disk activate [PATH]
```

Here, [PATH] is path to a block device or a directory.

An additional option `--activate-key` has to be used with this subcommand when a copy of `/var/lib/ceph/bootstrap-osd/{cluster}.keyring` isn't present in the OSD node.

Usage:

```
ceph-disk activate [PATH] [--activate-key PATH]
```

Another option `--mark-init` can also be used with this subcommand. `--mark-init` provides init system to manage the OSD directory.

activate-journal

Activate an OSD via it's journal device. `udev` triggers `ceph-disk activate-journal <dev>` based on the partition type.

Usage:

```
ceph-disk activate-journal [DEV]
```

Here, [DEV] is the path to a journal block device.

Others options like `--activate-key` and `--mark-init` can also be used with this subcommand.

`--mark-init` provides init system to manage the OSD directory.

Usage:

```
ceph-disk activate-journal [--activate-key PATH] [--mark-init INITSYSTEM] [DEV]
```

activate-all

Activate all tagged OSD partitions. `activate-all` relies on `/dev/disk/by-parttype-uuid/$typeuuid.$uuid` to find all partitions. Special `udev` rules are installed to create these links. It is triggered on ceph service start or run directly.

Usage:

```
ceph-disk activate-all
```

Others options like `--activate-key` and `--mark-init` can also be used with this subcommand.

`--mark-init` provides init system to manage the OSD directory.

Usage:

```
ceph-disk activate-all [--activate-key PATH] [--mark-init INITSYSTEM]
```

list

List disk partitions and Ceph OSDs. It is run directly or triggered by **`ceph-deploy`**.

Usage:

```
ceph-disk list
```

suppress-activate

Suppress activate on a device (prefix). Mark devices that you don't want to activate with a file like `/var/lib/ceph/tmp/suppress-activate.sdb` where the last bit is the sanitized device name (/dev/X without the /dev/ prefix). A function `is_suppressed()` checks for and matches a prefix (/dev/). It means suppressing sdb will stop activate on sdb1, sdb2, etc.

Usage:

```
ceph-disk suppress-activate [PATH]
```

Here, [PATH] is path to a block device or a directory.

unsuppress-activate

Stop suppressing activate on a device (prefix). It is used to activate a device that was earlier kept deactivated using `suppress-activate`.

Usage:

```
ceph-disk unsuppress-activate [PATH]
```

Here, [PATH] is path to a block device or a directory.

zap

Zap/erase/destroy a device's partition table and contents. It actually uses `sgdisk` and it's option `--zap-all` to destroy both GPT and MBR data structures so that the disk becomes suitable for repartitioning. `sgdisk` then uses `--mbrtogpt` to convert the MBR or BSD disklabel disk to a GPT disk. The `prepare` subcommand can now be executed which will create a new GPT partition. It is also run directly or triggered by **ceph-deploy**.

Usage:

```
ceph-disk zap [DEV]
```

Here, [DEV] is path to a block device.

Options

--prepend-to-path `PATH`
Prepend PATH to $PATH for backward compatibility (default `/usr/bin`).

--statedir `PATH`
Directory in which ceph configuration is preserved (default `/usr/lib/ceph`).

--sysconfdir `PATH`
Directory in which ceph configuration files are found (default `/etc/ceph`).

--cluster
Provide name of the ceph cluster in which the OSD is being prepared.

--cluster-uuid
Provide uuid of the ceph cluster in which the OSD is being prepared.

--fs-type
Provide the filesytem type for the OSD. e.g. `xfs/ext4/btrfs`.

--osd-uuid
Unique OSD uuid to assign to the disk.

--journal-uuid
Unique uuid to assign to the journal.

--zap-disk
Destroy the partition table and content of a disk.

--data-dir
Verify that `[data-path]` is of a directory.

--data-dev
Verify that `[data-path]` is of a block device.

--journal-file
Verify that journal is a file.

--journal-dev
Verify that journal is a block device.

--dmcrypt
Encrypt `[data-path]` and/or journal devices with `dm-crypt`.

--dmcrypt-key-dir
Directory where `dm-crypt` keys are stored.

--activate-key
Use when a copy of `/var/lib/ceph/bootstrap-osd/{cluster}.keyring` isn't present in the OSD node. Suffix the option by the path to the keyring.

--mark-init
Provide init system to manage the OSD directory.

Availability

ceph-disk is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

See also

ceph-osd(8), ceph-deploy(8)

4.4.2 ceph – ceph administration tool

Synopsis

ceph auth [*add* | *caps* | *del* | *export* | *get* | *get-key* | *get-or-create* | *get-or-create-key* | *import* | *list* | *print-key* | *print_key*] ...

ceph compact

ceph config-key [*del* | *exists* | *get* | *list* | *put*] ...

ceph daemon <*name*> | <*path*> <*command*> ...

ceph daemonperf <*name*> | <*path*> [*interval* [*count*]]

ceph df *{detail}*

ceph fs [*ls* | *new* | *reset* | *rm*] ...

ceph fsid

ceph health *{detail}*

ceph heap [*dump* | *start_profiler* | *stop_profiler* | *release* | *stats*] ...

ceph injectargs <*injectedargs*> [<*injectedargs*>...]

ceph log <*logtext*> [<*logtext*>...]

ceph mds [*add_data_pool* | *cluster_down* | *cluster_up* | *compat* | *deactivate* | *dump* | *fail* | *getmap* | *newfs* | *remove_data_pool* | *rm* | *rmfailed* | *set* | *set_max_mds* | *set_state* | *setmap* | *stat* | *stop* | *tell*] ...

ceph mon [*add* | *dump* | *getmap* | *remove* | *stat*] ...

ceph mon_status

ceph osd [*blacklist* | *blocked-by* | *create* | *deep-scrub* | *df* | *down* | *dump* | *erasure-code-profile* | *find* | *getcrushmap* | *getmap* | *getmaxosd* | *in* | *lspools* | *map* | *metadata* | *out* | *pause* | *perf* | *pg-temp* | *primary-affinity* | *primary-temp* | *repair* | *reweight* | *reweight-by-pg* | *rm* | *scrub* | *set* | *setcrushmap* | *setmaxosd* | *stat* | *thrash* | *tree* | *unpause* | *unset*] ...

ceph osd crush [*add* | *add-bucket* | *create-or-move* | *dump* | *get-tunable* | *link* | *move* | *remove* | *rename-bucket* | *reweight* | *reweight-all* | *reweight-subtree* | *rm* | *rule* | *set* | *set-tunable* | *show-tunables* | *tunables* | *unlink*] ...

ceph osd pool [*create* | *delete* | *get* | *get-quota* | *ls* | *mksnap* | *rename* | *rmsnap* | *set* | *set-quota* | *stats*] ...

ceph osd tier [*add* | *add-cache* | *cache-mode* | *remove* | *remove-overlay* | *set-overlay*] ...

ceph pg [*debug* | *deep-scrub* | *dump* | *dump_json* | *dump_pools_json* | *dump_stuck* | *force_create_pg* | *getmap* | *ls* | *ls-by-osd* | *ls-by-pool* | *ls-by-primary* | *map* | *repair* | *scrub* | *send_pg_creates* | *set_full_ratio* | *set_nearfull_ratio* | *stat*] ...

ceph quorum [*enter* | *exit*]

ceph quorum_status

ceph report { *<tags>* [*<tags>...*] }

ceph scrub

ceph status

ceph sync force {–yes-i-really-mean-it} {–i-know-what-i-am-doing}

ceph tell *<name (type.id)> <args> [<args>...]*

ceph version

Description

ceph is a control utility which is used for manual deployment and maintenance of a Ceph cluster. It provides a diverse set of commands that allows deployment of monitors, OSDs, placement groups, MDS and overall maintenance, administration of the cluster.

Commands

auth

Manage authentication keys. It is used for adding, removing, exporting or updating of authentication keys for a particular entity such as a monitor or OSD. It uses some additional subcommands.

Subcommand `add` adds authentication info for a particular entity from input file, or random key if no input is given and/or any caps specified in the command.

Usage:

```
ceph auth add <entity> {<caps> [<caps>...]}
```

Subcommand `caps` updates caps for **name** from caps specified in the command.

Usage:

```
ceph auth caps <entity> <caps> [<caps>...]
```

Subcommand `del` deletes all caps for `name`.

Usage:

```
ceph auth del <entity>
```

Subcommand `export` writes keyring for requested entity, or master keyring if none given.

Usage:

```
ceph auth export {<entity>}
```

Subcommand `get` writes keyring file with requested key.

Usage:

```
ceph auth get <entity>
```

Subcommand `get-key` displays requested key.

Usage:

```
ceph auth get-key <entity>
```

Subcommand `get-or-create` adds authentication info for a particular entity from input file, or random key if no input given and/or any caps specified in the command.

Usage:

```
ceph auth get-or-create <entity> {<caps> [<caps>...]}
```

Subcommand `get-or-create-key` gets or adds key for `name` from system/caps pairs specified in the command. If key already exists, any given caps must match the existing caps for that key.

Usage:

```
ceph auth get-or-create-key <entity> {<caps> [<caps>...]}
```

Subcommand `import` reads keyring from input file.

Usage:

```
ceph auth import
```

Subcommand `list` lists authentication state.

Usage:

```
ceph auth list
```

Subcommand `print-key` displays requested key.

Usage:

```
ceph auth print-key <entity>
```

Subcommand `print_key` displays requested key.

Usage:

```
ceph auth print_key <entity>
```

compact

Causes compaction of monitor's leveldb storage.

Usage:

```
ceph compact
```

config-key

Manage configuration key. It uses some additional subcommands.

Subcommand `del` deletes configuration key.

Usage:

```
ceph config-key del <key>
```

Subcommand `exists` checks for configuration keys existence.

Usage:

```
ceph config-key exists <key>
```

Subcommand `get` gets the configuration key.

Usage:

```
ceph config-key get <key>
```

Subcommand `list` lists configuration keys.

Usage:

```
ceph config-key list
```

Subcommand `put` puts configuration key and values.

Usage:

```
ceph config-key put <key> {<val>}
```

daemon

Submit admin-socket commands.

Usage:

```
ceph daemon {daemon_name|socket_path} {command} ...
```

Example:

```
ceph daemon osd.0 help
```

daemonperf

Watch performance counters from a Ceph daemon.

Usage:

```
ceph daemonperf {daemon_name|socket_path} [{interval} [{count}]]
```

df

Show cluster's free space status.

Usage:

```
ceph df {detail}
```

fs

Manage cephfs filesystems. It uses some additional subcommands.

Subcommand `ls` to list filesystems

Usage:

```
ceph fs ls
```

Subcommand `new` to make a new filesystem using named pools <metadata> and <data>

Usage:

```
ceph fs new <fs_name> <metadata> <data>
```

Subcommand `reset` is used for disaster recovery only: reset to a single-MDS map

Usage:

```
ceph fs reset <fs_name> {--yes-i-really-mean-it}
```

Subcommand `rm` to disable the named filesystem

Usage:

```
ceph fs rm <fs_name> {--yes-i-really-mean-it}
```

fsid

Show cluster's FSID/UUID.

Usage:

```
ceph fsid
```

health

Show cluster's health.

Usage:

```
ceph health {detail}
```

heap

Show heap usage info (available only if compiled with tcmalloc)

Usage:

```
ceph heap dump|start_profiler|stop_profiler|release|stats
```

injectargs

Inject configuration arguments into monitor.

Usage:

```
ceph injectargs <injected_args> [<injected_args>...]
```

log

Log supplied text to the monitor log.

Usage:

```
ceph log <logtext> [<logtext>...]
```

mds

Manage metadata server configuration and administration. It uses some additional subcommands.

Subcommand `add_data_pool` adds data pool.

Usage:

```
ceph mds add_data_pool <pool>
```

Subcommand `cluster_down` takes mds cluster down.

Usage:

```
ceph mds cluster_down
```

Subcommand `cluster_up` brings mds cluster up.

Usage:

```
ceph mds cluster_up
```

Subcommand `compat` manages compatible features. It uses some additional subcommands.

Subcommand `rm_compat` removes compatible feature.

Usage:

```
ceph mds compat rm_compat <int[0-]>
```

Subcommand `rm_incompat` removes incompatible feature.

Usage:

```
ceph mds compat rm_incompat <int[0-]>
```

Subcommand `show` shows mds compatibility settings.

Usage:

```
ceph mds compat show
```

Subcommand `deactivate` stops mds.

Usage:

```
ceph mds deactivate <who>
```

Subcommand `dump` dumps information, optionally from epoch.

Usage:

```
ceph mds dump {<int[0-]>}
```

Subcommand `fail` forces mds to status fail.

Usage:

```
ceph mds fail <who>
```

Subcommand `getmap` gets MDS map, optionally from epoch.

Usage:

```
ceph mds getmap {<int[0-]>}
```

Subcommand `newfs` makes new filesystem using pools <metadata> and <data>.

Usage:

```
ceph mds newfs <int[0-]> <int[0-]> {--yes-i-really-mean-it}
```

Subcommand `remove_data_pool` removes data pool.

Usage:

```
ceph mds remove_data_pool <pool>
```

Subcommand `rm` removes inactive mds.

Usage:

```
ceph mds rm <int[0-]> <name> (type.id)>
```

Subcommand `rmfailed` removes failed mds.

Usage:

```
ceph mds rmfailed <int[0-]>
```

Subcommand `set` set mds parameter <var> to <val>

Usage:

```
ceph mds set max_mds|max_file_size|allow_new_snaps|inline_data <va> {<confirm>}
```

Subcommand `set_max_mds` sets max MDS index.

Usage:

```
ceph mds set_max_mds <int[0-]>
```

Subcommand `set_state` sets mds state of <gid> to <numeric-state>.

Usage:

```
ceph mds set_state <int[0-]> <int[0-20]>
```

Subcommand `setmap` sets mds map; must supply correct epoch number.

Usage:

```
ceph mds setmap <int[0-]>
```

Subcommand `stat` shows MDS status.

Usage:

```
ceph mds stat
```

Subcommand `stop` stops mds.

Usage:

```
ceph mds stop <who>
```

Subcommand `tell` sends command to particular mds.

Usage:

```
ceph mds tell <who> <args> [<args>...]
```

mon

Manage monitor configuration and administration. It uses some additional subcommands.

Subcommand `add` adds new monitor named <name> at <addr>.

Usage:

```
ceph mon add <name> <IPaddr[:port]>
```

Subcommand `dump` dumps formatted monmap (optionally from epoch)

Usage:

```
ceph mon dump {<int[0-]>}
```

Subcommand `getmap` gets monmap.

Usage:

```
ceph mon getmap {<int[0-]>}
```

Subcommand `remove` removes monitor named <name>.

Usage:

```
ceph mon remove <name>
```

Subcommand `stat` summarizes monitor status.

Usage:

```
ceph mon stat
```

mon_status

Reports status of monitors.

Usage:

```
ceph mon_status
```

osd

Manage OSD configuration and administration. It uses some additional subcommands.

Subcommand `blacklist` manage blacklisted clients. It uses some additional subcommands.

Subcommand `add` add <addr> to blacklist (optionally until <expire> seconds from now)

Usage:

```
ceph osd blacklist add <EntityAddr> {<float[0.0-]>}
```

Subcommand `ls` show blacklisted clients

Usage:

```
ceph osd blacklist ls
```

Subcommand `rm` remove <addr> from blacklist

Usage:

```
ceph osd blacklist rm <EntityAddr>
```

Subcommand `blocked-by` prints a histogram of which OSDs are blocking their peers

Usage:

```
ceph osd blocked-by
```

Subcommand `create` creates new osd (with optional UUID).

Usage:

```
ceph osd create {<uuid>}
```

Subcommand `crush` is used for CRUSH management. It uses some additional subcommands.

Subcommand `add` adds or updates crushmap position and weight for <name> with <weight> and location <args>.

Usage:

```
ceph osd crush add <osdname (id|osd.id)> <float[0.0-]> <args> [<args>...]
```

Subcommand `add-bucket` adds no-parent (probably root) crush bucket <name> of type <type>.

Usage:

```
ceph osd crush add-bucket <name> <type>
```

Subcommand `create-or-move` creates entry or moves existing entry for <name> <weight> at/to location <args>.

Usage:

```
ceph osd crush create-or-move <osdname (id|osd.id)> <float[0.0-]> <args>
[<args>...]
```

Subcommand `dump` dumps crush map.

Usage:

```
ceph osd crush dump
```

Subcommand `get-tunable` get crush tunable straw_calc_version

Usage:

```
ceph osd crush get-tunable straw_calc_version
```

Subcommand `link` links existing entry for <name> under location <args>.

Usage:

```
ceph osd crush link <name> <args> [<args>...]
```

Subcommand `move` moves existing entry for <name> to location <args>.

Usage:

```
ceph osd crush move <name> <args> [<args>...]
```

Subcommand `remove` removes <name> from crush map (everywhere, or just at <ancestor>).

Usage:

```
ceph osd crush remove <name> {<ancestor>}
```

Subcommand `rename-bucket` renames buchket <srcname> to <stname>

Usage:

```
ceph osd crush rename-bucket <srcname> <dstname>
```

Subcommand `reweight` change <name>'s weight to <weight> in crush map.

Usage:

```
ceph osd crush reweight <name> <float[0.0-]>
```

Subcommand `reweight-all` recalculate the weights for the tree to ensure they sum correctly

Usage:

```
ceph osd crush reweight-all
```

Subcommand `reweight-subtree` changes all leaf items beneath <name> to <weight> in crush map

Usage:

```
ceph osd crush reweight-subtree <name> <weight>
```

Subcommand `rm` removes <name> from crush map (everywhere, or just at <ancestor>).

Usage:

```
ceph osd crush rm <name> {<ancestor>}
```

Subcommand `rule` is used for creating crush rules. It uses some additional subcommands.

Subcommand `create-erasure` creates crush rule <name> for erasure coded pool created with <profile> (default default).

Usage:

```
ceph osd crush rule create-erasure <name> {<profile>}
```

Subcommand `create-simple` creates crush rule <name> to start from <root>, replicate across buckets of type <type>, using a choose mode of <firstn|indep> (default firstn; indep best for erasure pools).

Usage:

```
ceph osd crush rule create-simple <name> <root> <type> {firstn|indep}
```

Subcommand `dump` dumps crush rule <name> (default all).

Usage:

```
ceph osd crush rule dump {<name>}
```

Subcommand `list` lists crush rules.

Usage:

```
ceph osd crush rule list
```

Subcommand `ls` lists crush rules.

Usage:

```
ceph osd crush rule ls
```

Subcommand `rm` removes crush rule <name>.

Usage:

```
ceph osd crush rule rm <name>
```

Subcommand `set` used alone, sets crush map from input file.

Usage:

```
ceph osd crush set
```

Subcommand `set` with osdname/osd.id update crushmap position and weight for <name> to <weight> with location <args>.

Usage:

```
ceph osd crush set <osdname (id|osd.id)> <float[0.0-]> <args> [<args>...]
```

Subcommand `set-tunable` set crush tunable <tunable> to <value>. The only tunable that can be set is straw_calc_version.

Usage:

```
ceph osd crush set-tunable straw_calc_version <value>
```

Subcommand `show-tunables` shows current crush tunables.

Usage:

```
ceph osd crush show-tunables
```

Subcommand `tunables` sets crush tunables values to <profile>.

Usage:

```
ceph osd crush tunables legacy|argonaut|bobtail|firefly|hammer|optimal|default
```

Subcommand `unlink` unlinks <name> from crush map (everywhere, or just at <ancestor>).

Usage:

```
ceph osd crush unlink <name> {<ancestor>}
```

Subcommand `df` shows OSD utilization

Usage:

```
ceph osd df {plain|tree}
```

Subcommand `deep-scrub` initiates deep scrub on specified osd.

Usage:

```
ceph osd deep-scrub <who>
```

Subcommand `down` sets osd(s) <id> [<id>...] down.

Usage:

```
ceph osd down <ids> [<ids>...]
```

Subcommand `dump` prints summary of OSD map.

Usage:

```
ceph osd dump {<int[0-]>}
```

Subcommand `erasure-code-profile` is used for managing the erasure code profiles. It uses some additional subcommands.

Subcommand `get` gets erasure code profile <name>.

Usage:

```
ceph osd erasure-code-profile get <name>
```

Subcommand `ls` lists all erasure code profiles.

Usage:

```
ceph osd erasure-code-profile ls
```

Subcommand `rm` removes erasure code profile <name>.

Usage:

```
ceph osd erasure-code-profile rm <name>
```

Subcommand `set` creates erasure code profile <name> with [<key[=value]> ...] pairs. Add a –force at the end to override an existing profile (IT IS RISKY).

Usage:

```
ceph osd erasure-code-profile set <name> {<profile> [<profile>...]}
```

Subcommand `find` find osd <id> in the CRUSH map and shows its location.

Usage:

```
ceph osd find <int[0-]>
```

Subcommand `getcrushmap` gets CRUSH map.

Usage:

```
ceph osd getcrushmap {<int[0-]>}
```

Subcommand `getmap` gets OSD map.

Usage:

```
ceph osd getmap {<int[0-]>}
```

Subcommand `getmaxosd` shows largest OSD id.

Usage:

```
ceph osd getmaxosd
```

Subcommand `in` sets osd(s) <id> [<id>...] in.

Usage:

```
ceph osd in <ids> [<ids>...]
```

Subcommand `lost` marks osd as permanently lost. THIS DESTROYS DATA IF NO MORE REPLICAS EXIST, BE CAREFUL.

Usage:

```
ceph osd lost <int[0-]> {--yes-i-really-mean-it}
```

Subcommand `ls` shows all OSD ids.

Usage:

```
ceph osd ls {<int[0-]>}
```

Subcommand `lspools` lists pools.

Usage:

```
ceph osd lspools {<int>}
```

Subcommand `map` finds pg for <object> in <pool>.

Usage:

```
ceph osd map <poolname> <objectname>
```

Subcommand `metadata` fetches metadata for osd <id>.

Usage:

```
ceph osd metadata <int[0-]>
```

Subcommand `out` sets osd(s) <id> [<id>...] out.

Usage:

```
ceph osd out <ids> [<ids>...]
```

Subcommand `pause` pauses osd.

Usage:

```
ceph osd pause
```

Subcommand `perf` prints dump of OSD perf summary stats.

Usage:

```
ceph osd perf
```

Subcommand `pg-temp` set pg_temp mapping pgid:[<id> [<id>...]] (developers only).

Usage:

```
ceph osd pg-temp <pgid> {<id> [<id>...]}
```

Subcommand `pool` is used for managing data pools. It uses some additional subcommands.

Subcommand `create` creates pool.

Usage:

```
ceph osd pool create <poolname> <int[0-]> {<int[0-]>} {replicated|erasure}
{<erasure_code_profile>} {<ruleset>} {<int>}
```

Subcommand `delete` deletes pool.

Usage:

```
ceph osd pool delete <poolname> {<poolname>} {--yes-i-really-really-mean-it}
```

Subcommand `get` gets pool parameter <var>.

Usage:

```
ceph osd pool get <poolname> size|min_size|crash_replay_interval|pg_num|
pgp_num|crush_ruleset|auid|write_fadvise_dontneed
```

Only for tiered pools:

```
ceph osd pool get <poolname> hit_set_type|hit_set_period|hit_set_count|hit_set_fpp|
target_max_objects|target_max_bytes|cache_target_dirty_ratio|
cache_target_full_ratio|cache_min_flush_age|cache_min_evict_age|
min_read_recency_for_promote
```

Only for erasure coded pools:

```
ceph osd pool get <poolname> erasure_code_profile
```

Use `all` to get all pool parameters that apply to the pool's type:

```
ceph osd pool get <poolname> all
```

Subcommand `get-quota` obtains object or byte limits for pool.

Usage:

```
ceph osd pool get-quota <poolname>
```

Subcommand `ls` list pools

Usage:

```
ceph osd pool ls {detail}
```

Subcommand `mksnap` makes snapshot <snap> in <pool>.

Usage:

```
ceph osd pool mksnap <poolname> <snap>
```

Subcommand `rename` renames <srcpool> to <destpool>.

Usage:

```
ceph osd pool rename <poolname> <poolname>
```

Subcommand `rmsnap` removes snapshot <snap> from <pool>.

Usage:

```
ceph osd pool rmsnap <poolname> <snap>
```

Subcommand `set` sets pool parameter <var> to <val>.

Usage:

```
ceph osd pool set <poolname> size|min_size|crash_replay_interval|pg_num|
pgp_num|crush_ruleset|hashpspool|nodelete|nopgchange|nosizechange|
hit_set_type|hit_set_period|hit_set_count|hit_set_fpp|debug_fake_ec_pool|
target_max_bytes|target_max_objects|cache_target_dirty_ratio|
cache_target_full_ratio|cache_min_flush_age|cache_min_evict_age|auid|
min_read_recency_for_promote|write_fadvise_dontneed
<val> {--yes-i-really-mean-it}
```

Subcommand `set-quota` sets object or byte limit on pool.

Usage:

```
ceph osd pool set-quota <poolname> max_objects|max_bytes <val>
```

Subcommand `stats` obtain stats from all pools, or from specified pool.

Usage:

```
ceph osd pool stats {<name>}
```

Subcommand `primary-affinity` adjust osd primary-affinity from 0.0 <=<weight> <= 1.0

Usage:

```
ceph osd primary-affinity <osdname (id|osd.id)> <float[0.0-1.0]>
```

Subcommand `primary-temp` sets primary_temp mapping pgid:<id>|-1 (developers only).

Usage:

```
ceph osd primary-temp <pgid> <id>
```

Subcommand `repair` initiates repair on a specified osd.

Usage:

```
ceph osd repair <who>
```

Subcommand `reweight` reweights osd to 0.0 < <weight> < 1.0.

Usage:

```
osd reweight <int[0-]> <float[0.0-1.0]>
```

Subcommand `reweight-by-pg` reweight OSDs by PG distribution [overload-percentage-for-consideration, default 120].

Usage:

```
ceph osd reweight-by-pg {<int[100-]>} {<poolname> [<poolname...]}
```

Subcommand `reweight-by-utilization` reweight OSDs by utilization [overload-percentage-for-consideration, default 120].

Usage:

```
ceph osd reweight-by-utilization {<int[100-]>}
```

Subcommand `rm` removes osd(s) <id> [<id>...] in the cluster.

Usage:

```
ceph osd rm <ids> [<ids>...]
```

Subcommand `scrub` initiates scrub on specified osd.

Usage:

```
ceph osd scrub <who>
```

Subcommand `set` sets <key>.

Usage:

```
ceph osd set full|pause|noup|nodown|noout|noin|nobackfill|
norebalance|norecover|noscrub|nodeep-scrub|notieragent
```

Subcommand `setcrushmap` sets crush map from input file.

Usage:

```
ceph osd setcrushmap
```

Subcommand `setmaxosd` sets new maximum osd value.

Usage:

```
ceph osd setmaxosd <int[0-]>
```

Subcommand `stat` prints summary of OSD map.

Usage:

```
ceph osd stat
```

Subcommand `thrash` thrashes OSDs for <num_epochs>.

Usage:

```
ceph osd thrash <int[0-]>
```

Subcommand `tier` is used for managing tiers. It uses some additional subcommands.

Subcommand `add` adds the tier <tierpool> (the second one) to base pool <pool> (the first one).

Usage:

```
ceph osd tier add <poolname> <poolname> {--force-nonempty}
```

Subcommand `add-cache` adds a cache <tierpool> (the second one) of size <size> to existing pool <pool> (the first one).

Usage:

```
ceph osd tier add-cache <poolname> <poolname> <int[0-]>
```

Subcommand `cache-mode` specifies the caching mode for cache tier <pool>.

Usage:

```
ceph osd tier cache-mode <poolname> none|writeback|forward|readonly|
readforward|readproxy
```

Subcommand `remove` removes the tier <tierpool> (the second one) from base pool <pool> (the first one).

Usage:

```
ceph osd tier remove <poolname> <poolname>
```

Subcommand `remove-overlay` removes the overlay pool for base pool <pool>.

Usage:

```
ceph osd tier remove-overlay <poolname>
```

Subcommand `set-overlay` set the overlay pool for base pool <pool> to be <overlaypool>.

Usage:

```
ceph osd tier set-overlay <poolname> <poolname>
```

Subcommand `tree` prints OSD tree.

Usage:

```
ceph osd tree {<int[0-]>}
```

Subcommand `unpause` unpauses osd.

Usage:

```
ceph osd unpause
```

Subcommand `unset` unsets <key>.

Usage:

```
ceph osd unset full|pause|noup|nodown|noout|noin|nobackfill|
norebalance|norecover|noscrub|nodeep-scrub|notieragent
```

pg

It is used for managing the placement groups in OSDs. It uses some additional subcommands.

Subcommand `debug` shows debug info about pgs.

Usage:

```
ceph pg debug unfound_objects_exist|degraded_pgs_exist
```

Subcommand `deep-scrub` starts deep-scrub on <pgid>.

Usage:

```
ceph pg deep-scrub <pgid>
```

Subcommand `dump` shows human-readable versions of pg map (only 'all' valid with plain).

Usage:

```
ceph pg dump {all|summary|sum|delta|pools|osds|pgs|pgs_brief} [{all|summary|sum|delta|pools|osds|pgs
```

Subcommand `dump_json` shows human-readable version of pg map in json only.

Usage:

```
ceph pg dump_json {all|summary|sum|delta|pools|osds|pgs|pgs_brief} [{all|summary|sum|delta|pools|osds
```

Subcommand `dump_pools_json` shows pg pools info in json only.

Usage:

```
ceph pg dump_pools_json
```

Subcommand `dump_stuck` shows information about stuck pgs.

Usage:

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

Subcommand `force_create_pg` forces creation of pg <pgid>.

Usage:

```
ceph pg force_create_pg <pgid>
```

Subcommand `getmap` gets binary pg map to -o/stdout.

Usage:

```
ceph pg getmap
```

Subcommand `ls` lists pg with specific pool, osd, state

Usage:

```
ceph pg ls {<int>} {active|clean|down|replay|splitting|
scrubbing|scrubq|degraded|inconsistent|peering|repair|
recovery|backfill_wait|incomplete|stale| remapped|
deep_scrub|backfill|backfill_toofull|recovery_wait|
undersized [active|clean|down|replay|splitting|
scrubbing|scrubq|degraded|inconsistent|peering|repair|
recovery|backfill_wait|incomplete|stale|remapped|
deep_scrub|backfill|backfill_toofull|recovery_wait|
undersized...]}
```

Subcommand `ls-by-osd` lists pg on osd [osd]

Usage:

```
ceph pg ls-by-osd <osdname (id|osd.id)> {<int>}
{active|clean|down|replay|splitting|
scrubbing|scrubq|degraded|inconsistent|peering|repair|
recovery|backfill_wait|incomplete|stale| remapped|
deep_scrub|backfill|backfill_toofull|recovery_wait|
undersized [active|clean|down|replay|splitting|
scrubbing|scrubq|degraded|inconsistent|peering|repair|
recovery|backfill_wait|incomplete|stale|remapped|
deep_scrub|backfill|backfill_toofull|recovery_wait|
undersized...]}
```

Subcommand `ls-by-pool` lists pg with pool = [poolname | poolid]

Usage:

```
ceph pg ls-by-pool <poolstr> {<int>} {active|
clean|down|replay|splitting|
scrubbing|scrubq|degraded|inconsistent|peering|repair|
recovery|backfill_wait|incomplete|stale| remapped|
deep_scrub|backfill|backfill_toofull|recovery_wait|
undersized [active|clean|down|replay|splitting|
scrubbing|scrubq|degraded|inconsistent|peering|repair|
recovery|backfill_wait|incomplete|stale|remapped|
deep_scrub|backfill|backfill_toofull|recovery_wait|
undersized...]}
```

Subcommand `ls-by-primary` lists pg with primary = [osd]

Usage:

```
ceph pg ls-by-primary <osdname (id|osd.id)> {<int>}
{active|clean|down|replay|splitting|
scrubbing|scrubq|degraded|inconsistent|peering|repair|
recovery|backfill_wait|incomplete|stale| remapped|
deep_scrub|backfill|backfill_toofull|recovery_wait|
undersized [active|clean|down|replay|splitting|
scrubbing|scrubq|degraded|inconsistent|peering|repair|
recovery|backfill_wait|incomplete|stale|remapped|
deep_scrub|backfill|backfill_toofull|recovery_wait|
undersized...]}
```

Subcommand `map` shows mapping of pg to osds.

Usage:

```
ceph pg map <pgid>
```

Subcommand `repair` starts repair on <pgid>.

Usage:

```
ceph pg repair <pgid>
```

Subcommand `scrub` starts scrub on <pgid>.

Usage:

```
ceph pg scrub <pgid>
```

Subcommand `send_pg_creates` triggers pg creates to be issued.

Usage:

```
ceph pg send_pg_creates
```

Subcommand `set_full_ratio` sets ratio at which pgs are considered full.

Usage:

```
ceph pg set_full_ratio <float[0.0-1.0]>
```

Subcommand `set_nearfull_ratio` sets ratio at which pgs are considered nearly full.

Usage:

```
ceph pg set_nearfull_ratio <float[0.0-1.0]>
```

Subcommand `stat` shows placement group status.

Usage:

```
ceph pg stat
```

quorum

Enter or exit quorum.

Usage:

```
ceph quorum enter|exit
```

quorum_status

Reports status of monitor quorum.

Usage:

```
ceph quorum_status
```

report

Reports full status of cluster, optional title tag strings.

Usage:

```
ceph report {<tags> [<tags>...]}
```

scrub

Scrubs the monitor stores.

Usage:

```
ceph scrub
```

status

Shows cluster status.

Usage:

```
ceph status
```

sync force

Forces sync of and clear monitor store.

Usage:

```
ceph sync force {--yes-i-really-mean-it} {--i-know-what-i-am-doing}
```

tell

Sends a command to a specific daemon.

Usage:

```
ceph tell <name (type.id)> <args> [<args>...]
```

version

Show mon daemon version

Usage:

```
ceph version
```

Options

-i `infile`
will specify an input file to be passed along as a payload with the command to the monitor cluster. This is only used for specific monitor commands.

-o `outfile`
will write any payload returned by the monitor cluster with its reply to outfile. Only specific monitor commands (e.g. osd getmap) return a payload.

-c ceph.conf, **--conf**=ceph.conf
: Use ceph.conf configuration file instead of the default `/etc/ceph/ceph.conf` to determine monitor addresses during startup.

--id CLIENT_ID, **--user** CLIENT_ID
: Client id for authentication.

--name CLIENT_NAME, **-n** CLIENT_NAME
: Client name for authentication.

--cluster CLUSTER
: Name of the Ceph cluster.

--admin-daemon ADMIN_SOCKET
: Submit admin-socket commands.

--admin-socket ADMIN_SOCKET_NOPE
: You probably mean –admin-daemon

-s, --status
: Show cluster status.

-w, --watch
: Watch live cluster changes.

--watch-debug
: Watch debug events.

--watch-info
: Watch info events.

--watch-sec
: Watch security events.

--watch-warn
: Watch warning events.

--watch-error
: Watch error events.

--version, -v
: Display version.

--verbose
: Make verbose.

--concise
: Make less verbose.

-f {json,json-pretty,xml,xml-pretty,plain}, **--format**
: Format of output.

--connect-timeout CLUSTER_TIMEOUT
: Set a timeout for connecting to the cluster.

Availability

ceph is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

See also

ceph-mon(8), ceph-osd(8), ceph-mds(8)

4.4.3 ceph-deploy – Ceph deployment tool

Synopsis

ceph-deploy new [*initial-monitor-node(s)*]

ceph-deploy install [*ceph-node*] [*ceph-node*...]

ceph-deploy mon *create-initial*

ceph-deploy osd *prepare* [*ceph-node*]:[*dir-path*]

ceph-deploy osd *activate* [*ceph-node*]:[*dir-path*]

ceph-deploy osd *create* [*ceph-node*]:[*dir-path*]

ceph-deploy admin [*admin-node*][*ceph-node*...]

ceph-deploy purgedata [*ceph-node*][*ceph-node*...]

ceph-deploy forgetkeys

Description

`ceph-deploy` is a tool which allows easy and quick deployment of a Ceph cluster without involving complex and detailed manual configuration. It uses ssh to gain access to other Ceph nodes from the admin node, sudo for administrator privileges on them and the underlying Python scripts automates the manual process of Ceph installation on each node from the admin node itself. It can be easily run on an workstation and doesn't require servers, databases or any other automated tools. With **`ceph-deploy`**, it is really easy to set up and take down a cluster. However, it is not a generic deployment tool. It is a specific tool which is designed for those who want to get Ceph up and running quickly with only the unavoidable initial configuration settings and without the overhead of installing other tools like `Chef`, `Puppet` or `Juju`. Those who want to customize security settings, partitions or directory locations and want to set up a cluster following detailed manual steps, should use other tools i.e, `Chef`, `Puppet`, `Juju` or `Crowbar`.

With **`ceph-deploy`**, you can install Ceph packages on remote nodes, create a cluster, add monitors, gather/forget keys, add OSDs and metadata servers, configure admin hosts or take down the cluster.

Commands

new

Start deploying a new cluster and write a configuration file and keyring for it. It tries to copy ssh keys from admin node to gain passwordless ssh to monitor node(s), validates host IP, creates a cluster with a new initial monitor node or nodes for monitor quorum, a ceph configuration file, a monitor secret keyring and a log file for the new cluster. It populates the newly created Ceph configuration file with `fsid` of cluster, hostnames and IP addresses of initial monitor members under `[global]` section.

Usage:

```
ceph-deploy new [MON][MON...]
```

Here, [MON] is initial monitor hostname (short hostname i.e, `hostname -s`).

Other options like *`--no-ssh-copykey`*, *`--fsid`*, *`--cluster-network`* and *`--public-network`* can also be used with this command.

If more than one network interface is used, `public network` setting has to be added under `[global]` section of Ceph configuration file. If the public subnet is given, `new` command will choose the one IP from the remote host that exists within the subnet range. Public network can also be added at runtime using *`--public-network`* option with the command as mentioned above.

install

Install Ceph packages on remote hosts. As a first step it installs `yum-plugin-priorities` in admin and other nodes using passwordless ssh and sudo so that Ceph packages from upstream repository get more priority. It then detects the platform and distribution for the hosts and installs Ceph normally by downloading distro compatible packages if adequate repo for Ceph is already added. `--release` flag is used to get the latest release for installation. During detection of platform and distribution before installation, if it finds the `distro.init` to be `sysvinit` (Fedora, CentOS/RHEL etc), it doesn't allow installation with custom cluster name and uses the default name `ceph` for the cluster.

If the user explicitly specifies a custom repo url with *`--repo-url`* for installation, anything detected from the configuration will be overridden and the custom repository location will be used for installation of Ceph packages. If required, valid custom repositories are also detected and installed. In case of installation from a custom repo a boolean is used to determine the logic needed to proceed with a custom repo installation. A custom repo install helper is used that goes through config checks to retrieve repos (and any extra repos defined) and installs them. `cd_conf` is the object built from `argparse` that holds the flags and information needed to determine what metadata from the configuration is to be used.

A user can also opt to install only the repository without installing Ceph and its dependencies by using *`--repo`* option.

Usage:

```
ceph-deploy install [HOST][HOST...]
```

Here, [HOST] is/are the host node(s) where Ceph is to be installed.

An option `--release` is used to install a release known as CODENAME (default: firefly).

Other options like *`--testing`*, *`--dev`*, *`--adjust-repos`*, *`--no-adjust-repos`*, *`--repo`*, *`--local-mirror`*, *`--repo-url`* and *`--gpg-url`* can also be used with this command.

mds

Deploy Ceph mds on remote hosts. A metadata server is needed to use CephFS and the `mds` command is used to create one on the desired host node. It uses the subcommand `create` to do so. `create` first gets the hostname and distro information of the desired mds host. It then tries to read the `bootstrap-mds` key for the cluster and deploy it in the desired host. The key generally has a format of `{cluster}.bootstrap-mds.keyring`. If it doesn't finds a keyring, it runs `gatherkeys` to get the keyring. It then creates a mds on the desired host under the path `/var/lib/ceph/mds/` in `/var/lib/ceph/mds/{cluster}-{name}` format and a bootstrap keyring under `/var/lib/ceph/bootstrap-mds/` in `/var/lib/ceph/bootstrap-mds/{cluster}.keyring` format. It then runs appropriate commands based on `distro.init` to start the `mds`. To remove the mds, subcommand `destroy` is used.

Usage:

```
ceph-deploy mds create [HOST[:DAEMON-NAME]] [HOST[:DAEMON-NAME]...]

ceph-deploy mds destroy [HOST[:DAEMON-NAME]] [HOST[:DAEMON-NAME]...]
```

The [DAEMON-NAME] is optional.

mon

Deploy Ceph monitor on remote hosts. `mon` makes use of certain subcommands to deploy Ceph monitors on other nodes.

Subcommand `create-initial` deploys for monitors defined in `mon initial members` under `[global]` section in Ceph configuration file, wait until they form quorum and then gatherkeys, reporting the monitor status along the process. If monitors don't form quorum the command will eventually time out.

Usage:

```
ceph-deploy mon create-initial
```

Subcommand `create` is used to deploy Ceph monitors by explicitly specifying the hosts which are desired to be made monitors. If no hosts are specified it will default to use the `mon initial members` defined under `[global]` section of Ceph configuration file. `create` first detects platform and distro for desired hosts and checks if hostname is compatible for deployment. It then uses the monitor keyring initially created using `new` command and deploys the monitor in desired host. If multiple hosts were specified during `new` command i.e, if there are multiple hosts in `mon initial members` and multiple keyrings were created then a concatenated keyring is used for deployment of monitors. In this process a keyring parser is used which looks for `[entity]` sections in monitor keyrings and returns a list of those sections. A helper is then used to collect all keyrings into a single blob that will be used to inject it to monitors with `--mkfs` on remote nodes. All keyring files are concatenated to be in a directory ending with `.keyring`. During this process the helper uses list of sections returned by keyring parser to check if an entity is already present in a keyring and if not, adds it. The concatenated keyring is used for deployment of monitors to desired multiple hosts.

Usage:

```
ceph-deploy mon create [HOST] [HOST...]
```

Here, [HOST] is hostname of desired monitor host(s).

Subcommand `add` is used to add a monitor to an existing cluster. It first detects platform and distro for desired host and checks if hostname is compatible for deployment. It then uses the monitor keyring, ensures configuration for new monitor host and adds the monitor to the cluster. If the section for the monitor exists and defines a mon addr that will be used, otherwise it will fallback by resolving the hostname to an IP. If *`--address`* is used it will override all other options. After adding the monitor to the cluster, it gives it some time to start. It then looks for any monitor errors and checks monitor status. Monitor errors arise if the monitor is not added in `mon initial members`, if it doesn't exist in `monmap` and if neither `public_addr` nor `public_network` keys were defined for monitors. Under such conditions, monitors may not be able to form quorum. Monitor status tells if the monitor is up and running normally. The status is checked by running `ceph daemon mon.hostname mon_status` on remote end which provides the output and returns a boolean status of what is going on. `False` means a monitor that is not fine even if it is up and running, while `True` means the monitor is up and running correctly.

Usage:

```
ceph-deploy mon add [HOST]

ceph-deploy mon add [HOST] --address [IP]
```

Here, [HOST] is the hostname and [IP] is the IP address of the desired monitor node.

Subcommand `destroy` is used to completely remove monitors on remote hosts. It takes hostnames as arguments. It stops the monitor, verifies if `ceph-mon` daemon really stopped, creates an archive directory `mon-remove` under `/var/lib/ceph/`, archives old monitor directory in `{cluster}-{hostname}-{stamp}` format in it and removes the monitor from cluster by running `ceph remove...` command.

Usage:

```
ceph-deploy mon destroy [HOST]
```

Here, [HOST] is hostname of monitor that is to be removed.

gatherkeys

Gather authentication keys for provisioning new nodes. It takes hostnames as arguments. It checks for and fetches `client.admin` keyring, monitor keyring and `bootstrap-mds/bootstrap-osd` keyring from monitor host. These authentication keys are used when new `monitors/OSDs/MDS` are added to the cluster.

Usage:

```
ceph-deploy gatherkeys [HOST] [HOST...]
```

Here, [HOST] is hostname of the monitor from where keys are to be pulled.

disk

Manage disks on a remote host. It actually triggers the `ceph-disk` utility and it's subcommands to manage disks.

Subcommand `list` lists disk partitions and Ceph OSDs.

Usage:

```
ceph-deploy disk list [HOST:[DISK]]
```

Here, [HOST] is hostname of the node and [DISK] is disk name or path.

Subcommand `prepare` prepares a directory, disk or drive for a Ceph OSD. It creates a GPT partition, marks the partition with Ceph type uuid, creates a file system, marks the file system as ready for Ceph consumption, uses entire partition and adds a new partition to the journal disk.

Usage:

```
ceph-deploy disk prepare [HOST:[DISK]]
```

Here, [HOST] is hostname of the node and [DISK] is disk name or path.

Subcommand `activate` activates the Ceph OSD. It mounts the volume in a temporary location, allocates an OSD id (if needed), remounts in the correct location `/var/lib/ceph/osd/$cluster-$id` and starts `ceph-osd`. It is triggered by `udev` when it sees the OSD GPT partition type or on ceph service start with `ceph disk activate-all`.

Usage:

```
ceph-deploy disk activate [HOST:[DISK]]
```

Here, [HOST] is hostname of the node and [DISK] is disk name or path.

Subcommand `zap` zaps/erases/destroys a device's partition table and contents. It actually uses `sgdisk` and it's option `--zap-all` to destroy both GPT and MBR data structures so that the disk becomes suitable for repartitioning. `sgdisk` then uses `--mbrtogpt` to convert the MBR or BSD disklabel disk to a GPT disk. The `prepare` subcommand can now be executed which will create a new GPT partition.

Usage:

```
ceph-deploy disk zap [HOST:[DISK]]
```

Here, [HOST] is hostname of the node and [DISK] is disk name or path.

osd

Manage OSDs by preparing data disk on remote host. `osd` makes use of certain subcommands for managing OSDs.

Subcommand `prepare` prepares a directory, disk or drive for a Ceph OSD. It first checks against multiple OSDs getting created and warns about the possibility of more than the recommended which would cause issues with max allowed PIDs in a system. It then reads the bootstrap-osd key for the cluster or writes the bootstrap key if not found. It then uses **`ceph-disk`** utility's `prepare` subcommand to prepare the disk, journal and deploy the OSD on the desired host. Once prepared, it gives some time to the OSD to settle and checks for any possible errors and if found, reports to the user.

Usage:

```
ceph-deploy osd prepare HOST:DISK[:JOURNAL] [HOST:DISK[:JOURNAL]...]
```

Subcommand `activate` activates the OSD prepared using `prepare` subcommand. It actually uses **`ceph-disk`** utility's `activate` subcommand with appropriate init type based on distro to activate the OSD. Once activated, it gives some time to the OSD to start and checks for any possible errors and if found, reports to the user. It checks the status of the prepared OSD, checks the OSD tree and makes sure the OSDs are up and in.

Usage:

```
ceph-deploy osd activate HOST:DISK[:JOURNAL] [HOST:DISK[:JOURNAL]...]
```

Subcommand `create` uses `prepare` and `activate` subcommands to create an OSD.

Usage:

```
ceph-deploy osd create HOST:DISK[:JOURNAL] [HOST:DISK[:JOURNAL]...]
```

Subcommand `list` lists disk partitions, Ceph OSDs and prints OSD metadata. It gets the osd tree from a monitor host, uses the `ceph-disk-list` output and gets the mount point by matching the line where the partition mentions the OSD name, reads metadata from files, checks if a journal path exists, if the OSD is in a OSD tree and prints the OSD metadata.

Usage:

```
ceph-deploy osd list HOST:DISK[:JOURNAL] [HOST:DISK[:JOURNAL]...]
```

admin

Push configuration and `client.admin` key to a remote host. It takes the `{cluster}.client.admin.keyring` from admin node and writes it under `/etc/ceph` directory of desired node.

Usage:

```
ceph-deploy admin [HOST] [HOST...]
```

Here, [HOST] is desired host to be configured for Ceph administration.

config

Push/pull configuration file to/from a remote host. It uses `push` subcommand to takes the configuration file from admin host and write it to remote host under `/etc/ceph` directory. It uses `pull` subcommand to do the opposite i.e, pull the configuration file under `/etc/ceph` directory of remote host to admin node.

Usage:

```
ceph-deploy push [HOST] [HOST...]

ceph-deploy pull [HOST] [HOST...]
```

Here, [HOST] is the hostname of the node where config file will be pushed to or pulled from.

uninstall

Remove Ceph packages from remote hosts. It detects the platform and distro of selected host and uninstalls Ceph packages from it. However, some dependencies like `librbd1` and `librados2` will not be removed because they can cause issues with `qemu-kvm`.

Usage:

```
ceph-deploy uninstall [HOST] [HOST...]
```

Here, [HOST] is hostname of the node from where Ceph will be uninstalled.

purge

Remove Ceph packages from remote hosts and purge all data. It detects the platform and distro of selected host, uninstalls Ceph packages and purges all data. However, some dependencies like `librbd1` and `librados2` will not be removed because they can cause issues with `qemu-kvm`.

Usage:

```
ceph-deploy purge [HOST] [HOST...]
```

Here, [HOST] is hostname of the node from where Ceph will be purged.

purgedata

Purge (delete, destroy, discard, shred) any Ceph data from `/var/lib/ceph`. Once it detects the platform and distro of desired host, it first checks if Ceph is still installed on the selected host and if installed, it won't purge data from it. If Ceph is already uninstalled from the host, it tries to remove the contents of `/var/lib/ceph`. If it fails then probably OSDs are still mounted and needs to be unmounted to continue. It unmount the OSDs and tries to remove the contents of `/var/lib/ceph` again and checks for errors. It also removes contents of `/etc/ceph`. Once all steps are successfully completed, all the Ceph data from the selected host are removed.

Usage:

```
ceph-deploy purgedata [HOST] [HOST...]
```

Here, [HOST] is hostname of the node from where Ceph data will be purged.

forgetkeys

Remove authentication keys from the local directory. It removes all the authentication keys i.e, monitor keyring, client.admin keyring, bootstrap-osd and bootstrap-mds keyring from the node.

Usage:

```
ceph-deploy forgetkeys
```

pkg

Manage packages on remote hosts. It is used for installing or removing packages from remote hosts. The package names for installation or removal are to be specified after the command. Two options *`--install`* and *`--remove`* are used for this purpose.

Usage:

```
ceph-deploy pkg --install [PKGs] [HOST] [HOST...]

ceph-deploy pkg --remove [PKGs] [HOST] [HOST...]
```

Here, [PKGs] is comma-separated package names and [HOST] is hostname of the remote node where packages are to be installed or removed from.

calamari

Install and configure Calamari nodes. It first checks if distro is supported for Calamari installation by ceph-deploy. An argument `connect` is used for installation and configuration. It checks for `ceph-deploy` configuration file (cd_conf) and Calamari release repo or `calamari-minion` repo. It relies on default for repo installation as it doesn't install Ceph unless specified otherwise. `options` dictionary is also defined because `ceph-deploy` pops items internally which causes issues when those items are needed to be available for every host. If the distro is Debian/Ubuntu, it is ensured that proxy is disabled for `calamari-minion` repo. `calamari-minion` package is then installed and custom repository files are added. minion config is placed prior to installation so that it is present when the minion first starts. config directory, calamari salt config are created and salt-minion package is installed. If the distro is Redhat/CentOS, the salt-minion service needs to be started.

Usage:

```
ceph-deploy calamari {connect} [HOST] [HOST...]
```

Here, [HOST] is the hostname where Calamari is to be installed.

An option `--release` can be used to use a given release from repositories defined in **ceph-deploy**'s configuration. Defaults to `calamari-minion`.

Another option *--master* can also be used with this command.

Options

--version
The current installed version of **ceph-deploy**.

--username
The username to connect to the remote host.

--overwrite-conf
Overwrite an existing conf file on remote host (if present).

--cluster
Name of the cluster.

--ceph-conf
Use (or reuse) a given `ceph.conf` file.

--no-ssh-copykey
Do not attempt to copy ssh keys.

--fsid
Provide an alternate FSID for `ceph.conf` generation.

--cluster-network
Specify the (internal) cluster network.

--public-network
Specify the public network for a cluster.

--testing
Install the latest development release.

--dev
Install a bleeding edge built from Git branch or tag (default: master).

--adjust-repos
Install packages modifying source repos.

--no-adjust-repos
Install packages without modifying source repos.

--repo
Install repo files only (skips package installation).

--local-mirror
Fetch packages and push them to hosts for a local repo mirror.

--repo-url
Specify a repo url that mirrors/contains Ceph packages.

--gpg-url
Specify a GPG key url to be used with custom repos (defaults to ceph.com).

--address
: IP address of the host node to be added to the cluster.

--keyrings
: Concatenate multiple keyrings to be seeded on new monitors.

--zap-disk
: Destroy the partition table and content of a disk.

--fs-type
: Filesystem to use to format disk `(xfs, btrfs or ext4)`.

--dmcrypt
: Encrypt [data-path] and/or journal devices with `dm-crypt`.

--dmcrypt-key-dir
: Directory where `dm-crypt` keys are stored.

--install
: Comma-separated package(s) to install on remote hosts.

--remove
: Comma-separated package(s) to remove from remote hosts.

--master
: The domain for the Calamari master server.

Availability

ceph-deploy is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the documentation at http://ceph.com/ceph-deploy/docs for more information.

See also

ceph-mon(8), ceph-osd(8), ceph-disk(8), ceph-mds(8)

4.4.4 ceph-rest-api – ceph RESTlike administration server

Synopsis

ceph-rest-api [-c *conffile*] [–cluster *clustername*] [-n *name*] [-i *id*]

Description

ceph-rest-api is a WSGI application that can run as a standalone web service or run under a web server that supports WSGI. It provides much of the functionality of the **ceph** command-line tool through an HTTP-accessible interface.

Options

-c/-conf conffile
: names the ceph.conf file to use for configuration. If -c is not specified, the default depends on the state of the –cluster option (default 'ceph'; see below). The configuration file is searched for in this order:

  - $CEPH_CONF

- /etc/ceph/${cluster}.conf
- ~/.ceph/${cluster}.conf
- ${cluster}.conf (in the current directory)

so you can also pass this option in the environment as CEPH_CONF.

`--cluster` `clustername`
set *clustername* for use in the $cluster metavariable, for locating the ceph.conf file. The default is 'ceph'.

`-n``/-name name`
specifies the client 'name', which is used to find the client-specific configuration options in the config file, and also is the name used for authentication when connecting to the cluster (the entity name appearing in ceph auth list output, for example). The default is 'client.restapi'.

`-i``/-id id`
specifies the client 'id', which will form the clientname as 'client.<id>' if clientname is not set. If -n/-name is set, that takes precedence.

Also, global Ceph options are supported.

Configuration parameters

Supported configuration parameters include:

- **keyring** the keyring file holding the key for 'clientname'
- **public addr** ip:port to listen on (default 0.0.0.0:5000)
- **log file** (usual Ceph default)
- **restapi base url** the base URL to answer requests on (default /api/v0.1)
- **restapi log level** critical, error, warning, info, debug (default warning)

Configuration parameters are searched in the standard order: first in the section named '<clientname>', then 'client', then 'global'.

<clientname> is either supplied by -n/–name, "client.<id>" where <id> is supplied by -i/–id, or 'client.restapi' if neither option is present.

A single-threaded server will run on **public addr** if the ceph-rest-api executed directly; otherwise, configuration is specified by the enclosing WSGI web server.

Commands

Commands are submitted with HTTP GET requests (for commands that primarily return data) or PUT (for commands that affect cluster state). HEAD and OPTIONS are also supported. Standard HTTP status codes are returned.

For commands that return bulk data, the request can include Accept: application/json or Accept: application/xml to select the desired structured output, or you may use a .json or .xml addition to the requested PATH. Parameters are supplied as query parameters in the request; for parameters that take more than one value, repeat the key=val construct. For instance, to remove OSDs 2 and 3, send a PUT request to `osd/rm?ids=2&ids=3`.

Discovery

Human-readable discovery of supported commands and parameters, along with a small description of each command, is provided when the requested path is incomplete/partially matching. Requesting / will redirect to the value of **restapi**

base url, and that path will give a full list of all known commands. For example, requesting `api/vX.X/mon` will return the list of API calls for monitors - `api/vX.X/osd` will return the list of API calls for OSD and so on.

The command set is very similar to the commands supported by the **ceph** tool. One notable exception is that the `ceph pg <pgid> <command>` style of commands is supported here as `tell/<pgid>/command?args`.

Deployment as WSGI application

When deploying as WSGI application (say, with Apache/mod_wsgi, or nginx/uwsgi, or gunicorn, etc.), use the `ceph_rest_api.py` module (`ceph-rest-api` is a thin layer around this module). The standalone web server is of course not used, so address/port configuration is done in the WSGI server. Use a python .wsgi module or the equivalent to call `app = generate_app(conf, cluster, clientname, clientid, args)` where:

- conf is as -c/–conf above
- cluster is as –cluster above
- clientname, -n/–name
- clientid, -i/–id, and
- args are any other generic Ceph arguments

When app is returned, it will have attributes 'ceph_addr' and 'ceph_port' set to what the address and port are in the Ceph configuration; those may be used for the server, or ignored.

Any errors reading configuration or connecting to the cluster cause an exception to be raised; see your WSGI server documentation for how to see those messages in case of problem.

Availability

ceph-rest-api is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

See also

ceph(8)

4.4.5 ceph-authtool – ceph keyring manipulation tool

Synopsis

ceph-authtool *keyringfile* [-l | –list] [-C | –create-keyring] [-p | –print] [-n | –name *entityname*] [–gen-key] [-a | –add-key *base64_key*] [–caps *capfile*]

Description

ceph-authtool is a utility to create, view, and modify a Ceph keyring file. A keyring file stores one or more Ceph authentication keys and possibly an associated capability specification. Each key is associated with an entity name, of the form `{client,mon,mds,osd}.name`.

WARNING Ceph provides authentication and protection against man-in-the-middle attacks once secret keys are in place. However, data over the wire is not encrypted, which may include the messages used to configure said keys. The system is primarily intended to be used in trusted environments.

Options

-l, --list
will list all keys and capabilities present in the keyring

-p, --print
will print an encoded key for the specified entityname. This is suitable for the `mount -o secret=` argument

-C, --create-keyring
will create a new keyring, overwriting any existing keyringfile

--gen-key
will generate a new secret key for the specified entityname

--add-key
will add an encoded key to the keyring

--cap `subsystem capability`
will set the capability for given subsystem

--caps `capsfile`
will set all of capabilities associated with a given key, for all subsystems

Capabilities

The subsystem is the name of a Ceph subsystem: `mon`, `mds`, or `osd`.

The capability is a string describing what the given user is allowed to do. This takes the form of a comma separated list of allow clauses with a permission specifier containing one or more of rwx for read, write, and execute permission. The `allow *` grants full superuser permissions for the given subsystem.

For example:

```
# can read, write, and execute objects
osd = "allow rwx"

# can access mds server
mds = "allow"

# can modify cluster state (i.e., is a server daemon)
mon = "allow rwx"
```

A librados user restricted to a single pool might look like:

```
mon = "allow r"

osd = "allow rw pool foo"
```

A client using rbd with read access to one pool and read/write access to another:

```
mon = "allow r"

osd = "allow class-read object_prefix rbd_children, allow pool templates r class-read, allow pool vms
```

A client mounting the file system with minimal permissions would need caps like:

```
mds = "allow"

osd = "allow rw pool data"

mon = "allow r"
```

OSD Capabilities

In general, an osd capability follows the grammar:

```
osdcap  := grant[,grant...]
grant   := allow (match capspec | capspec match)
match   := [pool[=]<poolname> | object_prefix <prefix>]
capspec := * | [r][w][x] [class-read] [class-write]
```

The capspec determines what kind of operations the entity can perform:

```
r           = read access to objects
w           = write access to objects
x           = can call any class method (same as class-read class-write)
class-read  = can call class methods that are reads
class-write = can call class methods that are writes
*           = equivalent to rwx, plus the ability to run osd admin commands,
              i.e. ceph osd tell ...
```

The match criteria restrict a grant based on the pool being accessed. Grants are additive if the client fulfills the match condition. For example, if a client has the osd capabilities: “allow r object_prefix prefix, allow w pool foo, allow x pool bar”, then it has rw access to pool foo, rx access to pool bar, and r access to objects whose names begin with ‘prefix’ in any pool.

Caps file format

The caps file format consists of zero or more key/value pairs, one per line. The key and value are separated by an =, and the value must be quoted (with `'` or `"`) if it contains any whitespace. The key is the name of the Ceph subsystem (`osd`, `mds`, `mon`), and the value is the capability string (see above).

Example

To create a new keyring containing a key for client.foo:

```
ceph-authtool -C -n client.foo --gen-key keyring
```

To associate some capabilities with the key (namely, the ability to mount a Ceph filesystem):

```
ceph-authtool -n client.foo --cap mds 'allow' --cap osd 'allow rw pool=data' --cap mon 'allow r' keyr
```

To display the contents of the keyring:

```
ceph-authtool -l keyring
```

When mounting a Ceph file system, you can grab the appropriately encoded secret key with:

```
mount -t ceph serverhost:/ mountpoint -o name=foo,secret=`ceph-authtool -p -n client.foo keyring`
```

Availability

ceph-authtool is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

See also

ceph(8)

4.4.6 ceph-clsinfo – show class object information

Synopsis

ceph-clsinfo [*options*] ... *filename*

Description

ceph-clsinfo can show name, version, and architecture information about a specific class object.

Options

`-n, --name`
Shows the class name

`-v, --version`
Shows the class version

`-a, --arch`
Shows the class architecture

Availability

ceph-clsinfo is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

See also

ceph(8)

4.4.7 ceph-conf – ceph conf file tool

Synopsis

ceph-conf -c *conffile* –list-all-sections
ceph-conf -c *conffile* -L
ceph-conf -c *conffile* -l *prefix*
ceph-conf *key* -s *section1* ...
ceph-conf [-s *section*] –lookup *key*
ceph-conf [-s *section*] *key*

Description

ceph-conf is a utility for getting information about a ceph configuration file. As with most Ceph programs, you can specify which Ceph configuration file to use with the `-c` flag.

Actions

ceph-conf will perform one of the following actions:

–list-all-sections or -L prints out a list of all the section names in the configuration file.

–list-sections or -l prints out a list of all the sections that begin with a given prefix. For example, –list-sections mon would list all sections beginning with mon.

–lookup will search the configuration for a given value. By default, the sections that are searched are determined by the Ceph name that we are using. The Ceph name defaults to client.admin. It can be specified with –name.

For example, if we specify –name osd.0, the following sections will be searched: [osd.0], [osd], [global]

You can specify additional sections to search with –section or -s. These additional sections will be searched before the sections that would normally be searched. As always, the first matching entry we find will be returned.

Note: –lookup is the default action. If no other actions are given on the command line, we will default to doing a lookup.

Examples

To find out what value osd 0 will use for the "osd data" option:

```
ceph-conf -c foo.conf  --name osd.0 --lookup "osd data"
```

To find out what value will mds a use for the "log file" option:

```
ceph-conf -c foo.conf  --name mds.a "log file"
```

To list all sections that begin with osd:

```
ceph-conf -c foo.conf -l osd
```

To list all sections:

```
ceph-conf -c foo.conf -L
```

Availability

ceph-conf is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

See also

ceph(8),

4.4.8 ceph-debugpack – ceph debug packer utility

Synopsis

ceph-debugpack [*options*] *filename.tar.gz*

Description

ceph-debugpack will build a tarball containing various items that are useful for debugging crashes. The resulting tarball can be shared with Ceph developers when debugging a problem.

The tarball will include the binaries for ceph-mds, ceph-osd, and ceph-mon, radosgw, any log files, the ceph.conf configuration file, any core files we can find, and (if the system is running) dumps of the current cluster state as reported by 'ceph report'.

Options

-c `ceph.conf`, **--conf**=`ceph.conf`
  Use *ceph.conf* configuration file instead of the default `/etc/ceph/ceph.conf` to determine monitor addresses during startup.

Availability

ceph-debugpack is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

See also

ceph(8) ceph-post-file(8)

4.4.9 ceph-dencoder – ceph encoder/decoder utility

Synopsis

ceph-dencoder [commands...]

Description

ceph-dencoder is a utility to encode, decode, and dump ceph data structures. It is used for debugging and for testing inter-version compatibility.

ceph-dencoder takes a simple list of commands and performs them in order.

Commands

`version`
  Print the version string for the **ceph-dencoder** binary.

import <file>
: Read a binary blob of encoded data from the given file. It will be placed in an in-memory buffer.

export <file>
: Write the contents of the current in-memory buffer to the given file.

list_types
: List the data types known to this build of **ceph-dencoder**.

type <name>
: Select the given type for future `encode` or `decode` operations.

skip <bytes>
: Seek <bytes> into the imported file before reading data structure, use this with objects that have a preamble/header before the object of interest.

decode
: Decode the contents of the in-memory buffer into an instance of the previously selected type. If there is an error, report it.

encode
: Encode the contents of the in-memory instance of the previously selected type to the in-memory buffer.

dump_json
: Print a JSON-formatted description of the in-memory object.

count_tests
: Print the number of built-in test instances of the previously selected type that **ceph-dencoder** is able to generate.

select_test <n>
: Select the given build-in test instance as a the in-memory instance of the type.

get_features
: Print the decimal value of the feature set supported by this version of **ceph-dencoder**. Each bit represents a feature. These correspond to CEPH_FEATURE_* defines in src/include/ceph_features.h.

set_features <f>
: Set the feature bits provided to `encode` to *f*. This allows you to encode objects such that they can be understood by old versions of the software (for those types that support it).

Example

Say you want to examine an attribute on an object stored by `ceph-osd`. You can do this:

```
$ cd /mnt/osd.12/current/2.b_head
$ attr -l foo_bar_head_EFE6384B
Attribute "ceph.snapset" has a 31 byte value for foo_bar_head_EFE6384B
Attribute "ceph._" has a 195 byte value for foo_bar_head_EFE6384B
$ attr foo_bar_head_EFE6384B -g ceph._ -q > /tmp/a
$ ceph-dencoder type object_info_t import /tmp/a decode dump_json
{ "oid": { "oid": "foo",
      "key": "bar",
      "snapid": -2,
      "hash": 4024842315,
      "max": 0},
  "locator": { "pool": 2,
      "preferred": -1,
      "key": "bar"},
  "category": "",
  "version": "9'1",
  "prior_version": "0'0",
  "last_reqid": "client.4116.0:1",
  "size": 1681,
  "mtime": "2012-02-21 08:58:23.666639",
  "lost": 0,
  "wrlock_by": "unknown.0.0:0",
  "snaps": [],
  "truncate_seq": 0,
  "truncate_size": 0,
  "watchers": {}}
```

Alternatively, perhaps you wish to dump an internal CephFS metadata object, you might do that like this:

```
$ rados -p metadata get mds_snaptable mds_snaptable.bin
$ ceph-dencoder type SnapServer skip 8 import mds_snaptable.bin decode dump_json
{ "snapserver": { "last_snap": 1,
   "pending_noop": [],
   "snaps": [],
   "need_to_purge": {},
   "pending_create": [],
   "pending_destroy": []}}
```

Availability

ceph-dencoder is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

See also

ceph(8)

4.4.10 ceph-mon – ceph monitor daemon

Synopsis

ceph-mon -i *monid* [–mon-data *mondatapath*]

Description

ceph-mon is the cluster monitor daemon for the Ceph distributed file system. One or more instances of **ceph-mon** form a Paxos part-time parliament cluster that provides extremely reliable and durable storage of cluster membership, configuration, and state.

The *mondatapath* refers to a directory on a local file system storing monitor data. It is normally specified via the `mon data` option in the configuration file.

Options

`-f, --foreground`
Foreground: do not daemonize after startup (run in foreground). Do not generate a pid file. Useful when run via ceph-run(8).

-d
Debug mode: like `-f`, but also send all log output to stderr.

-c `ceph.conf`, **--conf**=`ceph.conf`
Use *ceph.conf* configuration file instead of the default `/etc/ceph/ceph.conf` to determine monitor addresses during startup.

--mkfs
Initialize the `mon data` directory with seed information to form and initial ceph file system or to join an existing monitor cluster. Three pieces of information must be provided:

- The cluster fsid. This can come from a monmap (`--monmap <path>`) or explicitly via `--fsid <uuid>`.
- A list of monitors and their addresses. This list of monitors can come from a monmap (`--monmap <path>`), the `mon host` configuration value (in *ceph.conf* or via `-m host1,host2,...`), or `mon addr` lines in *ceph.conf*. If this monitor is to be part of the initial monitor quorum for a new Ceph cluster, then it must be included in the initial list, matching either the name or address of a monitor in the list. When matching by address, either the `public addr` or `public subnet` options may be used.
- The monitor secret key `mon.`. This must be included in the keyring provided via `--keyring <path>`.

--keyring
Specify a keyring for use with `--mkfs`.

Availability

ceph-mon is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

See also

ceph(8), ceph-mds(8), ceph-osd(8)

4.4.11 ceph-osd – ceph object storage daemon

Synopsis

ceph-osd -i *osdnum* [–osd-data *datapath*] [–osd-journal *journal*] [–mkfs] [–mkjournal] [–mkkey]

Description

ceph-osd is the object storage daemon for the Ceph distributed file system. It is responsible for storing objects on a local file system and providing access to them over the network.

The datapath argument should be a directory on a btrfs file system where the object data resides. The journal is optional, and is only useful performance-wise when it resides on a different disk than datapath with low latency (ideally, an NVRAM device).

Options

-f, --foreground
: Foreground: do not daemonize after startup (run in foreground). Do not generate a pid file. Useful when run via ceph-run(8).

-d
: Debug mode: like `-f`, but also send all log output to stderr.

--osd-data `osddata`
: Use object store at *osddata*.

--osd-journal `journal`
: Journal updates to *journal*.

--mkfs
: Create an empty object repository. This also initializes the journal (if one is defined).

--mkkey
: Generate a new secret key. This is normally used in combination with `--mkfs` as it is more convenient than generating a key by hand with ceph-authtool(8).

--mkjournal
: Create a new journal file to match an existing object repository. This is useful if the journal device or file is wiped out due to a disk or file system failure.

--flush-journal
: Flush the journal to permanent store. This runs in the foreground so you know when it's completed. This can be useful if you want to resize the journal or need to otherwise destroy it: this guarantees you won't lose data.

--get-cluster-fsid
: Print the cluster fsid (uuid) and exit.

--get-osd-fsid
: Print the OSD's fsid and exit. The OSD's uuid is generated at –mkfs time and is thus unique to a particular instantiation of this OSD.

--get-journal-fsid
: Print the journal's uuid. The journal fsid is set to match the OSD fsid at –mkfs time.

-c `ceph.conf`, **--conf**=`ceph.conf`
: Use *ceph.conf* configuration file instead of the default `/etc/ceph/ceph.conf` for runtime configuration options.

-m `monaddress[:port]`
: Connect to specified monitor (instead of looking through `ceph.conf`).

Availability

ceph-osd is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

See also

ceph(8), ceph-mds(8), ceph-mon(8), ceph-authtool(8)

4.4.12 ceph-run – restart daemon on core dump

Synopsis

ceph-run *command* ...

Description

ceph-run is a simple wrapper that will restart a daemon if it exits with a signal indicating it crashed and possibly core dumped (that is, signals 3, 4, 5, 6, 8, or 11).

The command should run the daemon in the foreground. For Ceph daemons, that means the `-f` option.

Options

None

Availability

ceph-run is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

See also

ceph(8), ceph-mon(8), ceph-mds(8), ceph-osd(8)

4.4.13 ceph-syn – ceph synthetic workload generator

Synopsis

ceph-syn [-m *monaddr*:*port*] –syn *command* ...

Description

ceph-syn is a simple synthetic workload generator for the Ceph distributed file system. It uses the userspace client library to generate simple workloads against a currently running file system. The file system need not be mounted via ceph-fuse(8) or the kernel client.

One or more `--syn` command arguments specify the particular workload, as documented below.

Options

`-d`
Detach from console and daemonize after startup.

`-c` `ceph.conf`, **`--conf`**`=ceph.conf`
Use *ceph.conf* configuration file instead of the default `/etc/ceph/ceph.conf` to determine monitor addresses during startup.

-m `monaddress[:port]`
  Connect to specified monitor (instead of looking through `ceph.conf`).

--num_client `num`
  Run num different clients, each in a separate thread.

--syn `workloadspec`
  Run the given workload. May be specified as many times as needed. Workloads will normally run sequentially.

Workloads

Each workload should be preceded by `--syn` on the command line. This is not a complete list.

`mknap` ***path snapname*** Create a snapshot called *snapname* on *path*.

`rmsnap` ***path snapname*** Delete snapshot called *snapname* on *path*.

`rmfile` ***path*** Delete/unlink *path*.

`writefile` ***sizeinmb blocksize*** Create a file, named after our client id, that is *sizeinmb* MB by writing *blocksize* chunks.

`readfile` ***sizeinmb blocksize*** Read file, named after our client id, that is *sizeinmb* MB by writing *blocksize* chunks.

`rw` ***sizeinmb blocksize*** Write file, then read it back, as above.

`makedirs` ***numsubdirs numfiles depth*** Create a hierarchy of directories that is *depth* levels deep. Give each directory *numsubdirs* subdirectories and *numfiles* files.

`walk` Recursively walk the file system (like find).

Availability

ceph-syn is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

See also

ceph(8), ceph-fuse(8)

4.4.14 crushtool – CRUSH map manipulation tool

Synopsis

crushtool (-d *map* | -c *map.txt* | –build –num_osds *numosds layer1 ...* | –test) [-o *outfile*]

Description

crushtool is a utility that lets you create, compile, decompile and test CRUSH map files.

CRUSH is a pseudo-random data distribution algorithm that efficiently maps input values (typically data objects) across a heterogeneous, hierarchically structured device map. The algorithm was originally described in detail in the following paper (although it has evolved some since then):

  http://www.ssrc.ucsc.edu/Papers/weil-sc06.pdf

The tool has four modes of operation.

--compile|-c map.txt
: will compile a plaintext map.txt into a binary map file.

--decompile|-d map
: will take the compiled map and decompile it into a plaintext source file, suitable for editing.

--build -num_osds {num-osds} layer1 ...
: will create map with the given layer structure. See below for a detailed explanation.

--test
: will perform a dry run of a CRUSH mapping for a range of input object names. See below for a detailed explanation.

Unlike other Ceph tools, **crushtool** does not accept generic options such as **–debug-crush** from the command line. They can, however, be provided via the CEPH_ARGS environment variable. For instance, to silence all output from the CRUSH subsystem:

```
CEPH_ARGS="--debug-crush 0" crushtool ...
```

Running tests with –test

The test mode will use the input crush map (as specified with **-i map**) and perform a dry run of CRUSH mapping or random placement (if **–simulate** is set). On completion, two kinds of reports can be created. 1) The **–show-...** option outputs human readable information on stderr. 2) The **–output-csv** option creates CSV files that are documented by the **–help-output** option.

--show-statistics
: For each rule, displays the mapping of each object. For instance:

```
    CRUSH rule 1 x 24 [11,6]
```

shows that object **24** is mapped to devices **[11,6]** by rule **1**. At the end of the mapping details, a summary of the distribution is displayed. For instance:

```
    rule 1 (metadata) num_rep 5 result size == 5:    1024/1024
```

shows that rule **1** which is named **metadata** successfully mapped **1024** objects to **result size == 5** devices when trying to map them to **num_rep 5** replicas. When it fails to provide the required mapping, presumably because the number of **tries** must be increased, a breakdown of the failures is displayed. For instance:

```
    rule 1 (metadata) num_rep 10 result size == 8:   4/1024
    rule 1 (metadata) num_rep 10 result size == 9:   93/1024
    rule 1 (metadata) num_rep 10 result size == 10:  927/1024
```

shows that although **num_rep 10** replicas were required, **4** out of **1024** objects (**4/1024**) were mapped to **result size == 8** devices only.

--show-bad-mappings
: Displays which object failed to be mapped to the required number of devices. For instance:

```
    bad mapping rule 1 x 781 num_rep 7 result [8,10,2,11,6,9]
```

shows that when rule **1** was required to map **7** devices, it could map only six : **[8,10,2,11,6,9]**.

--show-utilization
: Displays the expected and actual utilisation for each device, for each number of replicas. For instance:

```
    device 0: stored : 951      expected : 853.333
    device 1: stored : 963      expected : 853.333
    ...
```

shows that device **0** stored **951** objects and was expected to store **853**. Implies **–show-statistics**.

`--show-utilization-all`
Displays the same as **–show-utilization** but does not suppress output when the weight of a device is zero. Implies **–show-statistics**.

`--show-choose-tries`
Displays how many attempts were needed to find a device mapping. For instance:

```
    0:     95224
    1:      3745
    2:      2225
    ..
```

shows that **95224** mappings succeeded without retries, **3745** mappings succeeded with one attempts, etc. There are as many rows as the value of the **–set-choose-total-tries** option.

`--output-csv`
Creates CSV files (in the current directory) containing information documented by **–help-output**. The files are named after the rule used when collecting the statistics. For instance, if the rule : 'metadata' is used, the CSV files will be:

```
    metadata-absolute_weights.csv
    metadata-device_utilization.csv
    ...
```

The first line of the file shortly explains the column layout. For instance:

```
    metadata-absolute_weights.csv
    Device ID, Absolute Weight
    0,1
    ...
```

`--output-name` `NAME`
Prepend **NAME** to the file names generated when **–output-csv** is specified. For instance **–output-name FOO** will create files:

```
    FOO-metadata-absolute_weights.csv
    FOO-metadata-device_utilization.csv
    ...
```

The **–set-...** options can be used to modify the tunables of the input crush map. The input crush map is modified in memory. For example:

```
$ crushtool -i mymap --test --show-bad-mappings
bad mapping rule 1 x 781 num_rep 7 result [8,10,2,11,6,9]
```

could be fixed by increasing the **choose-total-tries** as follows:

$ crushtool -i mymap –test –show-bad-mappings –set-choose-total-tries 500

Building a map with –build

The build mode will generate hierarchical maps. The first argument specifies the number of devices (leaves) in the CRUSH hierarchy. Each layer describes how the layer (or devices) preceding it should be grouped.

Each layer consists of:

```
bucket ( uniform | list | tree | straw ) size
```

The **bucket** is the type of the buckets in the layer (e.g. “rack”). Each bucket name will be built by appending a unique number to the **bucket** string (e.g. “rack0”, “rack1”...).

The second component is the type of bucket: **straw** should be used most of the time.

The third component is the maximum size of the bucket. A size of zero means a bucket of infinite capacity.

Example

Suppose we have two rows with two racks each and 20 nodes per rack. Suppose each node contains 4 storage devices for Ceph OSD Daemons. This configuration allows us to deploy 320 Ceph OSD Daemons. Lets assume a 42U rack with 2U nodes, leaving an extra 2U for a rack switch.

To reflect our hierarchy of devices, nodes, racks and rows, we would execute the following:

```
$ crushtool -o crushmap --build --num_osds 320 \
       node straw 4 \
       rack straw 20 \
       row straw 2 \
       root straw 0
# id        weight  type name       reweight
-87 320     root root
-85 160             row row0
-81 80                      rack rack0
-1  4                               node node0
0   1                                       osd.0   1
1   1                                       osd.1   1
2   1                                       osd.2   1
3   1                                       osd.3   1
-2  4                               node node1
4   1                                       osd.4   1
5   1                                       osd.5   1
...
```

CRUSH rulesets are created so the generated crushmap can be tested. They are the same rulesets as the one created by default when creating a new Ceph cluster. They can be further edited with:

```
# decompile
crushtool -d crushmap -o map.txt

# edit
emacs map.txt

# recompile
crushtool -c map.txt -o crushmap
```

Availability

crushtool is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

See also

ceph(8), osdmaptool(8),

Authors

John Wilkins, Sage Weil, Loic Dachary

4.4.15 librados-config – display information about librados

Synopsis

librados-config [–version] [–vernum]

Description

librados-config is a utility that displays information about the installed `librados`.

Options

`--version`
Display `librados` version

`--vernum`
Display the `librados` version code

Availability

librados-config is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

See also

ceph(8), rados(8)

4.4.16 monmaptool – ceph monitor cluster map manipulation tool

Synopsis

monmaptool *mapfilename* [–clobber] [–print] [–create] [–add *ip*:*port* ...] [–rm *ip*:*port* ...]

Description

monmaptool is a utility to create, view, and modify a monitor cluster map for the Ceph distributed storage system. The monitor map specifies the only fixed addresses in the Ceph distributed system. All other daemons bind to arbitrary addresses and register themselves with the monitors.

When creating a map with –create, a new monitor map with a new, random UUID will be created. It should be followed by one or more monitor addresses.

The default Ceph monitor port is 6789.

Options

`--print`
will print a plaintext dump of the map, after any modifications are made.

`--clobber`
will allow monmaptool to overwrite mapfilename if changes are made.

`--create`
will create a new monitor map with a new UUID (and with it, a new, empty Ceph file system).

`--generate`
generate a new monmap based on the values on the command line or specified in the ceph configuration. This is, in order of preference,

1. `--monmap filename` to specify a monmap to load
2. `--mon-host 'host1,ip2'` to specify a list of hosts or ip addresses
3. `[mon.foo]` sections containing `mon addr` settings in the config

`--filter-initial-members`
filter the initial monmap by applying the `mon initial members` setting. Monitors not present in that list will be removed, and initial members not present in the map will be added with dummy addresses.

`--add` `name ip:port`
will add a monitor with the specified ip:port to the map.

`--rm` `name`
will remove the monitor with the specified ip:port from the map.

`--fsid` `uuid`
will set the fsid to the given uuid. If not specified with –create, a random fsid will be generated.

Example

To create a new map with three monitors (for a fresh Ceph file system):

```
monmaptool  --create  --add  mon.a 192.168.0.10:6789 --add mon.b 192.168.0.11:6789 \
  --add mon.c 192.168.0.12:6789 --clobber monmap
```

To display the contents of the map:

```
monmaptool --print monmap
```

To replace one monitor:

```
monmaptool --rm mon.a --add mon.a 192.168.0.9:6789 --clobber monmap
```

Availability

monmaptool is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

See also

ceph(8), crushtool(8),

4.4.17 osdmaptool – ceph osd cluster map manipulation tool

Synopsis

osdmaptool *mapfilename* [–print] [–createsimple *numosd* [–pgbits *bitsperosd*]] [–clobber]

Description

osdmaptool is a utility that lets you create, view, and manipulate OSD cluster maps from the Ceph distributed storage system. Notably, it lets you extract the embedded CRUSH map or import a new CRUSH map.

Options

`--print`
will simply make the tool print a plaintext dump of the map, after any modifications are made.

`--clobber`
will allow osdmaptool to overwrite mapfilename if changes are made.

`--import-crush` `mapfile`
will load the CRUSH map from mapfile and embed it in the OSD map.

`--export-crush` `mapfile`
will extract the CRUSH map from the OSD map and write it to mapfile.

`--createsimple` `numosd [-pgbits bitsperosd]`
will create a relatively generic OSD map with the numosd devices. If –pgbits is specified, the initial placement group counts will be set with bitsperosd bits per OSD. That is, the pg_num map attribute will be set to numosd shifted by bitsperosd.

Example

To create a simple map with 16 devices:

```
osdmaptool --createsimple 16 osdmap --clobber
```

To view the result:

```
osdmaptool --print osdmap
```

Availability

osdmaptool is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

See also

ceph(8), crushtool(8),

4.4.18 rados – rados object storage utility

Synopsis

rados [-m *monaddr*] [mkpool | rmpool *foo*] [-p | –pool *pool*] [-s | –snap *snap*] [-i *infile*] [-o *outfile*] *command* ...

Description

rados is a utility for interacting with a Ceph object storage cluster (RADOS), part of the Ceph distributed storage system.

Options

`-p` `pool`, **`--pool`** `pool`
Interact with the given pool. Required by most commands.

`-s` `snap`, **`--snap`** `snap`
Read from the given pool snapshot. Valid for all pool-specific read operations.

`-i` `infile`
will specify an input file to be passed along as a payload with the command to the monitor cluster. This is only used for specific monitor commands.

`-o` `outfile`
will write any payload returned by the monitor cluster with its reply to outfile. Only specific monitor commands (e.g. osd getmap) return a payload.

`-c` `ceph.conf`, **`--conf`**`=ceph.conf`
Use ceph.conf configuration file instead of the default /etc/ceph/ceph.conf to determine monitor addresses during startup.

`-m` `monaddress[:port]`
Connect to specified monitor (instead of looking through ceph.conf).

`-b` `block_size`
Set the block size for put/get ops and for write benchmarking.

Global commands

`lspools` List object pools

`df` Show utilization statistics, including disk usage (bytes) and object counts, over the entire system and broken down by pool.

`mkpool` *foo* Create a pool with name foo.

`rmpool` ***foo*** **[** ***foo*** **–yes-i-really-really-mean-it]** Delete the pool foo (and all its data)

Pool specific commands

`get` ***name outfile*** Read object name from the cluster and write it to outfile.

`put` ***name infile*** Write object name to the cluster with contents from infile.

`rm` ***name*** Remove object name.

`ls` ***outfile*** List objects in given pool and write to outfile.

`lssnap` List snapshots for given pool.

`clonedata` ***srcname dstname*** **–object-locator** ***key*** Clone object byte data from *srcname* to *dstname*. Both objects must be stored with the locator key *key* (usually either *srcname* or *dstname*). Object attributes and omap keys are not copied or cloned.

`mksnap` ***foo*** Create pool snapshot named *foo*.

`rmsnap` ***foo*** Remove pool snapshot named *foo*.

`bench` ***seconds mode*** **[-b** ***objsize*** **] [-t** ***threads*** **]** Benchmark for *seconds*. The mode can be *write*, *seq*, or *rand*. *seq* and *rand* are read benchmarks, either sequential or random. Before running one of the reading benchmarks, run a write benchmark with the *–no-cleanup* option. The default object size is 4 MB, and the default number of simulated threads (parallel writes) is 16. Note: -b *objsize* option is valid only in *write* mode.

`cleanup`

`listomapkeys` ***name*** List all the keys stored in the object map of object name.

`listomapvals` ***name*** List all key/value pairs stored in the object map of object name. The values are dumped in hexadecimal.

`getomapval` ***name key*** Dump the hexadecimal value of key in the object map of object name.

`setomapval` ***name key value*** Set the value of key in the object map of object name.

`rmomapkey` ***name key*** Remove key from the object map of object name.

`getomapheader` ***name*** Dump the hexadecimal value of the object map header of object name.

`setomapheader` ***name value*** Set the value of the object map header of object name.

Examples

To view cluster utilization:

```
rados df
```

To get a list object in pool foo sent to stdout:

```
rados -p foo ls -
```

To write an object:

```
rados -p foo put myobject blah.txt
```

To create a snapshot:

```
rados -p foo mksnap mysnap
```

To delete the object:

```
rados -p foo rm myobject
```

To read a previously snapshotted version of an object:

```
rados -p foo -s mysnap get myobject blah.txt.old
```

Availability

rados is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

See also

ceph(8)

4.4.19 ceph-post-file – post files for ceph developers

Synopsis

ceph-post-file [-d *description] [-u *user*] file or dir* ...

Description

ceph-post-file will upload files or directories to ceph.com for later analysis by Ceph developers.

Each invocation uploads files or directories to a separate directory with a unique tag. That tag can be passed to a developer or referenced in a bug report (http://tracker.ceph.com/). Once the upload completes, the directory is marked non-readable and non-writeable to prevent access or modification by other users.

Warning

Basic measures are taken to make posted data be visible only to developers with access to ceph.com infrastructure. However, users should think twice and/or take appropriate precautions before posting potentially sensitive data (for example, logs or data directories that contain Ceph secrets).

Options

-d *description*, **--description** *description*
Add a short description for the upload. This is a good opportunity to reference a bug number. There is no default value.

-u *user*
Set the user metadata for the upload. This defaults to *whoami*'@'*hostname -f*.

Examples

To upload a single log:

```
ceph-post-file /var/log/ceph/ceph-mon.`hostname`.log
```

To upload several directories:

```
ceph-post-file -d 'mon data directories' /var/log/ceph/mon/*
```

Availability

ceph-post-file is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

See also

ceph(8), ceph-debugpack(8),

4.5 Troubleshooting

Ceph is still on the leading edge, so you may encounter situations that require you to examine your configuration, modify your logging output, troubleshoot monitors and OSDs, profile memory and CPU usage, and reach out to the Ceph community for help.

4.5.1 The Ceph Community

The Ceph community is an excellent source of information and help. For operational issues with Ceph releases we recommend you subscribe to the ceph-users email list. When you no longer want to receive emails, you can unsubscribe from the ceph-users email list.

You may also subscribe to the ceph-devel email list. You should do so if your issue is:

- Likely related to a bug
- Related to a development release package
- Related to a development testing package
- Related to your own builds

If you no longer want to receive emails from the `ceph-devel` email list, you may unsubscribe from the ceph-devel email list.

Tip: The Ceph community is growing rapidly, and community members can help you if you provide them with detailed information about your problem. You can attach your ceph configuration file, log files, CRUSH map, and other details (e.g., `ceph osd tree`) to help people understand your issues.

4.5.2 Logging and Debugging

Typically, when you add debugging to your Ceph configuration, you do so at runtime. You can also add Ceph debug logging to your Ceph configuration file if you are encountering issues when starting your cluster. You may view Ceph log files under `/var/log/ceph` (the default location).

Tip: When debug output slows down your system, the latency can hide race conditions.

Logging is resource intensive. If you are encountering a problem in a specific area of your cluster, enable logging for that area of the cluster. For example, if your OSDs are running fine, but your metadata servers are not, you should start by enabling debug logging for the specific metadata server instance(s) giving you trouble. Enable logging for each subsystem as needed.

Important: Verbose logging can generate over 1GB of data per hour. If your OS disk reaches its capacity, the node will stop working.

If you enable or increase the rate of Ceph logging, ensure that you have sufficient disk space on your OS disk. See *Accelerating Log Rotation* for details on rotating log files. When your system is running well, remove unnecessary debugging settings to ensure your cluster runs optimally. Logging debug output messages is relatively slow, and a waste of resources when operating your cluster.

See *Subsystem, Log and Debug Settings* for details on available settings.

Runtime

If you would like to see the configuration settings at runtime, you must log in to a host with a running daemon and execute the following:

```
ceph --admin-daemon {/path/to/admin/socket} config show | less
ceph --admin-daemon /var/run/ceph/ceph-osd.0.asok config show | less
```

To activate Ceph's debugging output (*i.e.*, `dout()`) at runtime, use the `ceph tell` command to inject arguments into the runtime configuration:

```
ceph tell {daemon-type}.{daemon id or *} injectargs --{name} {value} [--{name} {value}]
```

Replace `{daemon-type}` with one of `osd`, `mon` or `mds`. You may apply the runtime setting to all daemons of a particular type with `*`, or specify a specific daemon's ID (i.e., its number or letter). For example, to increase debug logging for a `ceph-osd` daemon named `osd.0`, execute the following:

```
ceph tell osd.0 injectargs --debug-osd 0/5
```

The `ceph tell` command goes through the monitors. If you cannot bind to the monitor, you can still make the change by logging into the host of the daemon whose configuration you'd like to change using `ceph --admin-daemon`. For example:

```
sudo ceph --admin-daemon /var/run/ceph/ceph-osd.0.asok config set debug_osd 0/5
```

See *Subsystem, Log and Debug Settings* for details on available settings.

Boot Time

To activate Ceph's debugging output (*i.e.*, `dout()`) at boot time, you must add settings to your Ceph configuration file. Subsystems common to each daemon may be set under `[global]` in your configuration file. Subsystems for particular daemons are set under the daemon section in your configuration file (*e.g.*, `[mon]`, `[osd]`, `[mds]`). For example:

```
[global]
        debug ms = 1/5

[mon]
        debug mon = 20
        debug paxos = 1/5
        debug auth = 2
```

```
[osd]
        debug osd = 1/5
        debug filestore = 1/5
        debug journal = 1
        debug monc = 5/20

[mds]
        debug mds = 1
        debug mds balancer = 1
        debug mds log = 1
        debug mds migrator = 1
```

See *Subsystem, Log and Debug Settings* for details.

Accelerating Log Rotation

If your OS disk is relatively full, you can accelerate log rotation by modifying the Ceph log rotation file at `/etc/logrotate.d/ceph`. Add a size setting after the rotation frequency to accelerate log rotation (via cronjob) if your logs exceed the size setting. For example, the default setting looks like this:

```
rotate 7
weekly
compress
sharedscripts
```

Modify it by adding a `size` setting.

```
rotate 7
weekly
size 500M
compress
sharedscripts
```

Then, start the crontab editor for your user space.

```
crontab -e
```

Finally, add an entry to check the `etc/logrotate.d/ceph` file.

```
30 * * * * /usr/sbin/logrotate /etc/logrotate.d/ceph >/dev/null 2>&1
```

The preceding example checks the `etc/logrotate.d/ceph` file every 30 minutes.

Valgrind

Debugging may also require you to track down memory and threading issues. You can run a single daemon, a type of daemon, or the whole cluster with Valgrind. You should only use Valgrind when developing or debugging Ceph. Valgrind is computationally expensive, and will slow down your system otherwise. Valgrind messages are logged to `stderr`.

Subsystem, Log and Debug Settings

In most cases, you will enable debug logging output via subsystems.

Ceph Subsystems

Each subsystem has a logging level for its output logs, and for its logs in-memory. You may set different values for each of these subsystems by setting a log file level and a memory level for debug logging. Ceph's logging levels operate on a scale of `1` to `20`, where `1` is terse and `20` is verbose. In general, the logs in-memory are not sent to the output log unless:

- a fatal signal is raised or
- an `assert` in source code is triggered or
- upon requested. Please consult document on admin socket for more details.

A debug logging setting can take a single value for the log level and the memory level, which sets them both as the same value. For example, if you specify `debug ms = 5`, Ceph will treat it as a log level and a memory level of `5`. You may also specify them separately. The first setting is the log level, and the second setting is the memory level. You must separate them with a forward slash (/). For example, if you want to set the `ms` subsystem's debug logging level to `1` and its memory level to `5`, you would specify it as `debug ms = 1/5`. For example:

```
debug {subsystem} = {log-level}/{memory-level}
#for example
debug mds log = 1/20
```

The following table provides a list of Ceph subsystems and their default log and memory levels. Once you complete your logging efforts, restore the subsystems to their default level or to a level suitable for normal operations.

Subsystem	Log Level	Memory Level
`default`	0	5
`lockdep`	0	5
`context`	0	5
`crush`	1	5
`mds`	1	5
`mds balancer`	1	5
`mds locker`	1	5
`mds log`	1	5
`mds log expire`	1	5
`mds migrator`	1	5
`buffer`	0	0
`timer`	0	5
`filer`	0	5
`objecter`	0	0
`rados`	0	5
`rbd`	0	5
`journaler`	0	5
`objectcacher`	0	5
`client`	0	5
`osd`	0	5
`optracker`	0	5
`objclass`	0	5
`filestore`	1	5
`journal`	1	5
`ms`	0	5
`mon`	1	5
`monc`	0	5

Continued on next page

Table 4.2 – continued from previous page

Subsystem	Log Level	Memory Level
`paxos`	0	5
`tp`	0	5
`auth`	1	5
`finisher`	1	5
`heartbeatmap`	1	5
`perfcounter`	1	5
`rgw`	1	5
`javaclient`	1	5
`asok`	1	5
`throttle`	1	5

Logging Settings

Logging and debugging settings are not required in a Ceph configuration file, but you may override default settings as needed. Ceph supports the following settings:

`log file`

Description The location of the logging file for your cluster.

Type String

Required No

Default `/var/log/ceph/$cluster-$name.log`

`log max new`

Description The maximum number of new log files.

Type Integer

Required No

Default `1000`

`log max recent`

Description The maximum number of recent events to include in a log file.

Type Integer

Required No

Default `1000000`

`log to stderr`

Description Determines if logging messages should appear in `stderr`.

Type Boolean

Required No

Default `true`

`err to stderr`

Description Determines if error messages should appear in `stderr`.

Type Boolean

Required No

Default `true`

`log to syslog`

Description Determines if logging messages should appear in `syslog`.

Type Boolean

Required No

Default `false`

`err to syslog`

Description Determines if error messages should appear in `syslog`.

Type Boolean

Required No

Default `false`

`log flush on exit`

Description Determines if Ceph should flush the log files after exit.

Type Boolean

Required No

Default `true`

`clog to monitors`

Description Determines if `clog` messages should be sent to monitors.

Type Boolean

Required No

Default `true`

`clog to syslog`

Description Determines if `clog` messages should be sent to syslog.

Type Boolean

Required No

Default `false`

`mon cluster log to syslog`

Description Determines if the cluster log should be output to the syslog.

Type Boolean

Required No

Default `false`

`mon cluster log file`

Description The location of the cluster's log file.

Type String

Required No

Default `/var/log/ceph/$cluster.log`

OSD

`osd debug drop ping probability`

Description ?

Type Double

Required No

Default 0

`osd debug drop ping duration`

Description

Type Integer

Required No

Default 0

`osd debug drop pg create probability`

Description

Type Integer

Required No

Default 0

`osd debug drop pg create duration`

Description ?

Type Double

Required No

Default 1

`osd preserve trimmed log`

Description Preserves trimmed logs after trimming.

Type Boolean

Required No

Default `false`

`osd tmapput sets uses tmap`

Description Uses `tmap`. For debug only.

Type Boolean

Required No

Default `false`

`osd min pg log entries`

Description The minimum number of log entries for placement groups.

Type 32-bit Unsigned Integer

Required No

Default 1000

`osd op log threshold`

Description How many op log messages to show up in one pass.

Type Integer

Required No

Default 5

Filestore

`filestore debug omap check`

Description Debugging check on synchronization. This is an expensive operation.

Type Boolean

Required No

Default 0

MDS

`mds debug scatterstat`

Description Ceph will assert that various recursive stat invariants are true (for developers only).

Type Boolean

Required No

Default `false`

`mds debug frag`

Description Ceph will verify directory fragmentation invariants when convenient (developers only).

Type Boolean

Required No

Default `false`

`mds debug auth pins`

Description The debug auth pin invariants (for developers only).

Type Boolean

Required No

Default `false`

`mds debug subtrees`

Description The debug subtree invariants (for developers only).

Type Boolean

Required No

Default `false`

RADOS Gateway

`rgw log nonexistent bucket`

Description Should we log a non-existent buckets?

Type Boolean

Required No

Default `false`

`rgw log object name`

Description Should an object's name be logged. // man date to see codes (a subset are supported)

Type String

Required No

Default `%Y-%m-%d-%H-%i-%n`

`rgw log object name utc`

Description Object log name contains UTC?

Type Boolean

Required No

Default `false`

`rgw enable ops log`

Description Enables logging of every RGW operation.

Type Boolean

Required No

Default `true`

`rgw enable usage log`

Description Enable logging of RGW's bandwidth usage.

Type Boolean

Required No

Default `true`

`rgw usage log flush threshold`

Description Threshold to flush pending log data.

Type Integer

Required No

Default `1024`

`rgw usage log tick interval`

Description Flush pending log data every `s` seconds.

Type Integer

Required No

Default 30

`rgw intent log object name`

Description

Type String

Required No

Default `%Y-%m-%d-%i-%n`

`rgw intent log object name utc`

Description Include a UTC timestamp in the intent log object name.

Type Boolean

Required No

Default `false`

4.5.3 Troubleshooting Monitors

When a cluster encounters monitor-related troubles there's a tendency to panic, and some times with good reason. You should keep in mind that losing a monitor, or a bunch of them, don't necessarily mean that your cluster is down, as long as a majority is up, running and with a formed quorum. Regardless of how bad the situation is, the first thing you should do is to calm down, take a breath and try answering our initial troubleshooting script.

Initial Troubleshooting

Are the monitors running?

First of all, we need to make sure the monitors are running. You would be amazed by how often people forget to run the monitors, or restart them after an upgrade. There's no shame in that, but let's try not losing a couple of hours chasing an issue that is not there.

Are you able to connect to the monitor's servers?

Doesn't happen often, but sometimes people do have `iptables` rules that block accesses to monitor servers or monitor ports. Usually leftovers from monitor stress-testing that were forgotten at some point. Try ssh'ing into the server and, if that succeeds, try connecting to the monitor's port using you tool of choice (telnet, nc,...).

Does ''ceph -s'' run and obtain a reply from the cluster?

If the answer is yes then your cluster is up and running. One thing you can take for granted is that the monitors will only answer to a `status` request if there is a formed quorum.

If `ceph -s` blocked however, without obtaining a reply from the cluster or showing a lot of `fault` messages, then it is likely that your monitors are either down completely or just a portion is up – a portion that is not enough to form a quorum (keep in mind that a quorum if formed by a majority of monitors).

What if ''ceph -s'' doesn't finish?

If you haven't gone through all the steps so far, please go back and do.

For those running on Emperor 0.72-rc1 and forward, you will be able to contact each monitor individually asking them for their status, regardless of a quorum being formed. This an be achieved using `ceph ping mon.ID`, ID being the monitor's identifier. You should perform this for each monitor in the cluster. In section *Understanding mon_status* we will explain how to interpret the output of this command.

For the rest of you who don't tread on the bleeding edge, you will need to ssh into the server and use the monitor's admin socket. Please jump to *Using the monitor's admin socket*.

For other specific issues, keep on reading.

Using the monitor's admin socket

The admin socket allows you to interact with a given daemon directly using a Unix socket file. This file can be found in your monitor's `run` directory. By default, the admin socket will be kept in `/var/run/ceph/ceph-mon.ID.asok` but this can vary if you defined it otherwise. If you don't find it there, please check your `ceph.conf` for an alternative path or run:

```
ceph-conf --name mon.ID --show-config-value admin_socket
```

Please bear in mind that the admin socket will only be available while the monitor is running. When the monitor is properly shutdown, the admin socket will be removed. If however the monitor is not running and the admin socket still persists, it is likely that the monitor was improperly shutdown. Regardless, if the monitor is not running, you will not be able to use the admin socket, with `ceph` likely returning `Error 111:  Connection Refused`.

Accessing the admin socket is as simple as telling the `ceph` tool to use the `asok` file. In pre-Dumpling Ceph, this can be achieved by:

```
ceph --admin-daemon /var/run/ceph/ceph-mon.ID.asok <command>
```

while in Dumpling and beyond you can use the alternate (and recommended) format:

```
ceph daemon mon.ID <command>
```

Using `help` as the command to the `ceph` tool will show you the supported commands available through the admin socket. Please take a look at `config get`, `config show`, `mon_status` and `quorum_status`, as those can be enlightening when troubleshooting a monitor.

Understanding mon_status

`mon_status` can be obtained through the `ceph` tool when you have a formed quorum, or via the admin socket if you don't. This command will output a multitude of information about the monitor, including the same output you would get with `quorum_status`.

Take the following example of `mon_status`:

```
{ "name": "c",
  "rank": 2,
  "state": "peon",
  "election_epoch": 38,
  "quorum": [
        1,
        2],
  "outside_quorum": [],
  "extra_probe_peers": [],
  "sync_provider": [],
  "monmap": { "epoch": 3,
      "fsid": "5c4e9d53-e2e1-478a-8061-f543f8be4cf8",
      "modified": "2013-10-30 04:12:01.945629",
      "created": "2013-10-29 14:14:41.914786",
      "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": "127.0.0.1:6795\/0"}]}}
```

A couple of things are obvious: we have three monitors in the monmap (*a*, *b* and *c*), the quorum is formed by only two monitors, and *c* is in the quorum as a *peon*.

Which monitor is out of the quorum?

The answer would be **a**.

Why?

Take a look at the `quorum` set. We have two monitors in this set: *1* and *2*. These are not monitor names. These are monitor ranks, as established in the current monmap. We are missing the monitor with rank 0, and according to the monmap that would be `mon.a`.

By the way, how are ranks established?

Ranks are (re)calculated whenever you add or remove monitors and follow a simple rule: the **greater** the `IP:PORT` combination, the **lower** the rank is. In this case, considering that `127.0.0.1:6789` is lower than all the remaining `IP:PORT` combinations, `mon.a` has rank 0.

Most Common Monitor Issues

Have Quorum but at least one Monitor is down

When this happens, depending on the version of Ceph you are running, you should be seeing something similar to:

```
$ ceph health detail
[snip]
mon.a (rank 0) addr 127.0.0.1:6789/0 is down (out of quorum)
```

How to troubleshoot this?

First, make sure `mon.a` is running.

Second, make sure you are able to connect to `mon.a`'s server from the other monitors' servers. Check the ports as well. Check `iptables` on all your monitor nodes and make sure you're not dropping/rejecting connections.

If this initial troubleshooting doesn't solve your problems, then it's time to go deeper.

First, check the problematic monitor's `mon_status` via the admin socket as explained in *Using the monitor's admin socket* and *Understanding mon_status*.

Considering the monitor is out of the quorum, its state should be one of `probing`, `electing` or `synchronizing`. If it happens to be either `leader` or `peon`, then the monitor believes to be in quorum, while the remaining cluster is sure it is not; or maybe it got into the quorum while we were troubleshooting the monitor, so check you `ceph -s` again just to make sure. Proceed if the monitor is not yet in the quorum.

What if the state is `probing`?

This means the monitor is still looking for the other monitors. Every time you start a monitor, the monitor will stay in this state for some time while trying to find the rest of the monitors specified in the `monmap`. The time a monitor will spend in this state can vary. For instance, when on a single-monitor cluster, the monitor will pass through the probing state almost instantaneously, since there are no other monitors around. On a multi-monitor cluster, the monitors will stay in this state until they find enough monitors to form a quorum – this means that if you have 2 out of 3 monitors down, the one remaining monitor will stay in this state indefinitively until you bring one of the other monitors up.

If you have a quorum, however, the monitor should be able to find the remaining monitors pretty fast, as long as they can be reached. If your monitor is stuck probing and you've gone through with all the communication troubleshooting, then there is a fair chance that the monitor is trying to reach the other monitors on a wrong address. `mon_status` outputs the `monmap` known to the monitor: check if the other monitor's locations match reality. If they don't, jump to *Recovering a Monitor's Broken monmap*; if they do, then it may be related to severe clock skews amongst the monitor nodes and you should refer to *Clock Skews* first, but if that doesn't solve your problem then it is the time to prepare some logs and reach out to the community (please refer to *Preparing your logs* on how to best prepare your logs).

What if state is `electing`?

This means the monitor is in the middle of an election. These should be fast to complete, but at times the monitors can get stuck electing. This is usually a sign of a clock skew among the monitor nodes; jump to *Clock Skews* for more infos on that. If all your clocks are properly synchronized, it is best if you prepare some logs and reach out to the community. This is not a state that is likely to persist and aside from (*really*) old bugs there isn't an obvious reason besides clock skews on why this would happen.

What if state is `synchronizing`?

This means the monitor is synchronizing with the rest of the cluster in order to join the quorum. The synchronization process is as faster as smaller your monitor store is, so if you have a big store it may take a while. Don't worry, it should be finished soon enough.

However, if you notice that the monitor jumps from `synchronizing` to `electing` and then back to `synchronizing`, then you do have a problem: the cluster state is advancing (i.e., generating new maps) way too fast for the synchronization process to keep up. This used to be a thing in early Cuttlefish, but since then the synchronization process was quite refactored and enhanced to avoid just this sort of behavior. If this happens in later versions let us know. And bring some logs (see *Preparing your logs*).

What if state is `leader` or `peon`?

This should not happen. There is a chance this might happen however, and it has a lot to do with clock skews – see *Clock Skews*. If you're not suffering from clock skews, then please prepare your logs (see *Preparing your logs*) and reach out to us.

Recovering a Monitor's Broken monmap

This is how a `monmap` usually looks like, depending on the number of monitors:

```
epoch 3
fsid 5c4e9d53-e2e1-478a-8061-f543f8be4cf8
last_changed 2013-10-30 04:12:01.945629
created 2013-10-29 14:14:41.914786
0: 127.0.0.1:6789/0 mon.a
1: 127.0.0.1:6790/0 mon.b
2: 127.0.0.1:6795/0 mon.c
```

This may not be what you have however. For instance, in some versions of early Cuttlefish there was this one bug that could cause your `monmap` to be nullified. Completely filled with zeros. This means that not even `monmaptool` would be able to read it because it would find it hard to make sense of only-zeros. Some other times, you may end up with a monitor with a severely outdated monmap, thus being unable to find the remaining monitors (e.g., say `mon.c` is down; you add a new monitor `mon.d`, then remove `mon.a`, then add a new monitor `mon.e` and remove `mon.b`; you will end up with a totally different monmap from the one `mon.c` knows).

In this sort of situations, you have two possible solutions:

Scrap the monitor and create a new one

You should only take this route if you are positive that you won't lose the information kept by that monitor; that you have other monitors and that they are running just fine so that your new monitor is able to synchronize from the remaining monitors. Keep in mind that destroying a monitor, if there are no other copies of its contents, may lead to loss of data.

Inject a monmap into the monitor

Usually the safest path. You should grab the monmap from the remaining monitors and inject it into the monitor with the corrupted/lost monmap.

These are the basic steps:

1. Is there a formed quorum? If so, grab the monmap from the quorum:

```
$ ceph mon getmap -o /tmp/monmap
```

2. No quorum? Grab the monmap directly from another monitor (this assumes the monitor you're grabbing the monmap from has id ID-FOO and has been stopped):

```
$ ceph-mon -i ID-FOO --extract-monmap /tmp/monmap
```

3. Stop the monitor you're going to inject the monmap into.
4. Inject the monmap:

```
$ ceph-mon -i ID --inject-monmap /tmp/monmap
```

5. Start the monitor

Please keep in mind that the ability to inject monmaps is a powerful feature that can cause havoc with your monitors if misused as it will overwrite the latest, existing monmap kept by the monitor.

Clock Skews

Monitors can be severely affected by significant clock skews across the monitor nodes. This usually translates into weird behavior with no obvious cause. To avoid such issues, you should run a clock synchronization tool on your monitor nodes.

What's the maximum tolerated clock skew?

By default the monitors will allow clocks to drift up to `0.05 seconds`.

Can I increase the maximum tolerated clock skew?

This value is configurable via the `mon-clock-drift-allowed` option, and although you *CAN* it doesn't mean you *SHOULD*. The clock skew mechanism is in place because clock skewed monitor may not properly behave. We, as developers and QA afficcionados, are comfortable with the current default value, as it will alert the user before the monitors get out hand. Changing this value without testing it first may cause unforeseen effects on the stability of the monitors and overall cluster healthiness, although there is no risk of dataloss.

How do I know there's a clock skew?

The monitors will warn you in the form of a `HEALTH_WARN`. `ceph health detail` should show something in the form of:

```
mon.c addr 10.10.0.1:6789/0 clock skew 0.08235s > max 0.05s (latency 0.0045s)
```

That means that `mon.c` has been flagged as suffering from a clock skew.

What should I do if there's a clock skew?

> Synchronize your clocks. Running an NTP client may help. If you are already using one and you hit this sort of issues, check if you are using some NTP server remote to your network and consider hosting your own NTP server on your network. This last option tends to reduce the amount of issues with monitor clock skews.

Client Can't Connect or Mount

Check your IP tables. Some OS install utilities add a `REJECT` rule to `iptables`. The rule rejects all clients trying to connect to the host except for `ssh`. If your monitor host's IP tables have such a `REJECT` rule in place, clients connecting from a separate node will fail to mount with a timeout error. You need to address `iptables` rules that reject clients trying to connect to Ceph daemons. For example, you would need to address rules that look like this appropriately:

```
REJECT all -- anywhere anywhere reject-with icmp-host-prohibited
```

You may also need to add rules to IP tables on your Ceph hosts to ensure that clients can access the ports associated with your Ceph monitors (i.e., port 6789 by default) and Ceph OSDs (i.e., 6800 et. seq. by default). For example:

```
iptables -A INPUT -m multiport -p tcp -s {ip-address}/{netmask} --dports 6789,6800:6810 -j ACCEPT
```

Everything Failed! Now What?

Reaching out for help

You can find us on IRC at #ceph and #ceph-devel at OFTC (server irc.oftc.net) and on `ceph-devel@vger.kernel.org` and `ceph-users@lists.ceph.com`. Make sure you have grabbed your logs and have them ready if someone asks: the faster the interaction and lower the latency in response, the better chances everyone's time is optimized.

Preparing your logs

Monitor logs are, by default, kept in `/var/log/ceph/ceph-mon.FOO.log*`. We may want them. However, your logs may not have the necessary information. If you don't find your monitor logs at their default location, you can check where they should be by running:

```
ceph-conf --name mon.FOO --show-config-value log_file
```

The amount of information in the logs are subject to the debug levels being enforced by your configuration files. If you have not enforced a specific debug level then Ceph is using the default levels and your logs may not contain important information to track down you issue. A first step in getting relevant information into your logs will be to raise debug levels. In this case we will be interested in the information from the monitor. Similarly to what happens on other components, different parts of the monitor will output their debug information on different subsystems.

You will have to raise the debug levels of those subsystems more closely related to your issue. This may not be an easy task for someone unfamiliar with troubleshooting Ceph. For most situations, setting the following options on your monitors will be enough to pinpoint a potential source of the issue:

```
debug mon = 10
debug ms = 1
```

If we find that these debug levels are not enough, there's a chance we may ask you to raise them or even define other debug subsystems to obtain infos from – but at least we started off with some useful information, instead of a massively empty log without much to go on with.

Do I need to restart a monitor to adjust debug levels?

No. You may do it in one of two ways:

You have quorum

Either inject the debug option into the monitor you want to debug:

```
ceph tell mon.FOO injectargs --debug_mon 10/10
```

or into all monitors at once:

```
ceph tell mon.* injectargs --debug_mon 10/10
```

No quourm

Use the monitor's admin socket and directly adjust the configuration options:

```
ceph daemon mon.FOO config set debug_mon 10/10
```

Going back to default values is as easy as rerunning the above commands using the debug level `1/10` instead. You can check your current values using the admin socket and the following commands:

```
ceph daemon mon.FOO config show
```

or:

```
ceph daemon mon.FOO config get 'OPTION_NAME'
```

Reproduced the problem with appropriate debug levels. Now what?

Ideally you would send us only the relevant portions of your logs. We realise that figuring out the corresponding portion may not be the easiest of tasks. Therefore, we won't hold it to you if you provide the full log, but common sense should be employed. If your log has hundreds of thousands of lines, it may get tricky to go through the whole thing, specially if we are not aware at which point, whatever your issue is, happened. For instance, when reproducing, keep in mind to write down current time and date and to extract the relevant portions of your logs based on that.

Finally, you should reach out to us on the mailing lists, on IRC or file a new issue on the tracker.

4.5.4 Troubleshooting OSDs

Before troubleshooting your OSDs, check your monitors and network first. If you execute `ceph health` or `ceph -s` on the command line and Ceph returns a health status, the return of a status means that the monitors have a quorum. If you don't have a monitor quorum or if there are errors with the monitor status, address the monitor issues first. Check your networks to ensure they are running properly, because networks may have a significant impact on OSD operation and performance.

Obtaining Data About OSDs

A good first step in troubleshooting your OSDs is to obtain information in addition to the information you collected while monitoring your OSDs (e.g., `ceph osd tree`).

Ceph Logs

If you haven't changed the default path, you can find Ceph log files at `/var/log/ceph`:

```
ls /var/log/ceph
```

If you don't get enough log detail, you can change your logging level. See Logging and Debugging for details to ensure that Ceph performs adequately under high logging volume.

Admin Socket

Use the admin socket tool to retrieve runtime information. For details, list the sockets for your Ceph processes:

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

Then, execute the following, replacing `{socket-name}` with an actual socket name to show the list of available options:

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

The admin socket, among other things, allows you to:

- List your configuration at runtime
- Dump historic operations
- Dump the operation priority queue state
- Dump operations in flight
- Dump perfcounters

Display Freespace

Filesystem issues may arise. To display your filesystem's free space, execute `df`.

```
df -h
```

Execute `df --help` for additional usage.

I/O Statistics

Use iostat to identify I/O-related issues.

```
iostat -x
```

Diagnostic Messages

To retrieve diagnostic messages, use `dmesg` with `less`, `more`, `grep` or `tail`. For example:

```
dmesg | grep scsi
```

Stopping w/out Rebalancing

Periodically, you may need to perform maintenance on a subset of your cluster, or resolve a problem that affects a failure domain (e.g., a rack). If you do not want CRUSH to automatically rebalance the cluster as you stop OSDs for maintenance, set the cluster to `noout` first:

```
ceph osd set noout
```

Once the cluster is set to `noout`, you can begin stopping the OSDs within the failure domain that requires maintenance work.

```
stop ceph-osd id={num}
```

Note: Placement groups within the OSDs you stop will become `degraded` while you are addressing issues with within the failure domain.

Once you have completed your maintenance, restart the OSDs.

```
start ceph-osd id={num}
```

Finally, you must unset the cluster from `noout`.

```
ceph osd unset noout
```

OSD Not Running

Under normal circumstances, simply restarting the `ceph-osd` daemon will allow it to rejoin the cluster and recover.

An OSD Won't Start

If you start your cluster and an OSD won't start, check the following:

- **Configuration File:** If you were not able to get OSDs running from a new installation, check your configuration file to ensure it conforms (e.g., `host` not `hostname`, etc.).
- **Check Paths:** Check the paths in your configuration, and the actual paths themselves for data and journals. If you separate the OSD data from the journal data and there are errors in your configuration file or in the actual mounts, you may have trouble starting OSDs. If you want to store the journal on a block device, you should partition your journal disk and assign one partition per OSD.
- **Check Max Threadcount:** If you have a node with a lot of OSDs, you may be hitting the default maximum number of threads (e.g., usually 32k), especially during recovery. You can increase the number of threads using `sysctl` to see if increasing the maximum number of threads to the maximum possible number of threads allowed (i.e., 4194303) will help. For example:

  ```
  sysctl -w kernel.pid_max=4194303
  ```

  If increasing the maximum thread count resolves the issue, you can make it permanent by including a `kernel.pid_max` setting in the `/etc/sysctl.conf` file. For example:

  ```
  kernel.pid_max = 4194303
  ```

- **Kernel Version:** Identify the kernel version and distribution you are using. Ceph uses some third party tools by default, which may be buggy or may conflict with certain distributions and/or kernel versions (e.g., Google perftools). Check the OS recommendations to ensure you have addressed any issues related to your kernel.

- **Segment Fault:** If there is a segment fault, turn your logging up (if it isn't already), and try again. If it segment faults again, contact the ceph-devel email list and provide your Ceph configuration file, your monitor output and the contents of your log file(s).

If you cannot resolve the issue and the email list isn't helpful, you may contact Inktank for support.

An OSD Failed

When a `ceph-osd` process dies, the monitor will learn about the failure from surviving `ceph-osd` daemons and report it via the `ceph health` command:

```
ceph health
HEALTH_WARN 1/3 in osds are down
```

Specifically, you will get a warning whenever there are `ceph-osd` processes that are marked `in` and `down`. You can identify which `ceph-osds` are `down` with:

```
ceph health detail
HEALTH_WARN 1/3 in osds are down
osd.0 is down since epoch 23, last address 192.168.106.220:6800/11080
```

If there is a disk failure or other fault preventing `ceph-osd` from functioning or restarting, an error message should be present in its log file in `/var/log/ceph`.

If the daemon stopped because of a heartbeat failure, the underlying kernel file system may be unresponsive. Check `dmesg` output for disk or other kernel errors.

If the problem is a software error (failed assertion or other unexpected error), it should be reported to the ceph-devel email list.

No Free Drive Space

Ceph prevents you from writing to a full OSD so that you don't lose data. In an operational cluster, you should receive a warning when your cluster is getting near its full ratio. The `mon osd full ratio` defaults to `0.95`, or 95% of capacity before it stops clients from writing data. The `mon osd nearfull ratio` defaults to `0.85`, or 85% of capacity when it generates a health warning.

Full cluster issues usually arise when testing how Ceph handles an OSD failure on a small cluster. When one node has a high percentage of the cluster's data, the cluster can easily eclipse its nearfull and full ratio immediately. If you are testing how Ceph reacts to OSD failures on a small cluster, you should leave ample free disk space and consider temporarily lowering the `mon osd full ratio` and `mon osd nearfull ratio`.

Full `ceph-osds` will be reported by `ceph health`:

```
ceph health
HEALTH_WARN 1 nearfull osds
osd.2 is near full at 85%
```

Or:

```
ceph health
HEALTH_ERR 1 nearfull osds, 1 full osds
osd.2 is near full at 85%
osd.3 is full at 97%
```

The best way to deal with a full cluster is to add new `ceph-osds`, allowing the cluster to redistribute data to the newly available storage.

If you cannot start an OSD because it is full, you may delete some data by deleting some placement group directories in the full OSD.

Important: If you choose to delete a placement group directory on a full OSD, **DO NOT** delete the same placement group directory on another full OSD, or **YOU MAY LOSE DATA**. You **MUST** maintain at least one copy of your data on at least one OSD.

See Monitor Config Reference for additional details.

OSDs are Slow/Unresponsive

A commonly recurring issue involves slow or unresponsive OSDs. Ensure that you have eliminated other troubleshooting possibilities before delving into OSD performance issues. For example, ensure that your network(s) is working properly and your OSDs are running. Check to see if OSDs are throttling recovery traffic.

Tip: Newer versions of Ceph provide better recovery handling by preventing recovering OSDs from using up system resources so that `up` and `in` OSDs aren't available or are otherwise slow.

Networking Issues

Ceph is a distributed storage system, so it depends upon networks to peer with OSDs, replicate objects, recover from faults and check heartbeats. Networking issues can cause OSD latency and flapping OSDs. See *Flapping OSDs* for details.

Ensure that Ceph processes and Ceph-dependent processes are connected and/or listening.

```
netstat -a | grep ceph
netstat -l | grep ceph
sudo netstat -p | grep ceph
```

Check network statistics.

```
netstat -s
```

Drive Configuration

A storage drive should only support one OSD. Sequential read and sequential write throughput can bottleneck if other processes share the drive, including journals, operating systems, monitors, other OSDs and non-Ceph processes.

Ceph acknowledges writes *after* journaling, so fast SSDs are an attractive option to accelerate the response time–particularly when using the `ext4` or XFS filesystems. By contrast, the `btrfs` filesystem can write and journal simultaneously.

Note: Partitioning a drive does not change its total throughput or sequential read/write limits. Running a journal in a separate partition may help, but you should prefer a separate physical drive.

Bad Sectors / Fragmented Disk

Check your disks for bad sectors and fragmentation. This can cause total throughput to drop substantially.

Co-resident Monitors/OSDs

Monitors are generally light-weight processes, but they do lots of `fsync()`, which can interfere with other workloads, particularly if monitors run on the same drive as your OSDs. Additionally, if you run monitors on the same host as the OSDs, you may incur performance issues related to:

- Running an older kernel (pre-3.0)
- Running Argonaut with an old `glibc`
- Running a kernel with no syncfs(2) syscall.

In these cases, multiple OSDs running on the same host can drag each other down by doing lots of commits. That often leads to the bursty writes.

Co-resident Processes

Spinning up co-resident processes such as a cloud-based solution, virtual machines and other applications that write data to Ceph while operating on the same hardware as OSDs can introduce significant OSD latency. Generally, we recommend optimizing a host for use with Ceph and using other hosts for other processes. The practice of separating Ceph operations from other applications may help improve performance and may streamline troubleshooting and maintenance.

Logging Levels

If you turned logging levels up to track an issue and then forgot to turn logging levels back down, the OSD may be putting a lot of logs onto the disk. If you intend to keep logging levels high, you may consider mounting a drive to the default path for logging (i.e., `/var/log/ceph/$cluster-$name.log`).

Recovery Throttling

Depending upon your configuration, Ceph may reduce recovery rates to maintain performance or it may increase recovery rates to the point that recovery impacts OSD performance. Check to see if the OSD is recovering.

Kernel Version

Check the kernel version you are running. Older kernels may not receive new backports that Ceph depends upon for better performance.

Kernel Issues with SyncFS

Try running one OSD per host to see if performance improves. Old kernels might not have a recent enough version of `glibc` to support `syncfs(2)`.

Filesystem Issues

Currently, we recommend deploying clusters with XFS or ext4. The btrfs filesystem has many attractive features, but bugs in the filesystem may lead to performance issues.

Insufficient RAM

We recommend 1GB of RAM per OSD daemon. You may notice that during normal operations, the OSD only uses a fraction of that amount (e.g., 100-200MB). Unused RAM makes it tempting to use the excess RAM for co-resident applications, VMs and so forth. However, when OSDs go into recovery mode, their memory utilization spikes. If there is no RAM available, the OSD performance will slow considerably.

Old Requests or Slow Requests

If a `ceph-osd` daemon is slow to respond to a request, it will generate log messages complaining about requests that are taking too long. The warning threshold defaults to 30 seconds, and is configurable via the `osd op complaint time` option. When this happens, the cluster log will receive messages.

Legacy versions of Ceph complain about 'old requests':

```
osd.0 192.168.106.220:6800/18813 312 : [WRN] old request osd_op(client.5099.0:790 fatty_26485_object7
```

New versions of Ceph complain about 'slow requests':

```
{date} {osd.num} [WRN] 1 slow requests, 1 included below; oldest blocked for > 30.005692 secs
{date} {osd.num}  [WRN] slow request 30.005692 seconds old, received at {date-time}: osd_op(client.42
```

Possible causes include:

- A bad drive (check `dmesg` output)
- A bug in the kernel file system bug (check `dmesg` output)
- An overloaded cluster (check system load, iostat, etc.)
- A bug in the `ceph-osd` daemon.

Possible solutions

- Remove VMs Cloud Solutions from Ceph Hosts
- Upgrade Kernel
- Upgrade Ceph
- Restart OSDs

Flapping OSDs

We recommend using both a public (front-end) network and a cluster (back-end) network so that you can better meet the capacity requirements of object replication. Another advantage is that you can run a cluster network such that it isn't connected to the internet, thereby preventing some denial of service attacks. When OSDs peer and check heartbeats, they use the cluster (back-end) network when it's available. See Monitor/OSD Interaction for details.

However, if the cluster (back-end) network fails or develops significant latency while the public (front-end) network operates optimally, OSDs currently do not handle this situation well. What happens is that OSDs mark each other `down` on the monitor, while marking themselves `up`. We call this scenario 'flapping'.

If something is causing OSDs to 'flap' (repeatedly getting marked `down` and then `up` again), you can force the monitors to stop the flapping with:

```
ceph osd set noup      # prevent OSDs from getting marked up
ceph osd set nodown    # prevent OSDs from getting marked down
```

These flags are recorded in the osdmap structure:

```
ceph osd dump | grep flags
flags no-up,no-down
```

You can clear the flags with:

```
ceph osd unset noup
ceph osd unset nodown
```

Two other flags are supported, `noin` and `noout`, which prevent booting OSDs from being marked `in` (allocated data) or protect OSDs from eventually being marked `out` (regardless of what the current value for `mon osd down out interval` is).

Note: `noup`, `noout`, and `nodown` are temporary in the sense that once the flags are cleared, the action they were blocking should occur shortly after. The `noin` flag, on the other hand, prevents OSDs from being marked `in` on boot, and any daemons that started while the flag was set will remain that way.

4.5.5 Troubleshooting PGs

Placement Groups Never Get Clean

When you create a cluster and your cluster remains in `active`, `active+remapped` or `active+degraded` status and never achieve an `active+clean` status, you likely have a problem with your configuration.

You may need to review settings in the Pool, PG and CRUSH Config Reference and make appropriate adjustments.

As a general rule, you should run your cluster with more than one OSD and a pool size greater than 1 object replica.

One Node Cluster

Ceph no longer provides documentation for operating on a single node, because you would never deploy a system designed for distributed computing on a single node. Additionally, mounting client kernel modules on a single node containing a Ceph daemon may cause a deadlock due to issues with the Linux kernel itself (unless you use VMs for the clients). You can experiment with Ceph in a 1-node configuration, in spite of the limitations as described herein.

If you are trying to create a cluster on a single node, you must change the default of the `osd crush chooseleaf type` setting from `1` (meaning `host` or `node`) to `0` (meaning `osd`) in your Ceph configuration file before you create your monitors and OSDs. This tells Ceph that an OSD can peer with another OSD on the same host. If you are trying to set up a 1-node cluster and `osd crush chooseleaf type` is greater than `0`, Ceph will try to peer the PGs of one OSD with the PGs of another OSD on another node, chassis, rack, row, or even datacenter depending on the setting.

Tip: DO NOT mount kernel clients directly on the same node as your Ceph Storage Cluster, because kernel conflicts can arise. However, you can mount kernel clients within virtual machines (VMs) on a single node.

If you are creating OSDs using a single disk, you must create directories for the data manually first. For example:

```
mkdir /var/local/osd0 /var/local/osd1
ceph-deploy osd prepare {localhost-name}:/var/local/osd0 {localhost-name}:/var/local/osd1
ceph-deploy osd activate {localhost-name}:/var/local/osd0 {localhost-name}:/var/local/osd1
```

Fewer OSDs than Replicas

If you've brought up two OSDs to an `up` and `in` state, but you still don't see `active + clean` placement groups, you may have an `osd pool default size` set to greater than `2`.

There are a few ways to address this situation. If you want to operate your cluster in an `active + degraded` state with two replicas, you can set the `osd pool default min size` to `2` so that you can write objects in an `active + degraded` state. You may also set the `osd pool default size` setting to `2` so that you only have two stored replicas (the original and one replica), in which case the cluster should achieve an `active + clean` state.

Note: You can make the changes at runtime. If you make the changes in your Ceph configuration file, you may need to restart your cluster.

Pool Size = 1

If you have the `osd pool default size` set to `1`, you will only have one copy of the object. OSDs rely on other OSDs to tell them which objects they should have. If a first OSD has a copy of an object and there is no second copy, then no second OSD can tell the first OSD that it should have that copy. For each placement group mapped to the first OSD (see `ceph pg dump`), you can force the first OSD to notice the placement groups it needs by running:

```
ceph pg force_create_pg <pgid>
```

CRUSH Map Errors

Another candidate for placement groups remaining unclean involves errors in your CRUSH map.

Stuck Placement Groups

It is normal for placement groups to enter states like "degraded" or "peering" following a failure. Normally these states indicate the normal progression through the failure recovery process. However, if a placement group stays in one of these states for a long time this may be an indication of a larger problem. For this reason, the monitor will warn when placement groups get "stuck" in a non-optimal state. Specifically, we check for:

- `inactive` - The placement group has not been `active` for too long (i.e., it hasn't been able to service read/write requests).
- `unclean` - The placement group has not been `clean` for too long (i.e., it hasn't been able to completely recover from a previous failure).
- `stale` - The placement group status has not been updated by a `ceph-osd`, indicating that all nodes storing this placement group may be `down`.

You can explicitly list stuck placement groups with one of:

```
ceph pg dump_stuck stale
ceph pg dump_stuck inactive
ceph pg dump_stuck unclean
```

For stuck `stale` placement groups, it is normally a matter of getting the right `ceph-osd` daemons running again. For stuck `inactive` placement groups, it is usually a peering problem (see *Placement Group Down - Peering Failure*). For stuck `unclean` placement groups, there is usually something preventing recovery from completing, like unfound objects (see *Unfound Objects*);

Placement Group Down - Peering Failure

In certain cases, the `ceph-osd` *Peering* process can run into problems, preventing a PG from becoming active and usable. For example, `ceph health` might report:

```
ceph health detail
HEALTH_ERR 7 pgs degraded; 12 pgs down; 12 pgs peering; 1 pgs recovering; 6 pgs stuck unclean; 114/3
...
pg 0.5 is down+peering
pg 1.4 is down+peering
...
osd.1 is down since epoch 69, last address 192.168.106.220:6801/8651
```

We can query the cluster to determine exactly why the PG is marked `down` with:

```
ceph pg 0.5 query
```

```
{ "state": "down+peering",
  ...
  "recovery_state": [
       { "name": "Started\/Primary\/Peering\/GetInfo",
         "enter_time": "2012-03-06 14:40:16.169679",
         "requested_info_from": []},
       { "name": "Started\/Primary\/Peering",
         "enter_time": "2012-03-06 14:40:16.169659",
         "probing_osds": [
               0,
               1],
         "blocked": "peering is blocked due to down osds",
         "down_osds_we_would_probe": [
               1],
         "peering_blocked_by": [
               { "osd": 1,
                 "current_lost_at": 0,
                 "comment": "starting or marking this osd lost may let us proceed"}]},
       { "name": "Started",
         "enter_time": "2012-03-06 14:40:16.169513"}
   ]
}
```

The `recovery_state` section tells us that peering is blocked due to down `ceph-osd` daemons, specifically `osd.1`. In this case, we can start that `ceph-osd` and things will recover.

Alternatively, if there is a catastrophic failure of `osd.1` (e.g., disk failure), we can tell the cluster that it is `lost` and to cope as best it can.

Important: This is dangerous in that the cluster cannot guarantee that the other copies of the data are consistent and up to date.

To instruct Ceph to continue anyway:

```
ceph osd lost 1
```

Recovery will proceed.

Unfound Objects

Under certain combinations of failures Ceph may complain about `unfound` objects:

```
ceph health detail
HEALTH_WARN 1 pgs degraded; 78/3778 unfound (2.065%)
pg 2.4 is active+degraded, 78 unfound
```

This means that the storage cluster knows that some objects (or newer copies of existing objects) exist, but it hasn't found copies of them. One example of how this might come about for a PG whose data is on ceph-osds 1 and 2:

- 1 goes down
- 2 handles some writes, alone
- 1 comes up
- 1 and 2 repeer, and the objects missing on 1 are queued for recovery.
- Before the new objects are copied, 2 goes down.

Now 1 knows that these object exist, but there is no live `ceph-osd` who has a copy. In this case, IO to those objects will block, and the cluster will hope that the failed node comes back soon; this is assumed to be preferable to returning an IO error to the user.

First, you can identify which objects are unfound with:

```
ceph pg 2.4 list_missing [starting offset, in json]
```

```
{ "offset": { "oid": "",
     "key": "",
     "snapid": 0,
     "hash": 0,
     "max": 0},
 "num_missing": 0,
 "num_unfound": 0,
 "objects": [
    { "oid": "object 1",
      "key": "",
      "hash": 0,
      "max": 0 },
    ...
 ],
 "more": 0}
```

If there are too many objects to list in a single result, the `more` field will be true and you can query for more. (Eventually the command line tool will hide this from you, but not yet.)

Second, you can identify which OSDs have been probed or might contain data:

```
ceph pg 2.4 query
```

```
"recovery_state": [
     { "name": "Started\/Primary\/Active",
       "enter_time": "2012-03-06 15:15:46.713212",
       "might_have_unfound": [
             { "osd": 1,
               "status": "osd is down"}]},
```

In this case, for example, the cluster knows that `osd.1` might have data, but it is `down`. The full range of possible states include:

```
* already probed
* querying
* OSD is down
* not queried (yet)
```

Sometimes it simply takes some time for the cluster to query possible locations.

It is possible that there are other locations where the object can exist that are not listed. For example, if a ceph-osd is stopped and taken out of the cluster, the cluster fully recovers, and due to some future set of failures ends up with an unfound object, it won't consider the long-departed ceph-osd as a potential location to consider. (This scenario, however, is unlikely.)

If all possible locations have been queried and objects are still lost, you may have to give up on the lost objects. This, again, is possible given unusual combinations of failures that allow the cluster to learn about writes that were performed before the writes themselves are recovered. To mark the "unfound" objects as "lost":

```
ceph pg 2.5 mark_unfound_lost revert|delete
```

This the final argument specifies how the cluster should deal with lost objects.

The "delete" option will forget about them entirely.

The "revert" option (not available for erasure coded pools) will either roll back to a previous version of the object or (if it was a new object) forget about it entirely. Use this with caution, as it may confuse applications that expected the object to exist.

Homeless Placement Groups

It is possible for all OSDs that had copies of a given placement groups to fail. If that's the case, that subset of the object store is unavailable, and the monitor will receive no status updates for those placement groups. To detect this situation, the monitor marks any placement group whose primary OSD has failed as `stale`. For example:

```
ceph health
HEALTH_WARN 24 pgs stale; 3/300 in osds are down
```

You can identify which placement groups are `stale`, and what the last OSDs to store them were, with:

```
ceph health detail
HEALTH_WARN 24 pgs stale; 3/300 in osds are down
...
pg 2.5 is stuck stale+active+remapped, last acting [2,0]
...
osd.10 is down since epoch 23, last address 192.168.106.220:6800/11080
osd.11 is down since epoch 13, last address 192.168.106.220:6803/11539
osd.12 is down since epoch 24, last address 192.168.106.220:6806/11861
```

If we want to get placement group 2.5 back online, for example, this tells us that it was last managed by `osd.0` and `osd.2`. Restarting those `ceph-osd` daemons will allow the cluster to recover that placement group (and, presumably, many others).

Only a Few OSDs Receive Data

If you have many nodes in your cluster and only a few of them receive data, check the number of placement groups in your pool. Since placement groups get mapped to OSDs, a small number of placement groups will not distribute across your cluster. Try creating a pool with a placement group count that is a multiple of the number of OSDs. See Placement Groups for details. The default placement group count for pools isn't useful, but you can change it here.

Can't Write Data

If your cluster is up, but some OSDs are down and you cannot write data, check to ensure that you have the minimum number of OSDs running for the placement group. If you don't have the minimum number of OSDs running, Ceph will not allow you to write data because there is no guarantee that Ceph can replicate your data. See `osd pool default min size` in the Pool, PG and CRUSH Config Reference for details.

PGs Inconsistent

If you receive an `active + clean + inconsistent` state, this may happen due to an error during scrubbing. If the inconsistency is due to disk errors, check your disks.

You can repair the inconsistent placement group by executing:

```
ceph pg repair {placement-group-ID}
```

If you receive `active + clean + inconsistent` states periodically due to clock skew, you may consider configuring your NTP daemons on your monitor hosts to act as peers. See The Network Time Protocol and Ceph Clock Settings for additional details.

Erasure Coded PGs are not active+clean

When CRUSH fails to find enough OSDs to map to a PG, it will show as a `2147483647` which is ITEM_NONE or `no OSD found`. For instance:

```
[2,1,6,0,5,8,2147483647,7,4]
```

Not enough OSDs

If the Ceph cluster only has 8 OSDs and the erasure coded pool needs 9, that is what it will show. You can either create another erasure coded pool that requires less OSDs:

```
ceph osd erasure-code-profile set myprofile k=5 m=3
ceph osd pool create erasurepool 16 16 erasure myprofile
```

or add a new OSDs and the PG will automatically use them.

CRUSH constraints cannot be satisfied

If the cluster has enough OSDs, it is possible that the CRUSH ruleset imposes constraints that cannot be satisfied. If there are 10 OSDs on two hosts and the CRUSH rulesets require that no two OSDs from the same host are used in the same PG, the mapping may fail because only two OSD will be found. You can check the constraint by displaying the ruleset:

```
$ ceph osd crush rule ls
[
    "replicated_ruleset",
    "erasurepool"]
$ ceph osd crush rule dump erasurepool
{ "rule_id": 1,
  "rule_name": "erasurepool",
  "ruleset": 1,
  "type": 3,
  "min_size": 3,
  "max_size": 20,
  "steps": [
        { "op": "take",
          "item": -1,
          "item_name": "default"},
        { "op": "chooseleaf_indep",
          "num": 0,
          "type": "host"},
        { "op": "emit"}]}
```

You can resolve the problem by creating a new pool in which PGs are allowed to have OSDs residing on the same host with:

```
ceph osd erasure-code-profile set myprofile ruleset-failure-domain=osd
ceph osd pool create erasurepool 16 16 erasure myprofile
```

CRUSH gives up too soon

If the Ceph cluster has just enough OSDs to map the PG (for instance a cluster with a total of 9 OSDs and an erasure coded pool that requires 9 OSDs per PG), it is possible that CRUSH gives up before finding a mapping. It can be resolved by:

- lowering the erasure coded pool requirements to use less OSDs per PG (that requires the creation of another pool as erasure code profiles cannot be dynamically modified).
- adding more OSDs to the cluster (that does not require the erasure coded pool to be modified, it will become clean automatically)
- use a hand made CRUSH ruleset that tries more times to find a good mapping. It can be done by setting `set_choose_tries` to a value greater than the default.

You should first verify the problem with `crushtool` after extracting the crushmap from the cluster so your experiments do not modify the Ceph cluster and only work on a local files:

```
$ ceph osd crush rule dump erasurepool
{ "rule_name": "erasurepool",
  "ruleset": 1,
  "type": 3,
  "min_size": 3,
  "max_size": 20,
  "steps": [
        { "op": "take",
          "item": -1,
          "item_name": "default"},
        { "op": "chooseleaf_indep",
          "num": 0,
          "type": "host"},
        { "op": "emit"}]}
$ ceph osd getcrushmap > crush.map
got crush map from osdmap epoch 13
$ crushtool -i crush.map --test --show-bad-mappings \
   --rule 1 \
   --num-rep 9 \
   --min-x 1 --max-x $((1024 * 1024))
bad mapping rule 8 x 43 num_rep 9 result [3,2,7,1,2147483647,8,5,6,0]
bad mapping rule 8 x 79 num_rep 9 result [6,0,2,1,4,7,2147483647,5,8]
bad mapping rule 8 x 173 num_rep 9 result [0,4,6,8,2,1,3,7,2147483647]
```

Where `--num-rep` is the number of OSDs the erasure code crush ruleset needs, `--rule` is the value of the `ruleset` field displayed by `ceph osd crush rule dump`. The test will try mapping one million values (i.e. the range defined by `[--min-x,--max-x]`) and must display at least one bad mapping. If it outputs nothing it means all mappings are successfull and you can stop right there: the problem is elsewhere.

The crush ruleset can be edited by decompiling the crush map:

```
$ crushtool --decompile crush.map > crush.txt
```

and adding the following line to the ruleset:

```
step set_choose_tries 100
```

The relevant part of of the `crush.txt` file should look something like:

```
rule erasurepool {
        ruleset 1
        type erasure
        min_size 3
        max_size 20
        step set_chooseleaf_tries 5
        step set_choose_tries 100
        step take default
        step chooseleaf indep 0 type host
        step emit
}
```

It can then be compiled and tested again:

```
$ crushtool --compile crush.txt -o better-crush.map
```

When all mappings succeed, an histogram of the number of tries that were necessary to find all of them can be displayed with the `--show-choose-tries` option of `crushtool`:

```
$ crushtool -i better-crush.map --test --show-bad-mappings \
   --show-choose-tries \
   --rule 1 \
   --num-rep 9 \
   --min-x 1 --max-x $((1024 * 1024))
...
11:        42
12:        44
13:        54
14:        45
15:        35
16:        34
17:        30
18:        25
19:        19
20:        22
21:        20
22:        17
23:        13
24:        16
25:        13
26:        11
27:        11
28:        13
29:        11
30:        10
31:         6
32:         5
33:        10
34:         3
35:         7
```

```
36:        5
37:        2
38:        5
39:        5
40:        2
41:        5
42:        4
43:        1
44:        2
45:        2
46:        3
47:        1
48:        0
...
102:         0
103:         1
104:         0
...
```

It took 11 tries to map 42 PGs, 12 tries to map 44 PGs etc. The highest number of tries is the minimum value of `set_choose_tries` that prevents bad mappings (i.e. 103 in the above output because it did not take more than 103 tries for any PG to be mapped).

4.5.6 Memory Profiling

Ceph MON, OSD and MDS can generate heap profiles using `tcmalloc`. To generate heap profiles, ensure you have `google-perftools` installed:

```
sudo apt-get google-perftools
```

The profiler dumps output to your `log file` directory (i.e., `/var/log/ceph`). See Logging and Debugging for details. To view the profiler logs with Google's performance tools, execute the following:

```
google-pprof --text {path-to-daemon}  {log-path/filename}
```

For example:

```
$ ceph tell osd.0 heap start_profiler
$ ceph tell osd.0 heap dump
osd.0 tcmalloc heap stats:------------------------------------------------
MALLOC:        2632288 (    2.5 MiB) Bytes in use by application
MALLOC: +       499712 (    0.5 MiB) Bytes in page heap freelist
MALLOC: +       543800 (    0.5 MiB) Bytes in central cache freelist
MALLOC: +       327680 (    0.3 MiB) Bytes in transfer cache freelist
MALLOC: +      1239400 (    1.2 MiB) Bytes in thread cache freelists
MALLOC: +      1142936 (    1.1 MiB) Bytes in malloc metadata
MALLOC:   ------------
MALLOC: =      6385816 (    6.1 MiB) Actual memory used (physical + swap)
MALLOC: +            0 (    0.0 MiB) Bytes released to OS (aka unmapped)
MALLOC:   ------------
MALLOC: =      6385816 (    6.1 MiB) Virtual address space used
MALLOC:
MALLOC:            231              Spans in use
MALLOC:             56              Thread heaps in use
MALLOC:           8192              Tcmalloc page size
------------------------------------------------
Call ReleaseFreeMemory() to release freelist memory to the OS (via madvise()).
Bytes released to the OS take up virtual address space but no physical memory.
```

```
$ google-pprof --text \
               /usr/bin/ceph-osd  \
               /var/log/ceph/ceph-osd.0.profile.0001.heap
 Total: 3.7 MB
 1.9  51.1%  51.1%      1.9  51.1% ceph::log::Log::create_entry
 1.8  47.3%  98.4%      1.8  47.3% std::string::_Rep::_S_create
 0.0   0.4%  98.9%      0.0   0.6% SimpleMessenger::add_accept_pipe
 0.0   0.4%  99.2%      0.0   0.6% decode_message
 ...
```

Another heap dump on the same daemon will add another file. It is convenient to compare to a previous heap dump to show what has grown in the interval. For instance:

```
$ google-pprof --text --base out/osd.0.profile.0001.heap \
      ceph-osd out/osd.0.profile.0003.heap
 Total: 0.2 MB
 0.1  50.3%  50.3%      0.1  50.3% ceph::log::Log::create_entry
 0.1  46.6%  96.8%      0.1  46.6% std::string::_Rep::_S_create
 0.0   0.9%  97.7%      0.0  26.1% ReplicatedPG::do_op
 0.0   0.8%  98.5%      0.0   0.8% __gnu_cxx::new_allocator::allocate
```

Refer to Google Heap Profiler for additional details.

Once you have the heap profiler installed, start your cluster and begin using the heap profiler. You may enable or disable the heap profiler at runtime, or ensure that it runs continuously. For the following commandline usage, replace `{daemon-type}` with `mon`, `osd` or `mds`, and replace `{daemon-id}` with the OSD number or the MON or MDS id.

Starting the Profiler

To start the heap profiler, execute the following:

```
ceph tell {daemon-type}.{daemon-id} heap start_profiler
```

For example:

```
ceph tell osd.1 heap start_profiler
```

Alternatively the profile can be started when the daemon starts running if the `CEPH_HEAP_PROFILER_INIT=true` variable is found in the environment.

Printing Stats

To print out statistics, execute the following:

```
ceph  tell {daemon-type}.{daemon-id} heap stats
```

For example:

```
ceph tell osd.0 heap stats
```

Note: Printing stats does not require the profiler to be running and does not dump the heap allocation information to a file.

Dumping Heap Information

To dump heap information, execute the following:

```
ceph tell {daemon-type}.{daemon-id} heap dump
```

For example:

```
ceph tell mds.a heap dump
```

Note: Dumping heap information only works when the profiler is running.

Releasing Memory

To release memory that `tcmalloc` has allocated but which is not being used by the Ceph daemon itself, execute the following:

```
ceph tell {daemon-type}{daemon-id} heap release
```

For example:

```
ceph tell osd.2 heap release
```

Stopping the Profiler

To stop the heap profiler, execute the following:

```
ceph tell {daemon-type}.{daemon-id} heap stop_profiler
```

For example:

```
ceph tell osd.0 heap stop_profiler
```

4.5.7 CPU Profiling

If you built Ceph from source and compiled Ceph for use with oprofile you can profile Ceph's CPU usage. See Installing Oprofile for details.

Initializing oprofile

The first time you use `oprofile` you need to initialize it. Locate the `vmlinux` image corresponding to the kernel you are now running.

```
ls /boot
sudo opcontrol --init
sudo opcontrol --setup --vmlinux={path-to-image} --separate=library --callgraph=6
```

Starting oprofile

To start `oprofile` execute the following command:

```
opcontrol --start
```

Once you start `oprofile`, you may run some tests with Ceph.

Stopping oprofile

To stop `oprofile` execute the following command:

```
opcontrol --stop
```

Retrieving oprofile Results

To retrieve the top `cmon` results, execute the following command:

```
opreport -gal ./cmon | less
```

To retrieve the top `cmon` results with call graphs attached, execute the following command:

```
opreport -cal ./cmon | less
```

Important: After reviewing results, you should reset `oprofile` before running it again. Resetting `oprofile` removes data from the session directory.

Resetting oprofile

To reset `oprofile`, execute the following command:

```
sudo opcontrol --reset
```

Important: You should reset `oprofile` after analyzing data so that you do not commingle results from different tests.

Most Ceph deployments use Ceph Block Devices, Ceph Object Storage and/or the Ceph Filesystem. You may also develop applications that talk directly to the Ceph Storage Cluster.

4.6 Ceph Storage Cluster APIs

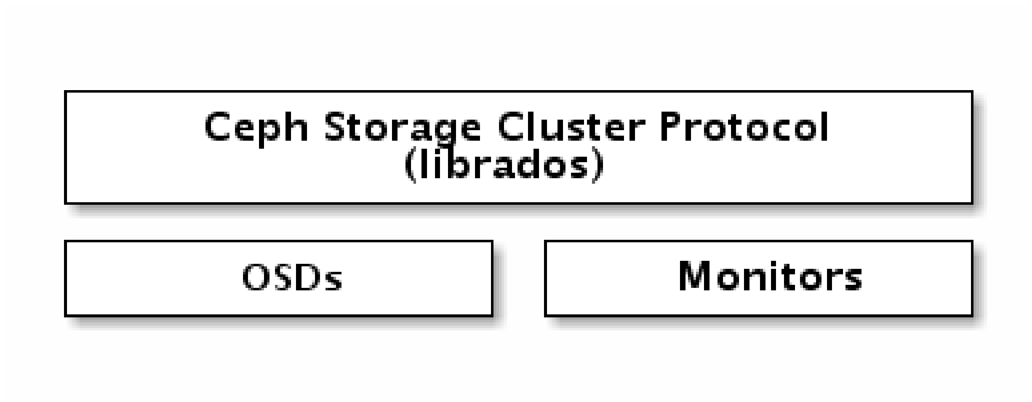
The *Ceph Storage Cluster* has a messaging layer protocol that enables clients to interact with a *Ceph Monitor* and a *Ceph OSD Daemon*. `librados` provides this functionality to *Ceph Clients* in the form of a library. All Ceph Clients either use `librados` or the same functionality encapsulated in `librados` to interact with the object store. For example, `librbd` and `libcephfs` leverage this functionality. You may use `librados` to interact with Ceph directly (e.g., an application that talks to Ceph, your own interface to Ceph, etc.).

4.6.1 Introduction to librados

The *Ceph Storage Cluster* provides the basic storage service that allows *Ceph* to uniquely deliver **object, block, and file storage** in one unified system. However, you are not limited to using the RESTful, block, or POSIX interfaces. Based upon RADOS, the `librados` API enables you to create your own interface to the Ceph Storage Cluster.

The `librados` API enables you to interact with the two types of daemons in the Ceph Storage Cluster:

- The *Ceph Monitor*, which maintains a master copy of the cluster map.
- The *Ceph OSD Daemon* (OSD), which stores data as objects on a storage node.



This guide provides a high-level introduction to using `librados`. Refer to Architecture for additional details of the Ceph Storage Cluster. To use the API, you need a running Ceph Storage Cluster. See Installation (Quick) for details.

Step 1: Getting librados

Your client application must bind with `librados` to connect to the Ceph Storage Cluster. You must install `librados` and any required packages to write applications that use `librados`. The `librados` API is written in C++, with additional bindings for C, Python, Java and PHP.

Getting librados for C/C++

To install `librados` development support files for C/C++ on Debian/Ubuntu distributions, execute the following:

```
sudo apt-get install librados-dev
```

To install `librados` development support files for C/C++ on RHEL/CentOS distributions, execute the following:

```
sudo yum install librados2-devel
```

Once you install `librados` for developers, you can find the required headers for C/C++ under `/usr/include/rados`.

```
ls /usr/include/rados
```

Getting librados for Python

The `rados.py` modules provides `librados` support to Python applications. The `librados-dev` package for Debian/Ubuntu and the `librados2-devel` package for RHEL/CentOS will install the `python-rados` package for you. You may install `python-rados` directly too.

To install `librados` development support files for Python on Debian/Ubuntu distributions, execute the following:

```
sudo apt-get install python-rados
```

To install `librados` development support files for C/C++ on RHEL/CentOS distributions, execute the following:

```
sudo yum install python-rados
```

You can find the module under `/usr/share/pyshared` on Debian systems, or under `/usr/lib/python*/site-packages` on CentOS/RHEL systems.

Getting librados for Java

To install `librados` for Java, you need to execute the following procedure:

1. Install `jna.jar`. For Debian/Ubuntu, execute:

   ```
   sudo apt-get install libjna-java
   ```

   For CentOS/RHEL, execute:

   ```
   sudo yum install jna
   ```

   The JAR files are located in `/usr/share/java`.

2. Clone the `rados-java` repository:

   ```
   git clone --recursive https://github.com/ceph/rados-java.git
   ```

3. Build the `rados-java` repository:

   ```
   cd rados-java
   ant
   ```

   The JAR file is located under `rados-java/target`.

4. Copy the JAR for RADOS to a common location (e.g., `/usr/share/java`) and ensure that it and the JNA JAR are in your JVM's classpath. For example:

   ```
   sudo cp target/rados-0.1.3.jar /usr/share/java/rados-0.1.3.jar
   sudo ln -s /usr/share/java/jna-3.2.7.jar /usr/lib/jvm/default-java/jre/lib/ext/jna-3.2.7.jar
   sudo ln -s /usr/share/java/rados-0.1.3.jar  /usr/lib/jvm/default-java/jre/lib/ext/rados-0.1.3.ja
   ```

To build the documentation, execute the following:

```
ant docs
```

Getting librados for PHP

To install the `librados` extension for PHP, you need to execute the following procedure:

1. Install php-dev. For Debian/Ubuntu, execute:

   ```
   sudo apt-get install php5-dev build-essential
   ```

   For CentOS/RHEL, execute:

   ```
   sudo yum install php-devel
   ```

2. Clone the `phprados` repository:

   ```
   git clone https://github.com/ceph/phprados.git
   ```

3. Build `phprados`:

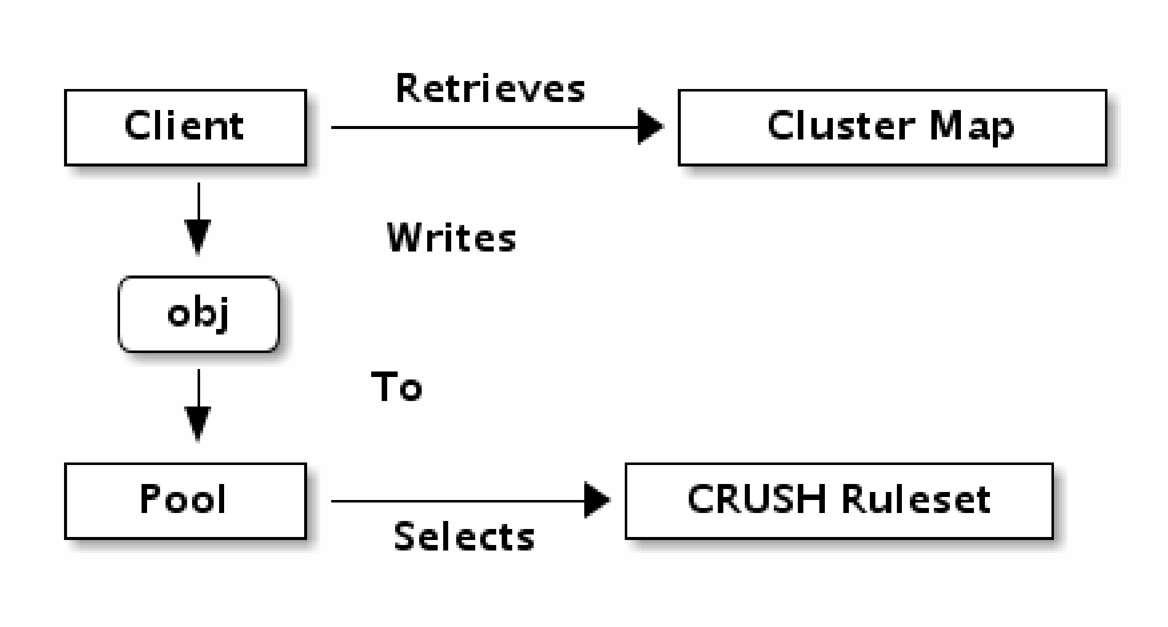
```
    cd phprados
    phpize
    ./configure
    make
    sudo make install
```

4. Enable `phprados` in php.ini by adding:

```
    extension=rados.so
```

Step 2: Configuring a Cluster Handle

A *Ceph Client*, via `librados`, interacts directly with OSDs to store and retrieve data. To interact with OSDs, the client app must invoke `librados` and connect to a Ceph Monitor. Once connected, `librados` retrieves the *Cluster Map* from the Ceph Monitor. When the client app wants to read or write data, it creates an I/O context and binds to a *pool*. The pool has an associated *ruleset* that defines how it will place data in the storage cluster. Via the I/O context, the client provides the object name to `librados`, which takes the object name and the cluster map (i.e., the topology of the cluster) and computes the placement group and OSD for locating the data. Then the client application can read or write data. The client app doesn't need to learn about the topology of the cluster directly.



The Ceph Storage Cluster handle encapsulates the client configuration, including:

- The user ID for `rados_create()` or user name for `rados_create2()` (preferred).
- The *cephx* authentication key
- The monitor ID and IP address
- Logging levels
- Debugging levels

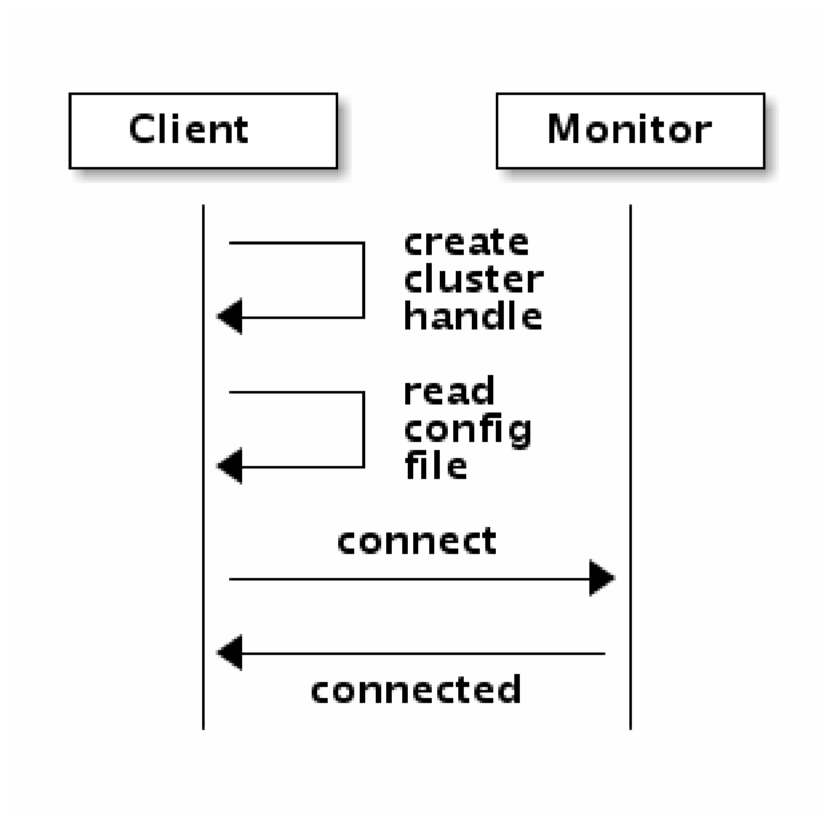
Thus, the first steps in using the cluster from your app are to 1) create a cluster handle that your app will use to connect to the storage cluster, and then 2) use that handle to connect. To connect to the cluster, the app must supply a monitor address, a username and an authentication key (cephx is enabled by default).

Tip: Talking to different Ceph Storage Clusters – or to the same cluster with different users – requires different cluster handles.

RADOS provides a number of ways for you to set the required values. For the monitor and encryption key settings, an easy way to handle them is to ensure that your Ceph configuration file contains a `keyring` path to a keyring file and at least one monitor address (e.g,. `mon host`). For example:

```
[global]
mon host = 192.168.1.1
keyring = /etc/ceph/ceph.client.admin.keyring
```

Once you create the handle, you can read a Ceph configuration file to configure the handle. You can also pass arguments to your app and parse them with the function for parsing command line arguments (e.g., `rados_conf_parse_argv()`), or parse Ceph environment variables (e.g., `rados_conf_parse_env()`). Some wrappers may not implement convenience methods, so you may need to implement these capabilities. The following diagram provides a high-level flow for the initial connection.



Once connected, your app can invoke functions that affect the whole cluster with only the cluster handle. For example, once you have a cluster handle, you can:

- Get cluster statistics
- Use Pool Operation (exists, create, list, delete)
- Get and set the configuration

One of the powerful features of Ceph is the ability to bind to different pools. Each pool may have a different number of placement groups, object replicas and replication strategies. For example, a pool could be set up as a “hot” pool that uses SSDs for frequently used objects or a “cold” pool that uses erasure coding.

The main difference in the various `librados` bindings is between C and the object-oriented bindings for C++, Java and Python. The object-oriented bindings use objects to represent cluster handles, IO Contexts, iterators, exceptions, etc.

C Example

For C, creating a simple cluster handle using the `admin` user, configuring it and connecting to the cluster might look something like this:

```
#include <stdio.h>
#include <string.h>
#include <rados/librados.h>

int main (int argc, char argv**)
{

        /* Declare the cluster handle and required arguments. */
        rados_t cluster;
        char cluster_name[] = "ceph";
        char user_name[] = "client.admin";
        uint64_t flags;

        /* Initialize the cluster handle with the "ceph" cluster name and the "client.admin" user */
        int err;
        err = rados_create2(&cluster, cluster_name, user_name, flags);

        if (err < 0) {
                fprintf(stderr, "%s: Couldn't create the cluster handle! %s\n", argv[0], strerror(-er
                exit(EXIT_FAILURE);
        } else {
                printf("\nCreated a cluster handle.\n");
        }


        /* Read a Ceph configuration file to configure the cluster handle. */
        err = rados_conf_read_file(cluster, "/etc/ceph/ceph.conf");
        if (err < 0) {
                fprintf(stderr, "%s: cannot read config file: %s\n", argv[0], strerror(-err));
                exit(EXIT_FAILURE);
        } else {
                printf("\nRead the config file.\n");
        }

        /* Read command line arguments */
        err = rados_conf_parse_argv(cluster, argc, argv);
        if (err < 0) {
                fprintf(stderr, "%s: cannot parse command line arguments: %s\n", argv[0], strerror(-e
                exit(EXIT_FAILURE);
        } else {
                printf("\nRead the command line arguments.\n");
        }

        /* Connect to the cluster */
        err = rados_connect(cluster);
        if (err < 0) {
                fprintf(stderr, "%s: cannot connect to cluster: %s\n", argv[0], strerror(-err));
                exit(EXIT_FAILURE);
        } else {
                printf("\nConnected to the cluster.\n");
        }

}
```

Compile your client and link to `librados` using `-lrados`. For example:

```
gcc ceph-client.c -lrados -o ceph-client
```

C++ Example

The Ceph project provides a C++ example in the `ceph/examples/librados` directory. For C++, a simple cluster handle using the `admin` user requires you to initialize a `librados::Rados` cluster handle object:

```
#include <iostream>
#include <string>
#include <rados/librados.hpp>

int main(int argc, const char **argv)
{

        int ret = 0;

        /* Declare the cluster handle and required variables. */
        librados::Rados cluster;
        char cluster_name[] = "ceph";
        char user_name[] = "client.admin";
        uint64_t flags;

        /* Initialize the cluster handle with the "ceph" cluster name and "client.admin" user */
        {
                ret = cluster.init2(user_name, cluster_name, flags);
                if (ret < 0) {
                        std::cerr << "Couldn't initialize the cluster handle! error " << ret << std:
                        ret = EXIT_FAILURE;
                        return 1;
                } else {
                        std::cout << "Created a cluster handle." << std::endl;
                }
        }

        /* Read a Ceph configuration file to configure the cluster handle. */
        {
                ret = cluster.conf_read_file("/etc/ceph/ceph.conf");
                if (ret < 0) {
                        std::cerr << "Couldn't read the Ceph configuration file! error " << ret << st
                        ret = EXIT_FAILURE;
                        return 1;
                } else {
                        std::cout << "Read the Ceph configuration file." << std::endl;
                }
        }

        /* Read command line arguments */
        {
                ret = cluster.conf_parse_argv(argc, argv);
                if (ret < 0) {
                        std::cerr << "Couldn't parse command line options! error " << ret << std::enc
```

```
                                ret = EXIT_FAILURE;
                                return 1;
                        } else {
                                std::cout << "Parsed command line options." << std::endl;
                        }
                }

                /* Connect to the cluster */
                {
                        ret = cluster.connect();
                        if (ret < 0) {
                                std::cerr << "Couldn't connect to cluster! error " << ret << std::endl;
                                ret = EXIT_FAILURE;
                                return 1;
                        } else {
                                std::cout << "Connected to the cluster." << std::endl;
                        }
                }

                return 0;
}
```

Compile the source; then, link `librados` using `-lrados`. For example:

```
g++ -g -c ceph-client.cc -o ceph-client.o
g++ -g ceph-client.o -lrados -o ceph-client
```

Python Example

Python uses the `admin` id and the `ceph` cluster name by default, and will read the standard `ceph.conf` file if the conffile parameter is set to the empty string. The Python binding converts C++ errors into exceptions.

```
import rados

try:
        cluster = rados.Rados(conffile='')
except TypeError as e:
        print 'Argument validation error: ', e
        raise e

print "Created cluster handle."

try:
        cluster.connect()
except Exception as e:
        print "connection error: ", e
        raise e
finally:
        print "Connected to the cluster."
```

Execute the example to verify that it connects to your cluster.

```
python ceph-client.py
```

Java Example

Java requires you to specify the user ID (`admin`) or user name (`client.admin`), and uses the `ceph` cluster name by default . The Java binding converts C++-based errors into exceptions.

```
import com.ceph.rados.Rados;
import com.ceph.rados.RadosException;

import java.io.File;

public class CephClient {
        public static void main (String args[]){

                try {
                        Rados cluster = new Rados("admin");
                        System.out.println("Created cluster handle.");

                        File f = new File("/etc/ceph/ceph.conf");
                        cluster.confReadFile(f);
                        System.out.println("Read the configuration file.");

                        cluster.connect();
                        System.out.println("Connected to the cluster.");

                } catch (RadosException e) {
                        System.out.println(e.getMessage() + ": " + e.getReturnValue());
                }
        }
}
```

Compile the source; then, run it. If you have copied the JAR to `/usr/share/java` and sym linked from your `ext` directory, you won't need to specify the classpath. For example:

```
javac CephClient.java
java CephClient
```

PHP Example

With the RADOS extension enabled in PHP you can start creating a new cluster handle very easily:

```
<?php

$r = rados_create();
rados_conf_read_file($r, '/etc/ceph/ceph.conf');
if (!rados_connect($r)) {
        echo "Failed to connect to Ceph cluster";
} else {
        echo "Successfully connected to Ceph cluster";
}
```

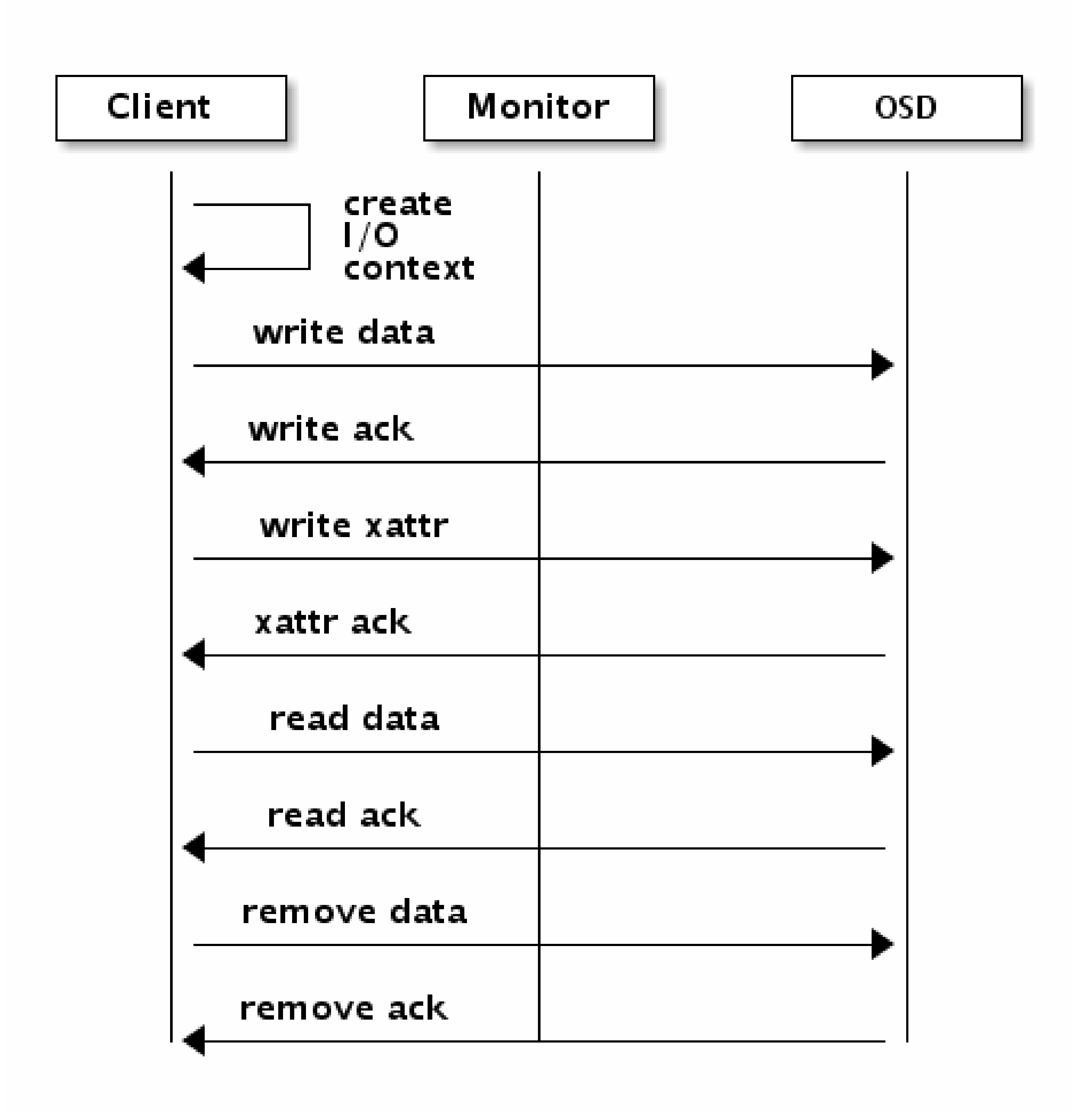
Save this as rados.php and run the code:

```
php rados.php
```

Step 3: Creating an I/O Context

Once your app has a cluster handle and a connection to a Ceph Storage Cluster, you may create an I/O Context and begin reading and writing data. An I/O Context binds the connection to a specific pool. The user must have appropriate CAPS permissions to access the specified pool. For example, a user with read access but not write access will only be able to read data. I/O Context functionality includes:

- Write/read data and extended attributes
- List and iterate over objects and extended attributes
- Snapshot pools, list snapshots, etc.



RADOS enables you to interact both synchronously and asynchronously. Once your app has an I/O Context, read/write operations only require you to know the object/xattr name. The CRUSH algorithm encapsulated in `librados` uses the cluster map to identify the appropriate OSD. OSD daemons handle the replication, as described in Smart Daemons Enable Hyperscale. The `librados` library also maps objects to placement groups, as described in Calculating PG IDs.

The following examples use the default `data` pool. However, you may also use the API to list pools, ensure they exist, or create and delete pools. For the write operations, the examples illustrate how to use synchronous mode. For the read operations, the examples illustrate how to use asynchronous mode.

Important: Use caution when deleting pools with this API. If you delete a pool, the pool and ALL DATA in the pool will be lost.

C Example

```
#include <stdio.h>
#include <string.h>
#include <rados/librados.h>

int main (int argc, const char argv**)
{
        /*
         * Continued from previous C example, where cluster handle and
         * connection are established. First declare an I/O Context.
         */

        rados_ioctx_t io;
        char *poolname = "data";

        err = rados_ioctx_create(cluster, poolname, &io);
        if (err < 0) {
                fprintf(stderr, "%s: cannot open rados pool %s: %s\n", argv[0], poolname, strerror(-e
                rados_shutdown(cluster);
                exit(EXIT_FAILURE);
        } else {
                printf("\nCreated I/O context.\n");
        }

        /* Write data to the cluster synchronously. */
        err = rados_write(io, "hw", "Hello World!", 12, 0);
        if (err < 0) {
                fprintf(stderr, "%s: Cannot write object \"hw\" to pool %s: %s\n", argv[0], poolname,
                rados_ioctx_destroy(io);
                rados_shutdown(cluster);
                exit(1);
        } else {
                printf("\nWrote \"Hello World\" to object \"hw\".\n");
        }

        char xattr[] = "en_US";
        err = rados_setxattr(io, "hw", "lang", xattr, 5);
        if (err < 0) {
                fprintf(stderr, "%s: Cannot write xattr to pool %s: %s\n", argv[0], poolname, strerro
                rados_ioctx_destroy(io);
                rados_shutdown(cluster);
                exit(1);
        } else {
                printf("\nWrote \"en_US\" to xattr \"lang\" for object \"hw\".\n");
        }

        /*
         * Read data from the cluster asynchronously.
         * First, set up asynchronous I/O completion.
         */
        rados_completion_t comp;
        err = rados_aio_create_completion(NULL, NULL, NULL, &comp);
        if (err < 0) {
                fprintf(stderr, "%s: Could not create aio completion: %s\n", argv[0], strerror(-err)
                rados_ioctx_destroy(io);
                rados_shutdown(cluster);
                exit(1);
        } else {
                printf("\nCreated AIO completion.\n");
        }

        /* Next, read data using rados_aio_read. */
        char read_res[100];
        err = rados_aio_read(io, "hw", comp, read_res, 12, 0);
        if (err < 0) {
                fprintf(stderr, "%s: Cannot read object. %s %s\n", argv[0], poolname, strerror(-err)
                rados_ioctx_destroy(io);
                rados_shutdown(cluster);
                exit(1);
        } else {
                printf("\nRead object \"hw\". The contents are:\n %s \n", read_res);
        }

        /* Wait for the operation to complete */
        rados_wait_for_complete(comp);

        /* Release the asynchronous I/O complete handle to avoid memory leaks. */
        rados_aio_release(comp);


        char xattr_res[100];
        err = rados_getxattr(io, "hw", "lang", xattr_res, 5);
        if (err < 0) {
                fprintf(stderr, "%s: Cannot read xattr. %s %s\n", argv[0], poolname, strerror(-err));
                rados_ioctx_destroy(io);
                rados_shutdown(cluster);
                exit(1);
        } else {
                printf("\nRead xattr \"lang\" for object \"hw\". The contents are:\n %s \n", xattr_re
        }

        err = rados_rmxattr(io, "hw", "lang");
        if (err < 0) {
                fprintf(stderr, "%s: Cannot remove xattr. %s %s\n", argv[0], poolname, strerror(-err
                rados_ioctx_destroy(io);
                rados_shutdown(cluster);
                exit(1);
        } else {
                printf("\nRemoved xattr \"lang\" for object \"hw\".\n");
        }

        err = rados_remove(io, "hw");
        if (err < 0) {
                fprintf(stderr, "%s: Cannot remove object. %s %s\n", argv[0], poolname, strerror(-err
                rados_ioctx_destroy(io);
                rados_shutdown(cluster);
                exit(1);
        } else {
                printf("\nRemoved object \"hw\".\n");
        }

}
```

C++ Example

```
#include <iostream>
#include <string>
#include <rados/librados.hpp>

int main(int argc, const char **argv)
{

        /* Continued from previous C++ example, where cluster handle and
         * connection are established. First declare an I/O Context.
         */

        librados::IoCtx io_ctx;
        const char *pool_name = "data";

        {
                ret = cluster.ioctx_create(pool_name, io_ctx);
                if (ret < 0) {
                        std::cerr << "Couldn't set up ioctx! error " << ret << std::endl;
                        exit(EXIT_FAILURE);
                } else {
                        std::cout << "Created an ioctx for the pool." << std::endl;
                }
        }


        /* Write an object synchronously. */
        {
                librados::bufferlist bl;
                bl.append("Hello World!");
                ret = io_ctx.write_full("hw", bl);
                if (ret < 0) {
                        std::cerr << "Couldn't write object! error " << ret << std::endl;
                        exit(EXIT_FAILURE);
                } else {
                        std::cout << "Wrote new object 'hw' " << std::endl;
                }
        }


        /*
         * Add an xattr to the object.
         */
        {
                librados::bufferlist lang_bl;
                lang_bl.append("en_US");
                ret = io_ctx.setxattr("hw", "lang", lang_bl);
                if (ret < 0) {
                        std::cerr << "failed to set xattr version entry! error "
                        << ret << std::endl;
                        exit(EXIT_FAILURE);
                } else {
                        std::cout << "Set the xattr 'lang' on our object!" << std::endl;
                }
        }


        /*
         * Read the object back asynchronously.
         */
        {
                librados::bufferlist read_buf;
                int read_len = 4194304;

                //Create I/O Completion.
                librados::AioCompletion *read_completion = librados::Rados::aio_create_completion();

                //Send read request.
                ret = io_ctx.aio_read("hw", read_completion, &read_buf, read_len, 0);
                if (ret < 0) {
                        std::cerr << "Couldn't start read object! error " << ret << std::endl;
                        exit(EXIT_FAILURE);
                }

                // Wait for the request to complete, and check that it succeeded.
                read_completion->wait_for_complete();
                ret = read_completion->get_return_value();
                if (ret < 0) {
                        std::cerr << "Couldn't read object! error " << ret << std::endl;
                        exit(EXIT_FAILURE);
                } else {
                        std::cout << "Read object hw asynchronously with contents.\n"
                        << read_buf.c_str() << std::endl;
                }
        }


        /*
         * Read the xattr.
         */
        {
                librados::bufferlist lang_res;
                ret = io_ctx.getxattr("hw", "lang", lang_res);
                if (ret < 0) {
                        std::cerr << "failed to get xattr version entry! error "
                        << ret << std::endl;
                        exit(EXIT_FAILURE);
                } else {
                        std::cout << "Got the xattr 'lang' from object hw!"
                        << lang_res.c_str() << std::endl;
                }
        }
        /*
         * Remove the xattr.
         */
        {
                ret = io_ctx.rmxattr("hw", "lang");
                if (ret < 0) {
                        std::cerr << "Failed to remove xattr! error "
                        << ret << std::endl;
                        exit(EXIT_FAILURE);
                } else {
                        std::cout << "Removed the xattr 'lang' from our object!" << std::endl;
                }
        }

        /*
         * Remove the object.
         */
        {
                ret = io_ctx.remove("hw");
                if (ret < 0) {
                        std::cerr << "Couldn't remove object! error " << ret << std::endl;
                        exit(EXIT_FAILURE);
                } else {
                        std::cout << "Removed object 'hw'." << std::endl;
                }
        }
}
```

Python Example

```
print "\n\nI/O Context and Object Operations"
print "================================="

print "\nCreating a context for the 'data' pool"
if not cluster.pool_exists('data'):
        raise RuntimeError('No data pool exists')
ioctx = cluster.open_ioctx('data')

print "\nWriting object 'hw' with contents 'Hello World!' to pool 'data'."
ioctx.write("hw", "Hello World!")
print "Writing XATTR 'lang' with value 'en_US' to object 'hw'"
ioctx.set_xattr("hw", "lang", "en_US")


print "\nWriting object 'bm' with contents 'Bonjour tout le monde!' to pool 'data'."
ioctx.write("bm", "Bonjour tout le monde!")
print "Writing XATTR 'lang' with value 'fr_FR' to object 'bm'"
ioctx.set_xattr("bm", "lang", "fr_FR")

print "\nContents of object 'hw'\n------------------------"
print ioctx.read("hw")

print "\n\nGetting XATTR 'lang' from object 'hw'"
print ioctx.get_xattr("hw", "lang")
print "\nContents of object 'bm'\n------------------------"
print ioctx.read("bm")

print "Getting XATTR 'lang' from object 'bm'"
print ioctx.get_xattr("bm", "lang")


print "\nRemoving object 'hw'"
ioctx.remove_object("hw")

print "Removing object 'bm'"
ioctx.remove_object("bm")
```

Java-Example

```
import com.ceph.rados.Rados;
import com.ceph.rados.RadosException;

import java.io.File;
import com.ceph.rados.IoCTX;

public class CephClient {
        public static void main (String args[]){

                try {
                        Rados cluster = new Rados("admin");
                        System.out.println("Created cluster handle.");

                        File f = new File("/etc/ceph/ceph.conf");
                        cluster.confReadFile(f);
                        System.out.println("Read the configuration file.");

                        cluster.connect();
                        System.out.println("Connected to the cluster.");

                        IoCTX io = cluster.ioCtxCreate("data");

                        String oidone = "hw";
                        String contentone = "Hello World!";
                        io.write(oidone, contentone);

                        String oidtwo = "bm";
                        String contenttwo = "Bonjour tout le monde!";
                        io.write(oidtwo, contenttwo);

                        String[] objects = io.listObjects();
                        for (String object: objects)
                                System.out.println(object);

                        io.remove(oidone);
                        io.remove(oidtwo);

                        cluster.ioCtxDestroy(io);

                } catch (RadosException e) {
                        System.out.println(e.getMessage() + ": " + e.getReturnValue());
                }
        }
}
```

PHP Example

```
<?php

$io = rados_ioctx_create($r, "mypool");
rados_write_full($io, "oidOne", "mycontents");
rados_remove("oidOne");
rados_ioctx_destroy($io);
```

Step 4: Closing Sessions

Once your app finishes with the I/O Context and cluster handle, the app should close the connection and shutdown the handle. For asynchronous I/O, the app should also ensure that pending asynchronous operations have completed.

C Example

```
rados_ioctx_destroy(io);
rados_shutdown(cluster);
```

C++ Example

```
io_ctx.close();
cluster.shutdown();
```

Python Example

```
print "\nClosing the connection."
ioctx.close()

print "Shutting down the handle."
cluster.shutdown()
```

PHP Example

```
rados_shutdown($r);
```

4.6.2 Librados (C)

librados provides low-level access to the RADOS service. For an overview of RADOS, see Architecture.

Example: connecting and writing an object

To use *Librados*, you instantiate a `rados_t` variable (a cluster handle) and call `rados_create()` with a pointer to it:

```
int err;
rados_t cluster;

err = rados_create(&cluster, NULL);
if (err < 0) {
        fprintf(stderr, "%s: cannot create a cluster handle: %s\n", argv[0], strerror(-err));
        exit(1);
}
```

Then you configure your `rados_t` to connect to your cluster, either by setting individual values (`rados_conf_set()`), using a configuration file (`rados_conf_read_file()`), using command line options (`rados_conf_parse_argv()`), or an environment variable (`rados_conf_parse_env()`):

```
err = rados_conf_read_file(cluster, "/path/to/myceph.conf");
if (err < 0) {
        fprintf(stderr, "%s: cannot read config file: %s\n", argv[0], strerror(-err));
        exit(1);
}
```

Once the cluster handle is configured, you can connect to the cluster with `rados_connect()`:

```
err = rados_connect(cluster);
if (err < 0) {
        fprintf(stderr, "%s: cannot connect to cluster: %s\n", argv[0], strerror(-err));
        exit(1);
}
```

Then you open an "IO context", a `rados_ioctx_t`, with `rados_ioctx_create()`:

```
rados_ioctx_t io;
char *poolname = "mypool";

err = rados_ioctx_create(cluster, poolname, &io);
if (err < 0) {
        fprintf(stderr, "%s: cannot open rados pool %s: %s\n", argv[0], poolname, strerror(-err));
        rados_shutdown(cluster);
        exit(1);
}
```

Note that the pool you try to access must exist.

Then you can use the RADOS data manipulation functions, for example write into an object called `greeting` with `rados_write_full()`:

```
err = rados_write_full(io, "greeting", "hello", 5);
if (err < 0) {
        fprintf(stderr, "%s: cannot write pool %s: %s\n", argv[0], poolname, strerror(-err));
        rados_ioctx_destroy(io);
        rados_shutdown(cluster);
        exit(1);
}
```

In the end, you'll want to close your IO context and connection to RADOS with `rados_ioctx_destroy()` and `rados_shutdown()`:

```
rados_ioctx_destroy(io);
rados_shutdown(cluster);
```

Asychronous IO

When doing lots of IO, you often don't need to wait for one operation to complete before starting the next one. *Librados* provides asynchronous versions of several operations:

- `rados_aio_write()`
- `rados_aio_append()`
- `rados_aio_write_full()`
- `rados_aio_read()`

For each operation, you must first create a `rados_completion_t` that represents what to do when the operation is safe or complete by calling `rados_aio_create_completion()`. If you don't need anything special to happen, you can pass NULL:

```
rados_completion_t comp;
err = rados_aio_create_completion(NULL, NULL, NULL, &comp);
if (err < 0) {
        fprintf(stderr, "%s: could not create aio completion: %s\n", argv[0], strerror(-err));
        rados_ioctx_destroy(io);
        rados_shutdown(cluster);
        exit(1);
}
```

Now you can call any of the aio operations, and wait for it to be in memory or on disk on all replicas:

```
err = rados_aio_write(io, "foo", comp, "bar", 3, 0);
if (err < 0) {
        fprintf(stderr, "%s: could not schedule aio write: %s\n", argv[0], strerror(-err));
        rados_aio_release(comp);
        rados_ioctx_destroy(io);
        rados_shutdown(cluster);
        exit(1);
}
rados_wait_for_complete(comp); // in memory
rados_wait_for_safe(comp); // on disk
```

Finally, we need to free the memory used by the completion with `rados_aio_release()`:

```
rados_aio_release(comp);
```

You can use the callbacks to tell your application when writes are durable, or when read buffers are full. For example, if you wanted to measure the latency of each operation when appending to several objects, you could schedule several writes and store the ack and commit time in the corresponding callback, then wait for all of them to complete using `rados_aio_flush()` before analyzing the latencies:

```
typedef struct {
        struct timeval start;
        struct timeval ack_end;
        struct timeval commit_end;
} req_duration;

void ack_callback(rados_completion_t comp, void *arg) {
        req_duration *dur = (req_duration *) arg;
        gettimeofday(&dur->ack_end, NULL);
}

void commit_callback(rados_completion_t comp, void *arg) {
        req_duration *dur = (req_duration *) arg;
        gettimeofday(&dur->commit_end, NULL);
}

int output_append_latency(rados_ioctx_t io, const char *data, size_t len, size_t num_writes) {
        req_duration times[num_writes];
        rados_completion_t comps[num_writes];
        for (size_t i = 0; i < num_writes; ++i) {
                gettimeofday(&times[i].start, NULL);
                int err = rados_aio_create_completion((void*) &times[i], ack_callback, commit_callbac
                if (err < 0) {
                        fprintf(stderr, "Error creating rados completion: %s\n", strerror(-err));
                        return err;
                }
                char obj_name[100];
                snprintf(obj_name, sizeof(obj_name), "foo%ld", (unsigned long)i);
                err = rados_aio_append(io, obj_name, comps[i], data, len);
                if (err < 0) {
                        fprintf(stderr, "Error from rados_aio_append: %s", strerror(-err));
                        return err;
                }
        }
        // wait until all requests finish *and* the callbacks complete
        rados_aio_flush(io);
        // the latencies can now be analyzed
        printf("Request # | Ack latency (s) | Commit latency (s)\n");
        for (size_t i = 0; i < num_writes; ++i) {
                // don't forget to free the completions
                rados_aio_release(comps[i]);
                struct timeval ack_lat, commit_lat;
                timersub(&times[i].ack_end, &times[i].start, &ack_lat);
                timersub(&times[i].commit_end, &times[i].start, &commit_lat);
                printf("%9ld | %8ld.%06ld | %10ld.%06ld\n", (unsigned long) i, ack_lat.tv_sec, ack_la
        }
        return 0;
}
```

Note that all the `rados_completion_t` must be freed with `rados_aio_release()` to avoid leaking memory.

API calls

Defines

`LIBRADOS_ALL_NSPACES`
Pass as nspace argument to *rados_ioctx_set_namespace()* before calling *rados_nobjects_list_open()* to return all objects in all namespaces.

struct `obj_watch_t`
#include <rados_types.h> One item from list_watchers

Public Members

char **addr**[256]

int64_t **watcher_id**

uint64_t **cookie**

uint32_t **timeout_seconds**

Defines

CEPH_OSD_TMAP_HDR

CEPH_OSD_TMAP_SET

CEPH_OSD_TMAP_CREATE

CEPH_OSD_TMAP_RM

LIBRADOS_VER_MAJOR

LIBRADOS_VER_MINOR

LIBRADOS_VER_EXTRA

LIBRADOS_VERSION(maj, min, extra)

LIBRADOS_VERSION_CODE

LIBRADOS_SUPPORTS_WATCH

LIBRADOS_LOCK_FLAG_RENEW

LIBRADOS_CREATE_EXCLUSIVE

LIBRADOS_CREATE_IDEMPOTENT

CEPH_RADOS_API

LIBRADOS_SNAP_HEAD

LIBRADOS_SNAP_DIR

Typedefs

typedef rados_t

A handle for interacting with a RADOS cluster. It encapsulates all RADOS client configuration, including username, key for authentication, logging, and debugging. Talking different clusters – or to the same cluster with different users – requires different cluster handles.

typedef void ***rados_config_t**

rados_config_t

A handle for the ceph configuration context for the rados_t cluster instance. This can be used to share configuration context/state (e.g., logging configuration) between librados instance.

Warning

The config context does not have independent reference counting. As such, a rados_config_t handle retrieved from a given rados_t is only valid as long as that rados_t.

typedef `rados_ioctx_t`

An io context encapsulates a few settings for all I/O operations done on it:

- pool - set when the io context is created (see *rados_ioctx_create()*)
- snapshot context for writes (see *rados_ioctx_selfmanaged_snap_set_write_ctx()*)
- snapshot id to read from (see *rados_ioctx_snap_set_read()*)
- object locator for all single-object operations (see *rados_ioctx_locator_set_key()*)
- namespace for all single-object operations (see *rados_ioctx_set_namespace()*). Set to LIBRADOS_ALL_NSPACES before *rados_nobjects_list_open()* will list all objects in all namespaces.

Warning

Changing any of these settings is not thread-safe - librados users must synchronize any of these changes on their own, or use separate io contexts for each thread

typedef `rados_list_ctx_t`

An iterator for listing the objects in a pool. Used with *rados_nobjects_list_open()*, *rados_nobjects_list_next()*, and *rados_nobjects_list_close()*.

typedef `rados_snap_t`

The id of a snapshot.

typedef `rados_xattrs_iter_t`

An iterator for listing extended attrbutes on an object. Used with *rados_getxattrs()*, *rados_getxattrs_next()*, and *rados_getxattrs_end()*.

typedef `rados_omap_iter_t`

An iterator for listing omap key/value pairs on an object. Used with *rados_read_op_omap_get_keys()*, *rados_read_op_omap_get_vals()*, *rados_read_op_omap_get_vals_by_keys()*, *rados_omap_get_next()*, and *rados_omap_get_end()*.

typedef `rados_write_op_t`

An object write operation stores a number of operations which can be executed atomically. For usage, see:

- Creation and deletion: *rados_create_write_op()* *rados_release_write_op()*
- Extended attribute manipulation: *rados_write_op_cmpxattr()* *rados_write_op_cmpxattr()*, *rados_write_op_setxattr()*, *rados_write_op_rmxattr()*
- Object map key/value pairs: *rados_write_op_omap_set()*, *rados_write_op_omap_rm_keys()*, *rados_write_op_omap_clear()*, *rados_write_op_omap_cmp()*
- Object properties: *rados_write_op_assert_exists()*, *rados_write_op_assert_version()*
- Creating objects: *rados_write_op_create()*
- IO on objects: *rados_write_op_append()*, *rados_write_op_write()*, rados_write_op_zero *rados_write_op_write_full()*, rados_write_op_remove, *rados_write_op_truncate()*, *rados_write_op_zero()*
- Hints: *rados_write_op_set_alloc_hint()*
- Performing the operation: *rados_write_op_operate()*, *rados_aio_write_op_operate()*

typedef `rados_read_op_t`

An object read operation stores a number of operations which can be executed atomically. For usage, see:

- Creation and deletion: *rados_create_read_op()* *rados_release_read_op()*

- Extended attribute manipulation: *rados_read_op_cmpxattr()*, rados_read_op_getxattr(), *rados_read_op_getxattrs()*
- Object map key/value pairs: *rados_read_op_omap_get_vals()*, *rados_read_op_omap_get_keys()*, *rados_read_op_omap_get_vals_by_keys()*, *rados_read_op_omap_cmp()*
- Object properties: *rados_read_op_stat()*, *rados_read_op_assert_exists()*, *rados_read_op_assert_version()*
- IO on objects: *rados_read_op_read()*
- Custom operations: *rados_read_op_exec()*, *rados_read_op_exec_user_buf()*
- Request properties: *rados_read_op_set_flags()*
- Performing the operation: *rados_read_op_operate()*, *rados_aio_read_op_operate()*

typedef `rados_completion_t`

Represents the state of an asynchronous operation - it contains the return value once the operation completes, and can be used to block until the operation is complete or safe.

typedef `rados_callback_t`

Callbacks for asynchrous operations take two parameters:

- cb the completion that has finished
- arg application defined data made available to the callback function

typedef `rados_watchcb_t`

Callback activated when a notify is received on a watched object.

Note

BUG: opcode is an internal detail that shouldn't be exposed

BUG: ver is unused

Parameters

- `opcode` - undefined
- `ver` - version of the watched object
- `arg` - application-specific data

typedef `rados_watchcb2_t`

Callback activated when a notify is received on a watched object.

Parameters

- `arg` - opaque user-defined value provided to *rados_watch2()*
- `notify_id` - an id for this notify event
- `handle` - the watcher handle we are notifying
- `notifier_id` - the unique client id for the notifier
- `data` - payload from the notifier
- `datalen` - length of payload buffer

typedef `rados_watcherrcb_t`

Callback activated when we encounter an error with the watch session. This can happen when the location of the objects moves within the cluster and we fail to register our watch with the new object location, or when our connection with the object OSD is otherwise interrupted and we may have missed notify events.

Parameters

- `pre` - opaque user-defined value provided to *rados_watch2()*
- `err` - error code

typedef `void(* rados_log_callback_t)(void *arg, const char *line, const char *who, uint64`

Enums

enum `[anonymous]`

Values:

`LIBRADOS_OP_FLAG_EXCL` = 0x1

`LIBRADOS_OP_FLAG_FAILOK` = 0x2

`LIBRADOS_OP_FLAG_FADVISE_RANDOM` = 0x4

`LIBRADOS_OP_FLAG_FADVISE_SEQUENTIAL` = 0x8

`LIBRADOS_OP_FLAG_FADVISE_WILLNEED` = 0x10

`LIBRADOS_OP_FLAG_FADVISE_DONTNEED` = 0x20

`LIBRADOS_OP_FLAG_FADVISE_NOCACHE` = 0x40

enum `[anonymous]`

Values:

`LIBRADOS_CMPXATTR_OP_EQ` = 1

`LIBRADOS_CMPXATTR_OP_NE` = 2

`LIBRADOS_CMPXATTR_OP_GT` = 3

`LIBRADOS_CMPXATTR_OP_GTE` = 4

`LIBRADOS_CMPXATTR_OP_LT` = 5

`LIBRADOS_CMPXATTR_OP_LTE` = 6

enum `[anonymous]`

Values:

`LIBRADOS_OPERATION_NOFLAG` = 0

`LIBRADOS_OPERATION_BALANCE_READS` = 1

`LIBRADOS_OPERATION_LOCALIZE_READS` = 2

`LIBRADOS_OPERATION_ORDER_READS_WRITES` = 4

`LIBRADOS_OPERATION_IGNORE_CACHE` = 8

`LIBRADOS_OPERATION_SKIPRWLOCKS` = 16

`LIBRADOS_OPERATION_IGNORE_OVERLAY` = 32

Functions

`CEPH_RADOS_API void rados_version(int * major, int * minor, int * extra)`

Get the version of librados.

The version number is major.minor.extra. Note that this is unrelated to the Ceph version number.

TODO: define version semantics, i.e.:

- incrementing major is for backwards-incompatible changes
- incrementing minor is for backwards-compatible changes
- incrementing extra is for bug fixes

Parameters

- `major` - where to store the major version number
- `minor` - where to store the minor version number
- `extra` - where to store the extra version number

`CEPH_RADOS_API int rados_create(rados_t * cluster, const char *const  id)`

Create a handle for communicating with a RADOS cluster.

Ceph environment variables are read when this is called, so if $CEPH_ARGS specifies everything you need to connect, no further configuration is necessary.

Return

0 on success, negative error code on failure

Parameters

- `cluster` - where to store the handle
- `id` - the user to connect as (i.e. admin, not client.admin)

`CEPH_RADOS_API int rados_create2(rados_t * pcluster, const char *const  clustername, c`

Extended version of rados_create.

Like rados_create, but 1) don't assume 'client\.'+id; allow full specification of name 2) allow specification of cluster name 3) flags for future expansion

`CEPH_RADOS_API int rados_create_with_context(rados_t * cluster, rados_config_t cct)`

Initialize a cluster handle from an existing configuration.

Share configuration state with another rados_t instance.

Return

0 on success, negative error code on failure

Parameters

- `cluster` - where to store the handle
- `cct` - the existing configuration to use

`CEPH_RADOS_API int rados_ping_monitor(rados_t cluster, const char * mon_id, char ** ou`

Ping the monitor with ID mon_id, storing the resulting reply in buf (if specified) with a maximum size of len.

The result buffer is allocated on the heap; the caller is expected to release that memory with *rados_buffer_free()*. The buffer and length pointers can be NULL, in which case they are not filled in.

Parameters

- `cluster` - cluster handle
- `mon_id` - ID of the monitor to ping
- `outstr` - double pointer with the resulting reply
- `outstrlen` - pointer with the size of the reply in outstr

`CEPH_RADOS_API int rados_connect(rados_t cluster)`

Connect to the cluster.

Note

BUG: Before calling this, calling a function that communicates with the cluster will crash.

Pre

The cluster handle is configured with at least a monitor address. If cephx is enabled, a client name and secret must also be set.

Post

If this succeeds, any function in librados may be used

Return

0 on sucess, negative error code on failure

Parameters

- `cluster` - The cluster to connect to.

`CEPH_RADOS_API void rados_shutdown(rados_t cluster)`

Disconnects from the cluster.

For clean up, this is only necessary after *rados_connect()* has succeeded.

Warning

This does not guarantee any asynchronous writes have completed. To do that, you must call *rados_aio_flush()* on all open io contexts.

We implicitly call *rados_watch_flush()* on shutdown. If there are watches being used, this should be done explicitly before destroying the relevant IoCtx. We do it here as a safety measure.

Post

the cluster handle cannot be used again

Parameters

- `cluster` - the cluster to shutdown

`CEPH_RADOS_API int rados_conf_read_file(rados_t cluster, const char * path)`

Configure the cluster handle using a Ceph config file

If path is NULL, the default locations are searched, and the first found is used. The locations are:

- $CEPH_CONF (environment variable)

•/etc/ceph/ceph.conf

•~/.ceph/config

•ceph.conf (in the current working directory)

Pre

rados_connect() has not been called on the cluster handle

Return

0 on success, negative error code on failure

Parameters

- `cluster` - cluster handle to configure
- `path` - path to a Ceph configuration file

`CEPH_RADOS_API int rados_conf_parse_argv(rados_t cluster, int argc, const char ** argv`

Configure the cluster handle with command line arguments

argv can contain any common Ceph command line option, including any configuration parameter prefixed by '–' and replacing spaces with dashes or underscores. For example, the following options are equivalent:

•–mon-host 10.0.0.1:6789

•–mon_host 10.0.0.1:6789

•-m 10.0.0.1:6789

Pre

rados_connect() has not been called on the cluster handle

Return

0 on success, negative error code on failure

Parameters

- `cluster` - cluster handle to configure
- `argc` - number of arguments in argv
- `argv` - arguments to parse

`CEPH_RADOS_API int rados_conf_parse_argv_remainder(rados_t cluster, int argc, const ch`

Configure the cluster handle with command line arguments, returning any remainders. Same rados_conf_parse_argv, except for extra remargv argument to hold returns unrecognized arguments.

Pre

rados_connect() has not been called on the cluster handle

Return

0 on success, negative error code on failure

Parameters

- `cluster` - cluster handle to configure
- `argc` - number of arguments in argv
- `argv` - arguments to parse

- `remargv` - char* array for returned unrecognized arguments

`CEPH_RADOS_API int rados_conf_parse_env(rados_t cluster, const char * var)`

Configure the cluster handle based on an environment variable

The contents of the environment variable are parsed as if they were Ceph command line options. If var is NULL, the CEPH_ARGS environment variable is used.

Pre

rados_connect() has not been called on the cluster handle

Note

BUG: this is not threadsafe - it uses a static buffer

Return

0 on success, negative error code on failure

Parameters

- `cluster` - cluster handle to configure
- `var` - name of the environment variable to read

`CEPH_RADOS_API int rados_conf_set(rados_t cluster, const char * option, const char * v`

Set a configuration option

Pre

rados_connect() has not been called on the cluster handle

Return

0 on success, negative error code on failure

-ENOENT when the option is not a Ceph configuration option

Parameters

- `cluster` - cluster handle to configure
- `option` - option to set
- `value` - value of the option

`CEPH_RADOS_API int rados_conf_get(rados_t cluster, const char * option, char * buf, si`

Get the value of a configuration option

Return

0 on success, negative error code on failure

-ENAMETOOLONG if the buffer is too short to contain the requested value

Parameters

- `cluster` - configuration to read
- `option` - which option to read
- `buf` - where to write the configuration value
- `len` - the size of buf in bytes

CEPH_RADOS_API int rados_cluster_stat(rados_t cluster, struct rados_cluster_stat_t *

Read usage info about the cluster

This tells you total space, space used, space available, and number of objects. These are not updated immediately when data is written, they are eventually consistent.

Return

0 on success, negative error code on failure

Parameters

- `cluster` - cluster to query
- `result` - where to store the results

CEPH_RADOS_API int rados_cluster_fsid(rados_t cluster, char * buf, size_t len)

Get the fsid of the cluster as a hexadecimal string.

The fsid is a unique id of an entire Ceph cluster.

Return

0 on success, negative error code on failure

-ERANGE if the buffer is too short to contain the fsid

Parameters

- `cluster` - where to get the fsid
- `buf` - where to write the fsid
- `len` - the size of buf in bytes (should be 37)

CEPH_RADOS_API int rados_wait_for_latest_osdmap(rados_t cluster)

Get/wait for the most recent osdmap

Return

0 on sucess, negative error code on failure

Parameters

- `cluster` - the cluster to shutdown

CEPH_RADOS_API int rados_pool_list(rados_t cluster, char * buf, size_t len)

List pools

Gets a list of pool names as NULL-terminated strings. The pool names will be placed in the supplied buffer one after another. After the last pool name, there will be two 0 bytes in a row.

If len is too short to fit all the pool name entries we need, we will fill as much as we can.

Return

length of the buffer we would need to list all pools

Parameters

- `cluster` - cluster handle
- `buf` - output buffer
- `len` - output buffer length

CEPH_RADOS_API rados_config_t rados_cct(rados_t cluster)

Get a configuration handle for a rados cluster handle

This handle is valid only as long as the cluster handle is valid.

Return

config handle for this cluster

Parameters

- `cluster` - cluster handle

CEPH_RADOS_API uint64_t rados_get_instance_id(rados_t cluster)

Get a global id for current instance

This id is a unique representation of current connection to the cluster

Return

instance global id

Parameters

- `cluster` - cluster handle

CEPH_RADOS_API int rados_ioctx_create(rados_t cluster, const char * pool_name, rados_i

Create an io context

The io context allows you to perform operations within a particular pool. For more details see rados_ioctx_t.

Return

0 on success, negative error code on failure

Parameters

- `cluster` - which cluster the pool is in
- `pool_name` - name of the pool
- `ioctx` - where to store the io context

CEPH_RADOS_API int rados_ioctx_create2(rados_t cluster, int64_t pool_id, rados_ioctx_t

CEPH_RADOS_API void rados_ioctx_destroy(rados_ioctx_t io)

The opposite of rados_ioctx_create

This just tells librados that you no longer need to use the io context. It may not be freed immediately if there are pending asynchronous requests on it, but you should not use an io context again after calling this function on it.

Warning

This does not guarantee any asynchronous writes have completed. You must call *rados_aio_flush()* on the io context before destroying it to do that.

If this ioctx is used by rados_watch, the caller needs to be sure that all registered watches are disconnected via *rados_unwatch()* and that *rados_watch_flush()* is called. This ensures that a racing watch callback does not make use of a destroyed ioctx.

Parameters

- `io` - the io context to dispose of

CEPH_RADOS_API rados_config_t rados_ioctx_cct(rados_ioctx_t io)
Get configuration hadnle for a pool handle

Return

rados_config_t for this cluster

Parameters

- `io` - pool handle

CEPH_RADOS_API rados_t rados_ioctx_get_cluster(rados_ioctx_t io)
Get the cluster handle used by this rados_ioctx_t Note that this is a weak reference, and should not be destroyed via *rados_shutdown()*.

Return

the cluster handle for this io context

Parameters

- `io` - the io context

CEPH_RADOS_API int rados_ioctx_pool_stat(rados_ioctx_t io, struct rados_pool_stat_t
Get pool usage statistics

Fills in a *rados_pool_stat_t* after querying the cluster.

Return

0 on success, negative error code on failure

Parameters

- `io` - determines which pool to query
- `stats` - where to store the results

CEPH_RADOS_API int64_t rados_pool_lookup(rados_t cluster, const char * pool_name)
Get the id of a pool

Return

id of the pool

-ENOENT if the pool is not found

Parameters

- `cluster` - which cluster the pool is in
- `pool_name` - which pool to look up

CEPH_RADOS_API int rados_pool_reverse_lookup(rados_t cluster, int64_t id, char * buf,
Get the name of a pool

Return

length of string stored, or -ERANGE if buffer too small

Parameters

- `cluster` - which cluster the pool is in
- `id` - the id of the pool

- `buf` - where to store the pool name
- `maxlen` - size of buffer where name will be stored

`CEPH_RADOS_API int rados_pool_create(rados_t cluster, const char * pool_name)`

Create a pool with default settings

The default owner is the admin user (auid 0). The default crush rule is rule 0.

Return

0 on success, negative error code on failure

Parameters

- `cluster` - the cluster in which the pool will be created
- `pool_name` - the name of the new pool

`CEPH_RADOS_API int rados_pool_create_with_auid(rados_t cluster, const char * pool_name`

Create a pool owned by a specific auid

The auid is the authenticated user id to give ownership of the pool. TODO: document auid and the rest of the auth system

Return

0 on success, negative error code on failure

Parameters

- `cluster` - the cluster in which the pool will be created
- `pool_name` - the name of the new pool
- `auid` - the id of the owner of the new pool

`CEPH_RADOS_API int rados_pool_create_with_crush_rule(rados_t cluster, const char * poo`

Create a pool with a specific CRUSH rule

Return

0 on success, negative error code on failure

Parameters

- `cluster` - the cluster in which the pool will be created
- `pool_name` - the name of the new pool
- `crush_rule_num` - which rule to use for placement in the new pool1

`CEPH_RADOS_API int rados_pool_create_with_all(rados_t cluster, const char * pool_name,`

Create a pool with a specific CRUSH rule and auid

This is a combination of *rados_pool_create_with_crush_rule()* and *rados_pool_create_with_auid()*.

Return

0 on success, negative error code on failure

Parameters

- `cluster` - the cluster in which the pool will be created
- `pool_name` - the name of the new pool

- `crush_rule_num` - which rule to use for placement in the new pool2
- `auid` - the id of the owner of the new pool

`CEPH_RADOS_API int rados_pool_get_base_tier(rados_t cluster, int64_t pool, int64_t * b`

Returns the pool that is the base tier for this pool.

The return value is the ID of the pool that should be used to read from/write to. If tiering is not set up for the pool, returns `pool`.

Return

0 on success, negative error code on failure

Parameters

- `cluster` - the cluster the pool is in
- `pool` - ID of the pool to query
- `base_tier` - base tier, or `pool` if tiering is not configured

`CEPH_RADOS_API int rados_pool_delete(rados_t cluster, const char * pool_name)`

Delete a pool and all data inside it

The pool is removed from the cluster immediately, but the actual data is deleted in the background.

Return

0 on success, negative error code on failure

Parameters

- `cluster` - the cluster the pool is in
- `pool_name` - which pool to delete

`CEPH_RADOS_API int rados_ioctx_pool_set_auid(rados_ioctx_t io, uint64_t auid)`

Attempt to change an io context's associated auid "owner."

Requires that you have write permission on both the current and new auid.

Return

0 on success, negative error code on failure

Parameters

- `io` - reference to the pool to change.
- `auid` - the auid you wish the io to have.

`CEPH_RADOS_API int rados_ioctx_pool_get_auid(rados_ioctx_t io, uint64_t * auid)`

Get the auid of a pool

Return

0 on success, negative error code on failure

Parameters

- `io` - pool to query
- `auid` - where to store the auid

`CEPH_RADOS_API int rados_ioctx_pool_requires_alignment(rados_ioctx_t io)`

CEPH_RADOS_API uint64_t rados_ioctx_pool_required_alignment(rados_ioctx_t io)

CEPH_RADOS_API int64_t rados_ioctx_get_id(rados_ioctx_t io)
Get the pool id of the io context

Return

the id of the pool the io context uses

Parameters

- `io` - the io context to query

CEPH_RADOS_API int rados_ioctx_get_pool_name(rados_ioctx_t io, char * buf, unsigned ma
Get the pool name of the io context

Return

length of string stored, or -ERANGE if buffer too small

Parameters

- `io` - the io context to query
- `buf` - pointer to buffer where name will be stored
- `maxlen` - size of buffer where name will be stored

CEPH_RADOS_API void rados_ioctx_locator_set_key(rados_ioctx_t io, const char * key)
Set the key for mapping objects to pgs within an io context.

The key is used instead of the object name to determine which placement groups an object is put in. This affects all subsequent operations of the io context - until a different locator key is set, all objects in this io context will be placed in the same pg.

This is useful if you need to do clone_range operations, which must be done with the source and destination objects in the same pg.

Parameters

- `io` - the io context to change
- `key` - the key to use as the object locator, or NULL to discard any previously set key

CEPH_RADOS_API void rados_ioctx_set_namespace(rados_ioctx_t io, const char * nspace)
Set the namespace for objects within an io context

The namespace specification further refines a pool into different domains. The mapping of objects to pgs is also based on this value.

Parameters

- `io` - the io context to change
- `nspace` - the name to use as the namespace, or NULL use the default namespace

CEPH_RADOS_API int rados_nobjects_list_open(rados_ioctx_t io, rados_list_ctx_t * ctx)
Start listing objects in a pool

Return

0 on success, negative error code on failure

Parameters

- `io` - the pool to list from
- `ctx` - the handle to store list context in

CEPH_RADOS_API uint32_t rados_nobjects_list_get_pg_hash_position(rados_list_ctx_t ctx)

Return hash position of iterator, rounded to the current PG

Return

current hash position, rounded to the current pg

Parameters

- `ctx` - iterator marking where you are in the listing

CEPH_RADOS_API uint32_t rados_nobjects_list_seek(rados_list_ctx_t ctx, uint32_t pos)

Reposition object iterator to a different hash position

Return

actual (rounded) position we moved to

Parameters

- `ctx` - iterator marking where you are in the listing
- `pos` - hash position to move to

CEPH_RADOS_API int rados_nobjects_list_next(rados_list_ctx_t ctx, const char ** entry,

Get the next object name and locator in the pool

*entry and *key are valid until next call to rados_objects_list_*

Return

0 on success, negative error code on failure

-ENOENT when there are no more objects to list

Parameters

- `ctx` - iterator marking where you are in the listing
- `entry` - where to store the name of the entry
- `key` - where to store the object locator (set to NULL to ignore)
- `nspace` - where to store the object namespace (set to NULL to ignore)

CEPH_RADOS_API void rados_nobjects_list_close(rados_list_ctx_t ctx)

Close the object listing handle.

This should be called when the handle is no longer needed. The handle should not be used after it has been closed.

Parameters

- `ctx` - the handle to close

CEPH_RADOS_API int rados_objects_list_open(rados_ioctx_t io, rados_list_ctx_t * ctx) Warning

Deprecated: Use *rados_nobjects_list_open()* instead

`CEPH_RADOS_API uint32_t rados_objects_list_get_pg_hash_position(rados_list_ctx_t ctx)` **Warning**

Deprecated: Use *rados_nobjects_list_get_pg_hash_position()* instead

`CEPH_RADOS_API uint32_t rados_objects_list_seek(rados_list_ctx_t ctx, uint32_t pos)` **Warning**

Deprecated: Use *rados_nobjects_list_seek()* instead

`CEPH_RADOS_API int rados_objects_list_next(rados_list_ctx_t ctx, const char ** entry,` **Warning**

Deprecated: Use *rados_nobjects_list_next()* instead

`CEPH_RADOS_API void rados_objects_list_close(rados_list_ctx_t ctx)` **Warning**

Deprecated: Use *rados_nobjects_list_close()* instead

`CEPH_RADOS_API int rados_ioctx_snap_create(rados_ioctx_t io, const char * snapname)`

Create a pool-wide snapshot

Return

0 on success, negative error code on failure

Parameters

- `io` - the pool to snapshot
- `snapname` - the name of the snapshot

`CEPH_RADOS_API int rados_ioctx_snap_remove(rados_ioctx_t io, const char * snapname)`

Delete a pool snapshot

Return

0 on success, negative error code on failure

Parameters

- `io` - the pool to delete the snapshot from
- `snapname` - which snapshot to delete

`CEPH_RADOS_API int rados_ioctx_snap_rollback(rados_ioctx_t io, const char * oid, const`

Rollback an object to a pool snapshot

The contents of the object will be the same as when the snapshot was taken.

Return

0 on success, negative error code on failure

Parameters

- `io` - the pool in which the object is stored
- `oid` - the name of the object to rollback
- `snapname` - which snapshot to rollback to

`CEPH_RADOS_API int rados_rollback(rados_ioctx_t io, const char * oid, const char * sna` **Warning**

Deprecated: Use *rados_ioctx_snap_rollback()* instead

`CEPH_RADOS_API void rados_ioctx_snap_set_read(rados_ioctx_t io, rados_snap_t snap)`

Set the snapshot from which reads are performed.

Subsequent reads will return data as it was at the time of that snapshot.

Parameters

- `io` - the io context to change
- `snap` - the id of the snapshot to set, or LIBRADOS_SNAP_HEAD for no snapshot (i.e. normal operation)

`CEPH_RADOS_API int rados_ioctx_selfmanaged_snap_create(rados_ioctx_t io, rados_snap_t`

Allocate an ID for a self-managed snapshot

Get a unique ID to put in the snaphot context to create a snapshot. A clone of an object is not created until a write with the new snapshot context is completed.

Return

0 on success, negative error code on failure

Parameters

- `io` - the pool in which the snapshot will exist
- `snapid` - where to store the newly allocated snapshot ID

`CEPH_RADOS_API int rados_ioctx_selfmanaged_snap_remove(rados_ioctx_t io, rados_snap_t`

Remove a self-managed snapshot

This increases the snapshot sequence number, which will cause snapshots to be removed lazily.

Return

0 on success, negative error code on failure

Parameters

- `io` - the pool in which the snapshot will exist
- `snapid` - where to store the newly allocated snapshot ID

`CEPH_RADOS_API int rados_ioctx_selfmanaged_snap_rollback(rados_ioctx_t io, const char`

Rollback an object to a self-managed snapshot

The contents of the object will be the same as when the snapshot was taken.

Return

0 on success, negative error code on failure

Parameters

- `io` - the pool in which the object is stored
- `oid` - the name of the object to rollback
- `snapid` - which snapshot to rollback to

`CEPH_RADOS_API int rados_ioctx_selfmanaged_snap_set_write_ctx(rados_ioctx_t io, rados_`

Set the snapshot context for use when writing to objects

This is stored in the io context, and applies to all future writes.

Return

0 on success, negative error code on failure

-EINVAL if snaps are not in descending order

Parameters

- `io` - the io context to change
- `seq` - the newest snapshot sequence number for the pool
- `snaps` - array of snapshots in sorted by descending id
- `num_snaps` - how many snaphosts are in the snaps array

`CEPH_RADOS_API int rados_ioctx_snap_list(rados_ioctx_t io, rados_snap_t * snaps, int ma`

List all the ids of pool snapshots

If the output array does not have enough space to fit all the snapshots, -ERANGE is returned and the caller should retry with a larger array.

Return

number of snapshots on success, negative error code on failure

-ERANGE is returned if the snaps array is too short

Parameters

- `io` - the pool to read from
- `snaps` - where to store the results
- `maxlen` - the number of rados_snap_t that fit in the snaps array

`CEPH_RADOS_API int rados_ioctx_snap_lookup(rados_ioctx_t io, const char * name, rados_`

Get the id of a pool snapshot

Return

0 on success, negative error code on failure

Parameters

- `io` - the pool to read from
- `name` - the snapshot to find
- `id` - where to store the result

`CEPH_RADOS_API int rados_ioctx_snap_get_name(rados_ioctx_t io, rados_snap_t id, char *`

Get the name of a pool snapshot

Return

0 on success, negative error code on failure

-ERANGE if the name array is too small

Parameters

- `io` - the pool to read from
- `id` - the snapshot to find
- `name` - where to store the result

- `maxlen` - the size of the name array

`CEPH_RADOS_API int rados_ioctx_snap_get_stamp(rados_ioctx_t io, rados_snap_t id, time_`

Find when a pool snapshot occurred

Return

0 on success, negative error code on failure

Parameters

- `io` - the pool the snapshot was taken in
- `id` - the snapshot to lookup
- `t` - where to store the result

`CEPH_RADOS_API uint64_t rados_get_last_version(rados_ioctx_t io)`

Return the version of the last object read or written to.

This exposes the internal version number of the last object read or written via this io context

Return

last read or written object version

Parameters

- `io` - the io context to check

`CEPH_RADOS_API int rados_write(rados_ioctx_t io, const char * oid, const char * buf, s`

Write *len* bytes from *buf* into the *oid* object, starting at offset *off*. The value of *len* must be <= UINT_MAX/2.

Note

This will never return a positive value not equal to len.

Return

0 on success, negative error code on failure

Parameters

- `io` - the io context in which the write will occur
- `oid` - name of the object
- `buf` - data to write
- `len` - length of the data, in bytes
- `off` - byte offset in the object to begin writing at

`CEPH_RADOS_API int rados_write_full(rados_ioctx_t io, const char * oid, const char * b`

Write *len* bytes from *buf* into the *oid* object. The value of *len* must be <= UINT_MAX/2.

The object is filled with the provided data. If the object exists, it is atomically truncated and then written.

Return

0 on success, negative error code on failure

Parameters

- `io` - the io context in which the write will occur

- `oid` - name of the object
- `buf` - data to write
- `len` - length of the data, in bytes

`CEPH_RADOS_API int rados_clone_range(rados_ioctx_t io, const char * dst, uint64_t dst_`

Efficiently copy a portion of one object to another

If the underlying filesystem on the OSD supports it, this will be a copy-on-write clone.

The src and dest objects must be in the same pg. To ensure this, the io context should have a locator key set (see *rados_ioctx_locator_set_key()*).

Return

0 on success, negative error code on failure

Parameters

- `io` - the context in which the data is cloned
- `dst` - the name of the destination object
- `dst_off` - the offset within the destination object (in bytes)
- `src` - the name of the source object
- `src_off` - the offset within the source object (in bytes)
- `len` - how much data to copy

`CEPH_RADOS_API int rados_append(rados_ioctx_t io, const char * oid, const char * buf,`

Append *len* bytes from *buf* into the *oid* object. The value of *len* must be <= UINT_MAX/2.

Return

0 on success, negative error code on failure

Parameters

- `io` - the context to operate in
- `oid` - the name of the object
- `buf` - the data to append
- `len` - length of buf (in bytes)

`CEPH_RADOS_API int rados_read(rados_ioctx_t io, const char * oid, char * buf, size_t l`

Read data from an object

The io context determines the snapshot to read from, if any was set by *rados_ioctx_snap_set_read()*.

Return

number of bytes read on success, negative error code on failure

Parameters

- `io` - the context in which to perform the read
- `oid` - the name of the object to read from
- `buf` - where to store the results
- `len` - the number of bytes to read

- `off` - the offset to start reading from in the object

`CEPH_RADOS_API int rados_remove(rados_ioctx_t io, const char * oid)`

Delete an object

Note

This does not delete any snapshots of the object.

Return

0 on success, negative error code on failure

Parameters

- `io` - the pool to delete the object from
- `oid` - the name of the object to delete

`CEPH_RADOS_API int rados_trunc(rados_ioctx_t io, const char * oid, uint64_t size)`

Resize an object

If this enlarges the object, the new area is logically filled with zeroes. If this shrinks the object, the excess data is removed.

Return

0 on success, negative error code on failure

Parameters

- `io` - the context in which to truncate
- `oid` - the name of the object
- `size` - the new size of the object in bytes

`CEPH_RADOS_API int rados_getxattr(rados_ioctx_t io, const char * o, const char * name,`

Get the value of an extended attribute on an object.

Return

length of xattr value on success, negative error code on failure

Parameters

- `io` - the context in which the attribute is read
- `o` - name of the object
- `name` - which extended attribute to read
- `buf` - where to store the result
- `len` - size of buf in bytes

`CEPH_RADOS_API int rados_setxattr(rados_ioctx_t io, const char * o, const char * name,`

Set an extended attribute on an object.

Return

0 on success, negative error code on failure

Parameters

- `io` - the context in which xattr is set

- `o` - name of the object
- `name` - which extended attribute to set
- `buf` - what to store in the xattr
- `len` - the number of bytes in buf

`CEPH_RADOS_API int rados_rmxattr(rados_ioctx_t io, const char * o, const char * name)`

Delete an extended attribute from an object.

Return

0 on success, negative error code on failure

Parameters

- `io` - the context in which to delete the xattr
- `o` - the name of the object
- `name` - which xattr to delete

`CEPH_RADOS_API int rados_getxattrs(rados_ioctx_t io, const char * oid, rados_xattrs_it`

Start iterating over xattrs on an object.

Post

iter is a valid iterator

Return

0 on success, negative error code on failure

Parameters

- `io` - the context in which to list xattrs
- `oid` - name of the object
- `iter` - where to store the iterator

`CEPH_RADOS_API int rados_getxattrs_next(rados_xattrs_iter_t iter, const char ** name,`

Get the next xattr on the object

Pre

iter is a valid iterator

Post

name is the NULL-terminated name of the next xattr, and val contains the value of the xattr, which is of length len. If the end of the list has been reached, name and val are NULL, and len is 0.

Return

0 on success, negative error code on failure

Parameters

- `iter` - iterator to advance
- `name` - where to store the name of the next xattr
- `val` - where to store the value of the next xattr
- `len` - the number of bytes in val

CEPH_RADOS_API void rados_getxattrs_end(rados_xattrs_iter_t iter)

Close the xattr iterator.

iter should not be used after this is called.

Parameters

- `iter` - the iterator to close

CEPH_RADOS_API int rados_omap_get_next(rados_omap_iter_t iter, char ** key, char ** va

Get the next omap key/value pair on the object

Pre

iter is a valid iterator

Post

key and val are the next key/value pair. key is null-terminated, and val has length len. If the end of the list has been reached, key and val are NULL, and len is 0. key and val will not be accessible after *rados_omap_get_end()* is called on iter, so if they are needed after that they should be copied.

Return

0 on success, negative error code on failure

Parameters

- `iter` - iterator to advance
- `key` - where to store the key of the next omap entry
- `val` - where to store the value of the next omap entry
- `len` - where to store the number of bytes in val

CEPH_RADOS_API void rados_omap_get_end(rados_omap_iter_t iter)

Close the omap iterator.

iter should not be used after this is called.

Parameters

- `iter` - the iterator to close

CEPH_RADOS_API int rados_stat(rados_ioctx_t io, const char * o, uint64_t * psize, time_

Get object stats (size/mtime)

TODO: when are these set, and by whom? can they be out of date?

Return

0 on success, negative error code on failure

Parameters

- `io` - ioctx
- `o` - object name
- `psize` - where to store object size
- `pmtime` - where to store modification time

CEPH_RADOS_API int rados_tmap_update(rados_ioctx_t io, const char * o, const char * cm

Update tmap (trivial map)

Do compound update to a tmap object, inserting or deleting some number of records. cmdbuf is a series of operation byte codes, following by command payload. Each command is a single-byte command code, whose value is one of CEPH_OSD_TMAP_*.

- update tmap 'header'
  - 1 byte = CEPH_OSD_TMAP_HDR
  - 4 bytes = data length (little endian)
  - N bytes = data
- insert/update one key/value pair
  - 1 byte = CEPH_OSD_TMAP_SET
  - 4 bytes = key name length (little endian)
  - N bytes = key name
  - 4 bytes = data length (little endian)
  - M bytes = data
- insert one key/value pair; return -EEXIST if it already exists.
  - 1 byte = CEPH_OSD_TMAP_CREATE
  - 4 bytes = key name length (little endian)
  - N bytes = key name
  - 4 bytes = data length (little endian)
  - M bytes = data
- remove one key/value pair
  - 1 byte = CEPH_OSD_TMAP_RM
  - 4 bytes = key name length (little endian)
  - N bytes = key name

Restrictions:

- The HDR update must preceed any key/value updates.
- All key/value updates must be in lexicographically sorted order in cmdbuf.
- You can read/write to a tmap object via the regular APIs, but you should be careful not to corrupt it. Also be aware that the object format may change without notice.

Return

0 on success, negative error code on failure

Parameters

- `io` - ioctx
- `o` - object name

- `cmdbuf` - command buffer
- `cmdbuflen` - command buffer length in bytes

CEPH_RADOS_API int rados_tmap_put(rados_ioctx_t io, const char * o, const char * buf,
Store complete tmap (trivial map) object

Put a full tmap object into the store, replacing what was there.

The format of buf is:

•4 bytes - length of header (little endian)

•N bytes - header data

•4 bytes - number of keys (little endian)

and for each key,

•4 bytes - key name length (little endian)

•N bytes - key name

•4 bytes - value length (little endian)

•M bytes - value data

Return

0 on success, negative error code on failure

Parameters

- `io` - ioctx
- `o` - object name
- `buf` - buffer
- `buflen` - buffer length in bytes

CEPH_RADOS_API int rados_tmap_get(rados_ioctx_t io, const char * o, char * buf, size_t
Fetch complete tmap (trivial map) object

Read a full tmap object. See *rados_tmap_put()* for the format the data is returned in.

Return

0 on success, negative error code on failure

-ERANGE if buf isn't big enough

Parameters

- `io` - ioctx
- `o` - object name
- `buf` - buffer
- `buflen` - buffer length in bytes

CEPH_RADOS_API int rados_exec(rados_ioctx_t io, const char * oid, const char * cls, co
Execute an OSD class method on an object

The OSD has a plugin mechanism for performing complicated operations on an object atomically. These plugins are called classes. This function allows librados users to call the custom methods.

The input and output formats are defined by the class. Classes in ceph.git can be found in src/cls subdirectories

Return

the length of the output, or -ERANGE if out_buf does not have enough space to store it (For methods that return data). For methods that don't return data, the return value is method-specific.

Parameters

- `io` - the context in which to call the method
- `oid` - the object to call the method on
- `cls` - the name of the class
- `method` - the name of the method
- `in_buf` - where to find input
- `in_len` - length of in_buf in bytes
- `buf` - where to store output
- `out_len` - length of buf in bytes

`CEPH_RADOS_API int rados_aio_create_completion(void * cb_arg, rados_callback_t cb_comp`

Constructs a completion to use with asynchronous operations

The complete and safe callbacks correspond to operations being acked and committed, respectively. The callbacks are called in order of receipt, so the safe callback may be triggered before the complete callback, and vice versa. This is affected by journalling on the OSDs.

TODO: more complete documentation of this elsewhere (in the RADOS docs?)

Note

Read operations only get a complete callback.

BUG: this should check for ENOMEM instead of throwing an exception

Return

0

Parameters

- `cb_arg` - application-defined data passed to the callback functions
- `cb_complete` - the function to be called when the operation is in memory on all relpicas
- `cb_safe` - the function to be called when the operation is on stable storage on all replicas
- `pc` - where to store the completion

`CEPH_RADOS_API int rados_aio_wait_for_complete(rados_completion_t c)`

Block until an operation completes

This means it is in memory on all replicas.

Note

BUG: this should be void

Return

0

Parameters

- `c` - operation to wait for

`CEPH_RADOS_API int rados_aio_wait_for_safe(rados_completion_t c)`

Block until an operation is safe

This means it is on stable storage on all replicas.

Note

BUG: this should be void

Return

0

Parameters

- `c` - operation to wait for

`CEPH_RADOS_API int rados_aio_is_complete(rados_completion_t c)`

Has an asynchronous operation completed?

Warning

This does not imply that the complete callback has finished

Return

whether c is complete

Parameters

- `c` - async operation to inspect

`CEPH_RADOS_API int rados_aio_is_safe(rados_completion_t c)`

Is an asynchronous operation safe?

Warning

This does not imply that the safe callback has finished

Return

whether c is safe

Parameters

- `c` - async operation to inspect

`CEPH_RADOS_API int rados_aio_wait_for_complete_and_cb(rados_completion_t c)`

Block until an operation completes and callback completes

This means it is in memory on all replicas and can be read.

Note

BUG: this should be void

Return

0

Parameters

- `c` - operation to wait for

`CEPH_RADOS_API int rados_aio_wait_for_safe_and_cb(rados_completion_t c)`

Block until an operation is safe and callback has completed

This means it is on stable storage on all replicas.

Note

BUG: this should be void

Return

0

Parameters

- `c` - operation to wait for

`CEPH_RADOS_API int rados_aio_is_complete_and_cb(rados_completion_t c)`

Has an asynchronous operation and callback completed

Return

whether c is complete

Parameters

- `c` - async operation to inspect

`CEPH_RADOS_API int rados_aio_is_safe_and_cb(rados_completion_t c)`

Is an asynchronous operation safe and has the callback completed

Return

whether c is safe

Parameters

- `c` - async operation to inspect

`CEPH_RADOS_API int rados_aio_get_return_value(rados_completion_t c)`

Get the return value of an asychronous operation

The return value is set when the operation is complete or safe, whichever comes first.

Pre

The operation is safe or complete

Note

BUG: complete callback may never be called when the safe message is received before the complete message

Return

return value of the operation

Parameters

- `c` - async operation to inspect

CEPH_RADOS_API void rados_aio_release(rados_completion_t c)

Release a completion

Call this when you no longer need the completion. It may not be freed immediately if the operation is not acked and committed.

Parameters

- `c` - completion to release

CEPH_RADOS_API int rados_aio_write(rados_ioctx_t io, const char * oid, rados_completio

Write data to an object asynchronously

Queues the write and returns. The return value of the completion will be 0 on success, negative error code on failure.

Return

0 on success, -EROFS if the io context specifies a snap_seq other than LIBRADOS_SNAP_HEAD

Parameters

- `io` - the context in which the write will occur
- `oid` - name of the object
- `completion` - what to do when the write is safe and complete
- `buf` - data to write
- `len` - length of the data, in bytes
- `off` - byte offset in the object to begin writing at

CEPH_RADOS_API int rados_aio_append(rados_ioctx_t io, const char * oid, rados_completi

Asychronously append data to an object

Queues the append and returns.

The return value of the completion will be 0 on success, negative error code on failure.

Return

0 on success, -EROFS if the io context specifies a snap_seq other than LIBRADOS_SNAP_HEAD

Parameters

- `io` - the context to operate in
- `oid` - the name of the object
- `completion` - what to do when the append is safe and complete
- `buf` - the data to append
- `len` - length of buf (in bytes)

CEPH_RADOS_API int rados_aio_write_full(rados_ioctx_t io, const char * oid, rados_comp

Asychronously write an entire object

The object is filled with the provided data. If the object exists, it is atomically truncated and then written. Queues the write_full and returns.

The return value of the completion will be 0 on success, negative error code on failure.

Return

0 on success, -EROFS if the io context specifies a snap_seq other than LIBRADOS_SNAP_HEAD

Parameters

- `io` - the io context in which the write will occur
- `oid` - name of the object
- `completion` - what to do when the write_full is safe and complete
- `buf` - data to write
- `len` - length of the data, in bytes

CEPH_RADOS_API int rados_aio_remove(rados_ioctx_t io, const char * oid, rados_completi

Asychronously remove an object

Queues the remove and returns.

The return value of the completion will be 0 on success, negative error code on failure.

Return

0 on success, -EROFS if the io context specifies a snap_seq other than LIBRADOS_SNAP_HEAD

Parameters

- `io` - the context to operate in
- `oid` - the name of the object
- `completion` - what to do when the remove is safe and complete

CEPH_RADOS_API int rados_aio_read(rados_ioctx_t io, const char * oid, rados_completion_

Asychronously read data from an object

The io context determines the snapshot to read from, if any was set by *rados_ioctx_snap_set_read()*.

The return value of the completion will be number of bytes read on success, negative error code on failure.

Note

only the 'complete' callback of the completion will be called.

Return

0 on success, negative error code on failure

Parameters

- `io` - the context in which to perform the read
- `oid` - the name of the object to read from
- `completion` - what to do when the read is complete
- `buf` - where to store the results
- `len` - the number of bytes to read
- `off` - the offset to start reading from in the object

CEPH_RADOS_API int rados_aio_flush(rados_ioctx_t io)

Block until all pending writes in an io context are safe

This is not equivalent to calling *rados_aio_wait_for_safe()* on all write completions, since this waits for the associated callbacks to complete as well.

Note

BUG: always returns 0, should be void or accept a timeout

Return

0 on success, negative error code on failure

Parameters

- `io` - the context to flush

CEPH_RADOS_API int rados_aio_flush_async(rados_ioctx_t io, rados_completion_t completi

Schedule a callback for when all currently pending aio writes are safe. This is a non-blocking version of *rados_aio_flush()*.

Return

0 on success, negative error code on failure

Parameters

- `io` - the context to flush
- `completion` - what to do when the writes are safe

CEPH_RADOS_API int rados_aio_stat(rados_ioctx_t io, const char * o, rados_completion_t

Asynchronously get object stats (size/mtime)

Return

0 on success, negative error code on failure

Parameters

- `io` - ioctx
- `o` - object name
- `psize` - where to store object size
- `pmtime` - where to store modification time

CEPH_RADOS_API int rados_aio_cancel(rados_ioctx_t io, rados_completion_t completion)

Cancel async operation

Return

0 on success, negative error code on failure

Parameters

- `io` - ioctx
- `completion` - completion handle

CEPH_RADOS_API int rados_watch(rados_ioctx_t io, const char * o, uint64_t ver, uint64_

Register an interest in an object

A watch operation registers the client as being interested in notifications on an object. OSDs keep track of watches on persistent storage, so they are preserved across cluster changes by the normal recovery process. If the client loses its connection to the primary OSD for a watched object, the watch will be removed after 30 seconds. Watches are automatically reestablished when a new connection is made, or a placement group switches OSDs.

Note

BUG: watch timeout should be configurable

BUG: librados should provide a way for watchers to notice connection resets

BUG: the ver parameter does not work, and -ERANGE will never be returned (See URL tracker.ceph.com/issues/2592)

Return

0 on success, negative error code on failure

-ERANGE if the version of the object is greater than ver

Parameters

- `io` - the pool the object is in
- `o` - the object to watch
- `ver` - expected version of the object
- `cookie` - where to store the internal id assigned to this watch
- `watchcb` - what to do when a notify is received on this object
- `arg` - application defined data to pass when watchcb is called

`CEPH_RADOS_API int rados_watch2(rados_ioctx_t io, const char * o, uint64_t * cookie, r`

Register an interest in an object

A watch operation registers the client as being interested in notifications on an object. OSDs keep track of watches on persistent storage, so they are preserved across cluster changes by the normal recovery process. If the client loses its connection to the primary OSD for a watched object, the watch will be removed after 30 seconds. Watches are automatically reestablished when a new connection is made, or a placement group switches OSDs.

Note

BUG: watch timeout should be configurable

Return

0 on success, negative error code on failure

Parameters

- `io` - the pool the object is in
- `o` - the object to watch
- `cookie` - where to store the internal id assigned to this watch
- `watchcb` - what to do when a notify is received on this object
- `watcherrcb` - what to do when the watch session encounters an error
- `arg` - opaque value to pass to the callback

`CEPH_RADOS_API int rados_watch_check(rados_ioctx_t io, uint64_t cookie)`

Check on the status of a watch

Return the number of milliseconds since the watch was last confirmed. Or, if there has been an error, return that.

If there is an error, the watch is no longer valid, and should be destroyed with *rados_unwatch2()*. The the user is still interested in the object, a new watch should be created with *rados_watch2()*.

Return

ms since last confirmed on success, negative error code on failure

Parameters

- `io` - the pool the object is in
- `cookie` - the watch handle

`CEPH_RADOS_API int rados_unwatch(rados_ioctx_t io, const char * o, uint64_t cookie)`

Unregister an interest in an object

Once this completes, no more notifies will be sent to us for this watch. This should be called to clean up unneeded watchers.

Return

0 on success, negative error code on failure

Parameters

- `io` - the pool the object is in
- `o` - the name of the watched object (ignored)
- `cookie` - which watch to unregister

`CEPH_RADOS_API int rados_unwatch2(rados_ioctx_t io, uint64_t cookie)`

Unregister an interest in an object

Once this completes, no more notifies will be sent to us for this watch. This should be called to clean up unneeded watchers.

Return

0 on success, negative error code on failure

Parameters

- `io` - the pool the object is in
- `cookie` - which watch to unregister

`CEPH_RADOS_API int rados_notify(rados_ioctx_t io, const char * o, uint64_t ver, const`

Sychronously notify watchers of an object

This blocks until all watchers of the object have received and reacted to the notify, or a timeout is reached.

Note

BUG: the timeout is not changeable via the C API

BUG: the bufferlist is inaccessible in a rados_watchcb_t

Return

0 on success, negative error code on failure

Parameters

- `io` - the pool the object is in
- `o` - the name of the object
- `ver` - obsolete - just pass zero
- `buf` - data to send to watchers
- `buf_len` - length of buf in bytes

CEPH_RADOS_API int rados_notify2(rados_ioctx_t io, const char * o, const char * buf, i

Sychronously notify watchers of an object

This blocks until all watchers of the object have received and reacted to the notify, or a timeout is reached.

The reply buffer is optional. If specified, the client will get back an encoded buffer that includes the ids of the clients that acknowledged the notify as well as their notify ack payloads (if any). Clients that timed out are not included. Even clients that do not include a notify ack payload are included in the list but have a 0-length payload associated with them. The format:

le32 num_acks { le64 gid global id for the client (for client.1234 that's 1234) le64 cookie cookie for the client le32 buflen length of reply message buffer u8 * buflen payload } * num_acks le32 num_timeouts { le64 gid global id for the client le64 cookie cookie for the client } * num_timeouts

Note: There may be multiple instances of the same gid if there are multiple watchers registered via the same client.

Note: The buffer must be released with *rados_buffer_free()* when the user is done with it.

Note: Since the result buffer includes clients that time out, it will be set even when *rados_notify()* returns an error code (like -ETIMEDOUT).

Return

0 on success, negative error code on failure

Parameters

- `io` - the pool the object is in
- `o` - the name of the object
- `buf` - data to send to watchers
- `buf_len` - length of buf in bytes
- `timeout_ms` - notify timeout (in ms)
- `reply_buffer` - pointer to reply buffer pointer (free with rados_buffer_free)
- `reply_buffer_len` - pointer to size of reply buffer

CEPH_RADOS_API int rados_notify_ack(rados_ioctx_t io, const char * o, uint64_t notify_

Acknolwedge receipt of a notify

Return

0 on success

Parameters

- `io` - the pool the object is in
- `o` - the name of the object
- `notify_id` - the notify_id we got on the watchcb2_t callback
- `cookie` - the watcher handle
- `buf` - payload to return to notifier (optional)
- `buf_len` - payload length

`CEPH_RADOS_API int rados_watch_flush(rados_t cluster)`

Flush watch/notify callbacks

This call will block until all pending watch/notify callbacks have been executed and the queue is empty. It should usually be called after shutting down any watches before shutting down the ioctx or librados to ensure that any callbacks do not misuse the ioctx (for example by calling rados_notify_ack after the ioctx has been destroyed).

Parameters

- `cluster` - the cluster handle

`CEPH_RADOS_API int rados_set_alloc_hint(rados_ioctx_t io, const char * o, uint64_t exp`

Set allocation hint for an object

This is an advisory operation, it will always succeed (as if it was submitted with a LIBRADOS_OP_FLAG_FAILOK flag set) and is not guaranteed to do anything on the backend.

Return

0 on success, negative error code on failure

Parameters

- `io` - the pool the object is in
- `o` - the name of the object
- `expected_object_size` - expected size of the object, in bytes
- `expected_write_size` - expected size of writes to the object, in bytes

`CEPH_RADOS_API rados_write_op_t rados_create_write_op(void)`

Create a new rados_write_op_t write operation. This will store all actions to be performed atomically. You must call rados_release_write_op when you are finished with it.

Return

non-NULL on success, NULL on memory allocation error.

`CEPH_RADOS_API void rados_release_write_op(rados_write_op_t write_op)`

Free a rados_write_op_t, must be called when you're done with it.

Parameters

- `write_op` - operation to deallocate, created with rados_create_write_op

`CEPH_RADOS_API void rados_write_op_set_flags(rados_write_op_t write_op, int flags)`

Set flags for the last operation added to this write_op. At least one op must have been added to the write_op.

Parameters

- `flags` - see librados.h constants beginning with LIBRADOS_OP_FLAG

CEPH_RADOS_API void rados_write_op_assert_exists(rados_write_op_t write_op)
Ensure that the object exists before writing

Parameters

- `write_op` - operation to add this action to

CEPH_RADOS_API void rados_write_op_assert_version(rados_write_op_t write_op, uint64_t v
Ensure that the object exists and that its internal version number is equal to "ver" before writing. "ver" should be a version number previously obtained with *rados_get_last_version()*.

•If the object's version is greater than the asserted version then rados_write_op_operate will return -ERANGE instead of executing the op.

•If the object's version is less than the asserted version then rados_write_op_operate will return -EOVERFLOW instead of executing the op.

Parameters

- `write_op` - operation to add this action to
- `ver` - object version number

CEPH_RADOS_API void rados_write_op_cmpxattr(rados_write_op_t write_op, const char * nam
Ensure that given xattr satisfies comparison. If the comparison is not satisfied, the return code of the operation will be -ECANCELED

Parameters

- `write_op` - operation to add this action to
- `name` - name of the xattr to look up
- `comparison_operator` - currently undocumented, look for LIBRADOS_CMPXATTR_OP_EQ in librados.h
- `value` - buffer to compare actual xattr value to
- `value_len` - length of buffer to compare actual xattr value to

CEPH_RADOS_API void rados_write_op_omap_cmp(rados_write_op_t write_op, const char * key
Ensure that the an omap value satisfies a comparison, with the supplied value on the right hand side (i.e. for OP_LT, the comparison is actual_value < value.

Parameters

- `write_op` - operation to add this action to
- `key` - which omap value to compare
- `comparison_operator` - one of LIBRADOS_CMPXATTR_OP_EQ, LIBRADOS_CMPXATTR_OP_LT, or LIBRADOS_CMPXATTR_OP_GT
- `val` - value to compare with
- `val_len` - length of value in bytes
- `prval` - where to store the return value from this action

CEPH_RADOS_API void rados_write_op_setxattr(rados_write_op_t write_op, const char * nam
Set an xattr

Parameters

- `write_op` - operation to add this action to
- `name` - name of the xattr

- `value` - buffer to set xattr to
- `value_len` - length of buffer to set xattr to

CEPH_RADOS_API void rados_write_op_rmxattr(rados_write_op_t write_op, const char * nam
Remove an xattr

Parameters

- `write_op` - operation to add this action to
- `name` - name of the xattr to remove

CEPH_RADOS_API void rados_write_op_create(rados_write_op_t write_op, int exclusive, co
Create the object

Parameters

- `write_op` - operation to add this action to
- `exclusive` - set to either LIBRADOS_CREATE_EXCLUSIVE or LIBRADOS_CREATE_IDEMPOTENT will error if the object already exists.
- `category` - category string (DEPRECATED, HAS NO EFFECT)

CEPH_RADOS_API void rados_write_op_write(rados_write_op_t write_op, const char * buffe
Write to offset

Parameters

- `write_op` - operation to add this action to
- `offset` - offset to write to
- `buffer` - bytes to write
- `len` - length of buffer

CEPH_RADOS_API void rados_write_op_write_full(rados_write_op_t write_op, const char *
Write whole object, atomically replacing it.

Parameters

- `write_op` - operation to add this action to
- `buffer` - bytes to write
- `len` - length of buffer

CEPH_RADOS_API void rados_write_op_append(rados_write_op_t write_op, const char * buff
Append to end of object.

Parameters

- `write_op` - operation to add this action to
- `buffer` - bytes to write
- `len` - length of buffer

CEPH_RADOS_API void rados_write_op_remove(rados_write_op_t write_op)
Remove object

Parameters

- `write_op` - operation to add this action to

CEPH_RADOS_API void rados_write_op_truncate(rados_write_op_t write_op, uint64_t offset
Truncate an object

Parameters

- `write_op` - operation to add this action to
- `offset` - Offset to truncate to

CEPH_RADOS_API void rados_write_op_zero(rados_write_op_t write_op, uint64_t offset, ui

Zero part of an object

Parameters

- `write_op` - operation to add this action to
- `offset` - Offset to zero
- `len` - length to zero

CEPH_RADOS_API void rados_write_op_exec(rados_write_op_t write_op, const char * cls, c

Execute an OSD class method on an object See *rados_exec()* for general description.

Parameters

- `write_op` - operation to add this action to
- `cls` - the name of the class
- `method` - the name of the method
- `in_buf` - where to find input
- `in_len` - length of in_buf in bytes
- `prval` - where to store the return value from the method

CEPH_RADOS_API void rados_write_op_omap_set(rados_write_op_t write_op, char const *con

Set key/value pairs on an object

Parameters

- `write_op` - operation to add this action to
- `keys` - array of null-terminated char arrays representing keys to set
- `vals` - array of pointers to values to set
- `lens` - array of lengths corresponding to each value
- `num` - number of key/value pairs to set

CEPH_RADOS_API void rados_write_op_omap_rm_keys(rados_write_op_t write_op, char const

Remove key/value pairs from an object

Parameters

- `write_op` - operation to add this action to
- `keys` - array of null-terminated char arrays representing keys to remove
- `keys_len` - number of key/value pairs to remove

CEPH_RADOS_API void rados_write_op_omap_clear(rados_write_op_t write_op)

Remove all key/value pairs from an object

Parameters

- `write_op` - operation to add this action to

CEPH_RADOS_API void rados_write_op_set_alloc_hint(rados_write_op_t write_op, uint64_t

Set allocation hint for an object

Parameters

- `write_op` - operation to add this action to
- `expected_object_size` - expected size of the object, in bytes
- `expected_write_size` - expected size of writes to the object, in bytes

CEPH_RADOS_API int rados_write_op_operate(rados_write_op_t write_op, rados_ioctx_t io,

Perform a write operation synchronously

Parameters

- `write_op` - operation to perform
- `io` - the ioctx that the object is in
- `oid` - the object id
- `mtime` - the time to set the mtime to, NULL for the current time
- `flags` - flags to apply to the entire operation (LIBRADOS_OPERATION_*)

CEPH_RADOS_API int rados_aio_write_op_operate(rados_write_op_t write_op, rados_ioctx_t

Perform a write operation asynchronously

Parameters

- `write_op` - operation to perform
- `io` - the ioctx that the object is in
- `completion` - what to do when operation has been attempted
- `oid` - the object id
- `mtime` - the time to set the mtime to, NULL for the current time
- `flags` - flags to apply to the entire operation (LIBRADOS_OPERATION_*)

CEPH_RADOS_API rados_read_op_t rados_create_read_op(void)

Create a new rados_read_op_t write operation. This will store all actions to be performed atomically. You must call rados_release_read_op when you are finished with it (after it completes, or you decide not to send it in the first place).

Return

non-NULL on success, NULL on memory allocation error.

CEPH_RADOS_API void rados_release_read_op(rados_read_op_t read_op)

Free a rados_read_op_t, must be called when you're done with it.

Parameters

- `read_op` - operation to deallocate, created with rados_create_read_op

CEPH_RADOS_API void rados_read_op_set_flags(rados_read_op_t read_op, int flags)

Set flags for the last operation added to this read_op. At least one op must have been added to the read_op.

Parameters

- `flags` - see librados.h constants beginning with LIBRADOS_OP_FLAG

CEPH_RADOS_API void rados_read_op_assert_exists(rados_read_op_t read_op)
Ensure that the object exists before reading

Parameters

- `read_op` - operation to add this action to

CEPH_RADOS_API void rados_read_op_assert_version(rados_read_op_t read_op, uint64_t ver
Ensure that the object exists and that its internal version number is equal to "ver" before reading. "ver" should be a version number previously obtained with *rados_get_last_version()*.

- If the object's version is greater than the asserted version then rados_read_op_operate will return -ERANGE instead of executing the op.
- If the object's version is less than the asserted version then rados_read_op_operate will return -EOVERFLOW instead of executing the op.

  Parameters

  - `read_op` - operation to add this action to
  - `ver` - object version number

CEPH_RADOS_API void rados_read_op_cmpxattr(rados_read_op_t read_op, const char * name,
Ensure that the an xattr satisfies a comparison If the comparison is not satisfied, the return code of the operation will be -ECANCELED

Parameters

- `read_op` - operation to add this action to
- `name` - name of the xattr to look up
- `comparison_operator` - currently undocumented, look for LIBRADOS_CMPXATTR_OP_EQ in librados.h
- `value` - buffer to compare actual xattr value to
- `value_len` - length of buffer to compare actual xattr value to

CEPH_RADOS_API void rados_read_op_getxattrs(rados_read_op_t read_op, rados_xattrs_iter
Start iterating over xattrs on an object.

Parameters

- `read_op` - operation to add this action to
- `iter` - where to store the iterator
- `prval` - where to store the return value of this action

CEPH_RADOS_API void rados_read_op_omap_cmp(rados_read_op_t read_op, const char * key,
Ensure that the an omap value satisfies a comparison, with the supplied value on the right hand side (i.e. for OP_LT, the comparison is actual_value < value.

Parameters

- `read_op` - operation to add this action to
- `key` - which omap value to compare
- `comparison_operator` - one of LIBRADOS_CMPXATTR_OP_EQ, LIBRADOS_CMPXATTR_OP_LT, or LIBRADOS_CMPXATTR_OP_GT
- `val` - value to compare with

- `val_len` - length of value in bytes
- `prval` - where to store the return value from this action

CEPH_RADOS_API void rados_read_op_stat(rados_read_op_t read_op, uint64_t * psize, time_

Get object size and mtime

Parameters

- `read_op` - operation to add this action to
- `psize` - where to store object size
- `pmtime` - where to store modification time
- `prval` - where to store the return value of this action

CEPH_RADOS_API void rados_read_op_read(rados_read_op_t read_op, uint64_t offset, size_

Read bytes from offset into buffer.

prlen will be filled with the number of bytes read if successful. A short read can only occur if the read reaches the end of the object.

Parameters

- `read_op` - operation to add this action to
- `offset` - offset to read from
- `len` - length of buffer
- `buffer` - where to put the data
- `bytes_read` - where to store the number of bytes read by this action
- `prval` - where to store the return value of this action

CEPH_RADOS_API void rados_read_op_exec(rados_read_op_t read_op, const char * cls, cons

Execute an OSD class method on an object See *rados_exec()* for general description.

The output buffer is allocated on the heap; the caller is expected to release that memory with *rados_buffer_free()*. The buffer and length pointers can all be NULL, in which case they are not filled in.

Parameters

- `read_op` - operation to add this action to
- `cls` - the name of the class
- `method` - the name of the method
- `in_buf` - where to find input
- `in_len` - length of in_buf in bytes
- `out_buf` - where to put librados-allocated output buffer
- `out_len` - length of out_buf in bytes
- `prval` - where to store the return value from the method

CEPH_RADOS_API void rados_read_op_exec_user_buf(rados_read_op_t read_op, const char *

Execute an OSD class method on an object See *rados_exec()* for general description.

If the output buffer is too small, prval will be set to -ERANGE and used_len will be 0.

Parameters

- `read_op` - operation to add this action to
- `cls` - the name of the class
- `method` - the name of the method
- `in_buf` - where to find input
- `in_len` - length of in_buf in bytes
- `out_buf` - user-provided buffer to read into
- `out_len` - length of out_buf in bytes
- `used_len` - where to store the number of bytes read into out_buf
- `prval` - where to store the return value from the method

`CEPH_RADOS_API void rados_read_op_omap_get_vals(rados_read_op_t read_op, const char *`

Start iterating over key/value pairs on an object.

They will be returned sorted by key.

Parameters

- `read_op` - operation to add this action to
- `start_after` - list keys starting after start_after
- `filter_prefix` - list only keys beginning with filter_prefix
- `max_return` - list no more than max_return key/value pairs
- `iter` - where to store the iterator
- `prval` - where to store the return value from this action

`CEPH_RADOS_API void rados_read_op_omap_get_keys(rados_read_op_t read_op, const char *`

Start iterating over keys on an object.

They will be returned sorted by key, and the iterator will fill in NULL for all values if specified.

Parameters

- `read_op` - operation to add this action to
- `start_after` - list keys starting after start_after
- `max_return` - list no more than max_return keys
- `iter` - where to store the iterator
- `prval` - where to store the return value from this action

`CEPH_RADOS_API void rados_read_op_omap_get_vals_by_keys(rados_read_op_t read_op, char`

Start iterating over specific key/value pairs

They will be returned sorted by key.

Parameters

- `read_op` - operation to add this action to
- `keys` - array of pointers to null-terminated keys to get
- `keys_len` - the number of strings in keys

- `iter` - where to store the iterator
- `prval` - where to store the return value from this action

CEPH_RADOS_API int rados_read_op_operate(rados_read_op_t read_op, rados_ioctx_t io, co

Perform a read operation synchronously

Parameters

- `read_op` - operation to perform
- `io` - the ioctx that the object is in
- `oid` - the object id
- `flags` - flags to apply to the entire operation (LIBRADOS_OPERATION_*)

CEPH_RADOS_API int rados_aio_read_op_operate(rados_read_op_t read_op, rados_ioctx_t io

Perform a read operation asynchronously

Parameters

- `read_op` - operation to perform
- `io` - the ioctx that the object is in
- `completion` - what to do when operation has been attempted
- `oid` - the object id
- `flags` - flags to apply to the entire operation (LIBRADOS_OPERATION_*)

CEPH_RADOS_API int rados_lock_exclusive(rados_ioctx_t io, const char * oid, const char

Take an exclusive lock on an object.

Return

0 on success, negative error code on failure

-EBUSY if the lock is already held by another (client, cookie) pair

-EEXIST if the lock is already held by the same (client, cookie) pair

Parameters

- `io` - the context to operate in
- `oid` - the name of the object
- `name` - the name of the lock
- `cookie` - user-defined identifier for this instance of the lock
- `desc` - user-defined lock description
- `duration` - the duration of the lock. Set to NULL for infinite duration.
- `flags` - lock flags

CEPH_RADOS_API int rados_lock_shared(rados_ioctx_t io, const char * o, const char * na

Take a shared lock on an object.

Return

0 on success, negative error code on failure

-EBUSY if the lock is already held by another (client, cookie) pair

-EEXIST if the lock is already held by the same (client, cookie) pair

Parameters

- `io` - the context to operate in
- `o` - the name of the object
- `name` - the name of the lock
- `cookie` - user-defined identifier for this instance of the lock
- `tag` - The tag of the lock
- `desc` - user-defined lock description
- `duration` - the duration of the lock. Set to NULL for infinite duration.
- `flags` - lock flags

CEPH_RADOS_API int rados_unlock(rados_ioctx_t io, const char * o, const char * name, c

Release a shared or exclusive lock on an object.

Return

0 on success, negative error code on failure

-ENOENT if the lock is not held by the specified (client, cookie) pair

Parameters

- `io` - the context to operate in
- `o` - the name of the object
- `name` - the name of the lock
- `cookie` - user-defined identifier for the instance of the lock

CEPH_RADOS_API ssize_t rados_list_lockers(rados_ioctx_t io, const char * o, const char

List clients that have locked the named object lock and information about the lock.

The number of bytes required in each buffer is put in the corresponding size out parameter. If any of the provided buffers are too short, -ERANGE is returned after these sizes are filled in.

Return

number of lockers on success, negative error code on failure

-ERANGE if any of the buffers are too short

Parameters

- `io` - the context to operate in
- `o` - the name of the object
- `name` - the name of the lock
- `exclusive` - where to store whether the lock is exclusive (1) or shared (0)
- `tag` - where to store the tag associated with the object lock
- `tag_len` - number of bytes in tag buffer
- `clients` - buffer in which locker clients are stored, separated by ‘’
- `clients_len` - number of bytes in the clients buffer
- `cookies` - buffer in which locker cookies are stored, separated by ‘’

- `cookies_len` - number of bytes in the cookies buffer
- `addrs` - buffer in which locker addresses are stored, separated by ‘’
- `addrs_len` - number of bytes in the clients buffer

`CEPH_RADOS_API int rados_break_lock(rados_ioctx_t io, const char * o, const char * nam`

Releases a shared or exclusive lock on an object, which was taken by the specified client.

Return

0 on success, negative error code on failure

-ENOENT if the lock is not held by the specified (client, cookie) pair

-EINVAL if the client cannot be parsed

Parameters

- `io` - the context to operate in
- `o` - the name of the object
- `name` - the name of the lock
- `client` - the client currently holding the lock
- `cookie` - user-defined identifier for the instance of the lock

`CEPH_RADOS_API int rados_blacklist_add(rados_t cluster, char * client_address, uint32_`

Blacklists the specified client from the OSDs

Return

0 on success, negative error code on failure

Parameters

- `cluster` - cluster handle
- `client_address` - client address
- `expire_seconds` - number of seconds to blacklist (0 for default)

`CEPH_RADOS_API int rados_mon_command(rados_t cluster, const char ** cmd, size_t cmdlen`

Send monitor command.

The result buffers are allocated on the heap; the caller is expected to release that memory with *rados_buffer_free()*. The buffer and length pointers can all be NULL, in which case they are not filled in.

Note

Takes command string in carefully-formatted JSON; must match defined commands, types, etc.

Return

0 on success, negative error code on failure

Parameters

- `cluster` - cluster handle
- `cmd` - an array of char *’s representing the command
- `cmdlen` - count of valid entries in cmd
- `inbuf` - any bulk input data (crush map, etc.)

- `outbuf` - double pointer to output buffer
- `outbuflen` - pointer to output buffer length
- `outs` - double pointer to status string
- `outslen` - pointer to status string length

`CEPH_RADOS_API int rados_mon_command_target(rados_t cluster, const char * name, const`
Send monitor command to a specific monitor.

The result buffers are allocated on the heap; the caller is expected to release that memory with *rados_buffer_free()*. The buffer and length pointers can all be NULL, in which case they are not filled in.

Note

Takes command string in carefully-formatted JSON; must match defined commands, types, etc.

Return

0 on success, negative error code on failure

Parameters

- `cluster` - cluster handle
- `name` - target monitor's name
- `cmd` - an array of char *'s representing the command
- `cmdlen` - count of valid entries in cmd
- `inbuf` - any bulk input data (crush map, etc.)
- `outbuf` - double pointer to output buffer
- `outbuflen` - pointer to output buffer length
- `outs` - double pointer to status string
- `outslen` - pointer to status string length

`CEPH_RADOS_API void rados_buffer_free(char * buf)`
free a rados-allocated buffer

Release memory allocated by librados calls like *rados_mon_command()*.

Parameters

- `buf` - buffer pointer

`CEPH_RADOS_API int rados_osd_command(rados_t cluster, int osdid, const char ** cmd, si`

`CEPH_RADOS_API int rados_pg_command(rados_t cluster, const char * pgstr, const char **`

`CEPH_RADOS_API int rados_monitor_log(rados_t cluster, const char * level, rados_log_ca`

struct `rados_pool_stat_t`
#include <librados.h> Usage information for a pool.

Public Members

uint64_t **`num_bytes`**
space used in bytes

uint64_t **num_kb**
: space used in KB

uint64_t **num_objects**
: number of objects in the pool

uint64_t **num_object_clones**
: number of clones of objects

uint64_t **num_object_copies**
: num_objects * num_replicas

uint64_t **num_objects_missing_on_primary**

uint64_t **num_objects_unfound**
: number of objects found on no OSDs

uint64_t **num_objects_degraded**
: number of objects replicated fewer times than they should be (but found on at least one OSD)

uint64_t **num_rd**

uint64_t **num_rd_kb**

uint64_t **num_wr**

uint64_t **num_wr_kb**

struct rados_cluster_stat_t
: *#include <librados.h>* Cluster-wide usage information

Public Members

uint64_t **kb**

uint64_t **kb_used**

uint64_t **kb_avail**

uint64_t **num_objects**

4.6.3 LibradosPP (C++)

Todo

write me!

4.6.4 Librados (Python)

The `rados` module is a thin Python wrapper for `librados`. The source is available in `/src/pybind/rados.py`. You may also install it as a package.

Installation

To install Python libraries for Ceph, see Getting librados for Python.

Getting Started

You can create your own Ceph client using Python. The following tutorial will show you how to import the Ceph Python module, connect to a Ceph cluster, and perform object operations as a `client.admin` user.

Note: To use the Ceph Python bindings, you must have access to a running Ceph cluster. To set one up quickly, see Getting Started.

First, create a Python source file for your Ceph client.

```
sudo vim client.py
```

Import the Module

To use the `rados.py` module, import it into your source file.

```
import rados
```

Configure a Cluster Handle

Before connecting to the Ceph Storage Cluster, create a cluster handle. By default, the cluster handle assumes a cluster named `ceph` (i.e., the default for deployment tools, and our Getting Started guides too), and a `client.admin` user name. You may change these defaults to suit your needs.

To connect to the Ceph Storage Cluster, your application needs to know where to find the Ceph Monitor. Provide this information to your application by specifying the path to your Ceph configuration file, which contains the location of the initial Ceph monitors.

```
import rados, sys

#Create Handle Examples.
cluster = rados.Rados(conffile='ceph.conf')
cluster = rados.Rados(conffile=sys.argv[1])
cluster = rados.Rados(conffile = 'ceph.conf', conf = dict (keyring = '/path/to/keyring'))
```

Ensure that the `conffile` argument provides the path and file name of your Ceph configuration file. You may use the `sys` module to avoid hard-coding the Ceph configuration path and file name.

Your Python client also requires a client keyring. For this example, we use the `client.admin` key by default. If you would like to specify the keyring when creating the cluster handle, you may use the `conf` argument. Alternatively, you may specify the keyring path in your Ceph configuration file. For example, you may add something like the following line to you Ceph configuration file:

```
keyring = /path/to/ceph.client.admin.keyring
```

For additional details on modifying your configuration via Python, see *Configuration*.

Connect to the Cluster

Once you have a cluster handle configured, you may connect to the cluster. With a connection to the cluster, you may execute methods that return information about the cluster.

```
import rados, sys

cluster = rados.Rados(conffile='ceph.conf')
print "\nlibrados version: " + str(cluster.version())
print "Will attempt to connect to: " + str(cluster.conf_get('mon initial members'))

cluster.connect()
print "\nCluster ID: " + cluster.get_fsid()

print "\n\nCluster Statistics"
print "=================="
cluster_stats = cluster.get_cluster_stats()

for key, value in cluster_stats.iteritems():
        print key, value
```

By default, Ceph authentication is `on`. Your application will need to know the location of the keyring. The `python-ceph` module doesn't have the default location, so you need to specify the keyring path. The easiest way to specify the keyring is to add it to the Ceph configuration file. The following Ceph configuration file example uses the `client.admin` keyring you generated with `ceph-deploy`.

```
[global]
...
keyring=/path/to/keyring/ceph.client.admin.keyring
```

Manage Pools

When connected to the cluster, the `Rados` API allows you to manage pools. You can list pools, check for the existence of a pool, create a pool and delete a pool.

```
print "\n\nPool Operations"
print "==============="

print "\nAvailable Pools"
print "----------------"
pools = cluster.list_pools()

for pool in pools:
        print pool

print "\nCreate 'test' Pool"
print "------------------"
cluster.create_pool('test')

print "\nPool named 'test' exists: " + str(cluster.pool_exists('test'))
print "\nVerify 'test' Pool Exists"
print "-------------------------"
pools = cluster.list_pools()

for pool in pools:
        print pool

print "\nDelete 'test' Pool"
print "------------------"
cluster.delete_pool('test')
print "\nPool named 'test' exists: " + str(cluster.pool_exists('test'))
```

Input/Output Context

Reading from and writing to the Ceph Storage Cluster requries an input/output context (ioctx). You can create an ioctx with the `open_ioctx()` method of the `Rados` class. The `ioctx_name` parameter is the name of the pool you wish to use.

```
ioctx = cluster.open_ioctx('data')
```

Once you have an I/O context, you can read/write objects, extended attributes, and perform a number of other operations. After you complete operations, ensure that you close the connection. For example:

```
print "\nClosing the connection."
ioctx.close()
```

Writing, Reading and Removing Objects

Once you create an I/O context, you can write objects to the cluster. If you write to an object that doesn't exist, Ceph creates it. If you write to an object that exists, Ceph overwrites it (except when you specify a range, and then it only overwrites the range). You may read objects (and object ranges) from the cluster. You may also remove objects from the cluster. For example:

```
print "\nWriting object 'hw' with contents 'Hello World!' to pool 'data'."
ioctx.write_full("hw", "Hello World!")

print "\n\nContents of object 'hw'\n------------------------\n"
print ioctx.read("hw")

print "\nRemoving object 'hw'"
ioctx.remove_object("hw")
```

Writing and Reading XATTRS

Once you create an object, you can write extended attributes (XATTRs) to the object and read XATTRs from the object. For example:

```
print "\n\nWriting XATTR 'lang' with value 'en_US' to object 'hw'"
ioctx.set_xattr("hw", "lang", "en_US")

print "\n\nGetting XATTR 'lang' from object 'hw'\n"
print ioctx.get_xattr("hw", "lang")
```

Listing Objects

If you want to examine the list of objects in a pool, you may retrieve the list of objects and iterate over them with the object iterator. For example:

```
object_iterator = ioctx.list_objects()

while True :

        try :
                rados_object = object_iterator.next()
                print "Object contents = " + rados_object.read()

        except StopIteration :
                break
```

The `Object` class provides a file-like interface to an object, allowing you to read and write content and extended attributes. Object operations using the I/O context provide additional functionality and asynchronous capabilities.

Cluster Handle API

The `Rados` class provides an interface into the Ceph Storage Daemon.

Configuration

The `Rados` class provides methods for getting and setting configuration values, reading the Ceph configuration file, and parsing arguments. You do not need to be connected to the Ceph Storage Cluster to invoke the following methods. See Storage Cluster Configuration for details on settings.

`Rados.`**`conf_get`**`(`*option*`)`

Get the value of a configuration option

Parameters **`option`** (*str*) – which option to read

Returns str - value of the option or None

Raises `TypeError`

`Rados.`**`conf_set`**`(`*option*, *val*`)`

Set the value of a configuration option

Parameters

- **`option`** (*str*) – which option to set
- **`option`** – value of the option

Raises `TypeError`, `ObjectNotFound`

`Rados.`**`conf_read_file`**`(`*path=None*`)`

Configure the cluster handle using a Ceph config file.

Parameters **`path`** (*str*) – path to the config file

`Rados.`**`conf_parse_argv`**`(`*args*`)`

Parse known arguments from args, and remove; returned args contain only those unknown to ceph

`Rados.`**`version`**`()`

Get the version number of the `librados` C library.

Returns a tuple of `(major, minor, extra)` components of the librados version

Connection Management

Once you configure your cluster handle, you may connect to the cluster, check the cluster `fsid`, retrieve cluster statistics, and disconnect (shutdown) from the cluster. You may also assert that the cluster handle is in a particular state (e.g., “configuring”, “connecting”, etc.).

`Rados.`**`connect`**`(`*timeout=0*`)`

Connect to the cluster. Use shutdown() to release resources.

`Rados.`**`shutdown`**`()`

Disconnects from the cluster. Call this explicitly when a Rados.connect()ed object is no longer used.

`Rados.`**`get_fsid`**`()`

Get the fsid of the cluster as a hexadecimal string.

Raises `Error`

Returns str - cluster fsid

`Rados.`**`get_cluster_stats`**`()`

Read usage info about the cluster

This tells you total space, space used, space available, and number of objects. These are not updated immediately when data is written, they are eventually consistent.

Returns

dict - contains the following keys:

- `kb` (int) - total space
- `kb_used` (int) - space used
- `kb_avail` (int) - free space available
- `num_objects` (int) - number of objects

`Rados.`**`require_state`**`(`**args*`)`

Checks if the Rados object is in a special state

Raises RadosStateError

Pool Operations

To use pool operation methods, you must connect to the Ceph Storage Cluster first. You may list the available pools, create a pool, check to see if a pool exists, and delete a pool.

`Rados.`**`list_pools`**`()`

Gets a list of pool names.

Returns list - of pool names.

`Rados.`**`create_pool`**`(`*pool_name*, *auid=None*, *crush_rule=None*`)`

Create a pool: - with default settings: if auid=None and crush_rule=None - owned by a specific auid: auid given and crush_rule=None - with a specific CRUSH rule: if auid=None and crush_rule given - with a specific CRUSH rule and auid: if auid and crush_rule given

Parameters

- **`pool_name`** (*str*) – name of the pool to create
- **`auid`** (*int*) – the id of the owner of the new pool
- **`crush_rule`** (*str*) – rule to use for placement in the new pool

Raises `TypeError`, `Error`

`Rados.`**`pool_exists`**`()`

Checks if a given pool exists.

Parameters **`pool_name`** (*str*) – name of the pool to check

Raises `TypeError`, `Error`

Returns bool - whether the pool exists, false otherwise.

`Rados.`**`delete_pool`**`(`*pool_name*`)`

Delete a pool and all data inside it.

The pool is removed from the cluster immediately, but the actual data is deleted in the background.

Parameters **`pool_name`** (*str*) – name of the pool to delete

Raises `TypeError`, `Error`

Input/Output Context API

To write data to and read data from the Ceph Object Store, you must create an Input/Output context (ioctx). The *Rados* class provides a *open_ioctx()* method. The remaining `ioctx` operations involve invoking methods of the *Ioctx* and other classes.

`Rados.`**`open_ioctx`**`(`*ioctx_name*`)`

Create an io context

The io context allows you to perform operations within a particular pool.

Parameters **`ioctx_name`** (*str*) – name of the pool

Raises `TypeError`, `Error`

Returns Ioctx - Rados Ioctx object

`Ioctx.`**`require_ioctx_open`**`()`

Checks if the rados.Ioctx object state is 'open'

Raises IoctxStateError

`Ioctx.`**`get_stats`**`()`

Get pool usage statistics

Returns

dict - contains the following keys:

- `num_bytes` (int) - size of pool in bytes
- `num_kb` (int) - size of pool in kbytes
- `num_objects` (int) - number of objects in the pool
- `num_object_clones` (int) - number of object clones
- `num_object_copies` (int) - number of object copies
- **`num_objects_missing_on_primary` (int) - number of objets** missing on primary
- `num_objects_unfound` (int) - number of unfound objects
- `num_objects_degraded` (int) - number of degraded objects
- `num_rd` (int) - bytes read
- `num_rd_kb` (int) - kbytes read
- `num_wr` (int) - bytes written
- `num_wr_kb` (int) - kbytes written

`Ioctx.`**`change_auid`**(*auid*)

Attempt to change an io context's associated auid "owner."

Requires that you have write permission on both the current and new auid.

Raises `Error`

`Ioctx.`**`get_last_version`**()

Return the version of the last object read or written to.

This exposes the internal version number of the last object read or written via this io context

Returns version of the last object used

`Ioctx.`**`close`**()

Close a rados.Ioctx object.

This just tells librados that you no longer need to use the io context. It may not be freed immediately if there are pending asynchronous requests on it, but you should not use an io context again after calling this function on it.

Object Operations

The Ceph Storage Cluster stores data as objects. You can read and write objects synchronously or asynchronously. You can read and write from offsets. An object has a name (or key) and data.

`Ioctx.`**`aio_write`**(*object_name*, *to_write*, *offset=0*, *oncomplete=None*, *onsafe=None*)

Write data to an object asynchronously

Queues the write and returns.

Parameters

- **`object_name`** (*str*) – name of the object
- **`to_write`** (*str*) – data to write
- **`offset`** (*int*) – byte offset in the object to begin writing at
- **`oncomplete`** (*completion*) – what to do when the write is safe and complete in memory on all replicas
- **`onsafe`** (*completion*) – what to do when the write is safe and complete on storage on all replicas

Raises `Error`

Returns completion object

`Ioctx.`**`aio_write_full`**(*object_name*, *to_write*, *oncomplete=None*, *onsafe=None*)

Asychronously write an entire object

The object is filled with the provided data. If the object exists, it is atomically truncated and then written. Queues the write and returns.

Parameters

- **`object_name`** (*str*) – name of the object
- **`to_write`** (*str*) – data to write
- **`oncomplete`** (*completion*) – what to do when the write is safe and complete in memory on all replicas
- **`onsafe`** (*completion*) – what to do when the write is safe and complete on storage on all replicas

Raises `Error`

Returns completion object

`Ioctx.`**`aio_append`**(*object_name*, *to_append*, *oncomplete=None*, *onsafe=None*)

Asychronously append data to an object

Queues the write and returns.

Parameters

- **`object_name`** (*str*) – name of the object
- **`to_append`** (*str*) – data to append
- **`offset`** (*int*) – byte offset in the object to begin writing at
- **`oncomplete`** (*completion*) – what to do when the write is safe and complete in memory on all replicas
- **`onsafe`** (*completion*) – what to do when the write is safe and complete on storage on all replicas

Raises `Error`

Returns completion object

`Ioctx.`**`write`**(*key*, *data*, *offset=0*)

Write data to an object synchronously

Parameters

- **`key`** (*str*) – name of the object
- **`data`** (*str*) – data to write
- **`offset`** (*int*) – byte offset in the object to begin writing at

Raises `TypeError`

Raises `LogicError`

Returns int - 0 on success

`Ioctx.`**`write_full`**(*key*, *data*)

Write an entire object synchronously.

The object is filled with the provided data. If the object exists, it is atomically truncated and then written.

Parameters

- **`key`** (*str*) – name of the object
- **`data`** (*str*) – data to write

Raises `TypeError`

Raises `Error`

Returns int - 0 on success

`Ioctx.`**`aio_flush`**()

Block until all pending writes in an io context are safe

Raises `Error`

`Ioctx.`**`set_locator_key`**`(`*loc_key*`)`
Set the key for mapping objects to pgs within an io context.

The key is used instead of the object name to determine which placement groups an object is put in. This affects all subsequent operations of the io context - until a different locator key is set, all objects in this io context will be placed in the same pg.

Parameters **`loc_key`** (*str*) – the key to use as the object locator, or NULL to discard any previously set key

Raises `TypeError`

`Ioctx.`**`aio_read`**`(`*object_name*, *length*, *offset*, *oncomplete*`)`
Asychronously read data from an object

oncomplete will be called with the returned read value as well as the completion:

oncomplete(completion, data_read)

Parameters

- **`object_name`** (*str*) – name of the object to read from
- **`length`** (*int*) – the number of bytes to read
- **`offset`** (*int*) – byte offset in the object to begin reading from
- **`oncomplete`** (*completion*) – what to do when the read is complete

Raises `Error`

Returns completion object

`Ioctx.`**`read`**`(`*key*, *length=8192*, *offset=0*`)`
Read data from an object synchronously

Parameters

- **`key`** (*str*) – name of the object
- **`length`** (*int*) – the number of bytes to read (default=8192)
- **`offset`** (*int*) – byte offset in the object to begin reading at

Raises `TypeError`

Raises `Error`

Returns str - data read from object

`Ioctx.`**`stat`**`(`*key*`)`
Get object stats (size/mtime)

Parameters **`key`** (*str*) – the name of the object to get stats from

Raises `TypeError`

Raises `Error`

Returns (size,timestamp)

`Ioctx.`**`trunc`**`(`*key*, *size*`)`
Resize an object

If this enlarges the object, the new area is logically filled with zeroes. If this shrinks the object, the excess data is removed.

Parameters

- **key** (*str*) – the name of the object to resize
- **size** (*int*) – the new size of the object in bytes

Raises `TypeError`

Raises `Error`

Returns int - 0 on success, otherwise raises error

`Ioctx.`**`remove_object`**`(`*key*`)`

Delete an object

This does not delete any snapshots of the object.

Parameters **key** (*str*) – the name of the object to delete

Raises `TypeError`

Raises `Error`

Returns bool - True on success

Object Extended Attributes

You may set extended attributes (XATTRs) on an object. You can retrieve a list of objects or XATTRs and iterate over them.

`Ioctx.`**`set_xattr`**`(`*key*, *xattr_name*, *xattr_value*`)`

Set an extended attribute on an object.

Parameters

- **key** (*str*) – the name of the object to set xattr to
- **xattr_name** (*str*) – which extended attribute to set
- **xattr_value** (*str*) – the value of the extended attribute

Raises `TypeError`

Raises `Error`

Returns bool - True on success, otherwise raise an error

`Ioctx.`**`get_xattrs`**`(`*oid*`)`

Start iterating over xattrs on an object.

Parameters **oid** – the name of the object to get xattrs from

Raises `TypeError`

Raises `Error`

Returns XattrIterator

`XattrIterator.`**`next`**`()`

Get the next xattr on the object

Raises StopIteration

Returns pair - of name and value of the next Xattr

`Ioctx.`**`get_xattr`**`(`*key*, *xattr_name*`)`

Get the value of an extended attribute on an object.

Parameters

- **`key`** (*str*) – the name of the object to get xattr from
- **`xattr_name`** (*str*) – which extended attribute to read

Raises `TypeError`

Raises `Error`

Returns str - value of the xattr

`Ioctx.`**`rm_xattr`**`(`*key*, *xattr_name*`)`

Removes an extended attribute on from an object.

Parameters

- **`key`** (*str*) – the name of the object to remove xattr from
- **`xattr_name`** (*str*) – which extended attribute to remove

Raises `TypeError`

Raises `Error`

Returns bool - True on success, otherwise raise an error

Object Interface

From an I/O context, you can retrieve a list of objects from a pool and iterate over them. The object interface provide makes each object look like a file, and you may perform synchronous operations on the objects. For asynchronous operations, you should use the I/O context methods.

`Ioctx.`**`list_objects`**`()`

Get ObjectIterator on rados.Ioctx object.

Returns ObjectIterator

`ObjectIterator.`**`next`**`()`

Get the next object name and locator in the pool

Raises StopIteration

Returns next rados.Ioctx Object

`Object.`**`read`**`(`*length = 1024*1024*`)`

`Object.`**`write`**`(`*string_to_write*`)`

`Object.`**`get_xattrs`**`()`

`Object.`**`get_xattr`**`(`*xattr_name*`)`

`Object.`**`set_xattr`**`(`*xattr_name*, *xattr_value*`)`

`Object.`**`rm_xattr`**`(`*xattr_name*`)`

`Object.`**`stat`**`()`

`Object.`**`remove`**`()`

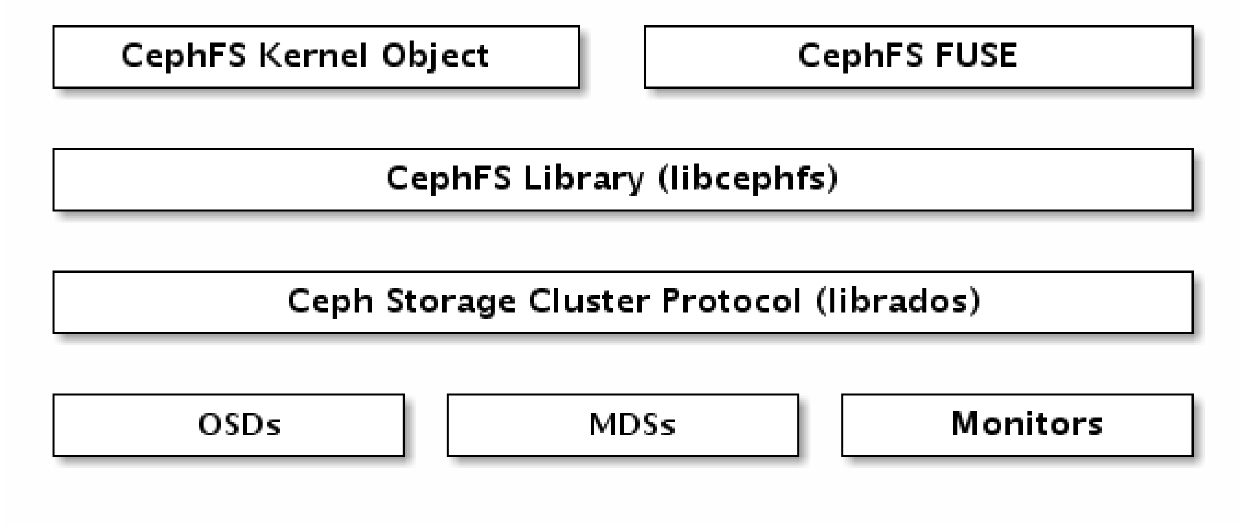
CHAPTER

FIVE

CEPH FILESYSTEM

The *Ceph Filesystem* (Ceph FS) is a POSIX-compliant filesystem that uses a Ceph Storage Cluster to store its data. The Ceph filesystem uses the same Ceph Storage Cluster system as Ceph Block Devices, Ceph Object Storage with its S3 and Swift APIs, or native bindings (librados).

Important: CephFS currently lacks a robust 'fsck' check and repair function. Please use caution when storing important data as the disaster recovery tools are still under development. For more information about using CephFS today, see `/cephfs/early-adopters`



CephFS Kernel Object
CephFS FUSE
CephFS Library (libcephfs)
Ceph Storage Cluster Protocol (librados)
OSDs
MDSs
Monitors

Using the Ceph Filesystem requires at least one *Ceph Metadata Server* in your Ceph Storage Cluster.

To run the Ceph Filesystem, you must have a running Ceph Storage Cluster with at least one *Ceph Metadata Server* running.

5.1 Add/Remove Metadata Server

With `ceph-deploy`, adding and removing metadata servers is a simple task. You just add or remove one or more metadata servers on the command line with one command.

Important: You must deploy at least one metadata server to use CephFS. There is experimental support for running multiple metadata servers. Do not run multiple metadata servers in production.

See MDS Config Reference for details on configuring metadata servers.

5.1.1 Add a Metadata Server

Once you deploy monitors and OSDs you may deploy the metadata server(s).

```
ceph-deploy mds create {host-name}[:{daemon-name}] [{host-name}[:{daemon-name}] ...]
```

You may specify a daemon instance a name (optional) if you would like to run multiple daemons on a single server.

5.1.2 Remove a Metadata Server

Coming soon...

5.2 MDS Config Reference

`mon force standby active`

Description If `true` monitors force standby-replay to be active. Set under `[mon]` or `[global]`.

Type Boolean

Default `true`

`max mds`

Description The number of active MDS daemons during cluster creation. Set under `[mon]` or `[global]`.

Type 32-bit Integer

Default `1`

`mds max file size`

Description The maximum allowed file size to set when creating a new file system.

Type 64-bit Integer Unsigned

Default `1ULL << 40`

`mds cache size`

Description The number of inodes to cache.

Type 32-bit Integer

Default `100000`

`mds cache mid`

Description The insertion point for new items in the cache LRU (from the top).

Type Float

Default `0.7`

`mds dir commit ratio`

Description The fraction of directory that is dirty before Ceph commits using a full update (instead of partial update).

Type Float

Default `0.5`

`mds dir max commit size`

Description The maximum size of a directory update before Ceph breaks it into smaller transactions) (MB).

Type 32-bit Integer

Default `90`

`mds decay halflife`

Description The half-life of MDS cache temperature.

Type Float

Default `5`

`mds beacon interval`

Description The frequency (in seconds) of beacon messages sent to the monitor.

Type Float

Default `4`

`mds beacon grace`

Description The interval without beacons before Ceph declares an MDS laggy (and possibly replace it).

Type Float

Default `15`

`mds blacklist interval`

Description The blacklist duration for failed MDSs in the OSD map.

Type Float

Default `24.0*60.0`

`mds session timeout`

Description The interval (in seconds) of client inactivity before Ceph times out capabilities and leases.

Type Float

Default `60`

`mds session autoclose`

Description The interval (in seconds) before Ceph closes a laggy client's session.

Type Float

Default `300`

`mds reconnect timeout`

Description The interval (in seconds) to wait for clients to reconnect during MDS restart.

Type Float

Default `45`

`mds tick interval`

Description How frequently the MDS performs internal periodic tasks.

Type Float

Default `5`

`mds dirstat min interval`

Description The minimum interval (in seconds) to try to avoid propagating recursive stats up the tree.

Type Float

Default `1`

`mds scatter nudge interval`

Description How quickly dirstat changes propagate up.

Type Float

Default `5`

`mds client prealloc inos`

Description The number of inode numbers to preallocate per client session.

Type 32-bit Integer

Default `1000`

`mds early reply`

Description Determines whether the MDS should allow clients to see request results before they commit to the journal.

Type Boolean

Default `true`

`mds use tmap`

Description Use trivialmap for directory updates.

Type Boolean

Default `true`

`mds default dir hash`

Description The function to use for hashing files across directory fragments.

Type 32-bit Integer

Default `2` (i.e., rjenkins)

`mds log`

Description Set to `true` if the MDS should journal metadata updates (disabled for benchmarking only).

Type Boolean

Default `true`

`mds log skip corrupt events`

Description Determines whether the MDS should try to skip corrupt journal events during journal replay.

Type Boolean

Default `false`

`mds log max events`

Description The maximum events in the journal before we initiate trimming. Set to `-1` to disable limits.

Type 32-bit Integer

Default `-1`

`mds log max segments`

Description The maximum number of segments (objects) in the journal before we initiate trimming. Set to `-1` to disable limits.

Type 32-bit Integer

Default `30`

`mds log max expiring`

Description The maximum number of segments to expire in parallels

Type 32-bit Integer

Default `20`

`mds log eopen size`

Description The maximum number of inodes in an EOpen event.

Type 32-bit Integer

Default `100`

`mds bal sample interval`

Description Determines how frequently to sample directory temperature (for fragmentation decisions).

Type Float

Default `3`

`mds bal replicate threshold`

Description The maximum temperature before Ceph attempts to replicate metadata to other nodes.

Type Float

Default `8000`

`mds bal unreplicate threshold`

Description The minimum temperature before Ceph stops replicating metadata to other nodes.

Type Float

Default `0`

`mds bal frag`

Description Determines whether the MDS will fragment directories.

Type Boolean

Default `false`

`mds bal split size`

Description The maximum directory size before the MDS will split a directory fragment into smaller bits.

Type 32-bit Integer

Default `10000`

`mds bal split rd`

Description The maximum directory read temperature before Ceph splits a directory fragment.

Type Float

Default `25000`

`mds bal split wr`

Description The maximum directory write temperature before Ceph splits a directory fragment.

Type Float

Default `10000`

`mds bal split bits`

Description The number of bits by which to split a directory fragment.

Type 32-bit Integer

Default `3`

`mds bal merge size`

Description The minimum directory size before Ceph tries to merge adjacent directory fragments.

Type 32-bit Integer

Default `50`

`mds bal merge rd`

Description The minimum read temperature before Ceph merges adjacent directory fragments.

Type Float

Default `1000`

`mds bal merge wr`

Description The minimum write temperature before Ceph merges adjacent directory fragments.

Type Float

Default `1000`

`mds bal interval`

Description The frequency (in seconds) of workload exchanges between MDSs.

Type 32-bit Integer

Default `10`

`mds bal fragment interval`

Description The frequency (in seconds) of adjusting directory fragmentation.

Type 32-bit Integer

Default `5`

`mds bal idle threshold`

Description The minimum temperature before Ceph migrates a subtree back to its parent.

Type Float

Default `0`

`mds bal max`

Description The number of iterations to run balancer before Ceph stops. (used for testing purposes only)

Type 32-bit Integer

Default `-1`

`mds bal max until`

Description The number of seconds to run balancer before Ceph stops. (used for testing purposes only)

Type 32-bit Integer

Default `-1`

`mds bal mode`

Description The method for calculating MDS load.

- `1` = Hybrid.
- `2` = Request rate and latency.
- `3` = CPU load.

Type 32-bit Integer

Default `0`

`mds bal min rebalance`

Description The minimum subtree temperature before Ceph migrates.

Type Float

Default `0.1`

`mds bal min start`

Description The minimum subtree temperature before Ceph searches a subtree.

Type Float

Default `0.2`

`mds bal need min`

Description The minimum fraction of target subtree size to accept.

Type Float

Default `0.8`

`mds bal need max`

Description The maximum fraction of target subtree size to accept.

Type Float

Default `1.2`

`mds bal midchunk`

Description Ceph will migrate any subtree that is larger than this fraction of the target subtree size.

Type Float

Default `0.3`

`mds bal minchunk`

Description Ceph will ignore any subtree that is smaller than this fraction of the target subtree size.

Type Float

Default `0.001`

`mds bal target removal min`

Description The minimum number of balancer iterations before Ceph removes an old MDS target from the MDS map.

Type 32-bit Integer

Default `5`

`mds bal target removal max`

Description The maximum number of balancer iteration before Ceph removes an old MDS target from the MDS map.

Type 32-bit Integer

Default `10`

`mds replay interval`

Description The journal poll interval when in standby-replay mode. ("hot standby")

Type Float

Default `1`

`mds shutdown check`

Description The interval for polling the cache during MDS shutdown.

Type 32-bit Integer

Default `0`

`mds thrash exports`

Description Ceph will randomly export subtrees between nodes (testing only).

Type 32-bit Integer

Default `0`

`mds thrash fragments`

Description Ceph will randomly fragment or merge directories.

Type 32-bit Integer

Default `0`

`mds dump cache on map`

Description Ceph will dump the MDS cache contents to a file on each MDSMap.

Type Boolean

Default `false`

`mds dump cache after rejoin`

Description Ceph will dump MDS cache contents to a file after rejoining the cache (during recovery).

Type Boolean

Default `false`

`mds verify scatter`

Description Ceph will assert that various scatter/gather invariants are `true` (developers only).

Type Boolean

Default `false`

`mds debug scatterstat`

Description Ceph will assert that various recursive stat invariants are `true` (for developers only).

Type Boolean

Default `false`

`mds debug frag`

Description Ceph will verify directory fragmentation invariants when convenient (developers only).

Type Boolean

Default `false`

`mds debug auth pins`

Description The debug auth pin invariants (for developers only).

Type Boolean

Default `false`

`mds debug subtrees`

Description The debug subtree invariants (for developers only).

Type Boolean

Default `false`

`mds kill mdstable at`

Description Ceph will inject MDS failure in MDSTable code (for developers only).

Type 32-bit Integer

Default `0`

`mds kill export at`

Description Ceph will inject MDS failure in the subtree export code (for developers only).

Type 32-bit Integer

Default `0`

`mds kill import at`

Description Ceph will inject MDS failure in the subtree import code (for developers only).

Type 32-bit Integer

Default `0`

`mds kill link at`

Description Ceph will inject MDS failure in hard link code (for developers only).

Type 32-bit Integer

Default `0`

`mds kill rename at`

Description Ceph will inject MDS failure in the rename code (for developers only).

Type 32-bit Integer

Default `0`

`mds wipe sessions`

Description Ceph will delete all client sessions on startup (for testing only).

Type Boolean

Default `0`

`mds wipe ino prealloc`

Description Ceph will delete ino preallocation metadata on startup (for testing only).

Type Boolean

Default `0`

`mds skip ino`

Description The number of inode numbers to skip on startup (for testing only).

Type 32-bit Integer

Default `0`

`mds standby for name`

Description An MDS daemon will standby for another MDS daemon of the name specified in this setting.

Type String

Default N/A

`mds standby for rank`

Description An MDS daemon will standby for an MDS daemon of this rank.

Type 32-bit Integer

Default `-1`

`mds standby replay`

Description Determines whether a `ceph-mds` daemon should poll and replay the log of an active MDS (hot standby).

Type Boolean

Default `false`

5.3 Journaler

`journaler allow split entries`

Description Allow an entry to span a stripe boundary

Type Boolean

Required No

Default `true`

`journaler write head interval`

Description How frequently to update the journal head object

Type Integer

Required No

Default `15`

`journaler prefetch periods`

Description How many stripe periods to read-ahead on journal replay

Type Integer

Required No

Default `10`

`journal prezero periods`

Description How mnay stripe periods to zero ahead of write position

Type Integer

Required No

Default `10`

`journaler batch interval`

Description Maximum additional latency in seconds we incur artificially.

Type Double

Required No

Default `.001`

`journaler batch max`

Description Maximum bytes we'll delay flushing.

Type 64-bit Unsigned Integer

Required No

Default `0`

5.4 ceph-mds – ceph metadata server daemon

5.4.1 Synopsis

ceph-mds -i *name* [[–hot-standby [*rank*]]|[–journal_check *rank*]]

5.4.2 Description

ceph-mds is the metadata server daemon for the Ceph distributed file system. One or more instances of ceph-mds collectively manage the file system namespace, coordinating access to the shared OSD cluster.

Each ceph-mds daemon instance should have a unique name. The name is used to identify daemon instances in the ceph.conf.

Once the daemon has started, the monitor cluster will normally assign it a logical rank, or put it in a standby pool to take over for another daemon that crashes. Some of the specified options can cause other behaviors.

If you specify hot-standby or journal-check, you must either specify the rank on the command line, or specify one of the mds_standby_for_[rank|name] parameters in the config. The command line specification overrides the config, and specifying the rank overrides specifying the name.

5.4.3 Options

-f, --foreground
Foreground: do not daemonize after startup (run in foreground). Do not generate a pid file. Useful when run via ceph-run(8).

-d
Debug mode: like `-f`, but also send all log output to stderr.

-c `ceph.conf`**, --conf**=`ceph.conf`
Use *ceph.conf* configuration file instead of the default `/etc/ceph/ceph.conf` to determine monitor addresses during startup.

-m `monaddress[:port]`
Connect to specified monitor (instead of looking through `ceph.conf`).

--journal-check `<rank>`
Attempt to replay the journal for MDS <rank>, then exit.

--hot-standby `<rank>`
Start as a hot standby for MDS <rank>.

5.4.4 Availability

ceph-mds is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

5.4.5 See also

ceph(8), ceph-mon(8), ceph-osd(8)

Once you have a healthy Ceph Storage Cluster with at least one Ceph Metadata Server, you may create and mount your Ceph Filesystem. Ensure that you client has network connectivity and the proper authentication keyring.

5.5 Create a Ceph filesystem

Tip: The `ceph fs new` command was introduced in Ceph 0.84. Prior to this release, no manual steps are required to create a filesystem, and pools named `data` and `metadata` exist by default.

The Ceph command line now includes commands for creating and removing filesystems, but at present only one filesystem may exist at a time.

A Ceph filesystem requires at least two RADOS pools, one for data and one for metadata. When configuring these pools, you might consider:

- Using a higher replication level for the metadata pool, as any data loss in this pool can render the whole filesystem inaccessible.
- Using lower-latency storage such as SSDs for the metadata pool, as this will directly affect the observed latency of filesystem operations on clients.

Refer to Pools to learn more about managing pools. For example, to create two pools with default settings for use with a filesystem, you might run the following commands:

```
$ ceph osd pool create cephfs_data <pg_num>
$ ceph osd pool create cephfs_metadata <pg_num>
```

Once the pools are created, you may enable the filesystem using the `fs new` command:

```
$ ceph fs new <fs_name> <metadata> <data>
```

For example:

```
$ ceph fs new cephfs cephfs_metadata cephfs_data
$ ceph fs ls
name: cephfs, metadata pool: cephfs_metadata, data pools: [cephfs_data ]
```

Once a filesystem has been created, your MDS(s) will be able to enter an *active* state. For example, in a single MDS system:

```
$ ceph mds stat
e5: 1/1/1 up {0=a=up:active}
```

Once the filesystem is created and the MDS is active, you are ready to mount the filesystem:

5.5.1 Mount Ceph FS with the Kernel Driver

To mount the Ceph file system you may use the `mount` command if you know the monitor host IP address(es), or use the `mount.ceph` utility to resolve the monitor host name(s) into IP address(es) for you. For example:

```
sudo mkdir /mnt/mycephfs
sudo mount -t ceph 192.168.0.1:6789:/ /mnt/mycephfs
```

To mount the Ceph file system with `cephx` authentication enabled, you must specify a user name and a secret.

```
sudo mount -t ceph 192.168.0.1:6789:/ /mnt/mycephfs -o name=admin,secret=AQATSKdNGBnwLhAAnNDKnH65FmV
```

The foregoing usage leaves the secret in the Bash history. A more secure approach reads the secret from a file. For example:

```
sudo mount -t ceph 192.168.0.1:6789:/ /mnt/mycephfs -o name=admin,secretfile=/etc/ceph/admin.secret
```

See Authentication for details on cephx.

To unmount the Ceph file system, you may use the `umount` command. For example:

```
sudo umount /mnt/mycephfs
```

Tip: Ensure that you are not within the file system directories before executing this command.

See mount.ceph for details.

5.5.2 Mount Ceph FS as a FUSE

For Ceph version 0.55 and later, `cephx` authentication is on by default. Before mounting a Ceph File System in User Space (FUSE), ensure that the client host has a copy of the Ceph configuration file and a keyring with CAPS for the Ceph metadata server.

1. From your client host, copy the Ceph configuration file from the monitor host to the `/etc/ceph` directory.

```
    sudo mkdir -p /etc/ceph
    sudo scp {user}@{server-machine}:/etc/ceph/ceph.conf /etc/ceph/ceph.conf
```

2. From your client host, copy the Ceph keyring from the monitor host to to the `/etc/ceph` directory.

```
    sudo scp {user}@{server-machine}:/etc/ceph/ceph.keyring /etc/ceph/ceph.keyring
```

3. Ensure that the Ceph configuration file and the keyring have appropriate permissions set on your client machine (e.g., `chmod 644`).

For additional details on `cephx` configuration, see CEPHX Config Reference.

To mount the Ceph file system as a FUSE, you may use the `ceph-fuse` command. For example:

```
sudo mkdir /home/usernname/cephfs
sudo ceph-fuse -m 192.168.0.1:6789 /home/username/cephfs
```

See ceph-fuse for additional details.

5.6 Mount Ceph FS with the Kernel Driver

To mount the Ceph file system you may use the `mount` command if you know the monitor host IP address(es), or use the `mount.ceph` utility to resolve the monitor host name(s) into IP address(es) for you. For example:

```
sudo mkdir /mnt/mycephfs
sudo mount -t ceph 192.168.0.1:6789:/ /mnt/mycephfs
```

To mount the Ceph file system with `cephx` authentication enabled, you must specify a user name and a secret.

```
sudo mount -t ceph 192.168.0.1:6789:/ /mnt/mycephfs -o name=admin,secret=AQATSKdNGBnwLhAAnNDKnH65FmV
```

The foregoing usage leaves the secret in the Bash history. A more secure approach reads the secret from a file. For example:

```
sudo mount -t ceph 192.168.0.1:6789:/ /mnt/mycephfs -o name=admin,secretfile=/etc/ceph/admin.secret
```

See Authentication for details on cephx.

To unmount the Ceph file system, you may use the `umount` command. For example:

```
sudo umount /mnt/mycephfs
```

Tip: Ensure that you are not within the file system directories before executing this command.

See mount.ceph for details.

5.7 Mount Ceph FS as a FUSE

For Ceph version 0.55 and later, `cephx` authentication is on by default. Before mounting a Ceph File System in User Space (FUSE), ensure that the client host has a copy of the Ceph configuration file and a keyring with CAPS for the Ceph metadata server.

1. From your client host, copy the Ceph configuration file from the monitor host to the `/etc/ceph` directory.

   ```
   sudo mkdir -p /etc/ceph
   sudo scp {user}@{server-machine}:/etc/ceph/ceph.conf /etc/ceph/ceph.conf
   ```

2. From your client host, copy the Ceph keyring from the monitor host to to the `/etc/ceph` directory.

   ```
   sudo scp {user}@{server-machine}:/etc/ceph/ceph.keyring /etc/ceph/ceph.keyring
   ```

3. Ensure that the Ceph configuration file and the keyring have appropriate permissions set on your client machine (e.g., `chmod 644`).

For additional details on `cephx` configuration, see CEPHX Config Reference.

To mount the Ceph file system as a FUSE, you may use the `ceph-fuse` command. For example:

```
sudo mkdir /home/usernname/cephfs
sudo ceph-fuse -m 192.168.0.1:6789 /home/username/cephfs
```

See ceph-fuse for additional details.

5.8 Mount Ceph FS in your File Systems Table

If you mount Ceph FS in your file systems table, the Ceph file system will mount automatically on startup.

5.8.1 Kernel Driver

To mount Ceph FS in your file systems table as a kernel driver, add the following to `/etc/fstab`:

```
{ipaddress}:{port}:/ {mount}/{mountpoint} {filesystem-name}     [name=username,secret=secretkey|secre
```

For example:

```
10.10.10.10:6789:/     /mnt/ceph    ceph    name=admin,secretfile=/etc/ceph/secret.key,noatime    0
```

Important: The `name` and `secret` or `secretfile` options are mandatory when you have Ceph authentication running.

See Authentication for details.

5.8.2 FUSE

To mount Ceph FS in your file systems table as a filesystem in user space, add the following to `/etc/fstab`:

```
#DEVICE                                  PATH         TYPE      OPTIONS
id={user-ID}[,conf={path/to/conf.conf}] /mount/path  fuse.ceph defaults 0 0
```

For example:

```
id=admin  /mnt/ceph  fuse.ceph defaults 0 0
id=myuser,conf=/etc/ceph/cluster.conf  /mnt/ceph2  fuse.ceph defaults 0 0
```

The `DEVICE` field is a comma-delimited list of options to pass to the command line. Ensure you use the ID (e.g., `admin`, not `client.admin`). You can pass any valid `ceph-fuse` option to the command line this way.

See Authentication for details.

5.9 cephfs – ceph file system options utility

5.9.1 Synopsis

cephfs [*path command options*]

5.9.2 Description

cephfs is a control utility for accessing and manipulating file layout and location data in the Ceph distributed storage system.

Choose one of the following three commands:

- `show_layout` View the layout information on a file or directory
- `set_layout` Set the layout information on a file or directory
- `show_location` View the location information on a file

5.9.3 Options

Your applicable options differ depending on whether you are setting or viewing layout/location.

Viewing options:

-l `-offset`
Specify an offset for which to retrieve location data

Setting options:

-u `-stripe_unit`
Set the size of each stripe

-c `-stripe_count`
Set the number of objects to stripe across

-s `-object_size`
Set the size of the objects to stripe across

-p `-pool`
Set the pool (by numeric value, not name!) to use

-o `-osd`
Set the preferred OSD to use as the primary (deprecated and ignored)

5.9.4 Limitations

When setting layout data, the specified object size must evenly divide by the specified stripe unit. Any parameters you don't set explicitly are left at the system defaults.

Obviously setting the layout of a file and a directory means different things. Setting the layout of a file specifies exactly how to place the individual file. This must be done before writing *any* data to it. Truncating a file does not allow you to change the layout either.

Setting the layout of a directory sets the "default layout", which is used to set the file layouts on any files subsequently created in the directory (or any subdirectory). Pre-existing files do not have their layouts changed.

You'll notice that the layout information allows you to specify a preferred OSD for placement. This feature is unsupported and ignored in modern versions of the Ceph servers; do not use it.

5.9.5 Availability

cephfs is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

5.9.6 See also

ceph(8)

5.10 ceph-fuse – FUSE-based client for ceph

5.10.1 Synopsis

ceph-fuse [-m *monaddr*:*port*] *mountpoint* [*fuse options*]

5.10.2 Description

ceph-fuse is a FUSE (File system in USErspace) client for Ceph distributed file system. It will mount a ceph file system (specified via the -m option for described by ceph.conf (see below) at the specific mount point.

The file system can be unmounted with:

```
fusermount -u mountpoint
```

or by sending `SIGINT` to the `ceph-fuse` process.

5.10.3 Options

Any options not recognized by ceph-fuse will be passed on to libfuse.

-d
Detach from console and daemonize after startup.

-c `ceph.conf`, **--conf**=`ceph.conf`
Use *ceph.conf* configuration file instead of the default `/etc/ceph/ceph.conf` to determine monitor addresses during startup.

-m `monaddress[:port]`
Connect to specified monitor (instead of looking through ceph.conf).

-r `root_directory`
Use root_directory as the mounted root, rather than the full Ceph tree.

5.10.4 Availability

ceph-fuse is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

5.10.5 See also

fusermount(8), ceph(8)

5.11 mount.ceph – mount a ceph file system

5.11.1 Synopsis

mount.ceph *monaddr1*[,*monaddr2*,...]:/[*subdir*] *dir* [-o *options*]

5.11.2 Description

mount.ceph is a simple helper for mounting the Ceph file system on a Linux host. It serves to resolve monitor hostname(s) into IP addresses and read authentication keys from disk; the Linux kernel client component does most of the real work. In fact, it is possible to mount a non-authenticated Ceph file system without mount.ceph by specifying monitor address(es) by IP:

```
mount -t ceph 1.2.3.4:/ mountpoint
```

Each monitor address monaddr takes the form host[:port]. If the port is not specified, the Ceph default of 6789 is assumed.

Multiple monitor addresses can be separated by commas. Only one responsible monitor is needed to successfully mount; the client will learn about all monitors from any responsive monitor. However, it is a good idea to specify more than one in case one happens to be down at the time of mount.

A subdirectory subdir may be specified if a subset of the file system is to be mounted.

Mount helper application conventions dictate that the first two options are device to be mounted and destination path. Options must be passed only after these fixed arguments.

5.11.3 Options

`wsize` int, max write size. Default: none (writeback uses smaller of wsize and stripe unit)

`rsize` int (bytes), max readahead, multiple of 1024, Default: 524288 (512*1024)

`osdtimeout` int (seconds), Default: 60

`osdkeepalivetimeout` int, Default: 5

`mount_timeout` int (seconds), Default: 60

`osd_idle_ttl` int (seconds), Default: 60

`caps_wanted_delay_min` int, cap release delay, Default: 5

`caps_wanted_delay_max` int, cap release delay, Default: 60

`cap_release_safety` int, Default: calculated

`readdir_max_entries` int, Default: 1024

`readdir_max_bytes` int, Default: 524288 (512*1024)

`write_congestion_kb` int (kb), max writeback in flight. scale with available memory. Default: calculated from available memory

`snapdirname` string, set the name of the hidden snapdir. Default: .snap

`name` RADOS user to authenticate as when using cephx. Default: guest

`secret` secret key for use with cephx. This option is insecure because it exposes the secret on the command line. To avoid this, use the secretfile option.

`secretfile` path to file containing the secret key to use with cephx

`ip` my ip

`noshare` create a new client instance, instead of sharing an existing instance of a client mounting the same cluster

`dirstat` funky *cat dirname* for stats, Default: off

`nodirstat` no funky *cat dirname* for stats

`rbytes` Report the recursive size of the directory contents for st_size on directories. Default: on

`norbytes` Do not report the recursive size of the directory contents for st_size on directories.

`nocrc` no data crc on writes

`noasyncreaddir` no dcache readdir

5.11.4 Examples

Mount the full file system:

```
mount.ceph monhost:/ /mnt/foo
```

If there are multiple monitors:

```
mount.ceph monhost1,monhost2,monhost3:/ /mnt/foo
```

If ceph-mon(8) is running on a non-standard port:

```
mount.ceph monhost1:7000,monhost2:7000,monhost3:7000:/ /mnt/foo
```

To mount only part of the namespace:

```
mount.ceph monhost1:/some/small/thing /mnt/thing
```

Assuming mount.ceph(8) is installed properly, it should be automatically invoked by mount(8) like so:

```
mount -t ceph monhost:/ /mnt/foo
```

5.11.5 Availability

mount.ceph is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

5.11.6 See also

ceph-fuse(8), ceph(8)

5.12 Using Hadoop with CephFS

The Ceph file system can be used as a drop-in replacement for the Hadoop File System (HDFS). This page describes the installation and configuration process of using Ceph with Hadoop.

5.12.1 Dependencies

- CephFS Java Interface
- Hadoop CephFS Plugin

Important: Currently requires Hadoop 1.1.X stable series

5.12.2 Installation

There are three requirements for using CephFS with Hadoop. First, a running Ceph installation is required. The details of setting up a Ceph cluster and the file system are beyond the scope of this document. Please refer to the Ceph documentation for installing Ceph.

The remaining two requirements are a Hadoop installation, and the Ceph file system Java packages, including the Java CephFS Hadoop plugin. The high-level steps are two add the dependencies to the Hadoop installation `CLASSPATH`, and configure Hadoop to use the Ceph file system.

CephFS Java Packages

- CephFS Hadoop plugin (hadoop-cephfs.jar)

Adding these dependencies to a Hadoop installation will depend on your particular deployment. In general the dependencies must be present on each node in the system that will be part of the Hadoop cluster, and must be in the `CLASSPATH` searched for by Hadoop. Typically approaches are to place the additional `jar` files into the `hadoop/lib` directory, or to edit the `HADOOP_CLASSPATH` variable in `hadoop-env.sh`.

The native Ceph file system client must be installed on each participating node in the Hadoop cluster.

5.12.3 Hadoop Configuration

This section describes the Hadoop configuration options used to control Ceph. These options are intended to be set in the Hadoop configuration file *conf/core-site.xml*.

Property	Value	Notes
fs.default.name	Ceph URI	ceph://[monaddr:port]/
ceph.conf.file	Local path to ceph.conf	/etc/ceph/ceph.conf
ceph.conf.options	Comma separated list of Ceph configuration key/value pairs	opt1=val1,opt2=val2
ceph.root.dir	Mount root directory	Default value: /
ceph.mon.address	Monitor address	host:port
ceph.auth.id	Ceph user id	Example: admin
ceph.auth.keyfile	Ceph key file	
ceph.auth.keyring	Ceph keyring file	
ceph.object.size	Default file object size in bytes	Default value (64MB): 67108864
ceph.data.pools	List of Ceph data pools for storing file.	Default value: default Ceph pool.
ceph.localize.reads	Allow reading from file replica objects	Default value: true

Support For Per-file Custom Replication

The Hadoop file system interface allows users to specify a custom replication factor (e.g. 3 copies of each block) when creating a file. However, object replication factors in the Ceph file system are controlled on a per-pool basis, and by default a Ceph file system will contain only a single pre-configured pool. Thus, in order to support per-file replication with Hadoop over Ceph, additional storage pools with non-default replications factors must be created, and Hadoop must be configured to choose from these additional pools.

Additional data pools can be specified using the `ceph.data.pools` configuration option. The value of the option is a comma separated list of pool names. The default Ceph pool will be used automatically if this configuration option is omitted or the value is empty. For example, the following configuration setting will consider the pools `pool1`, `pool2`, and `pool5` when selecting a target pool to store a file.

```
<property>
  <name>ceph.data.pools</name>
  <value>pool1,pool2,pool5</value>
</property>
```

Hadoop will not create pools automatically. In order to create a new pool with a specific replication factor use the `ceph osd pool create` command, and then set the `size` property on the pool using the `ceph osd pool set` command. For more information on creating and configuring pools see the RADOS Pool documentation.

Once a pool has been created and configured the metadata service must be told that the new pool may be used to store file data. A pool is be made available for storing file system data using the `ceph mds add_data_pool` command.

First, create the pool. In this example we create the `hadoop1` pool with replication factor 1.

```
ceph osd pool create hadoop1 100
ceph osd pool set hadoop1 size 1
```

Next, determine the pool id. This can be done by examining the output of the `ceph osd dump` command. For example, we can look for the newly created `hadoop1` pool.

```
ceph osd dump | grep hadoop1
```

The output should resemble:

```
pool 3 'hadoop1' rep size 1 min_size 1 crush_ruleset 0...
```

where `3` is the pool id. Next we will use the pool id reference to register the pool as a data pool for storing file system data.

```
ceph mds add_data_pool 3
```

The final step is to configure Hadoop to consider this data pool when selecting the target pool for new files.

```
<property>
        <name>ceph.data.pools</name>
        <value>hadoop1</value>
</property>
```

Pool Selection Rules

The following rules describe how Hadoop chooses a pool given a desired replication factor and the set of pools specified using the `ceph.data.pools` configuration option.

1. When no custom pools are specified the default Ceph data pool is used.
2. A custom pool with the same replication factor as the default Ceph data pool will override the default.
3. A pool with a replication factor that matches the desired replication will be chosen if it exists.
4. Otherwise, a pool with at least the desired replication factor will be chosen, or the maximum possible.

Debugging Pool Selection

Hadoop will produce log file entry when it cannot determine the replication factor of a pool (e.g. it is not configured as a data pool). The log message will appear as follows:

```
Error looking up replication of pool: <pool name>
```

Hadoop will also produce a log entry when it wasn't able to select an exact match for replication. This log entry will appear as follows:

```
selectDataPool path=<path> pool:repl=<name>:<value> wanted=<value>
```

5.13 Libcephfs (JavaDoc)

View the auto-generated JavaDoc pages for the CephFS Java bindings.

5.14 cephfs-journal-tool

5.14.1 Purpose

If a CephFS journal has become damaged, expert intervention may be required to restore the filesystem to a working state.

The `cephfs-journal-tool` utility provides functionality to aid experts in examining, modifying, and extracting data from journals.

Warning: This tool is **dangerous** because it directly modifies internal data structures of the filesystem. Make backups, be careful, and seek expert advice. If you are unsure, do not run this tool.

5.14.2 Syntax

```
cephfs-journal-tool journal <inspect|import|export|reset>
cephfs-journal-tool header <get|set>
cephfs-journal-tool event <get|splice|apply> [filter] <list|json|summary>
```

The tool operates in three modes: `journal`, `header` and `event`, meaning the whole journal, the header, and the events within the journal respectively.

5.14.3 Journal mode

This should be your starting point to assess the state of a journal.

- `inspect` reports on the health of the journal. This will identify any missing objects or corruption in the stored journal. Note that this does not identify inconsistencies in the events themselves, just that events are present and can be decoded.
- `import` and `export` read and write binary dumps of the journal in a sparse file format. Pass the filename as the last argument. The export operation may not work reliably for journals which are damaged (missing objects).
- `reset` truncates a journal, discarding any information within it.

Example: journal inspect

```
# cephfs-journal-tool journal inspect
Overall journal integrity: DAMAGED
Objects missing:
  0x1
Corrupt regions:
  0x400000-ffffffffffffffff
```

Example: Journal import/export

```
# cephfs-journal-tool journal export myjournal.bin
journal is 4194304~80643
read 80643 bytes at offset 4194304
wrote 80643 bytes at offset 4194304 to myjournal.bin
NOTE: this is a _sparse_ file; you can
    $ tar cSzf myjournal.bin.tgz myjournal.bin
      to efficiently compress it while preserving sparseness.

# cephfs-journal-tool journal import myjournal.bin
undump myjournal.bin
start 4194304 len 80643
writing header 200.00000000
 writing 4194304~80643
done.
```

Note: It is wise to use the `journal export <backup file>` command to make a journal backup before any further manipulation.

5.14.4 Header mode

- `get` outputs the current content of the journal header
- `set` modifies an attribute of the header. Allowed attributes are `trimmed_pos`, `expire_pos` and `write_pos`.

Example: header get/set

```
# cephfs-journal-tool header get
{ "magic": "ceph fs volume v011",
  "write_pos": 4274947,
  "expire_pos": 4194304,
  "trimmed_pos": 4194303,
  "layout": { "stripe_unit": 4194304,
      "stripe_count": 4194304,
      "object_size": 4194304,
      "cas_hash": 4194304,
      "object_stripe_unit": 4194304,
      "pg_pool": 4194304}}

# cephfs-journal-tool header set trimmed_pos 4194303
Updating trimmed_pos 0x400000 -> 0x3fffff
Successfully updated header.
```

5.14.5 Event mode

Event mode allows detailed examination and manipulation of the contents of the journal. Event mode can operate on all events in the journal, or filters may be applied.

The arguments following `cephfs-journal-tool event` consist of an action, optional filter parameters, and an output mode:

```
cephfs-journal-tool event <action> [filter] <output>
```

Actions:

- `get` read the events from the log
- `splice` erase events or regions in the journal
- `apply` extract filesystem metadata from events and attempt to apply it to the metadata store.

Filtering:

- `--range <int begin>..[int end]` only include events within the range begin (inclusive) to end (exclusive)
- `--path <path substring>` only include events referring to metadata containing the specified string
- `--inode <int>` only include events referring to metadata containing the specified string
- `--type <type string>` only include events of this type
- `--frag <ino>[.frag id]` only include events referring to this directory fragment
- `--dname <string>` only include events referring to this named dentry within a directory fragment (may only be used in conjunction with `--frag`
- `--client <int>` only include events from this client session ID

Filters may be combined on an AND basis (i.e. only the intersection of events from each filter).

Output modes:

- `binary`: write each event as a binary file, within a folder whose name is controlled by `--path`
- `json`: write all events to a single file, as a JSON serialized list of objects
- `summary`: write a human readable summary of the events read to standard out
- `list`: write a human readable terse listing of the type of each event, and which file paths the event affects.

Example: event mode

```
# cephfs-journal-tool event get json --path output.json
Wrote output to JSON file 'output.json'

# cephfs-journal-tool event get summary
Events by type:
  NOOP: 2
  OPEN: 2
  SESSION: 2
  SUBTREEMAP: 1
  UPDATE: 43

# cephfs-journal-tool event get list
0x400000 SUBTREEMAP:  ()
0x400308 SESSION:  ()
0x4003de UPDATE:  (setattr)
  /
0x40068b UPDATE:  (mkdir)
  diralpha
0x400d1b UPDATE:  (mkdir)
  diralpha/filealpha1
0x401666 UPDATE:  (unlink_local)
  stray0/10000000001
  diralpha/filealpha1
0x40228d UPDATE:  (unlink_local)
  diralpha
  stray0/10000000000
0x402bf9 UPDATE:  (scatter_writebehind)
  stray0
0x403150 UPDATE:  (mkdir)
  dirbravo
0x4037e0 UPDATE:  (openc)
  dirbravo/.filebravo1.swp
0x404032 UPDATE:  (openc)
  dirbravo/.filebravo1.swpx

# cephfs-journal-tool event get --path /filebravo1 list
0x40785a UPDATE:  (openc)
  dirbravo/filebravo1
0x4103ee UPDATE:  (cap update)
  dirbravo/filebravo1

# cephfs-journal-tool event splice --range 0x40f754..0x410bf1 summary
Events by type:
  OPEN: 1
  UPDATE: 2
```

```
# cephfs-journal-tool event apply --range 0x410bf1.. summary
Events by type:
  NOOP: 1
  SESSION: 1
  UPDATE: 9

# cephfs-journal-tool event get --inode=1099511627776 list
0x40068b UPDATE:  (mkdir)
  diralpha
0x400d1b UPDATE:  (mkdir)
  diralpha/filealpha1
0x401666 UPDATE:  (unlink_local)
  stray0/10000000001
  diralpha/filealpha1
0x40228d UPDATE:  (unlink_local)
  diralpha
  stray0/10000000000

# cephfs-journal-tool event get --frag=1099511627776 --dname=filealpha1 list
0x400d1b UPDATE:  (mkdir)
  diralpha/filealpha1
0x401666 UPDATE:  (unlink_local)
  stray0/10000000001
  diralpha/filealpha1

# cephfs-journal-tool event get binary --path bin_events
Wrote output to binary files in directory 'bin_events'
```

5.15 File layouts

The layout of a file controls how its contents are mapped to Ceph RADOS objects. You can read and write a file's layout using *virtual extended attributes* or xattrs.

The name of the layout xattrs depends on whether a file is a regular file or a directory. Regular files' layout xattrs are called `ceph.file.layout`, whereas directories' layout xattrs are called `ceph.dir.layout`. Where subsequent examples refer to `ceph.file.layout`, substitute `dir` as appropriate when dealing with directories.

Tip: Your linux distribution may not ship with commands for manipulating xattrs by default, the required package is usually called `attr`.

5.15.1 Layout fields

pool String, giving ID or name. Which RADOS pool a file's data objects will be stored in.

stripe_unit Integer in bytes. The size (in bytes) of a block of data used in the RAID 0 distribution of a file. All stripe units for a file have equal size. The last stripe unit is typically incomplete–i.e. it represents the data at the end of the file as well as unused "space" beyond it up to the end of the fixed stripe unit size.

stripe_count Integer. The number of consecutive stripe units that constitute a RAID 0 "stripe" of file data.

object_size Integer in bytes. File data is chunked into RADOS objects of this size.

5.15.2 Reading layouts with `getfattr`

Read the layout information as a single string:

```
$ touch file
$ getfattr -n ceph.file.layout file
# file: file
ceph.file.layout="stripe_unit=4194304 stripe_count=1 object_size=4194304 pool=cephfs_data"
```

Read individual layout fields:

```
$ getfattr -n ceph.file.layout.pool file
# file: file
ceph.file.layout.pool="cephfs_data"
$ getfattr -n ceph.file.layout.stripe_unit file
# file: file
ceph.file.layout.stripe_unit="4194304"
$ getfattr -n ceph.file.layout.stripe_count file
# file: file
ceph.file.layout.stripe_count="1"
$ getfattr -n ceph.file.layout.object_size file
# file: file
ceph.file.layout.object_size="4194304"
```

Note: When reading layouts, the pool will usually be indicated by name. However, in rare cases when pools have only just been created, the ID may be output instead.

Directories do not have an explicit layout until it is customized. Attempts to read the layout will fail if it has never been modified: this indicates that layout of the next ancestor directory with an explicit layout will be used.

```
$ mkdir dir
$ getfattr -n ceph.dir.layout dir
dir: ceph.dir.layout: No such attribute
$ setfattr -n ceph.dir.layout.stripe_count -v 2 dir
$ getfattr -n ceph.dir.layout dir
# file: dir
ceph.dir.layout="stripe_unit=4194304 stripe_count=2 object_size=4194304 pool=cephfs_data"
```

5.15.3 Writing layouts with `setfattr`

Layout fields are modified using `setfattr`:

```
$ ceph osd lspools
0 rbd,1 cephfs_data,2 cephfs_metadata,

$ setfattr -n ceph.file.layout.stripe_unit -v 1048576 file2
$ setfattr -n ceph.file.layout.stripe_count -v 8 file2
$ setfattr -n ceph.file.layout.object_size -v 10485760 file2
$ setfattr -n ceph.file.layout.pool -v 1 file2  # Setting pool by ID
$ setfattr -n ceph.file.layout.pool -v cephfs_data file2  # Setting pool by name
```

5.15.4 Inheritance of layouts

Files inherit the layout of their parent directory at creation time. However, subsequent changes to the parent directory's layout do not affect children.

```
$ getfattr -n ceph.dir.layout dir
# file: dir
ceph.dir.layout="stripe_unit=4194304 stripe_count=2 object_size=4194304 pool=cephfs_data"

# Demonstrate file1 inheriting its parent's layout
$ touch dir/file1
$ getfattr -n ceph.file.layout dir/file1
# file: dir/file1
ceph.file.layout="stripe_unit=4194304 stripe_count=2 object_size=4194304 pool=cephfs_data"

# Now update the layout of the directory before creating a second file
$ setfattr -n ceph.dir.layout.stripe_count -v 4 dir
$ touch dir/file2

# Demonstrate that file1's layout is unchanged
$ getfattr -n ceph.file.layout dir/file1
# file: dir/file1
ceph.file.layout="stripe_unit=4194304 stripe_count=2 object_size=4194304 pool=cephfs_data"

# ...while file2 has the parent directory's new layout
$ getfattr -n ceph.file.layout dir/file2
# file: dir/file2
ceph.file.layout="stripe_unit=4194304 stripe_count=4 object_size=4194304 pool=cephfs_data"
```

Files created as descendents of the directory also inherit the layout, if the intermediate directories do not have layouts set:

```
$ getfattr -n ceph.dir.layout dir
# file: dir
ceph.dir.layout="stripe_unit=4194304 stripe_count=4 object_size=4194304 pool=cephfs_data"
$ mkdir dir/childdir
$ getfattr -n ceph.dir.layout dir/childdir
dir/childdir: ceph.dir.layout: No such attribute
$ touch dir/childdir/grandchild
$ getfattr -n ceph.file.layout dir/childdir/grandchild
# file: dir/childdir/grandchild
ceph.file.layout="stripe_unit=4194304 stripe_count=4 object_size=4194304 pool=cephfs_data"
```

5.15.5 Adding a data pool to the MDS

Before you can use a pool with CephFS you have to add it to the Metadata Servers.

```
$ ceph mds add_data_pool cephfs_data_ssd
# Pool should now show up
$ ceph fs ls
.... data pools: [cephfs_data cephfs_data_ssd ]
```

Make sure that your cephx keys allows the client to access this new pool.

5.16 Ceph filesystem client eviction

When a filesystem client is unresponsive or otherwise misbehaving, it may be necessary to forcibly terminate its access to the filesystem. This process is called *eviction*.

This process is somewhat thorough in order to protect against data inconsistency resulting from misbehaving clients.

5.16.1 OSD blacklisting

First, prevent the client from performing any more data operations by *blacklisting* it at the RADOS level. You may be familiar with this concept as *fencing* in other storage systems.

Identify the client to evict from the MDS session list:

```
# ceph daemon mds.a session ls
[
    { "id": 4117,
      "num_leases": 0,
      "num_caps": 1,
      "state": "open",
      "replay_requests": 0,
      "reconnecting": false,
      "inst": "client.4117 172.16.79.251:0\/3271",
      "client_metadata": { "entity_id": "admin",
          "hostname": "fedoravm.localdomain",
          "mount_point": "\/home\/user\/mnt"}}]
```

In this case the 'fedoravm' client has address `172.16.79.251:0/3271`, so we blacklist it as follows:

```
# ceph osd blacklist add 172.16.79.251:0/3271
blacklisting 172.16.79.251:0/3271 until 2014-12-09 13:09:56.569368 (3600 sec)
```

5.16.2 OSD epoch barrier

While the evicted client is now marked as blacklisted in the central (mon) copy of the OSD map, it is now necessary to ensure that this OSD map update has propagated to all daemons involved in subsequent filesystem I/O. To do this, use the `osdmap barrier` MDS admin socket command.

First read the latest OSD epoch:

```
# ceph osd dump
epoch 12
fsid fd61ca96-53ff-4311-826c-f36b176d69ea
created 2014-12-09 12:03:38.595844
modified 2014-12-09 12:09:56.619957
...
```

In this case it is 12. Now request the MDS to barrier on this epoch:

```
# ceph daemon mds.a osdmap barrier 12
```

5.16.3 MDS session eviction

Finally, it is safe to evict the client's MDS session, such that any capabilities it held may be issued to other clients. The ID here is the `id` attribute from the `session ls` output:

```
# ceph daemon mds.a session evict 4117
```

That's it! The client has now been evicted, and any resources it had locked will now be available for other clients.

5.16.4 Background: OSD epoch barrier

The purpose of the barrier is to ensure that when we hand out any capabilities which might allow touching the same RADOS objects, the clients we hand out the capabilities to must have a sufficiently recent OSD map to not race with cancelled operations (from ENOSPC) or blacklisted clients (from evictions)

More specifically, the cases where we set an epoch barrier are:

- Client eviction (where the client is blacklisted and other clients must wait for a post-blacklist epoch to touch the same objects)
- OSD map full flag handling in the client (where the client may cancel some OSD ops from a pre-full epoch, so other clients must wait until the full epoch or later before touching the same objects).
- MDS startup, because we don't persist the barrier epoch, so must assume that latest OSD map is always required after a restart.

Note that this is a global value for simplicity: we could maintain this on a per-inode basis. We don't, because:

- It would be more complicated
- It would use an extra 4 bytes of memory for every inode
- It would not be much more efficient as almost always everyone has the latest OSD map anyway, in most cases everyone will breeze through this barrier rather than waiting.
- We only do this barrier in very rare cases, so any benefit from per-inode granularity would only very rarely be seen.

The epoch barrier is transmitted along with all capability messages, and instructs the receiver of the message to avoid sending any more RADOS operations to OSDs until it has seen this OSD epoch. This mainly applies to clients (doing their data writes directly to files), but also applies to the MDS because things like file size probing and file deletion are done directly from the MDS.

5.17 Handling a full Ceph filesystem

When a RADOS cluster reaches its `mon_osd_full_ratio` (default 95%) capacity, it is marked with the OSD full flag. This flag causes most normal RADOS clients to pause all operations until it is resolved (for example by adding more capacity to the cluster).

The filesystem has some special handling of the full flag, explained below.

5.17.1 Hammer and later

Since the hammer release, a full filesystem will lead to ENOSPC results from:

- Data writes on the client
- Metadata operations other than deletes and truncates

Because the full condition may not be encountered until data is flushed to disk (sometime after a `write` call has already returned 0), the ENOSPC error may not be seen until the application calls `fsync` or `fclose` (or equivalent) on the file handle.

Calling `fsync` is guaranteed to reliably indicate whether the data made it to disk, and will return an error if it doesn't. `fclose` will only return an error if buffered data happened to be flushed since the last write – a successful `fclose` does not guarantee that the data made it to disk, and in a full-space situation, buffered data may be discarded after an `fclose` if no space is available to persist it.

Warning: If an application appears to be misbehaving on a full filesystem, check that it is performing `fsync()` calls as necessary to ensure data is on disk before proceeding.

Data writes may be cancelled by the client if they are in flight at the time the OSD full flag is sent. Clients update the `osd_epoch_barrier` when releasing capabilities on files affected by cancelled operations, in order to ensure that these cancelled operations do not interfere with subsequent access to the data objects by the MDS or other clients. For more on the epoch barrier mechanism, see Ceph filesystem client eviction.

5.17.2 Legacy (pre-hammer) behavior

In versions of Ceph earlier than hammer, the MDS would ignore the full status of the RADOS cluster, and any data writes from clients would stall until the cluster ceased to be full.

There are two dangerous conditions to watch for with this behaviour:

- If a client had pending writes to a file, then it was not possible for the client to release the file to the MDS for deletion: this could lead to difficulty clearing space on a full filesystem
- If clients continued to create a large number of empty files, the resulting metadata writes from the MDS could lead to total exhaustion of space on the OSDs such that no further deletions could be performed.

5.18 Troubleshooting

5.18.1 Mount 5 Error

A mount 5 error typically occurs if a MDS server is laggy or if it crashed. Ensure at least one MDS is up and running, and the cluster is `active + healthy`.

5.18.2 Mount 12 Error

A mount 12 error with `cannot allocate memory` usually occurs if you have a version mismatch between the *Ceph Client* version and the *Ceph Storage Cluster* version. Check the versions using:

```
ceph -v
```

If the Ceph Client is behind the Ceph cluster, try to upgrade it:

```
sudo apt-get update && sudo apt-get install ceph-common
```

You may need to uninstall, autoclean and autoremove `ceph-common` and then reinstall it so that you have the latest version.

5.19 Disaster recovery

Danger: The notes in this section are aimed at experts, making a best effort to recovery what they can from damaged filesystems. These steps have the potential to make things worse as well as better. If you are unsure, do not proceed.

5.19.1 Journal export

Before attempting dangerous operations, make a copy of the journal like so:

```
cephfs-journal-tool journal export backup.bin
```

Note that this command may not always work if the journal is badly corrupted, in which case a RADOS-level copy should be made (http://tracker.ceph.com/issues/9902).

5.19.2 Dentry recovery from journal

If a journal is damaged or for any reason an MDS is incapable of replaying it, attempt to recover what file metadata we can like so:

```
cephfs-journal-tool event recover_dentries summary
```

This command by default acts on MDS rank 0, pass –rank=<n> to operate on other ranks.

This command will write any inodes/dentries recoverable from the journal into the backing store, if these inodes/dentries are higher-versioned than the previous contents of the backing store. If any regions of the journal are missing/damaged, they will be skipped.

Note that in addition to writing out dentries and inodes, this command will update the InoTables of each 'in' MDS rank, to indicate that any written inodes' numbers are now in use. In simple cases, this will result in an entirely valid backing store state.

Warning: The resulting state of the backing store is not guaranteed to be self-consistent, and an online MDS scrub will be required afterwards. The journal contents will not be modified by this command, you should truncate the journal separately after recovering what you can.

5.19.3 Journal truncation

If the journal is corrupt or MDSs cannot replay it for any reason, you can truncate it like so:

```
cephfs-journal-tool journal reset
```

Warning: Resetting the journal *will* lose metadata unless you have extracted it by other means such as `recover_dentries`. It is likely to leave some orphaned objects in the data pool. It may result in re-allocation of already-written inodes, such that permissions rules could be violated.

5.19.4 MDS table wipes

After the journal has been reset, it may no longer be consistent with respect to the contents of the MDS tables (InoTable, SessionMap, SnapServer).

To reset the SessionMap (erase all sessions), use:

```
cephfs-table-tool all reset session
```

This command acts on the tables of all 'in' MDS ranks. Replace 'all' with an MDS rank to operate on that rank only.

The session table is the table most likely to need resetting, but if you know you also need to reset the other tables then replace 'session' with 'snap' or 'inode'.

5.19.5 MDS map reset

Once the in-RADOS state of the filesystem (i.e. contents of the metadata pool) is somewhat recovered, it may be necessary to update the MDS map to reflect the contents of the metadata pool. Use the following command to reset the MDS map to a single MDS:

```
ceph fs reset <fs name> --yes-i-really-mean-it
```

Once this is run, any in-RADOS state for MDS ranks other than 0 will be ignored: as a result it is possible for this to result in data loss.

One might wonder what the difference is between 'fs reset' and 'fs remove; fs new'. The key distinction is that doing a remove/new will leave rank 0 in 'creating' state, such that it would overwrite any existing root inode on disk and orphan any existing files. In contrast, the 'reset' command will leave rank 0 in 'active' state such that the next MDS daemon to claim the rank will go ahead and use the existing in-RADOS metadata.

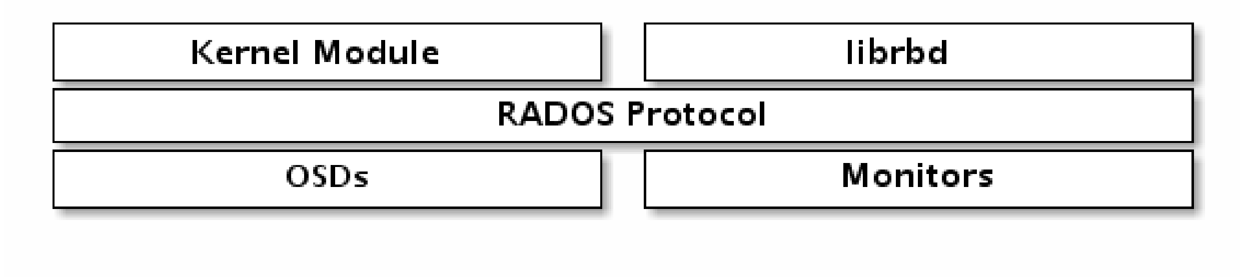
CHAPTER

SIX

CEPH BLOCK DEVICE

A block is a sequence of bytes (for example, a 512-byte block of data). Block-based storage interfaces are the most common way to store data with rotating media such as hard disks, CDs, floppy disks, and even traditional 9-track tape. The ubiquity of block device interfaces makes a virtual block device an ideal candidate to interact with a mass data storage system like Ceph.

Ceph block devices are thin-provisioned, resizable and store data striped over multiple OSDs in a Ceph cluster. Ceph block devices leverage RADOS capabilities such as snapshotting, replication and consistency. Ceph's RADOS Block Devices (RBD) interact with OSDs using kernel modules or the `librbd` library.



Note: Kernel modules can use Linux page caching. For `librbd`-based applications, Ceph supports RBD Caching.

Ceph's block devices deliver high performance with infinite scalability to kernel modules, or to KVMS (kernel virtual machines) such as Qemu, and cloud-based computing systems like OpenStack and CloudStack that rely on libvirt and Qemu to integrate with Ceph block devices. You can use the same cluster to operate the Ceph RADOS Gateway, the Ceph FS filesystem, and Ceph block devices simultaneously.

Important: To use Ceph Block Devices, you must have access to a running Ceph cluster.

6.1 Block Device Commands

The `rbd` command enables you to create, list, introspect and remove block device images. You can also use it to clone images, create snapshots, rollback an image to a snapshot, view a snapshot, etc. For details on using the `rbd` command, see RBD – Manage RADOS Block Device (RBD) Images for details.

Important: To use Ceph Block Device commands, you must have access to a running Ceph cluster.

6.1.1 Creating a Block Device Image

Before you can add a block device to a node, you must create an image for it in the *Ceph Storage Cluster* first. To create a block device image, execute the following:

```
rbd create {image-name} --size {megabytes} --pool {pool-name}
```

For example, to create a 1GB image named `bar` that stores information in a pool named `swimmingpool`, execute the following:

```
rbd create bar --size 1024 --pool swimmingpool
```

If you don't specify pool when creating an image, it will be stored in the default pool `rbd`. For example, to create a 1GB image named `foo` stored in the default pool `rbd`, execute the following:

```
rbd create foo --size 1024
```

Note: You must create a pool first before you can specify it as a source. See Storage Pools for details.

6.1.2 Listing Block Device Images

To list block devices in the `rbd` pool, execute the following (i.e., `rbd` is the default pool name):

```
rbd ls
```

To list block devices in a particular pool, execute the following, but replace `{poolname}` with the name of the pool:

```
rbd ls {poolname}
```

For example:

```
rbd ls swimmingpool
```

6.1.3 Retrieving Image Information

To retrieve information from a particular image, execute the following, but replace `{image-name}` with the name for the image:

```
rbd --image {image-name} info
```

For example:

```
rbd --image foo info
```

To retrieve information from an image within a pool, execute the following, but replace `{image-name}` with the name of the image and replace `{pool-name}` with the name of the pool:

```
rbd --image {image-name} -p {pool-name} info
```

For example:

```
rbd --image bar -p swimmingpool info
```

6.1.4 Resizing a Block Device Image

Ceph Block Device images are thin provisioned. They don't actually use any physical storage until you begin saving data to them. However, they do have a maximum capacity that you set with the `--size` option. If you want to increase (or decrease) the maximum size of a Ceph Block Device image, execute the following:

```
rbd resize --image foo --size 2048
```

6.1.5 Removing a Block Device Image

To remove a block device, execute the following, but replace `{image-name}` with the name of the image you want to remove:

```
rbd rm {image-name}
```

For example:

```
rbd rm foo
```

To remove a block device from a pool, execute the following, but replace `{image-name}` with the name of the image to remove and replace `{pool-name}` with the name of the pool:

```
rbd rm {image-name} -p {pool-name}
```

For example:

```
rbd rm bar -p swimmingpool
```

6.2 Kernel Module Operations

Important: To use kernel module operations, you must have a running Ceph cluster.

6.2.1 Get a List of Images

To mount a block device image, first return a list of the images.

```
rbd list
```

6.2.2 Map a Block Device

Use `rbd` to map an image name to a kernel module. You must specify the image name, the pool name, and the user name. `rbd` will load RBD kernel module on your behalf if it's not already loaded.

```
sudo rbd map {image-name} --pool {pool-name} --id {user-name}
```

For example:

```
sudo rbd map --pool rbd myimage --id admin
```

If you use cephx authentication, you must also specify a secret. It may come from a keyring or a file containing the secret.

```
sudo rbd map --pool rbd myimage --id admin --keyring /path/to/keyring
sudo rbd map --pool rbd myimage --id admin --keyfile /path/to/file
```

6.2.3 Show Mapped Block Devices

To show block device images mapped to kernel modules with the `rbd` command, specify the `showmapped` option.

```
rbd showmapped
```

6.2.4 Unmapping a Block Device

To unmap a block device image with the `rbd` command, specify the `unmap` option and the device name (i.e., by convention the same as the block device image name).

```
sudo rbd unmap /dev/rbd/{poolname}/{imagename}
```

For example:

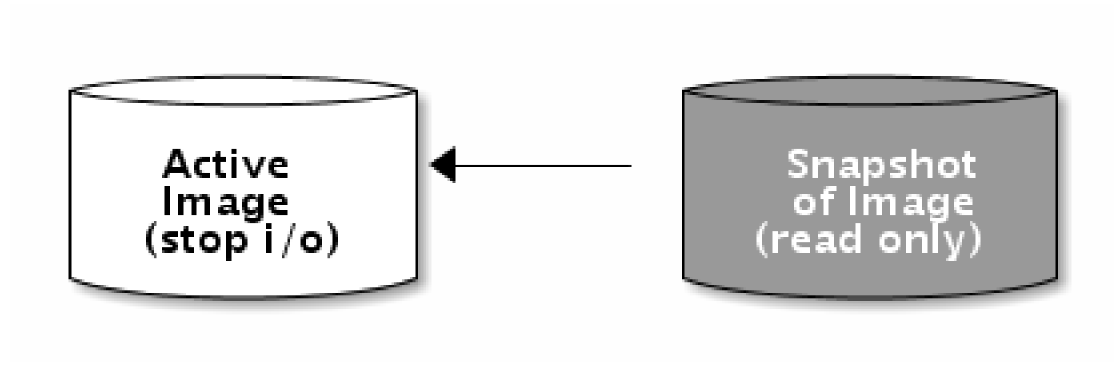
```
sudo rbd unmap /dev/rbd/rbd/foo
```

6.3 Snapshots

A snapshot is a read-only copy of the state of an image at a particular point in time. One of the advanced features of Ceph block devices is that you can create snapshots of the images to retain a history of an image's state. Ceph also supports snapshot layering, which allows you to clone images (e.g., a VM image) quickly and easily. Ceph supports block device snapshots using the `rbd` command and many higher level interfaces, including QEMU, libvirt, OpenStack and CloudStack.

Important: To use use RBD snapshots, you must have a running Ceph cluster.

Note: **STOP I/O BEFORE** snapshotting an image. If the image contains a filesystem, the filesystem must be in a consistent state **BEFORE** snapshotting.



6.3.1 Cephx Notes

When cephx is enabled (it is by default), you must specify a user name or ID and a path to the keyring containing the corresponding key for the user. See User Management for details. You may also add the `CEPH_ARGS` environment variable to avoid re-entry of the following parameters.

```
rbd --id {user-ID} --keyring=/path/to/secret [commands]
rbd --name {username} --keyring=/path/to/secret [commands]
```

For example:

```
rbd --id admin --keyring=/etc/ceph/ceph.keyring [commands]
rbd --name client.admin --keyring=/etc/ceph/ceph.keyring [commands]
```

Tip: Add the user and secret to the `CEPH_ARGS` environment variable so that you don't need to enter them each time.

6.3.2 Snapshot Basics

The following procedures demonstrate how to create, list, and remove snapshots using the `rbd` command on the command line.

Create Snapshot

To create a snapshot with `rbd`, specify the `snap create` option, the pool name and the image name.

```
rbd --pool {pool-name} snap create --snap {snap-name} {image-name}
rbd snap create {pool-name}/{image-name}@{snap-name}
```

For example:

```
rbd --pool rbd snap create --snap snapname foo
rbd snap create rbd/foo@snapname
```

List Snapshots

To list snapshots of an image, specify the pool name and the image name.

```
rbd --pool {pool-name} snap ls {image-name}
rbd snap ls {pool-name}/{image-name}
```

For example:

```
rbd --pool rbd snap ls foo
rbd snap ls rbd/foo
```

Rollback Snapshot

To rollback to a snapshot with `rbd`, specify the `snap rollback` option, the pool name, the image name and the snap name.

```
rbd --pool {pool-name} snap rollback --snap {snap-name} {image-name}
rbd snap rollback {pool-name}/{image-name}@{snap-name}
```

For example:

```
rbd --pool rbd snap rollback --snap snapname foo
rbd snap rollback rbd/foo@snapname
```

Note: Rolling back an image to a snapshot means overwriting the current version of the image with data from a snapshot. The time it takes to execute a rollback increases with the size of the image. It is **faster to clone** from a snapshot **than to rollback** an image to a snapshot, and it is the preferred method of returning to a pre-existing state.

Delete a Snapshot

To delete a snapshot with `rbd`, specify the `snap rm` option, the pool name, the image name and the snap name.

```
rbd --pool {pool-name} snap rm --snap {snap-name} {image-name}
rbd snap rm {pool-name}/{image-name}@{snap-name}
```

For example:

```
rbd --pool rbd snap rm --snap snapname foo
rbd snap rm rbd/foo@snapname
```

Note: Ceph OSDs delete data asynchronously, so deleting a snapshot doesn't free up the disk space immediately.

Purge Snapshots

To delete all snapshots for an image with `rbd`, specify the `snap purge` option and the image name.

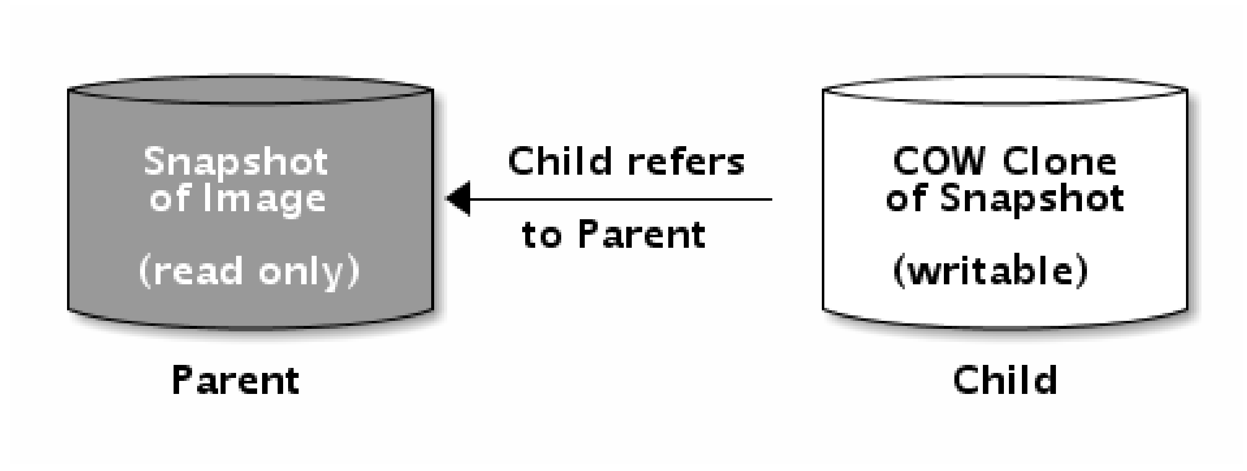
```
rbd --pool {pool-name} snap purge {image-name}
rbd snap purge {pool-name}/{image-name}
```

For example:

```
rbd --pool rbd snap purge foo
rbd snap purge rbd/foo
```

6.3.3 Layering

Ceph supports the ability to create many copy-on-write (COW) clones of a block device shapshot. Snapshot layering enables Ceph block device clients to create images very quickly. For example, you might create a block device image with a Linux VM written to it; then, snapshot the image, protect the snapshot, and create as many copy-on-write clones as you like. A snapshot is read-only, so cloning a snapshot simplifies semantics–making it possible to create clones rapidly.


Snapshot of Image (read only)
Child refers to Parent
COW Clone of Snapshot (writable)
Parent
Child

Note: The terms "parent" and "child" mean a Ceph block device snapshot (parent), and the corresponding image cloned from the snapshot (child). These terms are important for the command line usage below.

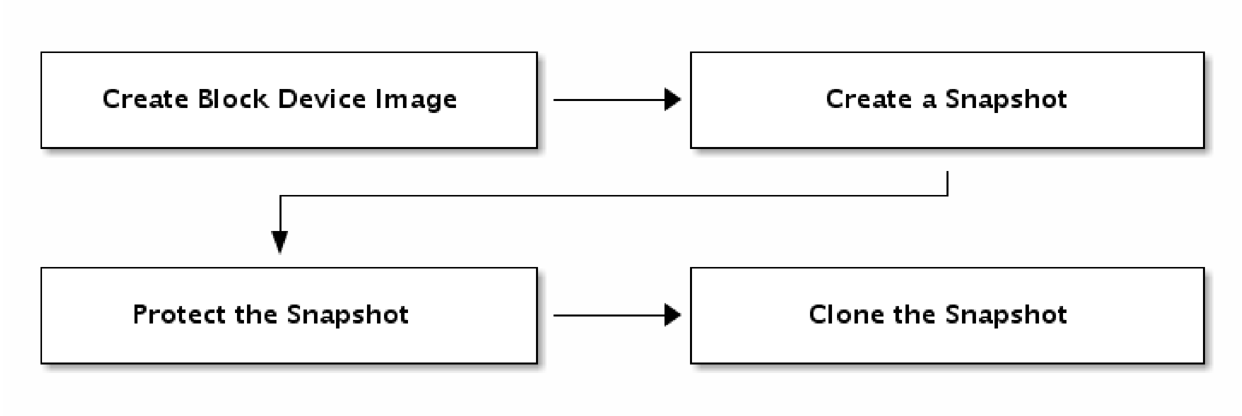
Each cloned image (child) stores a reference to its parent image, which enables the cloned image to open the parent snapshot and read it.

A COW clone of a snapshot behaves exactly like any other Ceph block device image. You can read to, write from, clone, and resize cloned images. There are no special restrictions with cloned images. However, the copy-on-write clone of a snapshot refers to the snapshot, so you **MUST** protect the snapshot before you clone it. The following diagram depicts the process.

Note: Ceph only supports cloning for `format 2` images (i.e., created with `rbd create --image-format 2`), and is not yet supported by the kernel `rbd` module. So you MUST use QEMU/KVM or `librbd` directly to access clones in the current release.

Getting Started with Layering

Ceph block device layering is a simple process. You must have an image. You must create a snapshot of the image. You must protect the snapshot. Once you have performed these steps, you can begin cloning the snapshot.



The cloned image has a reference to the parent snapshot, and includes the pool ID, image ID and snapshot ID. The inclusion of the pool ID means that you may clone snapshots from one pool to images in another pool.

1. **Image Template:** A common use case for block device layering is to create a a master image and a snapshot that

serves as a template for clones. For example, a user may create an image for a Linux distribution (e.g., Ubuntu 12.04), and create a snapshot for it. Periodically, the user may update the image and create a new snapshot (e.g., `sudo apt-get update`, `sudo apt-get upgrade`, `sudo apt-get dist-upgrade` followed by `rbd snap create`). As the image matures, the user can clone any one of the snapshots.

2. **Extended Template:** A more advanced use case includes extending a template image that provides more information than a base image. For example, a user may clone an image (e.g., a VM template) and install other software (e.g., a database, a content management system, an analytics system, etc.) and then snapshot the extended image, which itself may be updated just like the base image.
3. **Template Pool:** One way to use block device layering is to create a pool that contains master images that act as templates, and snapshots of those templates. You may then extend read-only privileges to users so that they may clone the snapshots without the ability to write or execute within the pool.
4. **Image Migration/Recovery:** One way to use block device layering is to migrate or recover data from one pool into another pool.

Protecting a Snapshot

Clones access the parent snapshots. All clones would break if a user inadvertently deleted the parent snapshot. To prevent data loss, you **MUST** protect the snapshot before you can clone it.

```
rbd --pool {pool-name} snap protect --image {image-name} --snap {snapshot-name}
rbd snap protect {pool-name}/{image-name}@{snapshot-name}
```

For example:

```
rbd --pool rbd snap protect --image my-image --snap my-snapshot
rbd snap protect rbd/my-image@my-snapshot
```

Note: You cannot delete a protected snapshot.

Cloning a Snapshot

To clone a snapshot, specify you need to specify the parent pool, image and snapshot; and, the child pool and image name. You must protect the snapshot before you can clone it.

```
rbd --pool {pool-name} --image {parent-image} --snap {snap-name} --dest-pool {pool-name} --dest {chi
rbd clone {pool-name}/{parent-image}@{snap-name} {pool-name}/{child-image-name}
```

For example:

```
rbd clone rbd/my-image@my-snapshot rbd/new-image
```

Note: You may clone a snapshot from one pool to an image in another pool. For example, you may maintain read-only images and snapshots as templates in one pool, and writeable clones in another pool.

Unprotecting a Snapshot

Before you can delete a snapshot, you must unprotect it first. Additionally, you may *NOT* delete snapshots that have references from clones. You must flatten each clone of a snapshot, before you can delete the snapshot.

```
rbd --pool {pool-name} snap unprotect --image {image-name} --snap {snapshot-name}
rbd snap unprotect {pool-name}/{image-name}@{snapshot-name}
```

For example:

```
rbd --pool rbd snap unprotect --image my-image --snap my-snapshot
rbd snap unprotect rbd/my-image@my-snapshot
```

Listing Children of a Snapshot

To list the children of a snapshot, execute the following:

```
rbd --pool {pool-name} children --image {image-name} --snap {snap-name}
rbd children {pool-name}/{image-name}@{snapshot-name}
```

For example:

```
rbd --pool rbd children --image my-image --snap my-snapshot
rbd children rbd/my-image@my-snapshot
```

Flattening a Cloned Image

Cloned images retain a reference to the parent snapshot. When you remove the reference from the child clone to the parent snapshot, you effectively "flatten" the image by copying the information from the snapshot to the clone. The time it takes to flatten a clone increases with the size of the snapshot. To delete a snapshot, you must flatten the child images first.

```
rbd --pool {pool-name} flatten --image {image-name}
rbd flatten {pool-name}/{image-name}
```

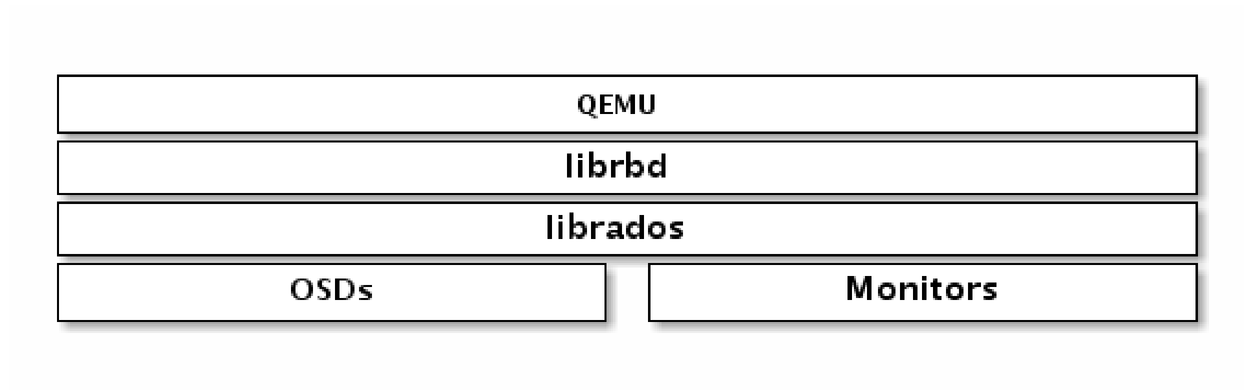
For example:

```
rbd --pool rbd flatten --image my-image
rbd flatten rbd/my-image
```

Note: Since a flattened image contains all the information from the snapshot, a flattened image will take up more storage space than a layered clone.

6.4 QEMU and Block Devices

The most frequent Ceph Block Device use case involves providing block device images to virtual machines. For example, a user may create a "golden" image with an OS and any relevant software in an ideal configuration. Then, the user takes a snapshot of the image. Finally, the user clones the snapshot (usually many times). See Snapshots for details. The ability to make copy-on-write clones of a snapshot means that Ceph can provision block device images to virtual machines quickly, because the client doesn't have to download an entire image each time it spins up a new virtual machine.



Ceph Block Devices can integrate with the QEMU virtual machine. For details on QEMU, see QEMU Open Source Processor Emulator. For QEMU documentation, see QEMU Manual. For installation details, see Installation.

Important: To use Ceph Block Devices with QEMU, you must have access to a running Ceph cluster.

6.4.1 Usage

The QEMU command line expects you to specify the pool name and image name. You may also specify a snapshot name.

QEMU will assume that the Ceph configuration file resides in the default location (e.g., `/etc/ceph/$cluster.conf`) and that you are executing commands as the default `client.admin` user unless you expressly specify another Ceph configuration file path or another user. When specifying a user, QEMU uses the `ID` rather than the full `TYPE:ID`. See User Management - User for details. Do not prepend the client type (i.e., `client.`) to the beginning of the user ID, or you will receive an authentication error. You should have the key for the `admin` user or the key of another user you specify with the `:id={user}` option in a keyring file stored in default path (i.e., `/etc/ceph` or the local directory with appropriate file ownership and permissions. Usage takes the following form:

```
qemu-img {command} [options] rbd:{pool-name}/{image-name}[@snapshot-name][:option1=value1][:option2=v
```

For example, specifying the `id` and `conf` options might look like the following:

```
qemu-img {command} [options] rbd:glance-pool/maipo:id=glance:conf=/etc/ceph/ceph.conf
```

Tip: Configuration values containing `:`, `@`, or = can be escaped with a leading \ character.

6.4.2 Creating Images with QEMU

You can create a block device image from QEMU. You must specify `rbd`, the pool name, and the name of the image you wish to create. You must also specify the size of the image.

```
qemu-img create -f raw rbd:{pool-name}/{image-name} {size}
```

For example:

```
qemu-img create -f raw rbd:data/foo 10G
```

Important: The `raw` data format is really the only sensible `format` option to use with RBD. Technically, you could use other QEMU-supported formats (such as `qcow2` or `vmdk`), but doing so would add additional overhead, and would also render the volume unsafe for virtual machine live migration when caching (see below) is enabled.

6.4.3 Resizing Images with QEMU

You can resize a block device image from QEMU. You must specify `rbd`, the pool name, and the name of the image you wish to resize. You must also specify the size of the image.

```
qemu-img resize rbd:{pool-name}/{image-name} {size}
```

For example:

```
qemu-img resize rbd:data/foo 10G
```

6.4.4 Retrieving Image Info with QEMU

You can retrieve block device image information from QEMU. You must specify `rbd`, the pool name, and the name of the image.

```
qemu-img info rbd:{pool-name}/{image-name}
```

For example:

```
qemu-img info rbd:data/foo
```

6.4.5 Running QEMU with RBD

QEMU can pass a block device from the host on to a guest, but since QEMU 0.15, there's no need to map an image as a block device on the host. Instead, QEMU can access an image as a virtual block device directly via `librbd`. This performs better because it avoids an additional context switch, and can take advantage of RBD caching.

You can use `qemu-img` to convert existing virtual machine images to Ceph block device images. For example, if you have a qcow2 image, you could run:

```
qemu-img convert -f qcow2 -O raw debian_squeeze.qcow2 rbd:data/squeeze
```

To run a virtual machine booting from that image, you could run:

```
qemu -m 1024 -drive format=raw,file=rbd:data/squeeze
```

RBD caching can significantly improve performance. Since QEMU 1.2, QEMU's cache options control `librbd` caching:

```
qemu -m 1024 -drive format=rbd,file=rbd:data/squeeze,cache=writeback
```

If you have an older version of QEMU, you can set the `librbd` cache configuration (like any Ceph configuration option) as part of the 'file' parameter:

```
qemu -m 1024 -drive format=raw,file=rbd:data/squeeze:rbd_cache=true,cache=writeback
```

Important: If you set rbd_cache=true, you must set cache=writeback or risk data loss. Without cache=writeback, QEMU will not send flush requests to librbd. If QEMU exits uncleanly in this configuration, filesystems on top of rbd can be corrupted.

6.4.6 Enabling Discard/TRIM

Since Ceph version 0.46 and QEMU version 1.1, Ceph Block Devices support the discard operation. This means that a guest can send TRIM requests to let a Ceph block device reclaim unused space. This can be enabled in the guest by mounting `ext4` or XFS with the `discard` option.

For this to be available to the guest, it must be explicitly enabled for the block device. To do this, you must specify a `discard_granularity` associated with the drive:

```
qemu -m 1024 -drive format=raw,file=rbd:data/squeeze,id=drive1,if=none \
     -device driver=ide-hd,drive=drive1,discard_granularity=512
```

Note that this uses the IDE driver. The virtio driver does not support discard.

If using libvirt, edit your libvirt domain's configuration file using `virsh edit` to include the `xmlns:qemu` value. Then, add a `qemu:commandline` block as a child of that domain. The following example shows how to set two devices with `qemu id=` to different `discard_granularity` values.

```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
        <qemu:commandline>
                <qemu:arg value='-set'/>
                <qemu:arg value='block.scsi0-0-0.discard_granularity=4096'/>
                <qemu:arg value='-set'/>
                <qemu:arg value='block.scsi0-0-1.discard_granularity=65536'/>
        </qemu:commandline>
</domain>
```

6.4.7 QEMU Cache Options

QEMU's cache options correspond to the following Ceph RBD Cache settings.

Writeback:

```
rbd_cache = true
```

Writethrough:

```
rbd_cache = true
rbd_cache_max_dirty = 0
```

None:

```
rbd_cache = false
```

QEMU's cache settings override Ceph's default settings (i.e., settings that are not explicitly set in the Ceph configuration file). If you explicitly set RBD Cache settings in your Ceph configuration file, your Ceph settings override the QEMU cache settings. If you set cache settings on the QEMU command line, the QEMU command line settings override the Ceph configuration file settings.

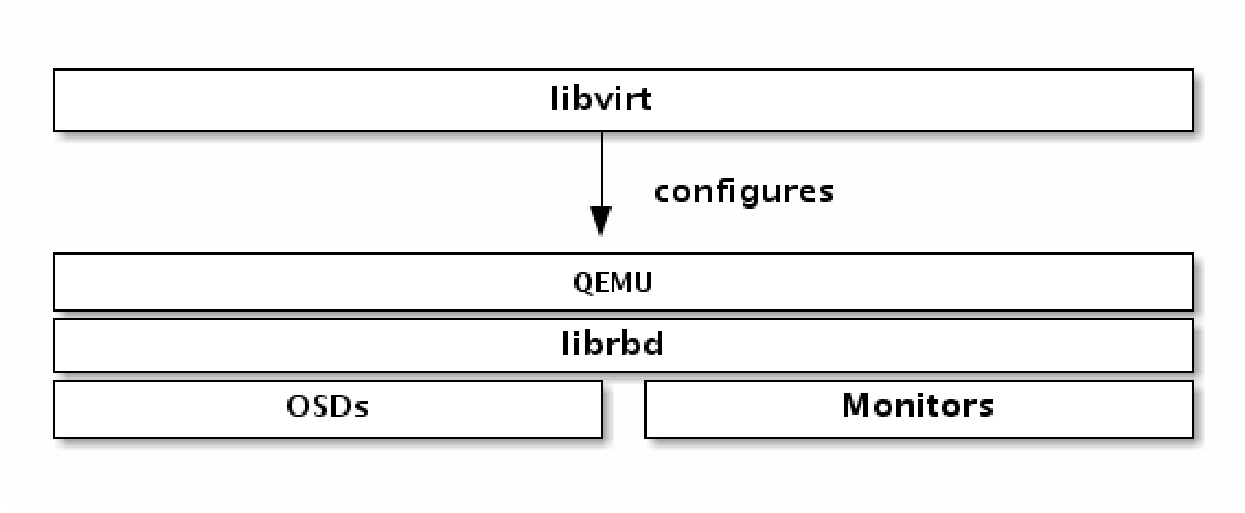
6.5 Using `libvirt` with Ceph RBD

The `libvirt` library creates a virtual machine abstraction layer between hypervisor interfaces and the software applications that use them. With `libvirt`, developers and system administrators can focus on a common management framework, common API, and common shell interface (i.e., `virsh`) to many different hypervisors, including:

- QEMU/KVM

- XEN
- LXC
- VirtualBox
- etc.

Ceph block devices support QEMU/KVM. You can use Ceph block devices with software that interfaces with `libvirt`. The following stack diagram illustrates how `libvirt` and QEMU use Ceph block devices via `librbd`.



The most common `libvirt` use case involves providing Ceph block devices to cloud solutions like OpenStack or CloudStack. The cloud solution uses `libvirt` to interact with QEMU/KVM, and QEMU/KVM interacts with Ceph block devices via `librbd`. See Block Devices and OpenStack and Block Devices and CloudStack for details. See Installation for installation details.

You can also use Ceph block devices with `libvirt`, `virsh` and the `libvirt` API. See libvirt Virtualization API for details.

To create VMs that use Ceph block devices, use the procedures in the following sections. In the exemplary embodiment, we've used `libvirt-pool` for the pool name, `client.libvirt` for the user name, and `new-libvirt-image` for the image name. You may use any value you like, but ensure you replace those values when executing commands in the subsequent procedures.

6.5.1 Configuring Ceph

To configure Ceph for use with `libvirt`, perform the following steps:

1. Create a pool (or use the default). The following example uses the pool name `libvirt-pool` with 128 placement groups.

```
ceph osd pool create libvirt-pool 128 128
```

   Verify the pool exists.

```
ceph osd lspools
```

2. Create a Ceph User (or use `client.admin` for version 0.9.7 and earlier). The following example uses the Ceph user name `client.libvirt` and references `libvirt-pool`.

```
ceph auth get-or-create client.libvirt mon 'allow r' osd 'allow class-read object_prefix rbd_chi
```

Verify the name exists.

```
ceph auth list
```

NOTE: `libvirt` will access Ceph using the ID `libvirt`, not the Ceph name `client.libvirt`. See User Management - User and User Management - CLI for a detailed explanation of the difference between ID and name.

3. Use QEMU to create an image in your RBD pool. The following example uses the image name `new-libvirt-image` and references `libvirt-pool`.

```
qemu-img create -f rbd rbd:libvirt-pool/new-libvirt-image 2G
```

Verify the image exists.

```
rbd -p libvirt-pool ls
```

NOTE: You can also use rbd create to create an image, but we recommend ensuring that QEMU is working properly.

6.5.2 Preparing the VM Manager

You may use `libvirt` without a VM manager, but you may find it simpler to create your first domain with `virt-manager`.

1. Install a virtual machine manager. See KVM/VirtManager for details.

```
sudo apt-get install virt-manager
```

2. Download an OS image (if necessary).
3. Launch the virtual machine manager.

```
sudo virt-manager
```

6.5.3 Creating a VM

To create a VM with `virt-manager`, perform the following steps:

1. Press the **Create New Virtual Machine** button.
2. Name the new virtual machine domain. In the exemplary embodiment, we use the name `libvirt-virtual-machine`. You may use any name you wish, but ensure you replace `libvirt-virtual-machine` with the name you choose in subsequent commandline and configuration examples.

```
libvirt-virtual-machine
```

3. Import the image.

```
/path/to/image/recent-linux.img
```

NOTE: Import a recent image. Some older images may not rescan for virtual devices properly.

4. Configure and start the VM.
5. You may use `virsh list` to verify the VM domain exists.

```
sudo virsh list
```

6. Login to the VM (root/root)
7. Stop the VM before configuring it for use with Ceph.

6.5.4 Configuring the VM

When configuring the VM for use with Ceph, it is important to use `virsh` where appropriate. Additionally, `virsh` commands often require root privileges (i.e., `sudo`) and will not return appropriate results or notify you that that root privileges are required. For a reference of `virsh` commands, refer to Virsh Command Reference.

1. Open the configuration file with `virsh edit`.

```
sudo virsh edit {vm-domain-name}
```

   Under `<devices>` there should be a `<disk>` entry.

```
<devices>
        <emulator>/usr/bin/kvm</emulator>
        <disk type='file' device='disk'>
                <driver name='qemu' type='raw'/>
                <source file='/path/to/image/recent-linux.img'/>
                <target dev='vda' bus='virtio'/>
                <address type='drive' controller='0' bus='0' unit='0'/>
        </disk>
```

   Replace `/path/to/image/recent-linux.img` with the path to the OS image. The minimum kernel for using the faster `virtio` bus is 2.6.25. See Virtio for details.

   IMPORTANT: Use `sudo virsh edit` instead of a text editor. If you edit the configuration file under `/etc/libvirt/qemu` with a text editor, `libvirt` may not recognize the change. If there is a discrepancy between the contents of the XML file under `/etc/libvirt/qemu` and the result of `sudo virsh dumpxml {vm-domain-name}`, then your VM may not work properly.

2. Add the Ceph RBD image you created as a `<disk>` entry.

```
<disk type='network' device='disk'>
        <source protocol='rbd' name='libvirt-pool/new-libvirt-image'>
                <host name='{monitor-host}' port='6789'/>
        </source>
        <target dev='vda' bus='virtio'/>
</disk>
```

   Replace `{monitor-host}` with the name of your host, and replace the pool and/or image name as necessary. You may add multiple `<host>` entries for your Ceph monitors. The `dev` attribute is the logical device name that will appear under the `/dev` directory of your VM. The optional `bus` attribute indicates the type of disk device to emulate. The valid settings are driver specific (e.g., "ide", "scsi", "virtio", "xen", "usb" or "sata").

   See Disks for details of the `<disk>` element, and its child elements and attributes.

3. Save the file.
4. If your Ceph Storage Cluster has Ceph Authentication enabled (it does by default), you must generate a secret.

```
cat > secret.xml <<EOF
<secret ephemeral='no' private='no'>
        <usage type='ceph'>
                <name>client.libvirt secret</name>
        </usage>
</secret>
EOF
```

5. Define the secret.

```
sudo virsh secret-define --file secret.xml
<uuid of secret is output here>
```

6. Get the `client.libvirt` key and save the key string to a file.

```
ceph auth get-key client.libvirt | sudo tee client.libvirt.key
```

7. Set the UUID of the secret.

```
sudo virsh secret-set-value --secret {uuid of secret} --base64 $(cat client.libvirt.key) && rm c
```

   You must also set the secret manually by adding the following `<auth>` entry to the `<disk>` element you entered earlier (replacing the `uuid` value with the result from the command line example above).

```
sudo virsh edit {vm-domain-name}
```

   Then, add `<auth></auth>` element to the domain configuration file:

```
...
</source>
<auth username='libvirt'>
        <secret type='ceph' uuid='9ec59067-fdbc-a6c0-03ff-df165c0587b8'/>
</auth>
<target ...
```

   NOTE: The exemplary ID is `libvirt`, not the Ceph name `client.libvirt` as generated at step 2 of *Configuring Ceph*. Ensure you use the ID component of the Ceph name you generated. If for some reason you need to regenerate the secret, you will have to execute `sudo virsh secret-undefine {uuid}` before executing `sudo virsh secret-set-value` again.

6.5.5 Summary

Once you have configured the VM for use with Ceph, you can start the VM. To verify that the VM and Ceph are communicating, you may perform the following procedures.

1. Check to see if Ceph is running:

```
ceph health
```

2. Check to see if the VM is running.

```
sudo virsh list
```

3. Check to see if the VM is communicating with Ceph. Replace `{vm-domain-name}` with the name of your VM domain:

```
sudo virsh qemu-monitor-command --hmp {vm-domain-name} 'info block'
```

4. Check to see if the device from `<target dev='hdb' bus='ide'/>` appears under `/dev` or under `proc/partitions`.

```
ls dev
cat proc/partitions
```

If everything looks okay, you may begin using the Ceph block device within your VM.

6.6 librbd Settings

See Block Device for additional details.

6.6.1 Cache Settings

Kernel Caching

The kernel driver for Ceph block devices can use the Linux page cache to improve performance.

The user space implementation of the Ceph block device (i.e., `librbd`) cannot take advantage of the Linux page cache, so it includes its own in-memory caching, called "RBD caching." RBD caching behaves just like well-behaved hard disk caching. When the OS sends a barrier or a flush request, all dirty data is written to the OSDs. This means that using write-back caching is just as safe as using a well-behaved physical hard disk with a VM that properly sends flushes (i.e. Linux kernel >= 2.6.32). The cache uses a Least Recently Used (LRU) algorithm, and in write-back mode it can coalesce contiguous requests for better throughput.

New in version 0.46.

Ceph supports write-back caching for RBD. To enable it, add `rbd cache = true` to the `[client]` section of your `ceph.conf` file. By default `librbd` does not perform any caching. Writes and reads go directly to the storage cluster, and writes return only when the data is on disk on all replicas. With caching enabled, writes return immediately, unless there are more than `rbd cache max dirty` unflushed bytes. In this case, the write triggers writeback and blocks until enough bytes are flushed.

New in version 0.47.

Ceph supports write-through caching for RBD. You can set the size of the cache, and you can set targets and limits to switch from write-back caching to write through caching. To enable write-through mode, set `rbd cache max dirty` to 0. This means writes return only when the data is on disk on all replicas, but reads may come from the cache. The cache is in memory on the client, and each RBD image has its own. Since the cache is local to the client, there's no coherency if there are others accessing the image. Running GFS or OCFS on top of RBD will not work with caching enabled.

The `ceph.conf` file settings for RBD should be set in the `[client]` section of your configuration file. The settings include:

`rbd cache`

Description Enable caching for RADOS Block Device (RBD).

Type Boolean

Required No

Default `true`

`rbd cache size`

Description The RBD cache size in bytes.

Type 64-bit Integer

Required No

Default `32 MiB`

`rbd cache max dirty`

Description The `dirty` limit in bytes at which the cache triggers write-back. If `0`, uses write-through caching.

Type 64-bit Integer

Required No

Constraint Must be less than `rbd cache size`.

Default `24 MiB`

`rbd cache target dirty`

Description The `dirty target` before the cache begins writing data to the data storage. Does not block writes to the cache.

Type 64-bit Integer

Required No

Constraint Must be less than `rbd cache max dirty`.

Default `16 MiB`

`rbd cache max dirty age`

Description The number of seconds dirty data is in the cache before writeback starts.

Type Float

Required No

Default `1.0`

New in version 0.60.

`rbd cache writethrough until flush`

Description Start out in write-through mode, and switch to write-back after the first flush request is received. Enabling this is a conservative but safe setting in case VMs running on rbd are too old to send flushes, like the virtio driver in Linux before 2.6.32.

Type Boolean

Required No

Default `true`

6.6.2 Read-ahead Settings

New in version 0.86.

RBD supports read-ahead/prefetching to optimize small, sequential reads. This should normally be handled by the guest OS in the case of a VM, but boot loaders may not issue efficient reads. Read-ahead is automatically disabled if caching is disabled.

`rbd readahead trigger requests`

Description Number of sequential read requests necessary to trigger read-ahead.

Type Integer

Required No

Default `10`

`rbd readahead max bytes`

Description Maximum size of a read-ahead request. If zero, read-ahead is disabled.

Type 64-bit Integer

Required No

Default `512 KiB`

`rbd readahead disable after bytes`

Description After this many bytes have been read from an RBD image, read-ahead is disabled for that image until it is closed. This allows the guest OS to take over read-ahead once it is booted. If zero, read-ahead stays enabled.

Type 64-bit Integer

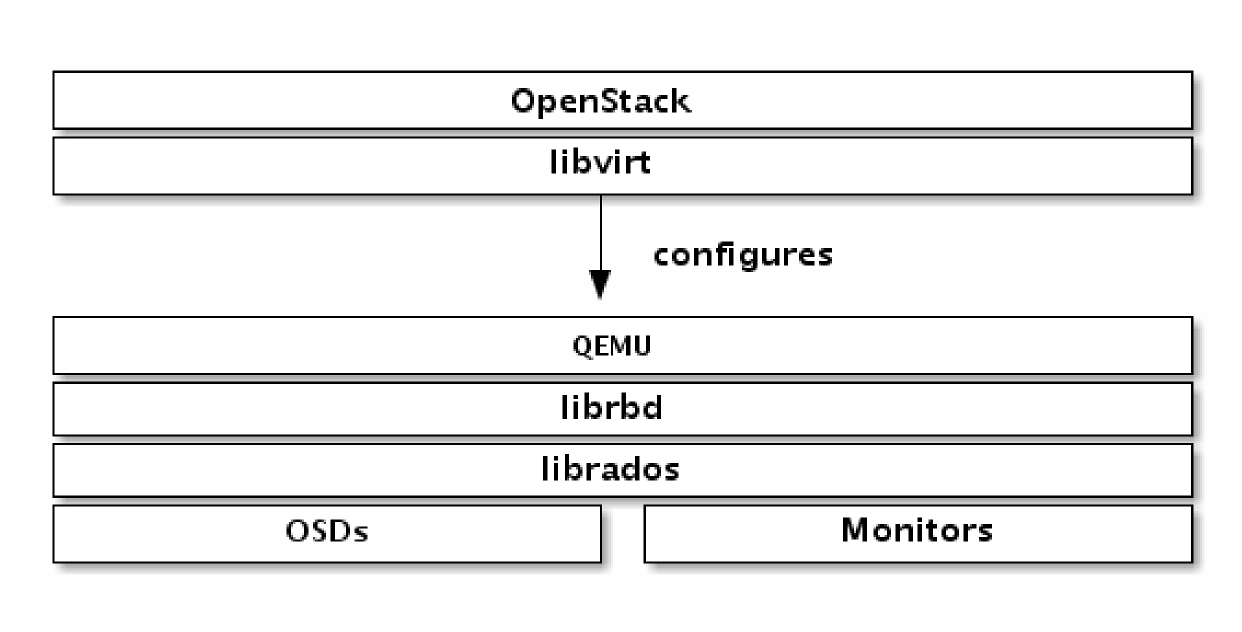
Required No

Default `50 MiB`

6.7 Block Devices and OpenStack

You may use Ceph Block Device images with OpenStack through `libvirt`, which configures the QEMU interface to `librbd`. Ceph stripes block device images as objects across the cluster, which means that large Ceph Block Device images have better performance than a standalone server!

To use Ceph Block Devices with OpenStack, you must install QEMU, `libvirt`, and OpenStack first. We recommend using a separate physical node for your OpenStack installation. OpenStack recommends a minimum of 8GB of RAM and a quad-core processor. The following diagram depicts the OpenStack/Ceph technology stack.



Important: To use Ceph Block Devices with OpenStack, you must have access to a running Ceph Storage Cluster.

Three parts of OpenStack integrate with Ceph's block devices:

- **Images**: OpenStack Glance manages images for VMs. Images are immutable. OpenStack treats images as binary blobs and downloads them accordingly.

- **Volumes**: Volumes are block devices. OpenStack uses volumes to boot VMs, or to attach volumes to running VMs. OpenStack manages volumes using Cinder services.
- **Guest Disks**: Guest disks are guest operating system disks. By default, when you boot a virtual machine, its disk appears as a file on the filesystem of the hypervisor (usually under `/var/lib/nova/instances/<uuid>/`). Prior to OpenStack Havana, the only way to boot a VM in Ceph was to use the boot-from-volume functionality of Cinder. However, now it is possible to boot every virtual machine inside Ceph directly without using Cinder, which is advantageous because it allows you to perform maintenance operations easily with the live-migration process. Additionally, if your hypervisor dies it is also convenient to trigger `nova evacuate` and run the virtual machine elsewhere almost seamlessly.

You can use OpenStack Glance to store images in a Ceph Block Device, and you can use Cinder to boot a VM using a copy-on-write clone of an image.

The instructions below detail the setup for Glance, Cinder and Nova, although they do not have to be used together. You may store images in Ceph block devices while running VMs using a local disk, or vice versa.

Important: Ceph doesn't support QCOW2 for hosting a virtual machine disk. Thus if you want to boot virtual machines in Ceph (ephemeral backend or boot from volume), the Glance image format must be `RAW`.

Tip: This document describes using Ceph Block Devices with OpenStack Havana. For earlier versions of OpenStack see Block Devices and OpenStack (Dumpling).

6.7.1 Create a Pool

By default, Ceph block devices use the `rbd` pool. You may use any available pool. We recommend creating a pool for Cinder and a pool for Glance. Ensure your Ceph cluster is running, then create the pools.

```
ceph osd pool create volumes 128
ceph osd pool create images 128
ceph osd pool create backups 128
ceph osd pool create vms 128
```

See Create a Pool for detail on specifying the number of placement groups for your pools, and Placement Groups for details on the number of placement groups you should set for your pools.

6.7.2 Configure OpenStack Ceph Clients

The nodes running `glance-api`, `cinder-volume`, `nova-compute` and `cinder-backup` act as Ceph clients. Each requires the `ceph.conf` file:

```
ssh {your-openstack-server} sudo tee /etc/ceph/ceph.conf </etc/ceph/ceph.conf
```

Install Ceph client packages

On the `glance-api` node, you'll need the Python bindings for `librbd`:

```
sudo apt-get install python-rbd
sudo yum install python-rbd
```

On the `nova-compute`, `cinder-backup` and on the `cinder-volume` node, use both the Python bindings and the client command line tools:

```
sudo apt-get install ceph-common
sudo yum install ceph
```

Setup Ceph Client Authentication

If you have cephx authentication enabled, create a new user for Nova/Cinder and Glance. Execute the following:

```
ceph auth get-or-create client.cinder mon 'allow r' osd 'allow class-read object_prefix rbd_children,
ceph auth get-or-create client.glance mon 'allow r' osd 'allow class-read object_prefix rbd_children,
ceph auth get-or-create client.cinder-backup mon 'allow r' osd 'allow class-read object_prefix rbd_ch
```

Add the keyrings for `client.cinder`, `client.glance`, and `client.cinder-backup` to the appropriate nodes and change their ownership:

```
ceph auth get-or-create client.glance | ssh {your-glance-api-server} sudo tee /etc/ceph/ceph.client.g
ssh {your-glance-api-server} sudo chown glance:glance /etc/ceph/ceph.client.glance.keyring
ceph auth get-or-create client.cinder | ssh {your-volume-server} sudo tee /etc/ceph/ceph.client.cinde
ssh {your-cinder-volume-server} sudo chown cinder:cinder /etc/ceph/ceph.client.cinder.keyring
ceph auth get-or-create client.cinder-backup | ssh {your-cinder-backup-server} sudo tee /etc/ceph/cep
ssh {your-cinder-backup-server} sudo chown cinder:cinder /etc/ceph/ceph.client.cinder-backup.keyring
```

Nodes running `nova-compute` need the keyring file for the `nova-compute` process:

```
ceph auth get-or-create client.cinder | ssh {your-nova-compute-server} sudo tee /etc/ceph/ceph.client
```

They also need to store the secret key of the `client.cinder` user in `libvirt`. The libvirt process needs it to access the cluster while attaching a block device from Cinder.

Create a temporary copy of the secret key on the nodes running `nova-compute`:

```
ceph auth get-key client.cinder | ssh {your-compute-node} tee client.cinder.key
```

Then, on the compute nodes, add the secret key to `libvirt` and remove the temporary copy of the key:

```
uuidgen
457eb676-33da-42ec-9a8c-9293d545c337

cat > secret.xml <<EOF
<secret ephemeral='no' private='no'>
  <uuid>457eb676-33da-42ec-9a8c-9293d545c337</uuid>
  <usage type='ceph'>
    <name>client.cinder secret</name>
  </usage>
</secret>
EOF
sudo virsh secret-define --file secret.xml
Secret 457eb676-33da-42ec-9a8c-9293d545c337 created
sudo virsh secret-set-value --secret 457eb676-33da-42ec-9a8c-9293d545c337 --base64 $(cat client.cinde
```

Save the uuid of the secret for configuring `nova-compute` later.

Important: You don't necessarily need the UUID on all the compute nodes. However from a platform consistency perspective, it's better to keep the same UUID.

6.7.3 Configure OpenStack to use Ceph

Configuring Glance

Glance can use multiple back ends to store images. To use Ceph block devices by default, configure Glance like the following.

Prior to Juno

Edit `/etc/glance/glance-api.conf` and add under the `[DEFAULT]` section:

```
default_store = rbd
rbd_store_user = glance
rbd_store_pool = images
rbd_store_chunk_size = 8
```

Juno

Edit `/etc/glance/glance-api.conf` and add under the `[glance_store]` section:

```
[DEFAULT]
...
default_store = rbd
...
[glance_store]
stores = rbd
rbd_store_pool = images
rbd_store_user = glance
rbd_store_ceph_conf = /etc/ceph/ceph.conf
rbd_store_chunk_size = 8
```

For more information about the configuration options available in Glance please see: http://docs.openstack.org/trunk/config-reference/content/section_glance-api.conf.html.

Important: Glance has not completely moved to 'store' yet. So we still need to configure the store in the DEFAULT section.

Any OpenStack version

If you want to enable copy-on-write cloning of images, also add under the `[DEFAULT]` section:

```
show_image_direct_url = True
```

Note that this exposes the back end location via Glance's API, so the endpoint with this option enabled should not be publicly accessible.

Disable the Glance cache management to avoid images getting cached under `/var/lib/glance/image-cache/`, assuming your configuration file has `flavor = keystone+cachemanagement`:

```
[paste_deploy]
flavor = keystone
```

Configuring Cinder

OpenStack requires a driver to interact with Ceph block devices. You must also specify the pool name for the block device. On your OpenStack node, edit `/etc/cinder/cinder.conf` by adding:

```
volume_driver = cinder.volume.drivers.rbd.RBDDriver
rbd_pool = volumes
rbd_ceph_conf = /etc/ceph/ceph.conf
rbd_flatten_volume_from_snapshot = false
rbd_max_clone_depth = 5
rbd_store_chunk_size = 4
rados_connect_timeout = -1
glance_api_version = 2
```

If you're using cephx authentication, also configure the user and uuid of the secret you added to `libvirt` as documented earlier:

```
rbd_user = cinder
rbd_secret_uuid = 457eb676-33da-42ec-9a8c-9293d545c337
```

Note that if you are configuring multiple cinder back ends, `glance_api_version = 2` must be in the `[DEFAULT]` section.

Configuring Cinder Backup

OpenStack Cinder Backup requires a specific daemon so don't forget to install it. On your Cinder Backup node, edit `/etc/cinder/cinder.conf` and add:

```
backup_driver = cinder.backup.drivers.ceph
backup_ceph_conf = /etc/ceph/ceph.conf
backup_ceph_user = cinder-backup
backup_ceph_chunk_size = 134217728
backup_ceph_pool = backups
backup_ceph_stripe_unit = 0
backup_ceph_stripe_count = 0
restore_discard_excess_bytes = true
```

Configuring Nova to attach Ceph RBD block device

In order to attach Cinder devices (either normal block or by issuing a boot from volume), you must tell Nova (and libvirt) which user and UUID to refer to when attaching the device. libvirt will refer to this user when connecting and authenticating with the Ceph cluster.

```
rbd_user = cinder
rbd_secret_uuid = 457eb676-33da-42ec-9a8c-9293d545c337
```

These two flags are also used by the Nova ephemeral backend.

Configuring Nova

In order to boot all the virtual machines directly into Ceph, you must configure the ephemeral backend for Nova.

It is recommended to enable the RBD cache in your Ceph configuration file (enabled by default since Giant). Moreover, enabling the admin socket brings a lot of benefits while troubleshoothing. Having one socket per virtual machine using a Ceph block device will help investigating performance and/or wrong behaviors.

This socket can be accessed like this:

```
ceph daemon /var/run/ceph/ceph-client.cinder.19195.32310016.asok help
```

Now on every compute nodes edit your Ceph configuration file:

```
[client]
    rbd cache = true
    rbd cache writethrough until flush = true
    admin socket = /var/run/ceph/$cluster-$type.$id.$pid.$cctid.asok
```

Tip: If your virtual machine is already running you can simply restart it to get the socket

Havana and Icehouse

Havana and Icehouse require patches to implement copy-on-write cloning and fix bugs with image size and live migration of ephemeral disks on rbd. These are available in branches based on upstream Nova stable/havana and stable/icehouse. Using them is not mandatory but **highly recommended** in order to take advantage of the copy-on-write clone functionality.

On every Compute node, edit `/etc/nova/nova.conf` and add:

```
libvirt_images_type = rbd
libvirt_images_rbd_pool = vms
libvirt_images_rbd_ceph_conf = /etc/ceph/ceph.conf
rbd_user = cinder
rbd_secret_uuid = 457eb676-33da-42ec-9a8c-9293d545c337
```

It is also a good practice to disable file injection. While booting an instance, Nova usually attempts to open the rootfs of the virtual machine. Then, Nova injects values such as password, ssh keys etc. directly into the filesystem. However, it is better to rely on the metadata service and `cloud-init`.

On every Compute node, edit `/etc/nova/nova.conf` and add:

```
libvirt_inject_password = false
libvirt_inject_key = false
libvirt_inject_partition = -2
```

To ensure a proper live-migration, use the following flags:

```
libvirt_live_migration_flag="VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRATE_PEER2PEER,VIR_MIGRATE_LIVE,VIR_
```

Juno

In Juno, Ceph block device was moved under the `[libvirt]` section. On every Compute node, edit `/etc/nova/nova.conf` under the `[libvirt]` section and add:

```
[libvirt]
images_type = rbd
images_rbd_pool = vms
images_rbd_ceph_conf = /etc/ceph/ceph.conf
rbd_user = cinder
rbd_secret_uuid = 457eb676-33da-42ec-9a8c-9293d545c337
```

It is also a good practice to disable file injection. While booting an instance, Nova usually attempts to open the rootfs of the virtual machine. Then, Nova injects values such as password, ssh keys etc. directly into the filesystem. However, it is better to rely on the metadata service and `cloud-init`.

On every Compute node, edit `/etc/nova/nova.conf` and add the following under the `[libvirt]` section:

```
inject_password = false
inject_key = false
inject_partition = -2
```

To ensure a proper live-migration, use the following flags:

```
live_migration_flag="VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRATE_PEER2PEER,VIR_MIGRATE_LIVE,VIR_MIGRATE_P
```

6.7.4 Restart OpenStack

To activate the Ceph block device driver and load the block device pool name into the configuration, you must restart OpenStack. Thus, for Debian based systems execute these commands on the appropriate nodes:

```
sudo glance-control api restart
sudo service nova-compute restart
sudo service cinder-volume restart
sudo service cinder-backup restart
```

For Red Hat based systems execute:

```
sudo service openstack-glance-api restart
sudo service openstack-nova-compute restart
sudo service openstack-cinder-volume restart
sudo service openstack-cinder-backup restart
```

Once OpenStack is up and running, you should be able to create a volume and boot from it.

6.7.5 Booting from a Block Device

You can create a volume from an image using the Cinder command line tool:

```
cinder create --image-id {id of image} --display-name {name of volume} {size of volume}
```

Note that image must be RAW format. You can use qemu-img to convert from one format to another. For example:

```
qemu-img convert -f {source-format} -O {output-format} {source-filename} {output-filename}
qemu-img convert -f qcow2 -O raw precise-cloudimg.img precise-cloudimg.raw
```

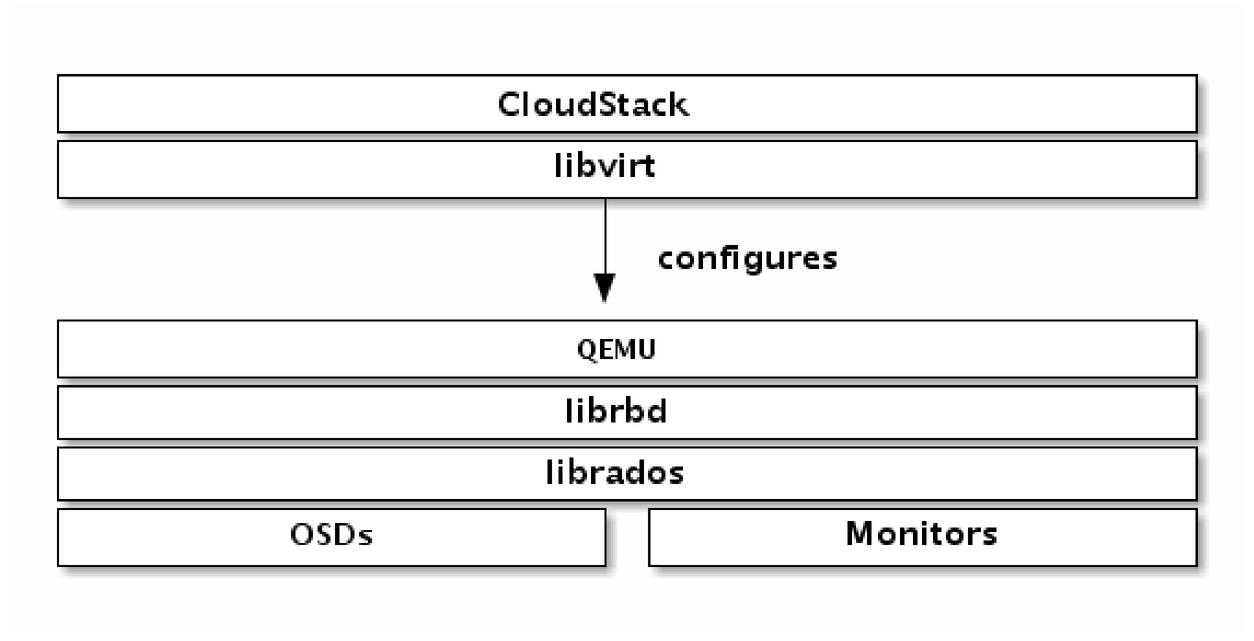
When Glance and Cinder are both using Ceph block devices, the image is a copy-on-write clone, so it can create a new volume quickly. In the OpenStack dashboard, you can boot from that volume by performing the following steps:

1. Launch a new instance.
2. Choose the image associated to the copy-on-write clone.
3. Select 'boot from volume'
4. Select the volume you created.

6.8 Block Devices and CloudStack

You may use Ceph Block Device images with CloudStack 4.0 and higher through `libvirt`, which configures the QEMU interface to `librbd`. Ceph stripes block device images as objects across the cluster, which means that large Ceph Block Device images have better performance than a standalone server!

To use Ceph Block Devices with CloudStack 4.0 and higher, you must install QEMU, `libvirt`, and CloudStack first. We recommend using a separate physical host for your CloudStack installation. CloudStack recommends a minimum of 4GB of RAM and a dual-core processor, but more CPU and RAM will perform better. The following diagram depicts the CloudStack/Ceph technology stack.



Important: To use Ceph Block Devices with CloudStack, you must have access to a running Ceph Storage Cluster.

CloudStack integrates with Ceph's block devices to provide CloudStack with a back end for CloudStack's Primary Storage. The instructions below detail the setup for CloudStack Primary Storage.

Note: We recommend installing with Ubuntu 14.04 or later so that you can use package installation instead of having to compile libvirt from source.

Note: Make sure the /tmp partition on your hypervisors is at least 25GB. When deploying from a template from the first time /tmp will be used for converting the template from QCOW2 to RAW for storage on RBD. This is no longer valid starting from CloudStack version 4.4.0

Note: To use RBD with CloudStack 4.4.0 you require at least librbd version 0.67.7 (Ceph Dumpling). Otherwise template deployments and template backups will fail. In case you use Ubuntu we recommend at least LTS version 14.04

Installing and configuring QEMU for use with CloudStack doesn't require any special handling. Ensure that you have a running Ceph Storage Cluster. Install QEMU and configure it for use with Ceph; then, install `libvirt` version 0.9.13 or higher (you may need to compile from source) and ensure it is running with Ceph.

Note: Raring Ringtail (13.04) will have `libvirt` version 0.9.13 or higher with RBD storage pool support enabled by default.

6.8.1 Create a Pool

By default, Ceph block devices use the `rbd` pool. Create a pool for CloudStack NFS Primary Storage. Ensure your Ceph cluster is running, then create the pool.

```
ceph osd pool create cloudstack
```

See Create a Pool for details on specifying the number of placement groups for your pools, and Placement Groups for details on the number of placement groups you should set for your pools.

6.8.2 Create a Ceph User

To access the Ceph cluster we require a Ceph user which has the correct credentials to access the `cloudstack` pool we just created. Although we could use `client.admin` for this, it's recommended to create a user with only access to the `cloudstack` pool.

```
ceph auth get-or-create client.cloudstack mon 'allow r' osd 'allow class-read object_prefix rbd_chil
```

Use the information returned by the command in the next step when adding the Primary Storage.

See User Management for additional details.

6.8.3 Add Primary Storage

To add primary storage, refer to Add Primary Storage (4.2.0) to add a Ceph block device, the steps include:

1. Log in to the CloudStack UI.
2. Click **Infrastructure** on the left side navigation bar.
3. Select the Zone you want to use for Primary Storage.
4. Click the **Compute** tab.
5. Select **View All** on the *Primary Storage* node in the diagram.
6. Click **Add Primary Storage**.
7. Follow the CloudStack instructions.
    - For **Protocol**, select `RBD`.
    - Add cluster information (cephx is supported). Note: Do not include the `client.` part of the user.
    - Add `rbd` as a tag.

6.8.4 Create a Disk Offering

To create a new disk offering, refer to Create a New Disk Offering (4.2.0). Create a disk offering so that it matches the `rbd` tag. The `StoragePoolAllocator` will choose the `rbd` pool when searching for a suitable storage pool. If the disk offering doesn't match the `rbd` tag, the `StoragePoolAllocator` may select the pool you created (e.g., `cloudstack`).

6.8.5 Limitations

- CloudStack will only bind to one monitor (You can however create a Round Robin DNS record over multiple monitors)
- You may need to compile `libvirt` to use version 0.9.13 with Ubuntu.

6.9 rbd – manage rados block device (RBD) images

6.9.1 Synopsis

rbd [-c *ceph.conf*] [-m *monaddr*] [-p | –pool *pool*] [–size *size*] [–order *bits*] [*command* ...]

6.9.2 Description

rbd is a utility for manipulating rados block device (RBD) images, used by the Linux rbd driver and the rbd storage driver for Qemu/KVM. RBD images are simple block devices that are striped over objects and stored in a RADOS object store. The size of the objects the image is striped over must be a power of two.

6.9.3 Options

`-c` `ceph.conf`**`, --conf`** `ceph.conf`
: Use ceph.conf configuration file instead of the default /etc/ceph/ceph.conf to determine monitor addresses during startup.

`-m` `monaddress[:port]`
: Connect to specified monitor (instead of looking through ceph.conf).

`-p` `pool`**`, --pool`** `pool`
: Interact with the given pool. Required by most commands.

`--no-progress`
: Do not output progress information (goes to standard error by default for some commands).

6.9.4 Parameters

`--image-format` `format`
: Specifies which object layout to use. The default is 1.

  - format 1 - Use the original format for a new rbd image. This format is understood by all versions of librbd and the kernel rbd module, but does not support newer features like cloning.
  - format 2 - Use the second rbd format, which is supported by librbd and kernel since version 3.11 (except for striping). This adds support for cloning and is more easily extensible to allow more features in the future.

`--size` `size-in-mb`
: Specifies the size (in megabytes) of the new rbd image.

`--order` `bits`
: Specifies the object size expressed as a number of bits, such that the object size is `1 << order`. The default is 22 (4 MB).

--stripe-unit `size-in-bytes`
: Specifies the stripe unit size in bytes. See striping section (below) for more details.

--stripe-count `num`
: Specifies the number of objects to stripe over before looping back to the first object. See striping section (below) for more details.

--snap `snap`
: Specifies the snapshot name for the specific operation.

--id `username`
: Specifies the username (without the `client.` prefix) to use with the map command.

--keyfile `filename`
: Specifies a file containing the secret to use with the map command. If not specified, `client.admin` will be used by default.

--keyring `filename`
: Specifies a keyring file containing a secret for the specified user to use with the map command. If not specified, the default keyring locations will be searched.

--shared `tag`
: Option for *lock add* that allows multiple clients to lock the same image if they use the same tag. The tag is an arbitrary string. This is useful for situations where an image must be open from more than one client at once, like during live migration of a virtual machine, or for use underneath a clustered filesystem.

--format `format`
: Specifies output formatting (default: plain, json, xml)

--pretty-format
: Make json or xml formatted output more human-readable.

-o `map-options`, **--options** `map-options`
: Specifies which options to use when mapping an image. map-options is a comma-separated string of options (similar to mount(8) mount options). See map options section below for more details.

--read-only
: Map the image read-only. Equivalent to -o ro.

--image-feature `feature`
: Specifies which RBD format 2 feature should be enabled when creating an image. Multiple features can be enabled by repeating this option multiple times. The following features are supported:

  layering: layering support striping: striping v2 support exclusive-lock: exclusive locking support object-map: object map support (requires exclusive-lock) fast-diff: fast diff calculations (requires object-map)

--image-shared
: Specifies that the image will be used concurrently by multiple clients. This will disable features that are dependent upon exclusive ownership of the image.

--object-extents
: Specifies that the diff should be limited to the extents of a full object instead of showing intra-object deltas. When the object map feature is enabled on an image, limiting the diff to the object extents will dramatically improve performance since the differences can be computed by examining the in-memory object map instead of querying RADOS for each object within the image.

6.9.5 Commands

ls [-l | –long] [pool-name] Will list all rbd images listed in the rbd_directory object. With -l, also show snapshots, and use longer-format output including size, parent (if clone), format, etc.

`info` [*image-name*] Will dump information (such as size and order) about a specific rbd image. If image is a clone, information about its parent is also displayed. If a snapshot is specified, whether it is protected is shown as well.

`create` [*image-name*] Will create a new rbd image. You must also specify the size via –size. The –stripe-unit and –stripe-count arguments are optional, but must be used together.

`clone` [*parent-snapname*] [*image-name*] Will create a clone (copy-on-write child) of the parent snapshot. Object order will be identical to that of the parent image unless specified. Size will be the same as the parent snapshot. The –stripe-unit and –stripe-count arguments are optional, but must be used together.

The parent snapshot must be protected (see *rbd snap protect*). This requires image format 2.

`flatten` [*image-name*] If image is a clone, copy all shared blocks from the parent snapshot and make the child independent of the parent, severing the link between parent snap and child. The parent snapshot can be unprotected and deleted if it has no further dependent clones.

This requires image format 2.

`children` [*image-name*] List the clones of the image at the given snapshot. This checks every pool, and outputs the resulting poolname/imagename.

This requires image format 2.

`resize` [*image-name*] [–allow-shrink] Resizes rbd image. The size parameter also needs to be specified. The –allow-shrink option lets the size be reduced.

`rm` [*image-name*] Deletes an rbd image (including all data blocks). If the image has snapshots, this fails and nothing is deleted.

`export` [*image-name*] [*dest-path*] Exports image to dest path (use - for stdout).

`import` [*path*] [*dest-image*] Creates a new image and imports its data from path (use - for stdin). The import operation will try to create sparse rbd images if possible. For import from stdin, the sparsification unit is the data block size of the destination image (1 << order).

The –stripe-unit and –stripe-count arguments are optional, but must be used together.

`export-diff` [*image-name*] [*dest-path*] [–from-snap *snapname*] [–object-extents] Exports an incremental diff for an image to dest path (use - for stdout). If an initial snapshot is specified, only changes since that snapshot are included; otherwise, any regions of the image that contain data are included. The end snapshot is specified using the standard –snap option or @snap syntax (see below). The image diff format includes metadata about image size changes, and the start and end snapshots. It efficiently represents discarded or 'zero' regions of the image.

`merge-diff` [*first-diff-path*] [*second-diff-path*] [*merged-diff-path*] Merge two continuous incremental diffs of an image into one single diff. The first diff's end snapshot must be equal with the second diff's start snapshot. The first diff could be - for stdin, and merged diff could be - for stdout, which enables multiple diff files to be merged using something like 'rbd merge-diff first second - | rbd merge-diff - third result'. Note this command currently only support the source incremental diff with stripe_count == 1

`import-diff` [*src-path*] [*image-name*] Imports an incremental diff of an image and applies it to the current image. If the diff was generated relative to a start snapshot, we verify that snapshot already exists before continuing. If there was an end snapshot we verify it does not already exist before applying the changes, and create the snapshot when we are done.

`diff` [*image-name*] [–from-snap *snapname*] [–object-extents] Dump a list of byte extents in the image that have changed since the specified start snapshot, or since the image was created. Each output line includes the starting offset (in bytes), the length of the region (in bytes), and either 'zero' or 'data' to indicate whether the region is known to be zeros or may contain other data.

`cp` [*src-image*] [*dest-image*] Copies the content of a src-image into the newly created dest-image. dest-image will have the same size, order, and image format as src-image.

`mv` [*src-image*] [*dest-image*] Renames an image. Note: rename across pools is not supported.

`image-meta list` [*image-name*] Show metadata held on the image. The first column is the key and the second column is the value.

`image-meta get` [*image-name*] [*key*] Get metadata value with the key.

`image-meta set` [*image-name*] [*key*] [*value*] Set metadata key with the value. They will displayed in *metadata-list*

`image-meta remove` [*image-name*] [*key*] Remove metadata key with the value.

`object-map` rebuild [*image-name*] Rebuilds an invalid object map for the specified image. An image snapshot can be specified to rebuild an invalid object map for a snapshot.

`snap` ls [*image-name*] Dumps the list of snapshots inside a specific image.

`snap` create [*image-name*] Creates a new snapshot. Requires the snapshot name parameter specified.

`snap` rollback [*image-name*] Rollback image content to snapshot. This will iterate through the entire blocks array and update the data head content to the snapshotted version.

`snap` rm [*image-name*] Removes the specified snapshot.

`snap` purge [*image-name*] Removes all snapshots from an image.

`snap` protect [*image-name*] Protect a snapshot from deletion, so that clones can be made of it (see *rbd clone*). Snapshots must be protected before clones are made; protection implies that there exist dependent cloned children that refer to this snapshot. *rbd clone* will fail on a nonprotected snapshot.

This requires image format 2.

`snap` unprotect [*image-name*] Unprotect a snapshot from deletion (undo *snap protect*). If cloned children remain, *snap unprotect* fails. (Note that clones may exist in different pools than the parent snapshot.)

This requires image format 2.

`map` [*image-name*] [-o | –options *map-options*] [–read-only] Maps the specified image to a block device via the rbd kernel module.

`unmap` [*image-name*] | [*device-path*] Unmaps the block device that was mapped via the rbd kernel module.

`showmapped` Show the rbd images that are mapped via the rbd kernel module.

`status` [*image-name*] Show the status of the image, including which clients have it open.

`feature` disable [*image-name*] [*feature*] Disables the specified feature on the specified image. Multiple features can be specified.

`feature` enable [*image-name*] [*feature*] Enables the specified feature on the specified image. Multiple features can be specified.

`lock` list [*image-name*] Show locks held on the image. The first column is the locker to use with the *lock remove* command.

`lock` add [*image-name*] [*lock-id*] Lock an image. The lock-id is an arbitrary name for the user's convenience. By default, this is an exclusive lock, meaning it will fail if the image is already locked. The –shared option changes this behavior. Note that locking does not affect any operation other than adding a lock. It does not protect an image from being deleted.

`lock` remove [*image-name*] [*lock-id*] [*locker*] Release a lock on an image. The lock id and locker are as output by lock ls.

`bench-write` [*image-name*] –io-size [*io-size-in-bytes*] –io-threads [*num-ios-in-flight*] –io-total [*total-bytes-to-write*] Generate a series of sequential writes to the image and measure the write throughput and latency. Defaults are: –io-size 4096, –io-threads 16, –io-total 1GB

6.9.6 Image name

In addition to using the –pool and the –snap options, the image name can include both the pool name and the snapshot name. The image name format is as follows:

```
[pool/]image-name[@snap]
```

Thus an image name that contains a slash character ('/') requires specifying the pool name explicitly.

6.9.7 Striping

RBD images are striped over many objects, which are then stored by the Ceph distributed object store (RADOS). As a result, read and write requests for the image are distributed across many nodes in the cluster, generally preventing any single node from becoming a bottleneck when individual images get large or busy.

The striping is controlled by three parameters:

`order`
: The size of objects we stripe over is a power of two, specifically 2^[*order*] bytes. The default is 22, or 4 MB.

`stripe_unit`
: Each [*stripe_unit*] contiguous bytes are stored adjacently in the same object, before we move on to the next object.

`stripe_count`
: After we write [*stripe_unit*] bytes to [*stripe_count*] objects, we loop back to the initial object and write another stripe, until the object reaches its maximum size (as specified by [*order*]. At that point, we move on to the next [*stripe_count*] objects.

By default, [*stripe_unit*] is the same as the object size and [*stripe_count*] is 1. Specifying a different [*stripe_unit*] requires that the STRIPINGV2 feature be supported (added in Ceph v0.53) and format 2 images be used.

6.9.8 Map options

Most of these options are useful mainly for debugging and benchmarking. The default values are set in the kernel and may therefore depend on the version of the running kernel.

- fsid=aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee - FSID that should be assumed by the client.
- ip=a.b.c.d[:p] - IP and, optionally, port the client should use.
- share - Enable sharing of client instances with other mappings (default).
- noshare - Disable sharing of client instances with other mappings.
- crc - Enable CRC32C checksumming for data writes (default).
- nocrc - Disable CRC32C checksumming for data writes.
- cephx_require_signatures - Require cephx message signing, i.e. MSG_AUTH feature bit (since 3.19, default).
- nocephx_require_signatures - Don't require cephx message signing (since 3.19).
- tcp_nodelay - Disable Nagle's algorithm on client sockets (since 4.0, default).
- notcp_nodelay - Enable Nagle's algorithm on client sockets (since 4.0).
- osdkeepalive=x - OSD keepalive timeout (default is 5 seconds).
- osd_idle_ttl=x - OSD idle TTL (default is 60 seconds).
- rw - Map the image read-write (default).

- ro - Map the image read-only. Equivalent to –read-only.

6.9.9 Examples

To create a new rbd image that is 100 GB:

```
rbd -p mypool create myimage --size 102400
```

or alternatively:

```
rbd create mypool/myimage --size 102400
```

To use a non-default object size (8 MB):

```
rbd create mypool/myimage --size 102400 --order 23
```

To delete an rbd image (be careful!):

```
rbd rm mypool/myimage
```

To create a new snapshot:

```
rbd snap create mypool/myimage@mysnap
```

To create a copy-on-write clone of a protected snapshot:

```
rbd clone mypool/myimage@mysnap otherpool/cloneimage
```

To see which clones of a snapshot exist:

```
rbd children mypool/myimage@mysnap
```

To delete a snapshot:

```
rbd snap rm mypool/myimage@mysnap
```

To map an image via the kernel with cephx enabled:

```
rbd map mypool/myimage --id admin --keyfile secretfile
```

To unmap an image:

```
rbd unmap /dev/rbd0
```

To create an image and a clone from it:

```
rbd import --image-format 2 image mypool/parent
rbd snap create --snap snapname mypool/parent
rbd snap protect mypool/parent@snap
rbd clone mypool/parent@snap otherpool/child
```

To create an image with a smaller stripe_unit (to better distribute small writes in some workloads):

```
rbd -p mypool create myimage --size 102400 --stripe-unit 65536 --stripe-count 16
```

To change an image from one image format to another, export it and then import it as the desired image format:

```
rbd export mypool/myimage@snap /tmp/img
rbd import --image-format 2 /tmp/img mypool/myimage2
```

To lock an image for exclusive use:

```
rbd lock add mypool/myimage mylockid
```

To release a lock:

```
rbd lock remove mypool/myimage mylockid client.2485
```

6.9.10 Availability

rbd is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

6.9.11 See also

ceph(8), rados(8)

6.10 rbd-fuse – expose rbd images as files

6.10.1 Synopsis

rbd-fuse [-p pool] [-c conffile] *mountpoint* [*fuse options*]

6.10.2 Description

rbd-fuse is a FUSE (File system in USErspace) client for RADOS block device (rbd) images. Given a pool containing rbd images, it will mount a userspace filesystem allowing access to those images as regular files at **mountpoint**.

The file system can be unmounted with:

```
fusermount -u mountpoint
```

or by sending `SIGINT` to the `rbd-fuse` process.

6.10.3 Options

Any options not recognized by rbd-fuse will be passed on to libfuse.

-c `ceph.conf`
Use *ceph.conf* configuration file instead of the default `/etc/ceph/ceph.conf` to determine monitor addresses during startup.

-p `pool`
Use *pool* as the pool to search for rbd images. Default is `rbd`.

6.10.4 Availability

rbd-fuse is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

6.10.5 See also

fusermount(8), rbd(8)

6.11 ceph-rbdnamer – udev helper to name RBD devices

6.11.1 Synopsis

ceph-rbdnamer *num*

6.11.2 Description

ceph-rbdnamer prints the pool and image name for the given RBD devices to stdout. It is used by *udev* (using a rule like the one below) to set up a device symlink.

```
KERNEL=="rbd[0-9]*", PROGRAM="/usr/bin/ceph-rbdnamer %n", SYMLINK+="rbd/%c{1}/%c{2}"
```

6.11.3 Availability

ceph-rbdnamer is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

6.11.4 See also

rbd(8), ceph(8)

6.12 RBD Replay

RBD Replay is a set of tools for capturing and replaying Rados Block Device (RBD) workloads. To capture an RBD workload, `lttng-tools` must be installed on the client, and `librbd` on the client must be the v0.87 (Giant) release or later. To replay an RBD workload, `librbd` on the client must be the Giant release or later.

Capture and replay takes three steps:

1. Capture the trace. Make sure to capture `pthread_id` context:

```
mkdir -p traces
lttng create -o traces librbd
lttng enable-event -u 'librbd:*'
lttng add-context -u -t pthread_id
lttng start
# run RBD workload here
lttng stop
```

2. Process the trace with rbd-replay-prep:

```
rbd-replay-prep traces/ust/uid/*/* replay.bin
```

3. Replay the trace with rbd-replay. Use read-only until you know it's doing what you want:

```
    rbd-replay --read-only replay.bin
```

Important: `rbd-replay` will destroy data by default. Do not use against an image you wish to keep, unless you use the `--read-only` option.

The replayed workload does not have to be against the same RBD image or even the same cluster as the captured workload. To account for differences, you may need to use the `--pool` and `--map-image` options of `rbd-replay`.

6.13 rbd-replay-prep – prepare captured rados block device (RBD) workloads for replay

6.13.1 Synopsis

rbd-replay-prep [–window *seconds*] [–anonymize] *trace_dir replay_file*

6.13.2 Description

rbd-replay-prep processes raw rados block device (RBD) traces to prepare them for **rbd-replay**.

6.13.3 Options

`--window` `seconds`
Requests further apart than 'seconds' seconds are assumed to be independent.

`--anonymize`
Anonymizes image and snap names.

6.13.4 Examples

To prepare workload1-trace for replay:

```
rbd-replay-prep workload1-trace/ust/uid/1000/64-bit workload1
```

6.13.5 Availability

rbd-replay-prep is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

6.13.6 See also

rbd-replay(8), rbd(8)

6.14 rbd-replay – replay rados block device (RBD) workloads

6.14.1 Synopsis

rbd-replay [*options*] *replay_file*

6.14.2 Description

rbd-replay is a utility for replaying rados block device (RBD) workloads.

6.14.3 Options

-c `ceph.conf`, **--conf** `ceph.conf`
: Use ceph.conf configuration file instead of the default /etc/ceph/ceph.conf to determine monitor addresses during startup.

-p `pool`, **--pool** `pool`
: Interact with the given pool. Defaults to 'rbd'.

--latency-multiplier
: Multiplies inter-request latencies. Default: 1.

--read-only
: Only replay non-destructive requests.

--map-image `rule`
: Add a rule to map image names in the trace to image names in the replay cluster. A rule of image1@snap1=image2@snap2 would map snap1 of image1 to snap2 of image2.

--dump-perf-counters
: **Experimental** Dump performance counters to standard out before an image is closed. Performance counters may be dumped multiple times if multiple images are closed, or if the same image is opened and closed multiple times. Performance counters and their meaning may change between versions.

6.14.4 Examples

To replay workload1 as fast as possible:

```
rbd-replay --latency-multiplier=0 workload1
```

To replay workload1 but use test_image instead of prod_image:

```
rbd-replay --map-image=prod_image=test_image workload1
```

6.14.5 Availability

rbd-replay is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

6.14.6 See also

rbd-replay-prep(8), rbd(8)

6.15 rbd-replay-many – replay a rados block device (RBD) workload on several clients

6.15.1 Synopsis

rbd-replay-many [*options*] –original-image *name host1* [*host2* [...]] – *rbd_replay_args*

6.15.2 Description

rbd-replay-many is a utility for replaying a rados block device (RBD) workload on several clients. Although all clients use the same workload, they replay against separate images. This matches normal use of librbd, where each original client is a VM with its own image.

Configuration and replay files are not automatically copied to clients. Replay images must already exist.

6.15.3 Options

`--original-image` `name`
Specifies the name (and snap) of the originally traced image. Necessary for correct name mapping.

`--image-prefix` `prefix`
Prefix of image names to replay against. Specifying –image-prefix=foo results in clients replaying against foo-0, foo-1, etc. Defaults to the original image name.

`--exec` `program`
Path to the rbd-replay executable.

`--delay` `seconds`
Delay between starting each client. Defaults to 0.

6.15.4 Examples

Typical usage:

```
rbd-replay-many host-0 host-1 --original-image=image -- -c ceph.conf replay.bin
```

This results in the following commands being executed:

```
ssh host-0 'rbd-replay' --map-image 'image=image-0' -c ceph.conf replay.bin
ssh host-1 'rbd-replay' --map-image 'image=image-1' -c ceph.conf replay.bin
```

6.15.5 Availability

rbd-replay-many is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

6.15.6 See also

rbd-replay(8), rbd(8)

6.16 Librbd (Python)

The *rbd* python module provides file-like access to RBD images.

6.16.1 Example: Creating and writing to an image

To use *rbd*, you must first connect to RADOS and open an IO context:

```
cluster = rados.Rados(conffile='my_ceph.conf')
cluster.connect()
ioctx = cluster.open_ioctx('mypool')
```

Then you instantiate an :class:rbd.RBD object, which you use to create the image:

```
rbd_inst = rbd.RBD()
size = 4 * 1024**3  # 4 GiB
rbd_inst.create(ioctx, 'myimage', size)
```

To perform I/O on the image, you instantiate an :class:rbd.Image object:

```
image = rbd.Image(ioctx, 'myimage')
data = 'foo' * 200
image.write(data, 0)
```

This writes 'foo' to the first 600 bytes of the image. Note that data cannot be :type:unicode - *Librbd* does not know how to deal with characters wider than a :c:type:char.

In the end, you'll want to close the image, the IO context and the connection to RADOS:

```
image.close()
ioctx.close()
cluster.shutdown()
```

To be safe, each of these calls would need to be in a separate :finally block:

```
cluster = rados.Rados(conffile='my_ceph_conf')
try:
    ioctx = cluster.open_ioctx('my_pool')
    try:
        rbd_inst = rbd.RBD()
        size = 4 * 1024**3  # 4 GiB
        rbd_inst.create(ioctx, 'myimage', size)
        image = rbd.Image(ioctx, 'myimage')
        try:
            data = 'foo' * 200
            image.write(data, 0)
        finally:
            image.close()
    finally:
        ioctx.close()
finally:
    cluster.shutdown()
```

This can be cumbersome, so the `Rados`, `Ioctx`, and `Image` classes can be used as context managers that close/shutdown automatically (see **PEP 343**). Using them as context managers, the above example becomes:

```
with rados.Rados(conffile='my_ceph.conf') as cluster:
    with cluster.open_ioctx('mypool') as ioctx:
        rbd_inst = rbd.RBD()
        size = 4 * 1024**3  # 4 GiB
        rbd_inst.create(ioctx, 'myimage', size)
        with rbd.Image(ioctx, 'myimage') as image:
            data = 'foo' * 200
            image.write(data, 0)
```

6.16.2 API Reference

This module is a thin wrapper around librbd.

It currently provides all the synchronous methods of librbd that do not use callbacks.

Error codes from librbd are turned into exceptions that subclass `Error`. Almost all methods may raise `Error` (the base class of all rbd exceptions), `PermissionError` and `IOError`, in addition to those documented for the method.

A number of methods have string arguments, which must not be unicode to interact correctly with librbd. If unicode is passed to these methods, a `TypeError` will be raised.

class `rbd.`**`RBD`**

This class wraps librbd CRUD functions.

`clone`(*p_ioctx*, *p_name*, *p_snapname*, *c_ioctx*, *c_name*, *features=0*, *order=None*)

Clone a parent rbd snapshot into a COW sparse child.

Parameters

- **`p_ioctx`** – the parent context that represents the parent snap
- **`p_name`** – the parent image name
- **`p_snapname`** – the parent image snapshot name
- **`c_ioctx`** – the child context that represents the new clone
- **`c_name`** – the clone (child) name
- **`features`** (*int*) – bitmask of features to enable; if set, must include layering
- **`order`** (*int*) – the image is split into (2**order) byte objects

Raises `TypeError`

Raises `InvalidArgument`

Raises `ImageExists`

Raises `FunctionNotSupported`

Raises `ArgumentOutOfRange`

`create`(*ioctx*, *name*, *size*, *order=None*, *old_format=True*, *features=0*, *stripe_unit=0*, *stripe_count=0*)

Create an rbd image.

Parameters

- **`ioctx`** (`rados.Ioctx`) – the context in which to create the image
- **`name`** (*str*) – what the image is called
- **`size`** (*int*) – how big the image is in bytes
- **`order`** (*int*) – the image is split into (2**order) byte objects

- **old_format** (*bool*) – whether to create an old-style image that is accessible by old clients, but can't use more advanced features like layering.
- **features** (*int*) – bitmask of features to enable
- **stripe_unit** (*int*) – stripe unit in bytes (default 0 for object size)
- **stripe_count** (*int*) – objects to stripe over before looping

Raises `ImageExists`

Raises `TypeError`

Raises `InvalidArgument`

Raises `FunctionNotSupported`

list(*ioctx*)
List image names.

Parameters **ioctx** (`rados.Ioctx`) – determines which RADOS pool is read

Returns list – a list of image names

remove(*ioctx*, *name*)
Delete an RBD image. This may take a long time, since it does not return until every object that comprises the image has been deleted. Note that all snapshots must be deleted before the image can be removed. If there are snapshots left, `ImageHasSnapshots` is raised. If the image is still open, or the watch from a crashed client has not expired, `ImageBusy` is raised.

Parameters

- **ioctx** (`rados.Ioctx`) – determines which RADOS pool the image is in
- **name** (*str*) – the name of the image to remove

Raises `ImageNotFound`, `ImageBusy`, `ImageHasSnapshots`

rename(*ioctx*, *src*, *dest*)
Rename an RBD image.

Parameters

- **ioctx** (`rados.Ioctx`) – determines which RADOS pool the image is in
- **src** (*str*) – the current name of the image
- **dest** (*str*) – the new name of the image

Raises `ImageNotFound`, `ImageExists`

version()
Get the version number of the `librbd` C library.

Returns a tuple of `(major, minor, extra)` components of the librbd version

class `rbd.`**Image**(*ioctx*, *name*, *snapshot=None*, *read_only=False*)
This class represents an RBD image. It is used to perform I/O on the image and interact with snapshots.

Note: Any method of this class may raise `ImageNotFound` if the image has been deleted.

break_lock(*client*, *cookie*)
Release a lock held by another rados client.

close()
Release the resources used by this image object.

After this is called, this object should not be used.

copy(*dest_ioctx*, *dest_name*)
Copy the image to another location.

Parameters

- **dest_ioctx** (`rados.Ioctx`) – determines which pool to copy into
- **dest_name** (*str*) – the name of the copy

Raises `ImageExists`

create_snap(*name*)
Create a snapshot of the image.

Parameters **name** (*str*) – the name of the snapshot

Raises `ImageExists`

diff_iterate(*offset*, *length*, *from_snapshot*, *iterate_cb*, *include_parent=True*, *whole_object=False*)
Iterate over the changed extents of an image.

This will call iterate_cb with three arguments:

(offset, length, exists)

where the changed extent starts at offset bytes, continues for length bytes, and is full of data (if exists is True) or zeroes (if exists is False).

If from_snapshot is None, it is interpreted as the beginning of time and this generates all allocated extents.

The end version is whatever is currently selected (via set_snap) for the image.

Raises `InvalidArgument` if from_snapshot is after the currently set snapshot.

Raises `ImageNotFound` if from_snapshot is not the name of a snapshot of the image.

Parameters

- **offset** (*int*) – start offset in bytes
- **length** (*int*) – size of region to report on, in bytes
- **from_snapshot** (*str or None*) – starting snapshot name, or None
- **iterate_cb** (*function acception arguments for offset, length, and exists*) – function to call for each extent
- **include_parent** (*bool*) – True if full history diff should include parent
- **whole_object** (*bool*) – True if diff extents should cover whole object

Raises `InvalidArgument`, `IOError`, `ImageNotFound`

discard(*offset*, *length*)
Trim the range from the image. It will be logically filled with zeroes.

features()
Gets the features bitmask of the image.

Returns int - the features bitmask of the image

flags()
Gets the flags bitmask of the image.

Returns int - the flags bitmask of the image

flatten()
Flatten clone image (copy all blocks from parent to child)

`flush``()`
Block until all writes are fully flushed if caching is enabled.

`invalidate_cache``()`
Drop any cached data for the image.

`is_exclusive_lock_owner``()`
Gets the status of the image exclusive lock.

Returns bool - true if the image is exclusively locked

`is_protected_snap``(`*name*`)`
Find out whether a snapshot is protected from deletion.

Parameters **`name`** (*str*) – the snapshot to check

Returns bool - whether the snapshot is protected

Raises `IOError`, `ImageNotFound`

`list_children``()`
List children of the currently set snapshot (set via set_snap()).

Returns list - a list of (pool name, image name) tuples

`list_lockers``()`
List clients that have locked the image and information about the lock.

Returns

dict - contains the following keys:

- `tag` - the tag associated with the lock (every additional locker must use the same tag)
- **`exclusive` - boolean indicating whether the** lock is exclusive or shared
- `lockers` - a list of (client, cookie, address) tuples

`list_snaps``()`
Iterate over the snapshots of an image.

Returns *`SnapIterator`*

`lock_exclusive``(`*cookie*`)`
Take an exclusive lock on the image.

Raises `ImageBusy` if a different client or cookie locked it `ImageExists` if the same client and cookie locked it

`lock_shared``(`*cookie*, *tag*`)`
Take a shared lock on the image. The tag must match that of the existing lockers, if any.

Raises `ImageBusy` if a different client or cookie locked it `ImageExists` if the same client and cookie locked it

`old_format``()`
Find out whether the image uses the old RBD format.

Returns bool - whether the image uses the old RBD format

`overlap``()`
Gets the number of overlapping bytes between the image and its parent image. If open to a snapshot, returns the overlap between the snapshot and the parent image.

Returns int - the overlap in bytes

Raises `ImageNotFound` if the image doesn't have a parent

`parent_info`()
Get information about a cloned image's parent (if any)

Returns tuple - `(pool name, image name, snapshot name)` components of the parent image

Raises `ImageNotFound` if the image doesn't have a parent

`protect_snap`(*name*)
Mark a snapshot as protected. This means it can't be deleted until it is unprotected.

Parameters **`name`** (*str*) – the snapshot to protect

Raises `IOError`, `ImageNotFound`

`read`(*offset*, *length*, *fadvise_flags=0*)
Read data from the image. Raises `InvalidArgument` if part of the range specified is outside the image.

Parameters

- **`offset`** (*int*) – the offset to start reading at
- **`length`** (*int*) – how many bytes to read
- **`fadvise_flags`** (*int*) – fadvise flags for this read

Returns str - the data read

Raises `InvalidArgument`, `IOError`

`remove_snap`(*name*)
Delete a snapshot of the image.

Parameters **`name`** (*str*) – the name of the snapshot

Raises `IOError`, `ImageBusy`

`resize`(*size*)
Change the size of the image.

Parameters **`size`** (*int*) – the new size of the image

`rollback_to_snap`(*name*)
Revert the image to its contents at a snapshot. This is a potentially expensive operation, since it rolls back each object individually.

Parameters **`name`** (*str*) – the snapshot to rollback to

Raises `IOError`

`set_snap`(*name*)
Set the snapshot to read from. Writes will raise ReadOnlyImage while a snapshot is set. Pass None to unset the snapshot (reads come from the current image) , and allow writing again.

Parameters **`name`** (*str or None*) – the snapshot to read from, or None to unset the snapshot

`size`()
Get the size of the image. If open to a snapshot, returns the size of that snapshot.

Returns the size of the image in bytes

`stat`()
Get information about the image. Currently parent pool and parent name are always -1 and ''.

Returns

dict - contains the following keys:

- `size` (int) - the size of the image in bytes
- `obj_size` (int) - the size of each object that comprises the image
- `num_objs` (int) - the number of objects in the image
- `order` (int) - log_2(object_size)
- `block_name_prefix` (str) - the prefix of the RADOS objects used to store the image
- `parent_pool` (int) - deprecated
- `parent_name` (str) - deprecated

See also `format()` and *`features()`*.

`stripe_count`()
Returns the stripe count used for the image.

`stripe_unit`()
Returns the stripe unit used for the image.

`unlock`(*cookie*)
Release a lock on the image that was locked by this rados client.

`unprotect_snap`(*name*)
Mark a snapshot unprotected. This allows it to be deleted if it was protected.

Parameters **`name`** (*str*) – the snapshot to unprotect

Raises `IOError`, `ImageNotFound`

`update_features`(*features*, *enabled*)
Updates the features bitmask of the image by enabling/disabling a single feature. The feature must support the ability to be dynamically enabled/disabled.

Parameters

- **`features`** (*int*) – feature bitmask to enable/disable
- **`enabled`** (*bool*) – whether to enable/disable the feature

Raises `InvalidArgument`

`write`(*data*, *offset*, *fadvise_flags=0*)
Write data to the image. Raises `InvalidArgument` if part of the write would fall outside the image.

Parameters

- **`data`** (*str*) – the data to be written
- **`offset`** (*int*) – where to start writing data
- **`fadvise_flags`** (*int*) – fadvise flags for this write

Returns int - the number of bytes written

Raises `IncompleteWriteError`, `LogicError`, `InvalidArgument`, `IOError`

class `rbd.`**`SnapIterator`**(*image*)
Iterator over snapshot info for an image.

Yields a dictionary containing information about a snapshot.

Keys are:

- `id` (int) - numeric identifier of the snapshot
- `size` (int) - size of the image at the time of snapshot (in bytes)

- `name` (str) - name of the snapshot

CHAPTER

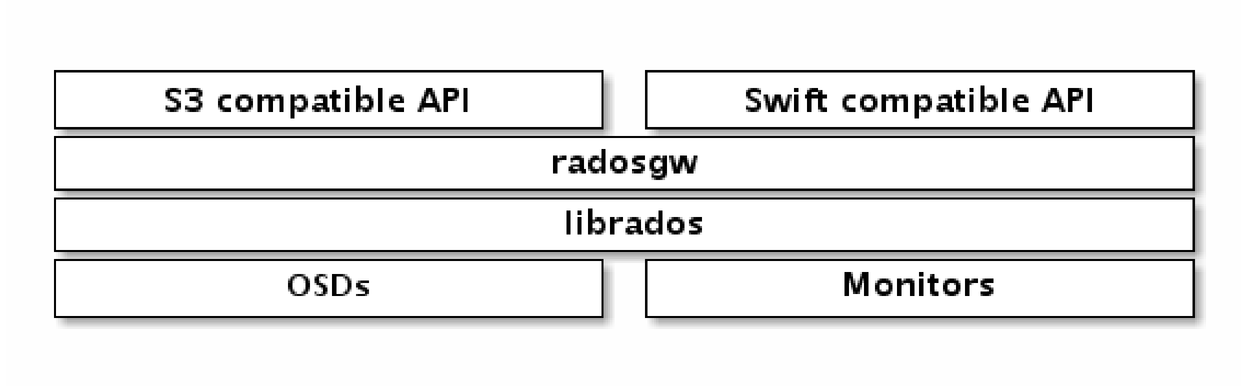
SEVEN

CEPH OBJECT GATEWAY

Ceph Object Gateway is an object storage interface built on top of `librados` to provide applications with a RESTful gateway to Ceph Storage Clusters. *Ceph Object Storage* supports two interfaces:

1. **S3-compatible:** Provides object storage functionality with an interface that is compatible with a large subset of the Amazon S3 RESTful API.
2. **Swift-compatible:** Provides object storage functionality with an interface that is compatible with a large subset of the OpenStack Swift API.

Ceph Object Storage uses the Ceph Object Gateway daemon (`radosgw`), which is a FastCGI module for interacting with a Ceph Storage Cluster. Since it provides interfaces compatible with OpenStack Swift and Amazon S3, the Ceph Object Gateway has its own user management. Ceph Object Gateway can store data in the same Ceph Storage Cluster used to store data from Ceph Filesystem clients or Ceph Block Device clients. The S3 and Swift APIs share a common namespace, so you may write data with one API and retrieve it with the other.



Note: Ceph Object Storage does **NOT** use the Ceph Metadata Server.

7.1 Install Ceph Object Gateway

Note: To run the Ceph object gateway service, you should have a running Ceph cluster, the gateway host should have access to storage and public networks, and SELinux should be in permissive mode in rpm-based distros.

The *Ceph Object Gateway* daemon runs on Apache and FastCGI.

To run a *Ceph Object Storage* service, you must install Apache and Ceph Object Gateway daemon on the host that is going to provide the gateway service, i.e, the `gateway host`. If you plan to run a Ceph Object Storage service with a federated architecture (multiple regions and zones), you must also install the synchronization agent.

Note: Previous versions of Ceph shipped with `mod_fastcgi`. The current version ships with `mod_proxy_fcgi` instead.

In distros that ship Apache 2.4 (such as RHEL 7, CentOS 7 or Ubuntu 14.04 `Trusty`), `mod_proxy_fcgi` is already present. When you install the `httpd` package with `yum` or the `apache2` package with `apt-get`, `mod_proxy_fcgi` becomes available for use on your server.

In distros that ship Apache 2.2 (such as RHEL 6, CentOS 6 or Ubuntu 12.04 `Precise`), `mod_proxy_fcgi` comes as a separate package. In **RHEL 6/CentOS 6**, it is available in `EPEL 6` repo and can be installed with `yum install mod_proxy_fcgi`. For **Ubuntu 12.04**, a backport for `mod_proxy_fcgi` is in progress and a bug has been filed for the same. See: ceph radosgw needs mod-proxy-fcgi for apache 2.2

7.1.1 Install Apache

To install Apache on the `gateway host`, execute the following:

On Debian-based distros, run:

```
sudo apt-get install apache2
```

On RPM-based distros, run:

```
sudo yum install httpd
```

7.1.2 Configure Apache

Make the following changes in Apache's configuration on the `gateway host`:

Debian-based distros

1. Add a line for the `ServerName` in `/etc/apache2/apache2.conf`. Provide the fully qualified domain name of the server machine (e.g., `hostname -f`):

```
ServerName {fqdn}
```

2. Load `mod_proxy_fcgi` module.

   Execute:

```
sudo a2enmod proxy_fcgi
```

3. Start Apache service:

```
sudo service apache2 start
```

RPM-based distros

1. Open the `httpd.conf` file:

```
sudo vim /etc/httpd/conf/httpd.conf
```

2. Uncomment `#ServerName` in the file and add the name of your server. Provide the fully qualified domain name of the server machine (e.g., `hostname -f`):

```
ServerName {fqdn}
```

3. Update /etc/httpd/conf/httpd.conf to load mod_proxy_fcgi module. Add the following to the file:

```
<IfModule !proxy_fcgi_module>
LoadModule proxy_fcgi_module modules/mod_proxy_fcgi.so
</IfModule>
```

4. Edit the line `Listen 80` in /etc/httpd/conf/httpd.conf with the public IP address of the host that you are configuring as a gateway server. Write `Listen {IP ADDRESS}:80` in place of `Listen 80`.

5. Start httpd service

   Execute:

```
sudo service httpd start
```

   Or:

```
sudo systemctl start httpd
```

7.1.3 Enable SSL

Some REST clients use HTTPS by default. So you should consider enabling SSL for Apache. Use the following procedures to enable SSL.

Note: You can use self-certified certificates. Some client APIs check for a trusted certificate authority. You may need to obtain a SSL certificate from a trusted authority to use those client APIs.

Debian-based distros

To enable SSL on Debian-based distros, execute the following steps:

1. Ensure that you have installed the dependencies:

```
sudo apt-get install openssl ssl-cert
```

2. Enable the SSL module:

```
sudo a2enmod ssl
```

3. Generate a certificate:

```
sudo mkdir /etc/apache2/ssl
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/apache2/ssl/apache.key -ou
```

4. Restart Apache:

```
sudo service apache2 restart
```

See the Ubuntu Server Guide for additional details.

RPM-based distros

To enable SSL on RPM-based distros, execute the following steps:

1. Ensure that you have installed the dependencies:

```
sudo yum install mod_ssl openssl
```

2. Generate private key:

```
openssl genrsa -out ca.key 2048
```

3. Generate CSR:

```
openssl req -new -key ca.key -out ca.csr
```

4. Generate a certificate:

```
openssl x509 -req -days 365 -in ca.csr -signkey ca.key -out ca.crt
```

5. Copy the files to appropriate locations:

```
sudo cp ca.crt /etc/pki/tls/certs
sudo cp ca.key /etc/pki/tls/private/ca.key
sudo cp ca.csr /etc/pki/tls/private/ca.csr
```

6. Update the Apache SSL configuration file `/etc/httpd/conf.d/ssl.conf`.

   Give the correct location of `SSLCertificateFile`:

```
SSLCertificateFile /etc/pki/tls/certs/ca.crt
```

   Give the correct location of `SSLCertificateKeyFile`:

```
SSLCertificateKeyFile /etc/pki/tls/private/ca.key
```

   Save the changes.

7. Restart Apache.

   Execute:

```
sudo service httpd restart
```

   Or:

```
sudo systemctl restart httpd
```

See Setting up an SSL secured Webserver with CentOS for additional details.

7.1.4 Install Ceph Object Gateway Daemon

Ceph Object Storage services use the Ceph Object Gateway daemon (`radosgw`) to enable the gateway. For federated architectures, the synchronization agent (`radosgw-agent`) provides data and metadata synchronization between zones and regions.

Debian-based distros

To install the Ceph Object Gateway daemon on the *gateway host*, execute the following:

```
sudo apt-get install radosgw
```

To install the Ceph Object Gateway synchronization agent, execute the following:

```
sudo apt-get install radosgw-agent
```

RPM-based distros

To install the Ceph Object Gateway daemon on the `gateway host`, execute the following:

```
sudo yum install ceph-radosgw
```

To install the Ceph Object Gateway synchronization agent, execute the following:

```
sudo yum install radosgw-agent
```

7.1.5 Configure The Gateway

Once you have installed the Ceph Object Gateway packages, the next step is to configure your Ceph Object Gateway. There are two approaches:

- **Simple:** A simple Ceph Object Gateway configuration implies that you are running a Ceph Object Storage service in a single data center. So you can configure the Ceph Object Gateway without regard to regions and zones.
- **Federated:** A federated Ceph Object Gateway configuration implies that you are running a Ceph Object Storage service in a geographically distributed manner for fault tolerance and failover. This involves configuring your Ceph Object Gateway instances with regions and zones.

Choose the approach that best reflects your cluster.

7.2 Configuring Ceph Object Gateway

Configuring a Ceph Object Gateway requires a running Ceph Storage Cluster, and an Apache web server with the FastCGI module.

The Ceph Object Gateway is a client of the Ceph Storage Cluster. As a Ceph Storage Cluster client, it requires:

- A name for the gateway instance. We use `gateway` in this guide.
- A storage cluster user name with appropriate permissions in a keyring.
- Pools to store its data.
- A data directory for the gateway instance.
- An instance entry in the Ceph Configuration file.
- A configuration file for the web server to interact with FastCGI.

7.2.1 Create a User and Keyring

Each instance must have a user name and key to communicate with a Ceph Storage Cluster. In the following steps, we use an admin node to create a keyring. Then, we create a client user name and key. Next, we add the key to the Ceph Storage Cluster. Finally, we distribute the key ring to the node containing the gateway instance.

Monitor Key CAPS

When you provide CAPS to the key, you MUST provide read capability. However, you have the option of providing write capability for the monitor. This is an important choice. If you provide write capability to the key, the Ceph Object Gateway will have the ability to create pools automatically; however, it will create pools with either the default number of placement groups (not ideal) or the number of placement groups you specified in your Ceph configuration file. If you allow the Ceph Object Gateway to create pools automatically, ensure that you have reasonable defaults for the number of placement groups first. See Pool Configuration for details.

See User Management for additional details on Ceph authentication.

1. Create a keyring for the gateway:

```
sudo ceph-authtool --create-keyring /etc/ceph/ceph.client.radosgw.keyring
sudo chmod +r /etc/ceph/ceph.client.radosgw.keyring
```

2. Generate a Ceph Object Gateway user name and key for each instance. For exemplary purposes, we will use the name `gateway` after `client.radosgw`:

```
sudo ceph-authtool /etc/ceph/ceph.client.radosgw.keyring -n client.radosgw.gateway --gen-key
```

3. Add capabilities to the key. See Configuration Reference - Pools for details on the effect of write permissions for the monitor and creating pools.

```
sudo ceph-authtool -n client.radosgw.gateway --cap osd 'allow rwx' --cap mon 'allow rwx' /etc/ce
```

4. Once you have created a keyring and key to enable the Ceph Object Gateway with access to the Ceph Storage Cluster, add the key to your Ceph Storage Cluster. For example:

```
sudo ceph -k /etc/ceph/ceph.client.admin.keyring auth add client.radosgw.gateway -i /etc/ceph/ce
```

5. Distribute the keyring to the node with the gateway instance.

```
sudo scp /etc/ceph/ceph.client.radosgw.keyring  ceph@{hostname}:/home/ceph
ssh {hostname}
sudo mv ceph.client.radosgw.keyring /etc/ceph/ceph.client.radosgw.keyring
```

Note: The 5th step is optional if `admin node` is the `gateway host`.

7.2.2 Create Pools

Ceph Object Gateways require Ceph Storage Cluster pools to store specific gateway data. If the user you created has permissions, the gateway will create the pools automatically. However, you should ensure that you have set an appropriate default number of placement groups per pool into your Ceph configuration file.

Note: Ceph Object Gateways have multiple pools, so don't make the number of PGs too high considering all of the pools assigned to the same CRUSH hierarchy, or performance may suffer.

When configuring a gateway with the default region and zone, the naming convention for pools typically omits region and zone naming, but you can use any naming convention you prefer. For example:

- `.rgw.root`
- `.rgw.control`
- `.rgw.gc`

- .rgw.buckets
- .rgw.buckets.index
- .rgw.buckets.extra
- .log
- .intent-log
- .usage
- .users
- .users.email
- .users.swift
- .users.uid

See Configuration Reference - Pools for details on the default pools for gateways. See Pools for details on creating pools. As already said, if write permission is given, Ceph Object Gateway will create pools automatically. To create a pool manually, execute the following:

```
ceph osd pool create {poolname} {pg-num} {pgp-num} {replicated | erasure} [{erasure-code-profile}]
```

Tip: Ceph supports multiple CRUSH hierarchies and CRUSH rulesets, enabling great flexibility in the way you configure your gateway. Pools such as `rgw.buckets.index` may benefit from a pool of SSDs for fast performance. Backing storage may benefit from the increased economy of erasure-coded storage, and/or the improved performance from cache tiering.

When you have completed this step, execute the following to ensure that you have created all of the foregoing pools:

```
rados lspools
```

7.2.3 Add a Gateway Configuration to Ceph

Add the Ceph Object Gateway configuration to your Ceph Configuration file in `admin node`. The Ceph Object Gateway configuration requires you to identify the Ceph Object Gateway instance. Then, you must specify the host name where you installed the Ceph Object Gateway daemon, a keyring (for use with cephx), the socket path for FastCGI and a log file.

For distros with Apache 2.2 and early versions of Apache 2.4 (RHEL 6, Ubuntu 12.04, 14.04 etc), append the following configuration to `/etc/ceph/ceph.conf` in your `admin node`:

```
[client.radosgw.gateway]
host = {hostname}
keyring = /etc/ceph/ceph.client.radosgw.keyring
rgw socket path = ""
log file = /var/log/radosgw/client.radosgw.gateway.log
rgw frontends = fastcgi socket_port=9000 socket_host=0.0.0.0
rgw print continue = false
```

Note: Apache 2.2 and early versions of Apache 2.4 do not use Unix Domain Sockets but use localhost TCP.

For distros with Apache 2.4.9 or later (RHEL 7, CentOS 7 etc), append the following configuration to `/etc/ceph/ceph.conf` in your `admin node`:

```
[client.radosgw.gateway]
host = {hostname}
keyring = /etc/ceph/ceph.client.radosgw.keyring
rgw socket path = /var/run/ceph/ceph.radosgw.gateway.fastcgi.sock
log file = /var/log/radosgw/client.radosgw.gateway.log
rgw print continue = false
```

Note: `Apache 2.4.9` supports Unix Domain Socket (UDS) but as `Ubuntu 14.04` ships with `Apache 2.4.7` it doesn't have UDS support and has to be configured for use with localhost TCP. A bug has been filed for backporting UDS support in `Apache 2.4.7` for `Ubuntu 14.04`. See: Backport support for UDS in Ubuntu Trusty

Here, `{hostname}` is the short hostname (output of command `hostname -s`) of the node that is going to provide the gateway service i.e, the `gateway host`.

The `[client.radosgw.gateway]` portion of the gateway instance identifies this portion of the Ceph configuration file as configuring a Ceph Storage Cluster client where the client type is a Ceph Object Gateway (i.e., `radosgw`).

Note: The last line in the configuration i.e, `rgw print continue = false` is added to avoid issues with `PUT` operations.

Once you finish the setup procedure, if you encounter issues with your configuration, you can add debugging to the `[global]` section of your Ceph configuration file and restart the gateway to help troubleshoot any configuration issues. For example:

```
[global]
#append the following in the global section.
debug ms = 1
debug rgw = 20
```

7.2.4 Distribute updated Ceph configuration file

The updated Ceph configuration file needs to be distributed to all Ceph cluster nodes from the `admin node`.

It involves the following steps:

1. Pull the updated `ceph.conf` from `/etc/ceph/` to the root directory of the cluster in admin node (e.g. `my-cluster` directory). The contents of `ceph.conf` in `my-cluster` will get overwritten. To do so, execute the following:

```
ceph-deploy --overwrite-conf config pull {hostname}
```

   Here, `{hostname}` is the short hostname of the Ceph admin node.

2. Push the updated `ceph.conf` file from the admin node to all other nodes in the cluster including the `gateway host`:

```
ceph-deploy --overwrite-conf config push [HOST] [HOST...]
```

   Give the hostnames of the other Ceph nodes in place of `[HOST] [HOST...]`.

7.2.5 Copy ceph.client.admin.keyring from admin node to gateway host

As the `gateway host` can be a different node that is not part of the cluster, the `ceph.client.admin.keyring` needs to be copied from the `admin node` to the `gateway host`. To do so, execute the following on `admin node`:

```
sudo scp /etc/ceph/ceph.client.admin.keyring  ceph@{hostname}:/home/ceph
ssh {hostname}
sudo mv ceph.client.admin.keyring /etc/ceph/ceph.client.admin.keyring
```

Note: The above step need not be executed if `admin node` is the `gateway host`.

7.2.6 Create a CGI wrapper script

The wrapper script provides the interface between the webserver and the radosgw process. This script needs to be in a web accessible location and should be executable.

Execute the following steps on the `gateway host`:

1. Create the script:

```
sudo vi /var/www/html/s3gw.fcgi
```

2. Add the following content to the script:

```
#!/bin/sh
exec /usr/bin/radosgw -c /etc/ceph/ceph.conf -n client.radosgw.gateway
```

3. Provide execute permissions to the script:

```
sudo chmod +x /var/www/html/s3gw.fcgi
```

7.2.7 Adjust CGI wrapper script permission

On some distros, `apache` should have execute permission on the `s3gw.fcgi` script. To change permission on the file, execute:

```
sudo chown apache:apache /var/www/html/s3gw.fcgi
```

7.2.8 Create Data Directory

Deployment scripts may not create the default Ceph Object Gateway data directory. Create data directories for each instance of a `radosgw` daemon (if you haven't done so already). The `host` variables in the Ceph configuration file determine which host runs each instance of a `radosgw` daemon. The typical form specifies the `radosgw` daemon, the cluster name and the daemon ID.

To create the directory on the `gateway host`, execute the following:

```
sudo mkdir -p /var/lib/ceph/radosgw/ceph-radosgw.gateway
```

7.2.9 Adjust Socket Directory Permissions

On some distros, the `radosgw` daemon runs as the unprivileged `apache` UID, and this UID must have write access to the location where it will write its socket file.

To grant permissions to the default socket location, execute the following on the `gateway host`:

```
sudo chown apache:apache /var/run/ceph
```

7.2.10 Change Log File Owner

On some distros, the `radosgw` daemon runs as the unprivileged `apache` UID, but the `root` user owns the log file by default. You must change it to the `apache` user so that Apache can populate the log file. To do so, execute the following:

```
sudo chown apache:apache /var/log/radosgw/client.radosgw.gateway.log
```

7.2.11 Start radosgw service

The Ceph Object gateway daemon needs to be started. To do so, execute the following on the `gateway host`:

On Debian-based distros:

```
sudo /etc/init.d/radosgw start
```

On RPM-based distros:

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

7.2.12 Create a Gateway Configuration file

On the host where you installed the Ceph Object Gateway i.e, `gateway host`, create an `rgw.conf` file. Place the file in `/etc/apache2/conf-available` directory for `Debian-based` distros and in `/etc/httpd/conf.d` directory for `RPM-based` distros. It is a Apache configuration file which is needed for the `radosgw` service. This file must be readable by the web server.

Execute the following steps:

1. Create the file:

   For Debian-based distros, execute:

   ```
   sudo vi /etc/apache2/conf-available/rgw.conf
   ```

   For RPM-based distros, execute:

   ```
   sudo vi /etc/httpd/conf.d/rgw.conf
   ```

2. For distros with Apache 2.2 and early versions of Apache 2.4 that use localhost TCP and do not support Unix Domain Socket, add the following contents to the file:

   ```
   <VirtualHost *:80>
   ServerName localhost
   DocumentRoot /var/www/html

   ErrorLog /var/log/httpd/rgw_error.log
   CustomLog /var/log/httpd/rgw_access.log combined

   # LogLevel debug

   RewriteEngine On

   RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization},L]

   SetEnv proxy-nokeepalive 1

   ProxyPass / fcgi://localhost:9000/
   ```

```
    </VirtualHost>
```

Note: For Debian-based distros replace `/var/log/httpd/` with `/var/log/apache2`.

3. For distros with Apache 2.4.9 or later that support Unix Domain Socket, add the following contents to the file:

```
    <VirtualHost *:80>
    ServerName localhost
    DocumentRoot /var/www/html

    ErrorLog /var/log/httpd/rgw_error.log
    CustomLog /var/log/httpd/rgw_access.log combined

    # LogLevel debug

    RewriteEngine On

    RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization},L]

    SetEnv proxy-nokeepalive 1

    ProxyPass / unix:///var/run/ceph/ceph.radosgw.gateway.fastcgi.sock|fcgi://localhost:9000/

    </VirtualHost>
```

7.2.13 Restart Apache

The Apache service needs to be restarted to accept the new configuration.

For Debian-based distros, run:

```
sudo service apache2 restart
```

For RPM-based distros, run:

```
sudo service httpd restart
```

Or:

```
sudo systemctl restart httpd
```

7.2.14 Using The Gateway

To use the REST interfaces, first create an initial Ceph Object Gateway user for the S3 interface. Then, create a subuser for the Swift interface. See the Admin Guide for more details on user management.

Create a radosgw user for S3 access

A `radosgw` user needs to be created and granted access. The command `man radosgw-admin` will provide information on additional command options.

To create the user, execute the following on the `gateway host`:

```
sudo radosgw-admin user create --uid="testuser" --display-name="First User"
```

The output of the command will be something like the following:

```
{"user_id": "testuser",
"display_name": "First User",
"email": "",
"suspended": 0,
"max_buckets": 1000,
"auid": 0,
"subusers": [],
"keys": [
{ "user": "testuser",
"access_key": "I0PJDPCIYZ665MW88W9R",
"secret_key": "dxaXZ8U90SXydYzyS5ivamEP20hkLSUViiaR+ZDA"}],
"swift_keys": [],
"caps": [],
"op_mask": "read, write, delete",
"default_placement": "",
"placement_tags": [],
"bucket_quota": { "enabled": false,
"max_size_kb": -1,
"max_objects": -1},
"user_quota": { "enabled": false,
"max_size_kb": -1,
"max_objects": -1},
"temp_url_keys": []}
```

Note: The values of `keys->access_key` and `keys->secret_key` are needed for access validation.

Create a Swift user

A Swift subuser needs to be created if this kind of access is needed. Creating a Swift user is a two step process. The first step is to create the user. The second is to create the secret key.

Execute the following steps on the `gateway host`:

Create the Swift user:

```
sudo radosgw-admin subuser create --uid=testuser --subuser=testuser:swift
```

The output will be something like the following:

```
--access=full
{ "user_id": "testuser",
"display_name": "First User",
"email": "",
"suspended": 0,
"max_buckets": 1000,
"auid": 0,
"subusers": [
{ "id": "testuser:swift",
"permissions": "full-control"}],
"keys": [
{ "user": "testuser:swift",
"access_key": "3Y1LNW4Q6X0Y53A52DET",
"secret_key": ""},
{ "user": "testuser",
"access_key": "I0PJDPCIYZ665MW88W9R",
"secret_key": "dxaXZ8U90SXydYzyS5ivamEP20hkLSUViiaR+ZDA"}],
"swift_keys": [],
"caps": [],
"op_mask": "read, write, delete",
"default_placement": "",
"placement_tags": [],
"bucket_quota": { "enabled": false,
"max_size_kb": -1,
"max_objects": -1},
"user_quota": { "enabled": false,
"max_size_kb": -1,
"max_objects": -1},
"temp_url_keys": []}
```

Create the secret key:

```
sudo radosgw-admin key create --subuser=testuser:swift --key-type=swift --gen-secret
```

The output will be something like the following:

```
{ "user_id": "testuser",
"display_name": "First User",
"email": "",
"suspended": 0,
"max_buckets": 1000,
"auid": 0,
"subusers": [
{ "id": "testuser:swift",
"permissions": "full-control"}],
"keys": [
{ "user": "testuser:swift",
"access_key": "3Y1LNW4Q6X0Y53A52DET",
"secret_key": ""},
{ "user": "testuser",
"access_key": "I0PJDPCIYZ665MW88W9R",
"secret_key": "dxaXZ8U90SXydYzyS5ivamEP20hkLSUViiaR+ZDA"}],
"swift_keys": [
{ "user": "testuser:swift",
"secret_key": "244+fz2gSqoHwR3lYtSbIyomyPHf3i7rgSJrF\/IA"}],
"caps": [],
"op_mask": "read, write, delete",
"default_placement": "",
"placement_tags": [],
"bucket_quota": { "enabled": false,
"max_size_kb": -1,
"max_objects": -1},
"user_quota": { "enabled": false,
"max_size_kb": -1,
"max_objects": -1},
"temp_url_keys": []}
```

7.2.15 Access Verification

You then need to verify if the created users are able to access the gateway.

Test S3 access

You need to write and run a Python test script for verifying S3 access. The S3 access test script will connect to the `radosgw`, create a new bucket and list all buckets. The values for `aws_access_key_id` and `aws_secret_access_key` are taken from the values of `access_key` and `secret_key` returned by the `radosgw_admin` command.

Execute the following steps:

1. You will need to install the `python-boto` package.

   For Debian-based distros, run:

```
sudo apt-get install python-boto
```

   For RPM-based distros, run:

```
sudo yum install python-boto
```

2. Create the Python script:

```
vi s3test.py
```

3. Add the following contents to the file:

```
import boto
import boto.s3.connection
access_key = 'I0PJDPCIYZ665MW88W9R'
secret_key = 'dxaXZ8U90SXydYzyS5ivamEP20hkLSUViiaR+ZDA'
conn = boto.connect_s3(
aws_access_key_id = access_key,
aws_secret_access_key = secret_key,
host = '{hostname}',
is_secure=False,
calling_format = boto.s3.connection.OrdinaryCallingFormat(),
)
bucket = conn.create_bucket('my-new-bucket')
for bucket in conn.get_all_buckets():
        print "{name}\t{created}".format(
                name = bucket.name,
                created = bucket.creation_date,
)
```

   Replace `{hostname}` with the hostname of the host where you have configured the gateway service i.e, the `gateway host`.

4. Run the script:

```
python s3test.py
```

   The output will be something like the following:

```
my-new-bucket 2015-02-16T17:09:10.000Z
```

Test swift access

Swift access can be verified via the `swift` command line client. The command `man swift` will provide more information on available command line options.

To install `swift` client, execute the following:

For Debian-based distros:

```
sudo apt-get install python-setuptools
sudo easy_install pip
sudo pip install --upgrade setuptools
sudo pip install --upgrade python-swiftclient
```

For RPM-based distros:

```
sudo yum install python-setuptools
sudo easy_install pip
sudo pip install --upgrade setuptools
sudo pip install --upgrade python-swiftclient
```

To test swift access, execute the following:

```
swift -A http://{IP ADDRESS}/auth/1.0 -U testuser:swift -K '{swift_secret_key}' list
```

Replace `{IP ADDRESS}` with the public IP address of the gateway server and `{swift_secret_key}` with its value from the output of `radosgw-admin key create` command executed for the `swift` user.

For example:

```
swift -A http://10.19.143.116/auth/1.0 -U testuser:swift -K '244+fz2gSqoHwR3lYtSbIyomyPHf3i7rgSJrF/IA
```

The output should be:

```
my-new-bucket
```

7.3 Configuring Federated Gateways

New in version 0.67: Dumpling

In Ceph version 0.67 Dumpling and beyond, you may configure each *Ceph Object Gateway* to participate in a federated architecture, with multiple regions, and with multiple zones for a region.

- **Region**: A region represents a *logical* geographic area and contains one or more zones. A cluster with multiple regions must specify a master region.
- **Zone**: A zone is a *logical* grouping of one or more Ceph Object Gateway instance(s). A region has a master zone that processes client requests.

Important: Only write objects to the master zone in a region. You may read objects from secondary zones. Currently, the Gateway does not prevent you from writing to a secondary zone, but **DON'T DO IT**.

7.3.1 Background

When you deploy a *Ceph Object Store* service that spans geographical locales, configuring Ceph Object Gateway regions and metadata synchronization agents enables the service to maintain a global namespace, even though Ceph Object Gateway instances run in different geographic locales and potentially on different Ceph Storage Clusters. When you separate one or more Ceph Object Gateway instances within a region into separate logical containers to maintain an extra copy (or copies) of the data, configuring Ceph Object Gateway zones and data synchronization agents enables the service to maintain one or more copy(ies) of the master zone's data. Extra copies of the data are important for failover, backup and disaster recovery.

You may deploy a single Ceph Storage Cluster with a federated architecture if you have low latency network connections (this isn't recommended). You may also deploy one Ceph Storage Cluster per region with a separate set of pools for each zone (typical). You may also deploy a separate Ceph Storage Cluster for each zone if your requirements and resources warrant this level of redundancy.

7.3.2 About this Guide

In the following sections, we will demonstrate how to configure a federated cluster in two logical steps:

- **Configure a Master Region:** This section of the guide describes how to set up a region with multiple zones, and how to synchronize data between the master zone and the secondary zone(s) within the master region.
- **Configure a Secondary Region:** This section of the guide describes how to repeat the section on setting up a master region and multiple zones so that you have two regions with intra-zone synchronization in each region. Finally, you will learn how to set up a metadata synchronization agent so that you can maintain a global namespace for the regions in your cluster.

7.3.3 Configure a Master Region

This section provides an exemplary procedure for setting up a region, and two zones within the region. The cluster will comprise two gateway daemon instances–one per zone. This region will serve as the master region.

Naming for the Master Region

Before configuring the cluster, defining region, zone and instance names will help you manage your cluster. Let's assume the region represents the United States, and we refer to it by its standard abbreviation.

- United States: `us`

Let's assume the zones represent the Eastern and Western United States. For continuity, our naming convention will use `{region name}-{zone name}` format, but you can use any naming convention you prefer.

- United States, East Region: `us-east`
- United States, West Region: `us-west`

Finally, let's assume that zones may have more than one Ceph Object Gateway instance per zone. For continuity, our naming convention will use `{region name}-{zone name}-{instance}` format, but you can use any naming convention you prefer.

- United States Region, Master Zone, Instance 1: `us-east-1`
- United States Region, Secondary Zone, Instance 1: `us-west-1`

Create Pools

You may have a Ceph Storage Cluster for the entire region or a Ceph Storage Cluster for each zone.

For continuity, our naming convention will use `{region name}-{zone name}` format prepended to the pool name, but you can use any naming convention you prefer. For example:

- `.us-east.rgw.root`
- `.us-east.rgw.control`
- `.us-east.rgw.gc`
- `.us-east.rgw.buckets`
- `.us-east.rgw.buckets.index`

- `.us-east.rgw.buckets.extra`
- `.us-east.log`
- `.us-east.intent-log`
- `.us-east.usage`
- `.us-east.users`
- `.us-east.users.email`
- `.us-east.users.swift`
- `.us-east.users.uid`

- `.us-west.rgw.root`
- `.us-west.rgw.control`
- `.us-west.rgw.gc`
- `.us-west.rgw.buckets`
- `.us-west.rgw.buckets.index`
- `.us-west.rgw.buckets.extra`
- `.us-west.log`
- `.us-west.intent-log`
- `.us-west.usage`
- `.us-west.users`
- `.us-west.users.email`
- `.us-west.users.swift`
- `.us-west.users.uid`

See Configuration Reference - Pools for details on the default pools for gateways. See Pools for details on creating pools. Execute the following to create a pool:

```
ceph osd pool create {poolname} {pg-num} {pgp-num} {replicated | erasure} [{erasure-code-profile}]
```

Tip: When adding a large number of pools, it may take some time for your cluster to return to a `active + clean` state.

CRUSH Maps

When deploying a Ceph Storage Cluster for the entire region, consider using a CRUSH rule for the zone such that you do NOT have overlapping failure domains. See CRUSH Map for details.
Ceph supports multiple CRUSH hierarchies and CRUSH rulesets, enabling great flexibility in the way you configure your gateway. Pools such as `rgw.buckets.index` may benefit from a modestly sized pool of SSDs for fast performance. Backing storage may benefit from the increased economy of erasure-coded storage, and/or the improved performance from cache tiering.

When you have completed this step, execute the following to ensure that you have created all of the foregoing pools:

```
rados lspools
```

Create a Keyring

Each instance must have a user name and key to communicate with a Ceph Storage Cluster. In the following steps, we use an admin node to create a keyring. Then, we create a client user name and key for each instance. Next, we add the keys to the Ceph Storage Cluster(s). Finally, we distribute the key ring to each node containing an instance.

1. Create a keyring.

```
sudo ceph-authtool --create-keyring /etc/ceph/ceph.client.radosgw.keyring
sudo chmod +r /etc/ceph/ceph.client.radosgw.keyring
```

2. Generate a Ceph Object Gateway user name and key for each instance.

```
sudo ceph-authtool /etc/ceph/ceph.client.radosgw.keyring -n client.radosgw.us-east-1 --gen-key
sudo ceph-authtool /etc/ceph/ceph.client.radosgw.keyring -n client.radosgw.us-west-1 --gen-key
```

3. Add capabilities to each key. See Configuration Reference - Pools for details on the effect of write permissions for the monitor and creating pools.

```
sudo ceph-authtool -n client.radosgw.us-east-1 --cap osd 'allow rwx' --cap mon 'allow rwx' /etc/
sudo ceph-authtool -n client.radosgw.us-west-1 --cap osd 'allow rwx' --cap mon 'allow rwx' /etc/
```

4. Once you have created a keyring and key to enable the Ceph Object Gateway with access to the Ceph Storage Cluster, add each key as an entry to your Ceph Storage Cluster(s). For example:

```
sudo ceph -k /etc/ceph/ceph.client.admin.keyring auth add client.radosgw.us-east-1 -i /etc/ceph/
sudo ceph -k /etc/ceph/ceph.client.admin.keyring auth add client.radosgw.us-west-1 -i /etc/ceph/
```

Note: When you use this procedure to configure the secondary region, replace `us-` with `eu-`. You will have a total of four users **after** you create the master region and the secondary region.

Install Apache/FastCGI

For each *Ceph Node* that runs a *Ceph Object Gateway* daemon instance, you must install Apache, FastCGI, the Ceph Object Gateway daemon (`radosgw`) and the Ceph Object Gateway Sync Agent (`radosgw-agent`). See Install Ceph Object Gateway for details.

Create Data Directories

Create data directories for each daemon instance on their respective hosts.

```
ssh {us-east-1}
sudo mkdir -p /var/lib/ceph/radosgw/ceph-radosgw.us-east-1

ssh {us-west-1}
sudo mkdir -p /var/lib/ceph/radosgw/ceph-radosgw.us-west-1
```

Note: When you use this procedure to configure the secondary region, replace `us-` with `eu-`. You will have a total of four data directories **after** you create the master region and the secondary region.

Create a Gateway Configuration

For each instance, create an Ceph Object Gateway configuration file under the `/etc/apache2/sites-available` directory on the host(s) where you installed the Ceph Object Gateway daemon(s). See below for an exemplary embodiment of a gateway configuration as discussed in the following text.

```
FastCgiExternalServer /var/www/s3gw.fcgi -socket /{path}/{socket-name}.sock


<VirtualHost *:80>

        ServerName {fqdn}
        <!--Remove the comment. Add a server alias with *.{fqdn} for S3 subdomains-->
        <!--ServerAlias *.{fqdn}-->
        ServerAdmin {email.address}
        DocumentRoot /var/www
        RewriteEngine On
        RewriteRule  ^/(.*) /s3gw.fcgi?%{QUERY_STRING} [E=HTTP_AUTHORIZATION:%{HTTP:Authorization},L]

        <IfModule mod_fastcgi.c>
           <Directory /var/www>
                        Options +ExecCGI
                        AllowOverride All
                        SetHandler fastcgi-script
                        Order allow,deny
                        Allow from all
                        AuthBasicAuthoritative Off
                </Directory>
        </IfModule>

        AllowEncodedSlashes On
        ErrorLog /var/log/apache2/error.log
        CustomLog /var/log/apache2/access.log combined
        ServerSignature Off

</VirtualHost>
```

1. Replace the `/{path}/{socket-name}` entry with path to the socket and the socket name. For example, `/var/run/ceph/client.radosgw.us-east-1.sock`. Ensure that you use the same path and socket name in your `ceph.conf` entry.
2. Replace the `{fqdn}` entry with the fully-qualified domain name of the server.
3. Replace the `{email.address}` entry with the email address for the server administrator.
4. Add a `ServerAlias` if you wish to use S3-style subdomains (of course you do).
5. Save the configuration to a file (e.g., `rgw-us-east.conf`).

Repeat the process for the secondary zone (e.g., `rgw-us-west.conf`).

Note: When you use this procedure to configure the secondary region, replace `us-` with `eu-`. You will have a total of four gateway configuration files on the respective nodes **after** you create the master region and the secondary region.

Finally, if you enabled SSL, make sure that you set the port to your SSL port (usually 443) and your configuration file includes the following:

```
SSLEngine on
SSLCertificateFile /etc/apache2/ssl/apache.crt
SSLCertificateKeyFile /etc/apache2/ssl/apache.key
SetEnv SERVER_PORT_SECURE 443
```

Enable the Configuration

For each instance, enable the gateway configuration and disable the default site.

1. Enable the site for the gateway configuration.

```
sudo a2ensite {rgw-conf-filename}
```

2. Disable the default site.

```
sudo a2dissite default
```

Note: Failure to disable the default site can lead to problems.

Add a FastCGI Script

FastCGI requires a script for each Ceph Object Gateway instance to enable the S3-compatible interface. To create the script, execute the following procedures.

1. Go to the `/var/www` directory.

```
cd /var/www
```

2. Open an editor with the file name `s3gw.fcgi`. **Note:** The configuration file specifies this filename.

```
sudo vim s3gw.fcgi
```

3. Add a shell script with `exec` and the path to the gateway binary, the path to the Ceph configuration file, and the user name (`-n`; the same user name created in step 2 of *Create a Keyring*. Copy the following into the editor.

```
#!/bin/sh
exec /usr/bin/radosgw -c /etc/ceph/ceph.conf -n client.radosgw.{ID}
```

For example:

```
#!/bin/sh
exec /usr/bin/radosgw -c /etc/ceph/ceph.conf -n client.radosgw.us-east-1
```

4. Save the file.

5. Change the permissions on the file so that it is executable.

```
sudo chmod +x s3gw.fcgi
```

Repeat the process for the secondary zone.

Note: When you use this procedure to configure the secondary region, replace `us-` with `eu-`. You will have a total of four FastCGI scripts **after** you create the master region and the secondary region.

Add Instances to Ceph Config File

On an admin node, add an entry for each instance in the Ceph configuration file for your Ceph Storage Cluster(s). For example:

```
...

[client.radosgw.us-east-1]
rgw region = us
rgw region root pool = .us.rgw.root
rgw zone = us-east
rgw zone root pool = .us-east.rgw.root
keyring = /etc/ceph/ceph.client.radosgw.keyring
rgw dns name = {hostname}
rgw socket path = /var/run/ceph/$name.sock
host = {host-name}

[client.radosgw.us-west-1]
rgw region = us
rgw region root pool = .us.rgw.root
rgw zone = us-west
rgw zone root pool = .us-west.rgw.root
keyring = /etc/ceph/ceph.client.radosgw.keyring
rgw dns name = {hostname}
rgw socket path = /var/run/ceph/$name.sock
host = {host-name}
```

Then, update each *Ceph Node* with the updated Ceph configuration file. For example:

```
ceph-deploy --overwrite-conf config push {node1} {node2} {nodex}
```

Note: When you use this procedure to configure the secondary region, replace `us` with `eu` for the name, region, pools and zones. You will have a total of four entries **after** you create the master region and the secondary region.

Create a Region

1. Configure a region infile called `us.json` for the `us` region.

   Copy the contents of the following example to a text editor. Set `is_master` to `true`. Replace `{fqdn}` with the fully-qualified domain name of the endpoint. It will specify a master zone as `us-east` and list it in the `zones` list along with the `us-west` zone. See Configuration Reference - Regions for details.:

```
     { "name": "us",
       "api_name": "us",
       "is_master": "true",
       "endpoints": [
             "http:\/\/{fqdn}:80\/"],
       "master_zone": "us-east",
       "zones": [
             { "name": "us-east",
               "endpoints": [
                     "http:\/\/{fqdn}:80\/"],
               "log_meta": "true",
               "log_data": "true"},
             { "name": "us-west",
               "endpoints": [
                     "http:\/\/{fqdn}:80\/"],
               "log_meta": "true",
               "log_data": "true"}],
       "placement_targets": [
        {
          "name": "default-placement",
          "tags": []
        }
       ],
       "default_placement": "default-placement"}
```

2. Create the `us` region using the `us.json` infile you just created.

```
     radosgw-admin region set --infile us.json --name client.radosgw.us-east-1
```

3. Delete the default region (if it exists).

```
     rados -p .us.rgw.root rm region_info.default
```

4. Set the `us` region as the default region.

```
     radosgw-admin region default --rgw-region=us --name client.radosgw.us-east-1
```

   Only one region can be the default region for a cluster.

5. Update the region map.

```
     radosgw-admin regionmap update --name client.radosgw.us-east-1
```

If you use different Ceph Storage Cluster instances for regions, you should repeat steps 2, 4 and 5 in by executing them with `--name client.radosgw-us-west-1`. You may also export the region map from the initial gateway instance and import it followed by updating the region map.

Note: When you use this procedure to configure the secondary region, replace `us` with `eu`. You will have a total of two regions **after** you create the master region and the secondary region.

Create Zones

1. Configure a zone infile called `us-east.json` for the `us-east` zone.

   Copy the contents of the following example to a text editor. This configuration uses pool names prepended with the region name and zone name. See Configuration Reference - Pools for additional details on gateway pools. See Configuration Reference - Zones for additional details on zones.

```
     { "domain_root": ".us-east.domain.rgw",
       "control_pool": ".us-east.rgw.control",
       "gc_pool": ".us-east.rgw.gc",
       "log_pool": ".us-east.log",
       "intent_log_pool": ".us-east.intent-log",
       "usage_log_pool": ".us-east.usage",
       "user_keys_pool": ".us-east.users",
       "user_email_pool": ".us-east.users.email",
       "user_swift_pool": ".us-east.users.swift",
       "user_uid_pool": ".us-east.users.uid",
       "system_key": { "access_key": "", "secret_key": ""},
       "placement_pools": [
         { "key": "default-placement",
           "val": { "index_pool": ".us-east.rgw.buckets.index",
                    "data_pool": ".us-east.rgw.buckets"}
```

```
        }
      ]
    }
```

2. Add the `us-east` zone using the `us-east.json` infile you just created in both the east and west pools by specifying their respective user names (i.e., `--name`).

```
radosgw-admin zone set --rgw-zone=us-east --infile us-east.json --name client.radosgw.us-east-1
radosgw-admin zone set --rgw-zone=us-east --infile us-east.json --name client.radosgw.us-west-1
```

   Repeat step 1 to create a zone infile for `us-west`. Then add the zone using the `us-west.json` infile in both the east and west pools by specifying their respective user names (i.e., `--name`).

```
radosgw-admin zone set --rgw-zone=us-west --infile us-west.json --name client.radosgw.us-east-1
radosgw-admin zone set --rgw-zone=us-west --infile us-west.json --name client.radosgw.us-west-1
```

3. Delete the default zone (if it exists).

```
rados -p .rgw.root rm zone_info.default
```

4. Update the region map.

```
radosgw-admin regionmap update --name client.radosgw.us-east-1
```

Note: When you use this procedure to configure the secondary region, replace `us-` with `eu-`. You will have a total of four zones **after** you create the master zone and the secondary zone in each region.

Create Zone Users

Ceph Object Gateway stores zone users in the zone pools. So you must create zone users after configuring the zones. Copy the `access_key` and `secret_key` fields for each user so you can update your zone configuration once you complete this step.

```
radosgw-admin user create --uid="us-east" --display-name="Region-US Zone-East" --name client.radosgw
radosgw-admin user create --uid="us-west" --display-name="Region-US Zone-West" --name client.radosgw
```

Note: When you use this procedure to configure the secondary region, replace `us-` with `eu-`. You will have a total of four zone users **after** you create the master region and the secondary region and their zones. These users are different from the users created in *Create a Keyring*.

Update Zone Configurations

You must update the zone configuration with zone users so that the synchronization agents can authenticate with the zones.

1. Open your `us-east.json` zone configuration file and paste the contents of the `access_key` and `secret_key` fields from the step of creating zone users into the `system_key` field of your zone configuration infile.

```
{ "domain_root": ".us-east.domain.rgw",
  "control_pool": ".us-east.rgw.control",
  "gc_pool": ".us-east.rgw.gc",
  "log_pool": ".us-east.log",
  "intent_log_pool": ".us-east.intent-log",
  "usage_log_pool": ".us-east.usage",
  "user_keys_pool": ".us-east.users",
  "user_email_pool": ".us-east.users.email",
  "user_swift_pool": ".us-east.users.swift",
  "user_uid_pool": ".us-east.users.uid",
  "system_key": {
    "access_key": "{paste-access_key-here}",
    "secret_key": "{paste-secret_key-here}"
         },
  "placement_pools": [
    { "key": "default-placement",
      "val": { "index_pool": ".us-east.rgw.buckets.index",
               "data_pool": ".us-east.rgw.buckets"}
    }
  ]
}
```

2. Save the `us-east.json` file. Then, update your zone configuration.

```
radosgw-admin zone set --rgw-zone=us-east --infile us-east.json --name client.radosgw.us-east-1
radosgw-admin zone set --rgw-zone=us-east --infile us-east.json --name client.radosgw.us-west-1
```

3. Repeat step 1 to update the zone infile for `us-west`. Then, update your zone configuration.

```
radosgw-admin zone set --rgw-zone=us-west --infile us-west.json --name client.radosgw.us-east-1
radosgw-admin zone set --rgw-zone=us-west --infile us-west.json --name client.radosgw.us-west-1
```

Note: When you use this procedure to configure the secondary region, replace `us-` with `eu-`. You will have a total of four zones **after** you create the master zone and the secondary zone in each region.

Restart Services

Once you have redeployed your Ceph configuration files, we recommend restarting your Ceph Storage Cluster(s) and Apache instances.

For Ubuntu, use the following on each *Ceph Node*:

```
sudo restart ceph-all
```

For Red Hat/CentOS, use the following:

```
sudo /etc/init.d/ceph restart
```

To ensure that all components have reloaded their configurations, for each gateway instance we recommend restarting the `apache2` service. For example:

```
sudo service apache2 restart
```

Start Gateway Instances

Start up the `radosgw` service.

```
sudo /etc/init.d/radosgw start
```

If you are running multiple instances on the same host, you must specify the user name.

```
sudo /etc/init.d/radosgw start --name client.radosgw.us-east-1
```

Open a browser and check the endpoints for each zone. A simple HTTP request to the domain name should return the following:

```
<ListAllMyBucketsResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
        <Owner>
                <ID>anonymous</ID>
                <DisplayName/>
        </Owner>
        <Buckets/>
</ListAllMyBucketsResult>
```

7.3.4 Configure a Secondary Region

This section provides an exemplary procedure for setting up a cluster with multiple regions. Configuring a cluster that spans regions requires maintaining a global namespace, so that there are no namespace clashes among object names stored across in different regions.

This section extends the procedure in *Configure a Master Region*, but changes the region name and modifies a few procedures. See the following sections for details.

Naming for the Secondary Region

Before configuring the cluster, defining region, zone and instance names will help you manage your cluster. Let's assume the region represents the European Union, and we refer to it by its standard abbreviation.

- European Union: `eu`

Let's assume the zones represent the Eastern and Western European Union. For continuity, our naming convention will use `{region name}-{zone name}` format, but you can use any naming convention you prefer.

- European Union, East Region: `eu-east`
- European Union, West Region: `eu-west`

Finally, let's assume that zones may have more than one Ceph Object Gateway instance per zone. For continuity, our naming convention will use `{region name}-{zone name}-{instance}` format, but you can use any naming convention you prefer.

- European Union Region, Master Zone, Instance 1: `eu-east-1`
- European Union Region, Secondary Zone, Instance 1: `eu-west-1`

Configuring a Secondary Region

Repeat the exemplary procedure of *Configure a Master Region* with the following differences:

1. Use *Naming for the Secondary Region* in lieu of *Naming for the Master Region*.
2. *Create Pools* using `eu` instead of `us`.
3. *Create a Keyring* and the corresponding keys using `eu` instead of `us`. You may use the same keyring if you desire, but ensure that you create the keys on the Ceph Storage Cluster for that region (or region and zone).
4. *Install Apache/FastCGI*.
5. *Create Data Directories* using `eu` instead of `us`.

6. *Create a Gateway Configuration* using `eu` instead of `us` for the socket names.
7. *Enable the Configuration*.
8. *Add a FastCGI Script* using `eu` instead of `us` for the user names.
9. *Add Instances to Ceph Config File* using `eu` instead of `us` for the pool names.
10. *Create a Region* using `eu` instead of `us`. Set `is_master` to `false`. For consistency, create the master region in the secondary region too.

```
radosgw-admin region set --infile us.json --name client.radosgw.eu-east-1
```

11. *Create Zones* using `eu` instead of `us`. Ensure that you update the user name (i.e., `--name`) so that you create the zones in the correct cluster.
12. *Update Zone Configurations* using `eu` instead of `us`.
13. Create zones from master region in the secondary region.

```
radosgw-admin zone set --rgw-zone=us-east --infile us-east.json --name client.radosgw.eu-east-1
radosgw-admin zone set --rgw-zone=us-east --infile us-east.json --name client.radosgw.eu-west-1
radosgw-admin zone set --rgw-zone=us-west --infile us-west.json --name client.radosgw.eu-east-1
radosgw-admin zone set --rgw-zone=us-west --infile us-west.json --name client.radosgw.eu-west-1
```

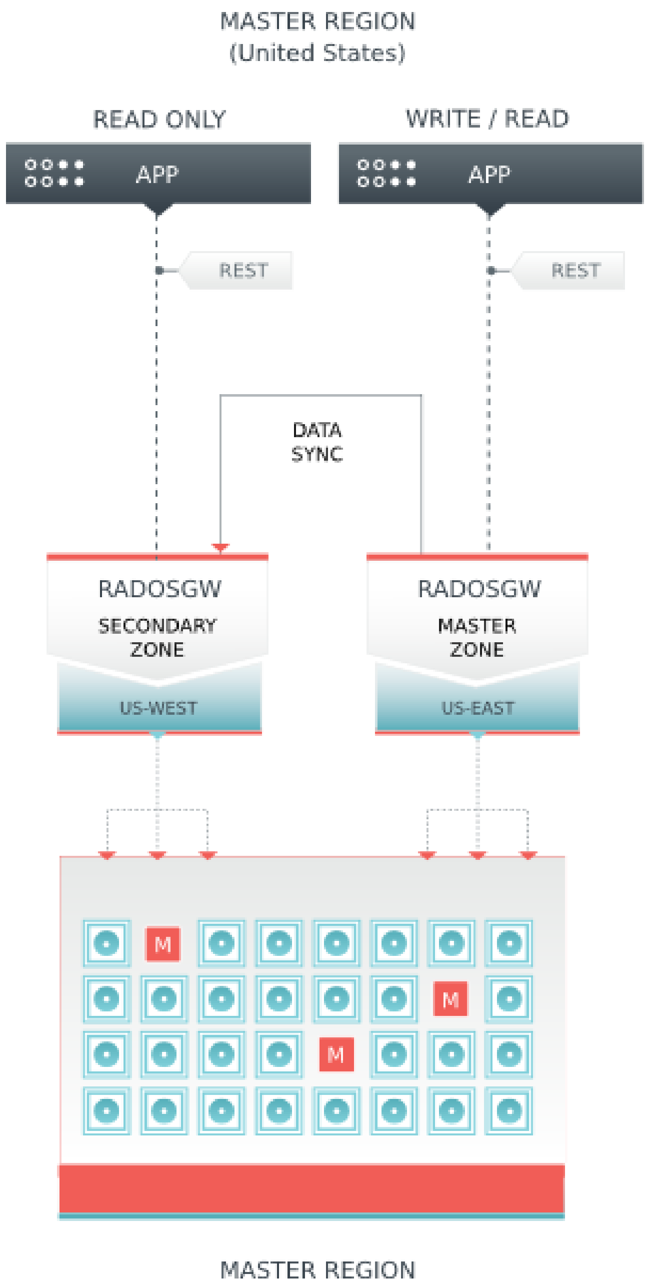
14. Create zones from secondary region in the master region.

```
radosgw-admin zone set --rgw-zone=eu-east --infile eu-east.json --name client.radosgw.us-east-1
radosgw-admin zone set --rgw-zone=eu-east --infile eu-east.json --name client.radosgw.us-west-1
radosgw-admin zone set --rgw-zone=eu-west --infile eu-west.json --name client.radosgw.us-east-1
radosgw-admin zone set --rgw-zone=eu-west --infile eu-west.json --name client.radosgw.us-west-1
```

15. *Restart Services*.
16. *Start Gateway Instances*.

7.3.5 Multi-Site Data Replication

The data synchronization agent replicates the data of a master zone to a secondary zone. The master zone of a region is the source for the secondary zone of the region and it gets selected automatically.



To configure the synchronization agent, retrieve the access key and secret for the source and destination, and the destination URL and port.

You may use `radosgw-admin zone list` to get a list of zone names. You may use `radosgw-admin zone get` to identify the key and secret for the zone. You may refer to the gateway configuration file you created under *Create a Gateway Configuration* to identify the port number.

You only need the hostname and port for a single instance (assuming all gateway instances in a region/zone access the same Ceph Storage Cluster). Specify these values in a configuration file (e.g., `cluster-data-sync.conf`), and include a `log_file` name.

For example:

```
src_access_key: {source-access-key}
src_secret_key: {source-secret-key}
```

```
destination: https://zone-name.fqdn.com:port
dest_access_key: {destination-access-key}
dest_secret_key: {destination-secret-key}
log_file: {log.filename}
```

A concrete example may look like this:

```
src_access_key: DG8RE354EFPZBICHIAF0
src_secret_key: i3U0HiRP8CXaBWrcF8bbh6CbsxGYuPPwRkixfFSb
destination: https://us-west.storage.net:80
dest_access_key: U60RFI6B08F32T2PD30G
dest_secret_key: W3HuUor7Gl1Ee93pA2pq2wFk1JMQ7hTrSDecYExl
log_file: /var/log/radosgw/radosgw-sync-us-east-west.log
```

To activate the data synchronization agent, open a terminal and execute the following:

```
radosgw-agent -c region-data-sync.conf
```

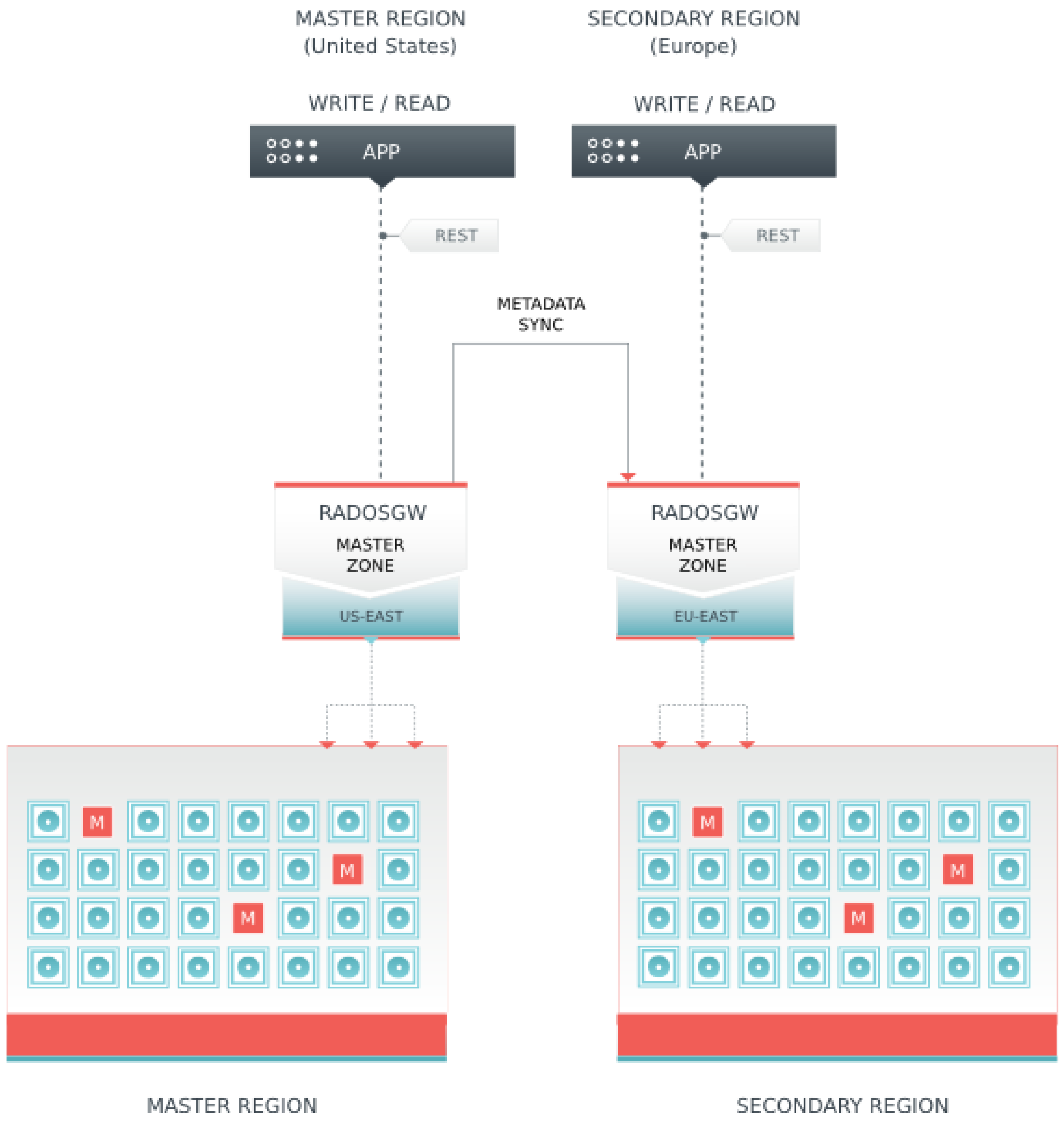
When the synchronization agent is running, you should see output indicating that the agent is synchronizing shards of data.

```
INFO:radosgw_agent.sync:Starting incremental sync
INFO:radosgw_agent.worker:17910 is processing shard number 0
INFO:radosgw_agent.worker:shard 0 has 0 entries after ''
INFO:radosgw_agent.worker:finished processing shard 0
INFO:radosgw_agent.worker:17910 is processing shard number 1
INFO:radosgw_agent.sync:1/64 shards processed
INFO:radosgw_agent.worker:shard 1 has 0 entries after ''
INFO:radosgw_agent.worker:finished processing shard 1
INFO:radosgw_agent.sync:2/64 shards processed
...
```

Note: You must have an agent for each source-destination pair.

7.3.6 Inter-Region Metadata Replication

The data synchronization agent replicates the metadata of master zone in the master region to a master zone in a secondary region. Metadata consists of gateway users and buckets, but not the objects within the buckets–ensuring a unified namespace across the cluster. The master zone of the master region is the source for the master zone of the secondary region and it gets selected automatically.



Follow the same steps in *Multi-Site Data Replication* by specifying the master zone of the master region as the source zone and the master zone of the secondary region as the secondary zone. When activating the `radosgw-agent`, specify `--metadata-only` so that it only copies metadata. For example:

```
radosgw-agent -c inter-region-data-sync.conf --metadata-only
```

Once you have completed the foregoing procedure, you should have a cluster consisting of a master region (`us`) and a secondary region (`eu`) where there is a unified namespace between the two regions.

7.4 Ceph Object Gateway Config Reference

The following settings may added to the Ceph configuration file (i.e., usually `ceph.conf`) under the `[client.radosgw.{instance-name}]` section. The settings may contain default values. If you do not specify each setting in the Ceph configuration file, the default value will be set automatically.

`rgw data`

Description Sets the location of the data files for Ceph Object Gateway.

Type String

Default `/var/lib/ceph/radosgw/$cluster-$id`

`rgw enable apis`

Description Enables the specified APIs.

Type String

Default `s3, swift, swift_auth, admin` All APIs.

`rgw cache enabled`

Description Whether the Ceph Object Gateway cache is enabled.

Type Boolean

Default `true`

`rgw cache lru size`

Description The number of entries in the Ceph Object Gateway cache.

Type Integer

Default `10000`

`rgw socket path`

Description The socket path for the domain socket. `FastCgiExternalServer` uses this socket. If you do not specify a socket path, Ceph Object Gateway will not run as an external server. The path you specify here must be the same as the path specified in the `rgw.conf` file.

Type String

Default N/A

`rgw host`

Description The host for the Ceph Object Gateway instance. Can be an IP address or a hostname.

Type String

Default `0.0.0.0`

`rgw port`

Description Port the instance listens for requests. If not specified, Ceph Object Gateway runs external FastCGI.

Type String

Default None

`rgw dns name`

Description The DNS name of the served domain. See also the `hostnames` setting within regions.

Type String

Default None

`rgw script uri`

Description The alternative value for the `SCRIPT_URI` if not set in the request.

Type String

Default None

`rgw request uri`

Description The alternative value for the `REQUEST_URI` if not set in the request.

Type String

Default None

`rgw print continue`

Description Enable `100-continue` if it is operational.

Type Boolean

Default `true`

`rgw remote addr param`

Description The remote address parameter. For example, the HTTP field containing the remote address, or the `X-Forwarded-For` address if a reverse proxy is operational.

Type String

Default `REMOTE_ADDR`

`rgw op thread timeout`

Description The timeout in seconds for open threads.

Type Integer

Default 600

`rgw op thread suicide timeout`

Description The time `timeout` in seconds before a Ceph Object Gateway process dies. Disabled if set to `0`.

Type Integer

Default `0`

`rgw thread pool size`

Description The size of the thread pool.

Type Integer

Default 100 threads.

`rgw num control oids`

Description The number of notification objects used for cache synchronization between different `rgw` instances.

Type Integer

Default `8`

`rgw init timeout`

Description The number of seconds before Ceph Object Gateway gives up on initialization.

Type Integer

Default `30`

`rgw mime types file`

Description The path and location of the MIME types. Used for Swift auto-detection of object types.

Type String

Default `/etc/mime.types`

`rgw gc max objs`

Description The maximum number of objects that may be handled by garbage collection in one garbage collection processing cycle.

Type Integer

Default `32`

`rgw gc obj min wait`

Description The minimum wait time before the object may be removed and handled by garbage collection processing.

Type Integer

Default `2 * 3600`

`rgw gc processor max time`

Description The maximum time between the beginning of two consecutive garbage collection processing cycles.

Type Integer

Default `3600`

`rgw gc processor period`

Description The cycle time for garbage collection processing.

Type Integer

Default `3600`

`rgw s3 success create obj status`

Description The alternate success status response for `create-obj`.

Type Integer

Default `0`

`rgw resolve cname`

Description Whether `rgw` should use DNS CNAME record of the request hostname field (if hostname is not equal to `rgw dns name`).

Type Boolean

Default `false`

`rgw object stripe size`

Description The size of an object stripe for Ceph Object Gateway objects. See Architecture for details on striping.

Type Integer

Default `4 << 20`

`rgw extended http attrs`

Description Add new set of attributes that could be set on an object. These extra attributes can be set through HTTP header fields when putting the objects. If set, these attributes will return as HTTP fields when doing GET/HEAD on the object.

Type String

Default None

Example “content_foo, content_bar”

`rgw exit timeout secs`

Description Number of seconds to wait for a process before exiting unconditionally.

Type Integer

Default `120`

`rgw get obj window size`

Description The window size in bytes for a single object request.

Type Integer

Default `16 << 20`

`rgw get obj max req size`

Description The maximum request size of a single get operation sent to the Ceph Storage Cluster.

Type Integer

Default `4 << 20`

`rgw relaxed s3 bucket names`

Description Enables relaxed S3 bucket names rules for US region buckets.

Type Boolean

Default `false`

`rgw list buckets max chunk`

Description The maximum number of buckets to retrieve in a single operation when listing user buckets.

Type Integer

Default `1000`

`rgw num zone opstate shards`

Description The maximum number of shards for keeping inter-region copy progress information.

Type Integer

Default `128`

`rgw opstate ratelimit sec`

Description The minimum time between opstate updates on a single upload. `0` disables the ratelimit.

Type Integer

Default `30`

`rgw curl wait timeout ms`

Description The timeout in milliseconds for certain `curl` calls.

Type Integer

Default `1000`

`rgw copy obj progress`

Description Enables output of object progress during long copy operations.

Type Boolean

Default `true`

`rgw copy obj progress every bytes`

Description The minimum bytes between copy progress output.

Type Integer

Default `1024 * 1024`

`rgw admin entry`

Description The entry point for an admin request URL.

Type String

Default `admin`

7.4.1 Regions

In Ceph v0.67 and beyond, Ceph Object Gateway supports federated deployments and a global namespace via the notion of regions. A region defines the geographic location of one or more Ceph Object Gateway instances within one or more zones.

Configuring regions differs from typical configuration procedures, because not all of the settings end up in a Ceph configuration file. In Ceph v0.67 and beyond, you can list regions, get a region configuration and set a region configuration.

List Regions

A Ceph cluster contains a list of regions. To list the regions, execute:

```
sudo radosgw-admin regions list
```

The `radosgw-admin` returns a JSON formatted list of regions.

```
{ "default_info": { "default_region": "default"},
  "regions": [
        "default"]}
```

Get a Region Map

To list the details of each region, execute:

```
sudo radosgw-admin region-map get
```

Note: If you receive a `failed to read region map` error, run `sudo radosgw-admin region-map update` first.

Get a Region

To view the configuration of a region, execute:

```
radosgw-admin region get [--rgw-region=<region>]
```

The `default` region looks like this:

```
{"name": "default",
 "api_name": "",
 "is_master": "true",
 "endpoints": [],
 "hostnames": [],
 "master_zone": "",
 "zones": [
   {"name": "default",
    "endpoints": [],
    "log_meta": "false",
    "log_data": "false"}
  ],
 "placement_targets": [
   {"name": "default-placement",
    "tags": [] }],
 "default_placement": "default-placement"}
```

Set a Region

Defining a region consists of creating a JSON object, specifying at least the required settings:

1. `name`: The name of the region. Required.
2. `api_name`: The API name for the region. Optional.
3. `is_master`: Determines if the region is the master region. Required. **note:** You can only have one master region.
4. `endpoints`: A list of all the endpoints in the region. For example, you may use multiple domain names to refer to the same region. Remember to escape the forward slashes (`\/`). You may also specify a port (`fqdn:port`) for each endpoint. Optional.
5. `hostnames`: A list of all the hostnames in the region. For example, you may use multiple domain names to refer to the same region. Optional. The `rgw dns name` setting will automatically be included in this list. You should restart the `radosgw` daemon(s) after changing this setting.
6. `master_zone`: The master zone for the region. Optional. Uses the default zone if not specified. **note:** You can only have one master zone per region.
7. `zones`: A list of all zones within the region. Each zone has a name (required), a list of endpoints (optional), and whether or not the gateway will log metadata and data operations (false by default).
8. `placement_targets`: A list of placement targets (optional). Each placement target contains a name (required) for the placement target and a list of tags (optional) so that only users with the tag can use the placement target (i.e., the user's `placement_tags` field in the user info).
9. `default_placement`: The default placement target for the object index and object data. Set to `default-placement` by default. You may also set a per-user default placement in the user info for each user.

To set a region, create a JSON object consisting of the required fields, save the object to a file (e.g., `region.json`); then, execute the following command:

```
sudo radosgw-admin region set --infile region.json
```

Where region.json is the JSON file you created.

Important: The default region is_master setting is true by default. If you create a new region and want to make it the master region, you must either set the default region is_master setting to false, or delete the default region.

Finally, update the map.

```
sudo radosgw-admin region-map update
```

Set a Region Map

Setting a region map consists of creating a JSON object consisting of one or more regions, and setting the master_region for the cluster. Each region in the region map consists of a key/value pair, where the key setting is equivalent to the name setting for an individual region configuration, and the val is a JSON object consisting of an individual region configuration.

You may only have one region with is_master equal to true, and it must be specified as the master_region at the end of the region map. The following JSON object is an example of a default region map.

```
{ "regions": [
     { "key": "default",
       "val": { "name": "default",
       "api_name": "",
       "is_master": "true",
       "endpoints": [],
       "hostnames": [],
       "master_zone": "",
       "zones": [
         { "name": "default",
           "endpoints": [],
           "log_meta": "false",
            "log_data": "false"}],
            "placement_targets": [
              { "name": "default-placement",
                "tags": []}],
                "default_placement": "default-placement"
              }
         }
       ],
   "master_region": "default"
}
```

To set a region map, execute the following:

```
sudo radosgw-admin region-map set --infile regionmap.json
```

Where regionmap.json is the JSON file you created. Ensure that you have zones created for the ones specified in the region map. Finally, update the map.

```
sudo radosgw-admin regionmap update
```

7.4.2 Zones

In Ceph v0.67 and beyond, Ceph Object Gateway supports the notion of zones. A zone defines a logical group consisting of one or more Ceph Object Gateway instances.

Configuring zones differs from typical configuration procedures, because not all of the settings end up in a Ceph configuration file. In Ceph v0.67 and beyond, you can list zones, get a zone configuration and set a zone configuration.

List Zones

To list the zones in a cluster, execute:

```
sudo radosgw-admin zone list
```

Get a Zone

To get the configuration of a zone, execute:

```
sudo radosgw-admin zone get [--rgw-zone=<zone>]
```

The `default` zone looks like this:

```
{ "domain_root": ".rgw",
  "control_pool": ".rgw.control",
  "gc_pool": ".rgw.gc",
  "log_pool": ".log",
  "intent_log_pool": ".intent-log",
  "usage_log_pool": ".usage",
  "user_keys_pool": ".users",
  "user_email_pool": ".users.email",
  "user_swift_pool": ".users.swift",
  "user_uid_pool": ".users.uid",
  "system_key": { "access_key": "", "secret_key": ""},
  "placement_pools": [
      {  "key": "default-placement",
         "val": { "index_pool": ".rgw.buckets.index",
                  "data_pool": ".rgw.buckets"}
      }
    ]
  }
```

Set a Zone

Configuring a zone involves specifying a series of Ceph Object Gateway pools. For consistency, we recommend using a pool prefix that is the same as the zone name. See *Pools* for details of configuring pools.

To set a zone, create a JSON object consisting of the pools, save the object to a file (e.g., `zone.json`); then, execute the following command, replacing `{zone-name}` with the name of the zone:

```
sudo radosgw-admin zone set --rgw-zone={zone-name} --infile zone.json
```

Where `zone.json` is the JSON file you created.

7.4.3 Region/Zone Settings

You may include the following settings in your Ceph configuration file under each `[client.radosgw.{instance-name}]` instance.

New in version v.67.

`rgw zone`

Description The name of the zone for the gateway instance.

Type String

Default None

New in version v.67.

`rgw region`

Description The name of the region for the gateway instance.

Type String

Default None

New in version v.67.

`rgw default region info oid`

Description The OID for storing the default region. We do not recommend changing this setting.

Type String

Default `default.region`

7.4.4 Pools

Ceph zones map to a series of Ceph Storage Cluster pools.

Manually Created Pools vs. Generated Pools

If you provide write capabilities to the user key for your Ceph Object Gateway, the gateway has the ability to create pools automatically. This is convenient, but the Ceph Object Storage Cluster uses the default values for the number of placement groups (which may not be ideal) or the values you specified in your Ceph configuration file. If you allow the Ceph Object Gateway to create pools automatically, ensure that you have reasonable defaults for the number of placement groups. See Pool Configuration for details. See Cluster Pools for details on creating pools.

The default pools for the Ceph Object Gateway's default zone include:

- `.rgw`
- `.rgw.control`
- `.rgw.gc`
- `.log`
- `.intent-log`
- `.usage`
- `.users`

- `.users.email`
- `.users.swift`
- `.users.uid`

You have significant discretion in determining how you want a zone to access pools. You can create pools on a per zone basis, or use the same pools for multiple zones. As a best practice, we recommend having a separate set of pools for your master zone and your secondary zones in each region. When creating pools for a specific zone, consider prepending the region name and zone name to the default pool names. For example:

- `.region1-zone1.domain.rgw`
- `.region1-zone1.rgw.control`
- `.region1-zone1.rgw.gc`
- `.region1-zone1.log`
- `.region1-zone1.intent-log`
- `.region1-zone1.usage`
- `.region1-zone1.users`
- `.region1-zone1.users.email`
- `.region1-zone1.users.swift`
- `.region1-zone1.users.uid`

Ceph Object Gateways store data for the bucket index (`index_pool`) and bucket data (`data_pool`) in placement pools. These may overlap–i.e., you may use the same pool for the index and the data. The index pool for default placement is `.rgw.buckets.index` and for the data pool for default placement is `.rgw.buckets`. See *Zones* for details on specifying pools in a zone configuration.

Deprecated since version v.67.

`rgw cluster root pool`

Description The Ceph Storage Cluster pool to store `radosgw` metadata for this instance. Not used in Ceph version v.67 and later. Use `rgw zone root pool` instead.

Type String

Required No

Default `.rgw.root`

Replaced By `rgw zone root pool`

New in version v.67.

`rgw region root pool`

Description The pool for storing all region-specific information.

Type String

Default `.rgw.root`

New in version v.67.

`rgw zone root pool`

Description The pool for storing zone-specific information.

Type String

Default `.rgw.root`

7.4.5 Swift Settings

`rgw enforce swift acls`

Description Enforces the Swift Access Control List (ACL) settings.

Type Boolean

Default `true`

`rgw swift token expiration`

Description The time in seconds for expiring a Swift token.

Type Integer

Default `24 * 3600`

`rgw swift url`

Description The URL for the Ceph Object Gateway Swift API.

Type String

Default None

`rgw swift url prefix`

Description The URL prefix for the Swift API.

Default `swift`

Example http://fqdn.com/swift

`rgw swift auth url`

Description Default URL for verifying v1 auth tokens (if not using internal Swift auth).

Type String

Default None

`rgw swift auth entry`

Description The entry point for a Swift auth URL.

Type String

Default `auth`

7.4.6 Logging Settings

`rgw log nonexistent bucket`

Description Enables Ceph Object Gateway to log a request for a non-existent bucket.

Type Boolean

Default `false`

`rgw log object name`

Description The logging format for an object name. See manpage *`date`* for details about format specifiers.

Type Date

Default `%Y-%m-%d-%H-%i-%n`

`rgw log object name utc`

Description Whether a logged object name includes a UTC time. If `false`, it uses the local time.

Type Boolean

Default `false`

`rgw usage max shards`

Description The maximum number of shards for usage logging.

Type Integer

Default `32`

`rgw usage max user shards`

Description The maximum number of shards used for a single user's usage logging.

Type Integer

Default `1`

`rgw enable ops log`

Description Enable logging for each successful Ceph Object Gateway operation.

Type Boolean

Default `false`

`rgw enable usage log`

Description Enable the usage log.

Type Boolean

Default `false`

`rgw ops log rados`

Description Whether the operations log should be written to the Ceph Storage Cluster backend.

Type Boolean

Default `true`

`rgw ops log socket path`

Description The Unix domain socket for writing operations logs.

Type String

Default None

`rgw ops log data backlog`

Description The maximum data backlog data size for operations logs written to a Unix domain socket.

Type Integer

Default `5 << 20`

`rgw usage log flush threshold`

Description The number of dirty merged entries in the usage log before flushing synchronously.

Type Integer

Default 1024

`rgw usage log tick interval`

Description Flush pending usage log data every `n` seconds.

Type Integer

Default `30`

`rgw intent log object name`

Description The logging format for the intent log object name. See manpage *`date`* for details about format specifiers.

Type Date

Default `%Y-%m-%d-%i-%n`

`rgw intent log object name utc`

Description Whether the intent log object name includes a UTC time. If `false`, it uses the local time.

Type Boolean

Default `false`

`rgw data log window`

Description The data log entries window in seconds.

Type Integer

Default `30`

`rgw data log changes size`

Description The number of in-memory entries to hold for the data changes log.

Type Integer

Default `1000`

`rgw data log num shards`

Description The number of shards (objects) on which to keep the data changes log.

Type Integer

Default `128`

`rgw data log obj prefix`

Description The object name prefix for the data log.

Type String

Default `data_log`

`rgw replica log obj prefix`

Description The object name prefix for the replica log.

Type String

Default `replica log`

`rgw md log max shards`

Description The maximum number of shards for the metadata log.

Type Integer

Default `64`

7.4.7 Keystone Settings

`rgw keystone url`

Description The URL for the Keystone server.

Type String

Default None

`rgw keystone admin token`

Description The Keystone admin token (shared secret).

Type String

Default None

`rgw keystone accepted roles`

Description The roles requires to serve requests.

Type String

Default `Member, admin`

`rgw keystone token cache size`

Description The maximum number of entries in each Keystone token cache.

Type Integer

Default `10000`

`rgw keystone revocation interval`

Description The number of seconds between token revocation checks.

Type Integer

Default `15 * 60`

7.5 Admin Guide

Once you have your Ceph Object Storage service up and running, you may administer the service with user management, access controls, quotas and usage tracking among other features.

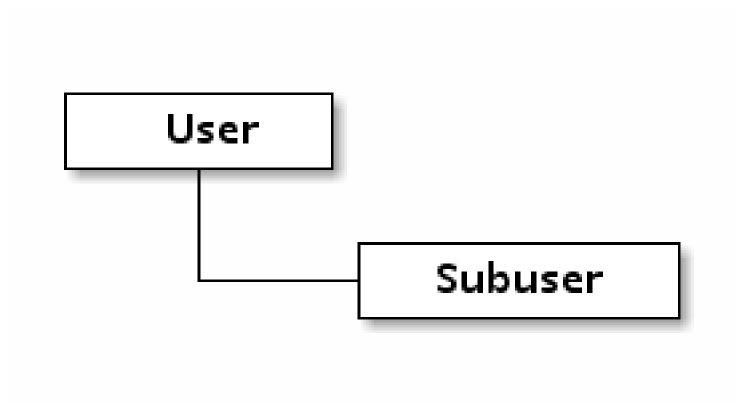
7.5.1 User Management

Ceph Object Storage user management refers to users of the Ceph Object Storage service (i.e., not the Ceph Object Gateway as a user of the Ceph Storage Cluster). You must create a user, access key and secret to enable end users to interact with Ceph Object Gateway services.

There are two user types:

- **User:** The term 'user' reflects a user of the S3 interface.

- **Subuser:** The term 'subuser' reflects a user of the Swift interface. A subuser is associated to a user .



You can create, modify, view, suspend and remove users and subusers. In addition to user and subuser IDs, you may add a display name and an email address for a user. You can specify a key and secret, or generate a key and secret automatically. When generating or specifying keys, note that user IDs correspond to an S3 key type and subuser IDs correspond to a swift key type. Swift keys also have access levels of `read`, `write`, `readwrite` and `full`.

Create a User

To create a user (S3 interface), execute the following:

```
radosgw-admin user create --uid={username} --display-name="{display-name}" [--email={email}]
```

For example:

```
radosgw-admin user create --uid=johndoe --display-name="John Doe" --email=john@example.com
```

```
{ "user_id": "johndoe",
  "display_name": "John Doe",
  "email": "john@example.com",
  "suspended": 0,
  "max_buckets": 1000,
  "auid": 0,
  "subusers": [],
  "keys": [
        { "user": "johndoe",
          "access_key": "11BS02LGFB6AL6H1ADMW",
          "secret_key": "vzCEkuryfn060dfee4fgQPqFrncKEIkh3ZcdOANY"}],
  "swift_keys": [],
  "caps": [],
  "op_mask": "read, write, delete",
  "default_placement": "",
  "placement_tags": [],
  "bucket_quota": { "enabled": false,
      "max_size_kb": -1,
      "max_objects": -1},
  "user_quota": { "enabled": false,
      "max_size_kb": -1,
      "max_objects": -1},
  "temp_url_keys": []}
```

Creating a user also creates an `access_key` and `secret_key` entry for use with any S3 API-compatible client.

Important: Check the key output. Sometimes `radosgw-admin` generates a JSON escape (`\`) character, and some clients do not know how to handle JSON escape characters. Remedies include removing the JSON escape character (`\`), encapsulating the string in quotes, regenerating the key and ensuring that it does not have a JSON escape character or specify the key and secret manually.

Create a Subuser

To create a subuser (Swift interface) for the user, you must specify the user ID (`--uid={username}`), a subuser ID and the access level for the subuser.

```
radosgw-admin subuser create --uid={uid} --subuser={uid} --access=[ read | write | readwrite | full ]
```

For example:

```
radosgw-admin subuser create --uid=johndoe --subuser=johndoe:swift --access=full
```

Note: `full` is not `readwrite`, as it also includes the access control policy.

```
{ "user_id": "johndoe",
  "display_name": "John Doe",
  "email": "john@example.com",
  "suspended": 0,
  "max_buckets": 1000,
  "auid": 0,
  "subusers": [
        { "id": "johndoe:swift",
          "permissions": "full-control"}],
  "keys": [
        { "user": "johndoe",
          "access_key": "11BS02LGFB6AL6H1ADMW",
          "secret_key": "vzCEkuryfn060dfee4fgQPqFrncKEIkh3ZcdOANY"}],
  "swift_keys": [],
  "caps": [],
  "op_mask": "read, write, delete",
  "default_placement": "",
  "placement_tags": [],
  "bucket_quota": { "enabled": false,
      "max_size_kb": -1,
      "max_objects": -1},
  "user_quota": { "enabled": false,
      "max_size_kb": -1,
      "max_objects": -1},
  "temp_url_keys": []}
```

Get User Info

To get information about a user, you must specify `user info` and the user ID (`--uid={username}`).

```
radosgw-admin user info --uid=johndoe
```

Modify User Info

To modify information about a user, you must specify the user ID (`--uid={username}`) and the attributes you want to modify. Typical modifications are to keys and secrets, email addresses, display names and access levels. For example:

```
radosgw-admin user modify --uid=johndoe --display-name="John E. Doe"
```

To modify subuser values, specify `subuser modify` and the subuser ID. For example:

```
radosgw-admin subuser modify --uid=johndoe:swift --access=full
```

User Enable/Suspend

When you create a user, the user is enabled by default. However, you may suspend user privileges and re-enable them at a later time. To suspend a user, specify `user suspend` and the user ID.

```
radosgw-admin user suspend --uid=johndoe
```

To re-enable a suspended user, specify `user enable` and the user ID.

```
radosgw-admin user enable --uid=johndoe
```

Note: Disabling the user disables the subuser.

Remove a User

When you remove a user, the user and subuser are removed from the system. However, you may remove just the subuser if you wish. To remove a user (and subuser), specify `user rm` and the user ID.

```
radosgw-admin user rm --uid=johndoe
```

To remove the subuser only, specify `subuser rm` and the subuser ID.

```
radosgw-admin subuser rm --uid=johndoe:swift
```

Options include:

- **Purge Data:** The `--purge-data` option purges all data associated to the UID.
- **Purge Keys:** The `--purge-keys` option purges all keys associated to the UID.

Remove a Subuser

When you remove a sub user, you are removing access to the Swift interface. The user will remain in the system. The Ceph Object Gateway To remove the subuser, specify `subuser rm` and the subuser ID.

```
radosgw-admin subuser rm --uid=johndoe:swift
```

Options include:

- **Purge Keys:** The `--purge-keys` option purges all keys associated to the UID.

Create a Key

To create a key for a user, you must specify `key create`. For a user, specify the user ID and the `s3` key type. To create a key for subuser, you must specify the subuser ID and the `swift` keytype. For example:

```
radosgw-admin key create --subuser=johndoe:swift --key-type=swift --gen-secret
```

```
{ "user_id": "johndoe",
  "rados_uid": 0,
  "display_name": "John Doe",
  "email": "john@example.com",
  "suspended": 0,
  "subusers": [
     { "id": "johndoe:swift",
       "permissions": "full-control"}],
  "keys": [
    { "user": "johndoe",
      "access_key": "QFAMEDSJP5DEKJO0DDXY",
      "secret_key": "iaSFLDVvDdQt6lkNzHyW4fPLZugBAI1g17LO0+87"}],
  "swift_keys": [
    { "user": "johndoe:swift",
      "secret_key": "E9T2rUZNu2gxUjcwUBO8n\/Ev4KX6\/GprEuH4qhu1"}]}
```

Add / Remove Access Keys

Users and subusers must have access keys to use the S3 and Swift interfaces. When you create a user or subuser and you do not specify an access key and secret, the key and secret get generated automatically. You may create a key and either specify or generate the access key and/or secret. You may also remove an access key and secret. Options include:

- `--secret=<key>` specifies a secret key (e.g,. manually generated).
- `--gen-access-key` generates random access key (for S3 user by default).
- `--gen-secret` generates a random secret key.
- `--key-type=<type>` specifies a key type. The options are: swift, s3

To add a key, specify the user.

```
radosgw-admin key create --uid=johndoe --key-type=s3 --gen-access-key --gen-secret
```

You may also specify a key and a secret.

To remove an access key, specify the user.

```
radosgw-admin key rm --uid=johndoe
```

Add / Remove Admin Capabilities

The Ceph Storage Cluster provides an administrative API that enables users to execute administrative functions via the REST API. By default, users do NOT have access to this API. To enable a user to exercise administrative functionality, provide the user with administrative capabilities.

To add administrative capabilities to a user, execute the following:

```
radosgw-admin caps add --uid={uid} --caps={caps}
```

You can add read, write or all capabilities to users, buckets, metadata and usage (utilization). For example:

```
--caps="[users|buckets|metadata|usage|zone]=[*|read|write|read, write]"
```

For example:

```
radosgw-admin caps add --uid=johndoe --caps="users=*"
```

To remove administrative capabilities from a user, execute the following:

```
radosgw-admin caps remove --uid=johndoe --caps={caps}
```

7.5.2 Quota Management

The Ceph Object Gateway enables you to set quotas on users and buckets owned by users. Quotas include the maximum number of objects in a bucket and the maximum storage size in megabytes.

- **Bucket:** The `--bucket` option allows you to specify a quota for buckets the user owns.
- **Maximum Objects:** The `--max-objects` setting allows you to specify the maximum number of objects. A negative value disables this setting.
- **Maximum Size:** The `--max-size` option allows you to specify a quota for the maximum number of bytes. A negative value disables this setting.
- **Quota Scope:** The `--quota-scope` option sets the scope for the quota. The options are `bucket` and `user`. Bucket quotas apply to buckets a user owns. User quotas apply to a user.

Set User Quota

Before you enable a quota, you must first set the quota parameters. For example:

```
radosgw-admin quota set --quota-scope=user --uid=<uid> [--max-objects=<num objects>] [--max-size=<ma
```

For example:

```
radosgw-admin quota set --quota-scope=user --uid=johndoe --max-objects=1024 --max-size=1024
```

A negative value for num objects and / or max size means that the specific quota attribute check is disabled.

Enable/Disable User Quota

Once you set a user quota, you may enable it. For example:

```
radosgw-admin quota enable --quota-scope=user --uid=<uid>
```

You may disable an enabled user quota. For example:

```
radosgw-admin quota-disable --quota-scope=user --uid=<uid>
```

Set Bucket Quota

Bucket quotas apply to the buckets owned by the specified `uid`. They are independent of the user.

```
radosgw-admin quota set --uid=<uid> --quota-scope=bucket [--max-objects=<num objects>] [--max-size=<
```

A negative value for num objects and / or max size means that the specific quota attribute check is disabled.

Enable/Disable Bucket Quota

Once you set a bucket quota, you may enable it. For example:

```
radosgw-admin quota enable --quota-scope=bucket --uid=<uid>
```

You may disable an enabled bucket quota. For example:

```
radosgw-admin quota-disable --quota-scope=bucket --uid=<uid>
```

Get Quota Settings

You may access each user's quota settings via the user information API. To read user quota setting information with the CLI interface, execute the following:

```
radosgw-admin user info --uid=<uid>
```

Update Quota Stats

Quota stats get updated asynchronously. You can update quota statistics for all users and all buckets manually to retrieve the latest quota stats.

```
radosgw-admin user stats --uid=<uid> --sync-stats
```

Get User Usage Stats

To see how much of the quota a user has consumed, execute the following:

```
radosgw-admin user stats --uid=<uid>
```

Note: You should execute `radosgw-admin user stats` with the `--sync-stats` option to receive the latest data.

Reading / Writing Global Quotas

You can read and write quota settings in a region map. To get a region map, execute the following.

```
radosgw-admin regionmap get > regionmap.json
```

To set quota settings for the entire region, simply modify the quota settings in the region map. Then, use `region set` to update the region map.

```
radosgw-admin region set < regionmap.json
```

Note: After updating the region map, you must restart the gateway.

7.5.3 Usage

The Ceph Object Gateway logs usage for each user. You can track user usage within date ranges too.

Options include:

- **Start Date:** The `--start-date` option allows you to filter usage stats from a particular start date (**format:** `yyyy-mm-dd[HH:MM:SS]`).
- **End Date:** The `--end-date` option allows you to filter usage up to a particular date (**format:** `yyyy-mm-dd[HH:MM:SS]`).
- **Log Entries:** The `--show-log-entries` option allows you to specify whether or not to include log entries with the usage stats (options: `true` | `false`).

Note: You may specify time with minutes and seconds, but it is stored with 1 hour resolution.

Show Usage

To show usage statistics, specify the `usage show`. To show usage for a particular user, you must specify a user ID. You may also specify a start date, end date, and whether or not to show log entries.:

```
radosgw-admin usage show --uid=johndoe --start-date=2012-03-01 --end-date=2012-04-01
```

You may also show a summary of usage information for all users by omitting a user ID.

```
radosgw-admin usage show --show-log-entries=false
```

Trim Usage

With heavy use, usage logs can begin to take up storage space. You can trim usage logs for all users and for specific users. You may also specify date ranges for trim operations.

```
radosgw-admin usage trim --start-date=2010-01-01 --end-date=2010-12-31
radosgw-admin usage trim --uid=johndoe
radosgw-admin usage trim --uid=johndoe --end-date=2013-12-31
```

7.6 Purging Temporary Data

Deprecated since version 0.52.

When you delete objects (and buckets/containers), the Gateway marks the data for removal, but it is still available to users until it is purged. Since data still resides in storage until it is purged, it may take up available storage space. To ensure that data marked for deletion isn't taking up a significant amount of storage space, you should run the following command periodically:

```
radosgw-admin temp remove
```

Important: Data marked for deletion may still be read. So consider executing the foregoing command a reasonable interval after data was marked for deletion.

Tip: Consider setting up a `cron` job to purge data.

7.7 Ceph Object Gateway S3 API

Ceph supports a RESTful API that is compatible with the basic data access model of the Amazon S3 API.

7.7.1 API

Common Entities

Bucket and Host Name

There are two different modes of accessing the buckets. The first (preferred) method identifies the bucket as the top-level directory in the URI.

```
GET /mybucket HTTP/1.1
Host: cname.domain.com
```

The second method identifies the bucket via a virtual bucket host name. For example:

```
GET / HTTP/1.1
Host: mybucket.cname.domain.com
```

Tip: We prefer the first method, because the second method requires expensive domain certification and DNS wild cards.

Common Request Headers

Request Header	Description
`CONTENT_LENGTH`	Length of the request body.
`DATE`	Request time and date (in UTC).
`HOST`	The name of the host server.
`AUTHORIZATION`	Authorization token.

Common Response Status

HTTP Status	Response Code
`100`	Continue
`200`	Success
`201`	Created
`202`	Accepted
`204`	NoContent
`206`	Partial content
`304`	NotModified
`400`	InvalidArgument
`400`	InvalidDigest
`400`	BadDigest
`400`	InvalidBucketName
`400`	InvalidObjectName
`400`	UnresolvableGrantByEmailAddress
`400`	InvalidPart
`400`	InvalidPartOrder
`400`	RequestTimeout
`400`	EntityTooLarge
`403`	AccessDenied

Continued on next page

Table 7.1 – continued from previous page

HTTP Status	Response Code
403	UserSuspended
403	RequestTimeTooSkewed
404	NoSuchKey
404	NoSuchBucket
404	NoSuchUpload
405	MethodNotAllowed
408	RequestTimeout
409	BucketAlreadyExists
409	BucketNotEmpty
411	MissingContentLength
412	PreconditionFailed
416	InvalidRange
422	UnprocessableEntity
500	InternalError

Authentication and ACLs

Requests to the RADOS Gateway (RGW) can be either authenticated or unauthenticated. RGW assumes unauthenticated requests are sent by an anonymous user. RGW supports canned ACLs.

Authentication

Authenticating a request requires including an access key and a Hash-based Message Authentication Code (HMAC) in the request before it is sent to the RGW server. RGW uses an S3-compatible authentication approach.

```
HTTP/1.1
PUT /buckets/bucket/object.mpeg
Host: cname.domain.com
Date: Mon, 2 Jan 2012 00:01:01 +0000
Content-Encoding: mpeg
Content-Length: 9999999

Authorization: AWS {access-key}:{hash-of-header-and-secret}
```

In the foregoing example, replace `{access-key}` with the value for your access key ID followed by a colon (`:`). Replace `{hash-of-header-and-secret}` with a hash of the header string and the secret corresponding to the access key ID.

To generate the hash of the header string and secret, you must:

1. Get the value of the header string.
2. Normalize the request header string into canonical form.
3. Generate an HMAC using a SHA-1 hashing algorithm. See RFC 2104 and HMAC for details.
4. Encode the `hmac` result as base-64.

To normalize the header into canonical form:

1. Get all fields beginning with `x-amz-`.
2. Ensure that the fields are all lowercase.
3. Sort the fields lexicographically.

4. Combine multiple instances of the same field name into a single field and separate the field values with a comma.
5. Replace white space and line breaks in field values with a single space.
6. Remove white space before and after colons.
7. Append a new line after each field.
8. Merge the fields back into the header.

Replace the `{hash-of-header-and-secret}` with the base-64 encoded HMAC string.

Access Control Lists (ACLs)

RGW supports S3-compatible ACL functionality. An ACL is a list of access grants that specify which operations a user can perform on a bucket or on an object. Each grant has a different meaning when applied to a bucket versus applied to an object:

Permission	Bucket	Object
`READ`	Grantee can list the objects in the bucket.	Grantee can read the object.
`WRITE`	Grantee can write or delete objects in the bucket.	N/A
`READ_ACP`	Grantee can read bucket ACL.	Grantee can read the object ACL.
`WRITE_ACP`	Grantee can write bucket ACL.	Grantee can write to the object ACL.
`FULL_CONTROL`	Grantee has full permissions for object in the bucket.	Grantee can read or write to the object ACL.

Service Operations

List Buckets

`GET /` returns a list of buckets created by the user making the request. `GET /` only returns buckets created by an authenticated user. You cannot make an anonymous request.

Syntax

```
GET / HTTP/1.1
Host: cname.domain.com

Authorization: AWS {access-key}:{hash-of-header-and-secret}
```

Response Entities

Name	Type	Description
`Buckets`	Container	Container for list of buckets.
`Bucket`	Container	Container for bucket information.
`Name`	String	Bucket name.
`CreationDate`	Date	UTC time when the bucket was created.
`ListAllMyBucketsResult`	Container	A container for the result.
`Owner`	Container	A container for the bucket owner's `ID` and `DisplayName`.
`ID`	String	The bucket owner's ID.
`DisplayName`	String	The bucket owner's display name.

Bucket Operations

PUT Bucket

Creates a new bucket. To create a bucket, you must have a user ID and a valid AWS Access Key ID to authenticate requests. You may not create buckets as an anonymous user.

Note: We do not support request entities for `PUT /{bucket}` in this release.

Constraints In general, bucket names should follow domain name constraints.

- Bucket names must be unique.
- Bucket names must begin and end with a lowercase letter.
- Bucket names may contain a dash (-).

Syntax

```
PUT /{bucket} HTTP/1.1
Host: cname.domain.com
x-amz-acl: public-read-write

Authorization: AWS {access-key}:{hash-of-header-and-secret}
```

Parameters

Name	Description	Valid Values	Required
`x-amz-acl`	Canned ACLs.	`private`, `public-read`, `public-read-write`, `authenticated-read`	No

HTTP Response If the bucket name is unique, within constraints and unused, the operation will succeed. If a bucket with the same name already exists and the user is the bucket owner, the operation will succeed. If the bucket name is already in use, the operation will fail.

HTTP Status	Status Code	Description
`409`	BucketAlreadyExists	Bucket already exists under different user's ownership.

DELETE Bucket

Deletes a bucket. You can reuse bucket names following a successful bucket removal.

Syntax

```
DELETE /{bucket} HTTP/1.1
Host: cname.domain.com

Authorization: AWS {access-key}:{hash-of-header-and-secret}
```

HTTP Response

HTTP Status	Status Code	Description
`204`	No Content	Bucket removed.

GET Bucket

Returns a list of bucket objects.

Syntax

```
GET /{bucket}?max-keys=25 HTTP/1.1
Host: cname.domain.com
```

Parameters

Name	Type	Description
`prefix`	String	Only returns objects that contain the specified prefix.
`delimiter`	String	The delimiter between the prefix and the rest of the object name.
`marker`	String	A beginning index for the list of objects returned.
`max-keys`	Integer	The maximum number of keys to return. Default is 1000.

HTTP Response

HTTP Status	Status Code	Description
`200`	OK	Buckets retrieved

Bucket Response Entities `GET /{bucket}` returns a container for buckets with the following fields.

Name	Type	Description
`ListBucketResult`	Entity	The container for the list of objects.
`Name`	String	The name of the bucket whose contents will be returned.
`Prefix`	String	A prefix for the object keys.
`Marker`	String	A beginning index for the list of objects returned.
`MaxKeys`	Integer	The maximum number of keys returned.
`Delimiter`	String	If set, objects with the same prefix will appear in the `CommonPrefixes` list.
`IsTruncated`	Boolean	If `true`, only a subset of the bucket's contents were returned.
`CommonPrefixes`	Container	If multiple objects contain the same prefix, they will appear in this list.

Object Response Entities The `ListBucketResult` contains objects, where each object is within a `Contents` container.

Name	Type	Description
`Contents`	Object	A container for the object.
`Key`	String	The object's key.
`LastModified`	Date	The object's last-modified date/time.
`ETag`	String	An MD-5 hash of the object. (entity tag)
`Size`	Integer	The object's size.
`StorageClass`	String	Should always return `STANDARD`.

Get Bucket Location

Retrieves the bucket's region. The user needs to be the bucket owner to call this. A bucket can be constrained to a region by providing `LocationConstraint` during a PUT request.

Syntax Add the `location` subresource to bucket resource as shown below

```
GET /{bucket}?location HTTP/1.1
Host: cname.domain.com

Authorization: AWS {access-key}:{hash-of-header-and-secret}
```

Response Entities

Name	Type	Description
`LocationConstraint`	String	The region where bucket resides, empty string for defult region

Get Bucket ACL

Retrieves the bucket access control list. The user needs to be the bucket owner or to have been granted `READ_ACP` permission on the bucket.

Syntax Add the `acl` subresource to the bucket request as shown below.

```
GET /{bucket}?acl HTTP/1.1
Host: cname.domain.com

Authorization: AWS {access-key}:{hash-of-header-and-secret}
```

Response Entities

Name	Type	Description
`AccessControlPolicy`	Container	A container for the response.
`AccessControlList`	Container	A container for the ACL information.
`Owner`	Container	A container for the bucket owner's `ID` and `DisplayName`.
`ID`	String	The bucket owner's ID.
`DisplayName`	String	The bucket owner's display name.
`Grant`	Container	A container for `Grantee` and `Permission`.
`Grantee`	Container	A container for the `DisplayName` and `ID` of the user receiving a grant permission.
`Permission`	String	The permission given to the `Grantee` bucket.

PUT Bucket ACL

Sets an access control to an existing bucket. The user needs to be the bucket owner or to have been granted `WRITE_ACP` permission on the bucket.

Syntax Add the `acl` subresource to the bucket request as shown below.

```
PUT /{bucket}?acl HTTP/1.1
```

Request Entities

Name	Type	Description
`AccessControlPolicy`	Container	A container for the request.
`AccessControlList`	Container	A container for the ACL information.
`Owner`	Container	A container for the bucket owner's `ID` and `DisplayName`.
`ID`	String	The bucket owner's ID.
`DisplayName`	String	The bucket owner's display name.
`Grant`	Container	A container for `Grantee` and `Permission`.
`Grantee`	Container	A container for the `DisplayName` and `ID` of the user receiving a grant of permission.
`Permission`	String	The permission given to the `Grantee` bucket.

List Bucket Multipart Uploads

`GET /?uploads` returns a list of the current in-progress multipart uploads–i.e., the application initiates a multipart upload, but the service hasn't completed all the uploads yet.

Syntax

```
GET /{bucket}?uploads HTTP/1.1
```

Parameters You may specify parameters for `GET /{bucket}?uploads`, but none of them are required.

Name	Type	Description
`prefix`	String	Returns in-progress uploads whose keys contains the specified prefix.
`delimiter`	String	The delimiter between the prefix and the rest of the object name.
`key-marker`	String	The beginning marker for the list of uploads.
`max-keys`	Integer	The maximum number of in-progress uploads. The default is 1000.
`max-uploads`	Integer	The maximum number of multipart uploads. The range from 1-1000. The default is 1000.
`upload-id-marker`	String	Ignored if `key-marker` isn't specified. Specifies the `ID` of first upload to list in lexicographical order at or following the `ID`.

Response Entities

Name	Type	Description
`ListMultipartUploadsResult`	Container	A container for the results.
`ListMultipartUploadsResult.Prefix`	String	The prefix specified by the `prefix` request parameter (if any).
`Bucket`	String	The bucket that will receive the bucket contents.
`KeyMarker`	String	The key marker specified by the `key-marker` request parameter (if any).
`UploadIdMarker`	String	The marker specified by the `upload-id-marker` request parameter (if any).
`NextKeyMarker`	String	The key marker to use in a subsequent request if `IsTruncated` i `true`.
`NextUploadIdMarker`	String	The upload ID marker to use in a subsequent request if `IsTruncated` is `true`.
`MaxUploads`	Integer	The max uploads specified by the `max-uploads` request parameter.
`Delimiter`	String	If set, objects with the same prefix will appear in the `CommonPrefixes` list.
`IsTruncated`	Boolean	If `true`, only a subset of the bucket's upload contents were returne
`Upload`	Container	A container for `Key`, `UploadId`, `InitiatorOwner`, `StorageClass`, and `Initiated` elements.
`Key`	String	The key of the object once the multipart upload is complete.
`UploadId`	String	The `ID` that identifies the multipart upload.
`Initiator`	Container	Contains the `ID` and `DisplayName` of the user who initiated the upload.
`DisplayName`	String	The initiator's display name.
`ID`	String	The initiator's ID.
`Owner`	Container	A container for the `ID` and `DisplayName` of the user who owns the uploaded object.
`StorageClass`	String	The method used to store the resulting object. `STANDARD` or `REDUCED_REDUNDANCY`
`Initiated`	Date	The date and time the user initiated the upload.
`CommonPrefixes`	Container	If multiple objects contain the same prefix, they will appear in this list.
`CommonPrefixes.Prefix`	String	The substring of the key after the prefix as defined by the `prefix` request parameter.

Object Operations

Put Object

Adds an object to a bucket. You must have write permissions on the bucket to perform this operation.

Syntax

```
PUT /{bucket}/{object} HTTP/1.1
```

Request Headers

Name	Description	Valid Values	Re- quire
content-md5	A base64 encoded MD-5 hash of the message.	A string. No defaults or constraints.	No
content-type	A standard MIME type.	Any MIME type. Default: `binary/octet-stream`	No
x-amz-meta-<...>	User metadata. Stored with the object.	A string up to 8kb. No defaults.	No
x-amz-acl	A canned ACL.	`private`, `public-read`, `public-read-write`, `authenticated-read`	No

Copy Object

To copy an object, use `PUT` and specify a destination bucket and the object name.

Syntax

```
PUT /{dest-bucket}/{dest-object} HTTP/1.1
x-amz-copy-source: {source-bucket}/{source-object}
```

Request Headers

Name	Description	Valid Values	Re- quire
x-amz-copy-source	The source bucket name + object name.	{bucket}/{obj}	Yes
x-amz-acl	A canned ACL.	`private`, `public-read`, `public-read-write`, `authenticated-read`	No
x-amz-copy-if-modified-since	Copies only if modified since the timestamp.	Timestamp	No
x-amz-copy-if-unmodified-since	Copies only if unmodified since the timestamp.	Timestamp	No
x-amz-copy-if-match	Copies only if object ETag matches ETag.	Entity Tag	No
x-amz-copy-if-none-match	Copies only if object ETag doesn't match.	Entity Tag	No

Response Entities

Name	Type	Description
CopyObjectResult	Container	A container for the response elements.
LastModified	Date	The last modified date of the source object.
Etag	String	The ETag of the new object.

Remove Object

Removes an object. Requires WRITE permission set on the containing bucket.

Syntax

```
DELETE /{bucket}/{object} HTTP/1.1
```

Get Object

Retrieves an object from a bucket within RADOS.

Syntax

```
GET /{bucket}/{object} HTTP/1.1
```

Request Headers

Name	Description	Valid Values	Re-quired
range	The range of the object to retrieve.	Range: bytes=beginbyte-endbyte	No
if-modified-since	Gets only if modified since the timestamp.	Timestamp	No
if-unmodified-since	Gets only if not modified since the timestamp.	Timestamp	No
if-match	Gets only if object ETag matches ETag.	Entity Tag	No
if-none-match	Gets only if object ETag matches ETag.	Entity Tag	No

Response Headers

Name	Description
Content-Range	Data range, will only be returned if the range header field was specified in the request

Get Object Info

Returns information about object. This request will return the same header information as with the Get Object request, but will include the metadata only, not the object data payload.

Syntax

```
HEAD /{bucket}/{object} HTTP/1.1
```

Request Headers

Name	Description	Valid Values	Re-quired
range	The range of the object to retrieve.	Range: bytes=beginbyte-endbyte	No
if-modified-since	Gets only if modified since the timestamp.	Timestamp	No
if-unmodified-since	Gets only if not modified since the timestamp.	Timestamp	No
if-match	Gets only if object ETag matches ETag.	Entity Tag	No
if-none-match	Gets only if object ETag matches ETag.	Entity Tag	No

Get Object ACL

Syntax

```
GET /{bucket}/{object}?acl HTTP/1.1
```

Response Entities

Name	Type	Description
`AccessControlPolicy`	Container	A container for the response.
`AccessControlList`	Container	A container for the ACL information.
`Owner`	Container	A container for the object owner's `ID` and `DisplayName`.
`ID`	String	The object owner's ID.
`DisplayName`	String	The object owner's display name.
`Grant`	Container	A container for `Grantee` and `Permission`.
`Grantee`	Container	A container for the `DisplayName` and `ID` of the user receiving a grant o permission.
`Permission`	String	The permission given to the `Grantee` object.

Set Object ACL

Syntax

```
PUT /{bucket}/{object}?acl
```

Request Entities

Name	Type	Description
`AccessControlPolicy`	Container	A container for the response.
`AccessControlList`	Container	A container for the ACL information.
`Owner`	Container	A container for the object owner's `ID` and `DisplayName`.
`ID`	String	The object owner's ID.
`DisplayName`	String	The object owner's display name.
`Grant`	Container	A container for `Grantee` and `Permission`.
`Grantee`	Container	A container for the `DisplayName` and `ID` of the user receiving a grant of permission.
`Permission`	String	The permission given to the `Grantee` object.

Initiate Multi-part Upload

Initiate a multi-part upload process.

Syntax

```
POST /{bucket}/{object}?uploads
```

Request Headers

Name	Description	Valid Values	Re-quired
content-md5	A base64 encoded MD-5 hash of the message.	A string. No defaults or constraints.	No
content-type	A standard MIME type.	Any MIME type. Default: `binary/octet-stream`	No
x-amz-meta-<...>	User metadata. Stored with the object.	A string up to 8kb. No defaults.	No
x-amz-acl	A canned ACL.	`private`, `public-read`, `public-read-write`, `authenticated-read`	No

Response Entities

Name	Type	Description
`InitiatedMultipartUploadsResult`	Container	A container for the results.
`Bucket`	String	The bucket that will receive the object contents.
`Key`	String	The key specified by the `key` request parameter (if any).
`UploadId`	String	The ID specified by the `upload-id` request parameter identifying the multipart upload (if any).

Multipart Upload Part

Syntax

```
PUT /{bucket}/{object}?partNumber=&uploadId= HTTP/1.1
```

HTTP Response The following HTTP response may be returned:

HTTP Status	Status Code	Description
404	NoSuchUpload	Specified upload-id does not match any initiated upload on this object

List Multipart Upload Parts

Syntax

```
GET /{bucket}/{object}?uploadId=123 HTTP/1.1
```

Response Entities

Name	Type	Description
`InitiatedMultipartUploadsResult`	Container	A container for the results.
`Bucket`	String	The bucket that will receive the object contents.
`Key`	String	The key specified by the `key` request parameter (if any).
`UploadId`	String	The ID specified by the `upload-id` request parameter identifying the multipart upload (if any).
`Initiator`	Container	Contains the `ID` and `DisplayName` of the user who initiated the upload.
`ID`	String	The initiator's ID.
`DisplayName`	String	The initiator's display name.
`Owner`	Container	A container for the `ID` and `DisplayName` of the user who owns the uploaded object.
`StorageClass`	String	The method used to store the resulting object. `STANDARD` or `REDUCED_REDUNDANCY`
`PartNumberMarker`	String	The part marker to use in a subsequent request if `IsTruncated` i `true`. Precedes the list.
`NextPartNumberMarker`	String	The next part marker to use in a subsequent request if `IsTruncated` is `true`. The end of the list.
`MaxParts`	Integer	The max parts allowed in the response as specified by the `max-parts` request parameter.
`IsTruncated`	Boolean	If `true`, only a subset of the object's upload contents were returne
`Part`	Container	A container for `Key`, `Part`, `InitiatorOwner`, `StorageClass`, and `Initiated` elements.
`PartNumber`	Integer	The identification number of the part.
`ETag`	String	The part's entity tag.
`Size`	Integer	The size of the uploaded part.

Complete Multipart Upload

Assembles uploaded parts and creates a new object, thereby completing a multipart upload.

Syntax

```
POST /{bucket}/{object}?uploadId= HTTP/1.1
```

Request Entities

Name	Type	Description	Required
`CompleteMultipartUpload`	Container	A container consisting of one or more parts.	Yes
`Part`	Container	A container for the `PartNumber` and `ETag`.	Yes
`PartNumber`	Integer	The identifier of the part.	Yes
`ETag`	String	The part's entity tag.	Yes

Response Entities

Name	Type	Description
CompleteMultipartUploadResult	Container	A container for the response.
Location	URI	The resource identifier (path) of the new object.
Bucket	String	The name of the bucket that contains the new object.
Key	String	The object's key.
ETag	String	The entity tag of the new object.

Abort Multipart Upload

Syntax

```
DELETE /{bucket}/{object}?uploadId= HTTP/1.1
```

C++ S3 Examples

Setup

The following contains includes and globals that will be used in later examples:

```
#include "libs3.h"
#include <stdlib.h>
#include <iostream>
#include <fstream>

const char access_key[] = "ACCESS_KEY";
const char secret_key[] = "SECRET_KEY";
const char host[] = "HOST";
const char sample_bucket[] = "sample_bucket";
const char sample_key[] = "hello.txt";
const char sample_file[] = "resource/hello.txt";

S3BucketContext bucketContext =
{
        host,
        sample_bucket,
        S3ProtocolHTTP,
        S3UriStylePath,
        access_key,
        secret_key
};

S3Status responsePropertiesCallback(
                const S3ResponseProperties *properties,
                void *callbackData)
{
        return S3StatusOK;
}

static void responseCompleteCallback(
                S3Status status,
                const S3ErrorDetails *error,
                void *callbackData)
{
        return;
}

S3ResponseHandler responseHandler =
{
        &responsePropertiesCallback,
        &responseCompleteCallback
};
```

Creating (and Closing) a Connection

This creates a connection so that you can interact with the server.

```
S3_initialize("s3", S3_INIT_ALL, host);
// Do stuff...
S3_deinitialize();
```

Listing Owned Buckets

This gets a list of Buckets that you own. This also prints out the bucket name, owner ID, and display name for each bucket.

```
static S3Status listServiceCallback(
                const char *ownerId,
                const char *ownerDisplayName,
                const char *bucketName,
                int64_t creationDate, void *callbackData)
{
        bool *header_printed = (bool*) callbackData;
        if (!*header_printed) {
                *header_printed = true;
                printf("%-22s", "       Bucket");
                printf("  %-20s  %-12s", "     Owner ID", "Display Name");
                printf("\n");
                printf("----------------------");
                printf("  ---------------------" "  ------------");
                printf("\n");
        }

        printf("%-22s", bucketName);
        printf("  %-20s  %-12s", ownerId ? ownerId : "", ownerDisplayName ? ownerDisplayName : "");
        printf("\n");

        return S3StatusOK;
}

S3ListServiceHandler listServiceHandler =
{
        responseHandler,
        &listServiceCallback
};
bool header_printed = false;
S3_list_service(S3ProtocolHTTP, access_key, secret_key, host, 0, &listServiceHandler, &header_printe
```

Creating a Bucket

This creates a new bucket.

```
S3_create_bucket(S3ProtocolHTTP, access_key, secret_key, host, sample_bucket, S3CannedAclPrivate, NU
```

Listing a Bucket's Content

This gets a list of objects in the bucket. This also prints out each object's name, the file size, and last modified date.

```
static S3Status listBucketCallback(
                int isTruncated,
                const char *nextMarker,
                int contentsCount,
                const S3ListBucketContent *contents,
                int commonPrefixesCount,
                const char **commonPrefixes,
                void *callbackData)
{
        printf("%-22s", "      Object Name");
        printf("  %-5s  %-20s", "Size", "   Last Modified");
        printf("\n");
        printf("----------------------");
        printf("  -----" "  --------------------");
        printf("\n");

    for (int i = 0; i < contentsCount; i++) {
        char timebuf[256];
                char sizebuf[16];
        const S3ListBucketContent *content = &(contents[i]);
                time_t t = (time_t) content->lastModified;

                strftime(timebuf, sizeof(timebuf), "%Y-%m-%dT%H:%M:%SZ", gmtime(&t));
                sprintf(sizebuf, "%5llu", (unsigned long long) content->size);
                printf("%-22s  %s  %s\n", content->key, sizebuf, timebuf);
    }

    return S3StatusOK;
}

S3ListBucketHandler listBucketHandler =
{
        responseHandler,
        &listBucketCallback
};
S3_list_bucket(&bucketContext, NULL, NULL, NULL, 0, NULL, &listBucketHandler, NULL);
```

The output will look something like this:

```
myphoto1.jpg 251262  2011-08-08T21:35:48.000Z
myphoto2.jpg 262518  2011-08-08T21:38:01.000Z
```

Deleting a Bucket

Note: The Bucket must be empty! Otherwise it won't work!

```
S3_delete_bucket(S3ProtocolHTTP, S3UriStylePath, access_key, secret_key, host, sample_bucket, NULL, &
```

Creating an Object (from a file)

This creates a file `hello.txt`.

```
#include <sys/stat.h>
typedef struct put_object_callback_data
```

```
{
    FILE *infile;
    uint64_t contentLength;
} put_object_callback_data;


static int putObjectDataCallback(int bufferSize, char *buffer, void *callbackData)
{
    put_object_callback_data *data = (put_object_callback_data *) callbackData;

    int ret = 0;

    if (data->contentLength) {
        int toRead = ((data->contentLength > (unsigned) bufferSize) ? (unsigned) bufferSize : data->
                ret = fread(buffer, 1, toRead, data->infile);
    }
    data->contentLength -= ret;
    return ret;
}

put_object_callback_data data;
struct stat statbuf;
if (stat(sample_file, &statbuf) == -1) {
        fprintf(stderr, "\nERROR: Failed to stat file %s: ", sample_file);
        perror(0);
        exit(-1);
}

int contentLength = statbuf.st_size;
data.contentLength = contentLength;

if (!(data.infile = fopen(sample_file, "r"))) {
        fprintf(stderr, "\nERROR: Failed to open input file %s: ", sample_file);
        perror(0);
        exit(-1);
}

S3PutObjectHandler putObjectHandler =
{
        responseHandler,
        &putObjectDataCallback
};

S3_put_object(&bucketContext, sample_key, contentLength, NULL, NULL, &putObjectHandler, &data);
```

Download an Object (to a file)

This downloads a file and prints the contents.

```
static S3Status getObjectDataCallback(int bufferSize, const char *buffer, void *callbackData)
{
        FILE *outfile = (FILE *) callbackData;
        size_t wrote = fwrite(buffer, 1, bufferSize, outfile);
        return ((wrote < (size_t) bufferSize) ? S3StatusAbortedByCallback : S3StatusOK);
}

S3GetObjectHandler getObjectHandler =
{
        responseHandler,
        &getObjectDataCallback
};
FILE *outfile = stdout;
S3_get_object(&bucketContext, sample_key, NULL, 0, 0, NULL, &getObjectHandler, outfile);
```

Delete an Object

This deletes an object.

```
S3ResponseHandler deleteResponseHandler =
{
        0,
        &responseCompleteCallback
};
S3_delete_object(&bucketContext, sample_key, 0, &deleteResponseHandler, 0);
```

Change an Object's ACL

This changes an object's ACL to grant full control to another user.

```
#include <string.h>
char ownerId[] = "owner";
char ownerDisplayName[] = "owner";
char granteeId[] = "grantee";
char granteeDisplayName[] = "grantee";

S3AclGrant grants[] = {
        {
                S3GranteeTypeCanonicalUser,
                {{}},
                S3PermissionFullControl
        },
        {
                S3GranteeTypeCanonicalUser,
                {{}},
                S3PermissionReadACP
        },
        {
                S3GranteeTypeAllUsers,
                {{}},
                S3PermissionRead
        }
};

strncpy(grants[0].grantee.canonicalUser.id, ownerId, S3_MAX_GRANTEE_USER_ID_SIZE);
strncpy(grants[0].grantee.canonicalUser.displayName, ownerDisplayName, S3_MAX_GRANTEE_DISPLAY_NAME_SI

strncpy(grants[1].grantee.canonicalUser.id, granteeId, S3_MAX_GRANTEE_USER_ID_SIZE);
strncpy(grants[1].grantee.canonicalUser.displayName, granteeDisplayName, S3_MAX_GRANTEE_DISPLAY_NAME_

S3_set_acl(&bucketContext, sample_key, ownerId, ownerDisplayName, 3, grants, 0, &responseHandler, 0);
```

Generate Object Download URL (signed)

This generates a signed download URL that will be valid for 5 minutes.

```
#include <time.h>
char buffer[S3_MAX_AUTHENTICATED_QUERY_STRING_SIZE];
int64_t expires = time(NULL) + 60 * 5; // Current time + 5 minutes

S3_generate_authenticated_query_string(buffer, &bucketContext, sample_key, expires, NULL);
```

C# S3 Examples

Creating a Connection

This creates a connection so that you can interact with the server.

```
using System;
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;

string accessKey = "put your access key here!";
string secretKey = "put your secret key here!";

AmazonS3Config config = new AmazonS3Config();
config.ServiceURL = "objects.dreamhost.com";

AmazonS3 client = Amazon.AWSClientFactory.CreateAmazonS3Client(
        accessKey,
        secretKey,
        config
        );
```

Listing Owned Buckets

This gets a list of Buckets that you own. This also prints out the bucket name and creation date of each bucket.

```
ListBucketResponse response = client.ListBuckets();
foreach (S3Bucket b in response.Buckets)
{
        Console.WriteLine("{0}\t{1}", b.BucketName, b.CreationDate);
}
```

The output will look something like this:

```
mahbuckat1   2011-04-21T18:05:39.000Z
mahbuckat2   2011-04-21T18:05:48.000Z
mahbuckat3   2011-04-21T18:07:18.000Z
```

Creating a Bucket

This creates a new bucket called `my-new-bucket`

```
PutBucketRequest request = new PutBucketRequest();
request.BucketName = "my-new-bucket";
client.PutBucket(request);
```

Listing a Bucket's Content

This gets a list of objects in the bucket. This also prints out each object's name, the file size, and last modified date.

```
ListObjectsRequest request = new ListObjectsRequest();
request.BucketName = "my-new-bucket";
ListObjectsResponse response = client.ListObjects(request);
foreach (S3Object o in response.S3Objects)
{
        Console.WriteLine("{0}\t{1}\t{2}", o.Key, o.Size, o.LastModified);
}
```

The output will look something like this:

```
myphoto1.jpg 251262  2011-08-08T21:35:48.000Z
myphoto2.jpg 262518  2011-08-08T21:38:01.000Z
```

Deleting a Bucket

Note: The Bucket must be empty! Otherwise it won't work!

```
DeleteBucketRequest request = new DeleteBucketRequest();
request.BucketName = "my-new-bucket";
client.DeleteBucket(request);
```

Forced Delete for Non-empty Buckets

Attention: not available

Creating an Object

This creates a file `hello.txt` with the string `"Hello World!"`

```
PutObjectRequest request = new PutObjectRequest();
request.Bucket      = "my-new-bucket";
request.Key         = "hello.txt";
request.ContentType = "text/plain";
request.ContentBody = "Hello World!";
client.PutObject(request);
```

Change an Object's ACL

This makes the object `hello.txt` to be publicly readable, and `secret_plans.txt` to be private.

```
SetACLRequest request = new SetACLRequest();
request.BucketName = "my-new-bucket";
request.Key        = "hello.txt";
request.CannedACL  = S3CannedACL.PublicRead;
client.SetACL(request);

SetACLRequest request2 = new SetACLRequest();
request2.BucketName = "my-new-bucket";
request2.Key        = "secret_plans.txt";
request2.CannedACL  = S3CannedACL.Private;
client.SetACL(request2);
```

Download an Object (to a file)

This downloads the object `perl_poetry.pdf` and saves it in `C:\Users\larry\Documents`

```
GetObjectRequest request = new GetObjectRequest();
request.BucketName = "my-new-bucket";
request.Key        = "perl_poetry.pdf"
GetObjectResponse response = client.GetObject(request);
response.WriteResponseStreamToFile("C:\\Users\\larry\\Documents\\perl_poetry.pdf");
```

Delete an Object

This deletes the object `goodbye.txt`

```
DeleteObjectRequest request = new DeleteObjectRequest();
request.BucketName = "my-new-bucket";
request.Key        = "goodbye.txt";
client.DeleteObject(request);
```

Generate Object Download URLs (signed and unsigned)

This generates an unsigned download URL for `hello.txt`. This works because we made `hello.txt` public by setting the ACL above. This then generates a signed download URL for `secret_plans.txt` that will work for 1 hour. Signed download URLs will work for the time period even if the object is private (when the time period is up, the URL will stop working).

Note: The C# S3 Library does not have a method for generating unsigned URLs, so the following example only shows generating signed URLs.

```
GetPreSignedUrlRequest request = new GetPreSignedUrlRequest();
request.BucketName = "my-bucket-name";
request.Key        = "secret_plans.txt";
request.Expires    = DateTime.Now.AddHours(1);
request.Protocol   = Protocol.HTTP;
string url = client.GetPreSignedURL(request);
Console.WriteLine(url);
```

The output of this will look something like:

```
http://objects.dreamhost.com/my-bucket-name/secret_plans.txt?Signature=XXXXXXXXXXXXXXXXXXXXXXXXXXX&E
```

Java S3 Examples

Setup

The following examples may require some or all of the following java classes to be imported:

```
import java.io.ByteArrayInputStream;
import java.io.File;
import java.util.List;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.util.StringUtils;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.Bucket;
import com.amazonaws.services.s3.model.CannedAccessControlList;
import com.amazonaws.services.s3.model.GeneratePresignedUrlRequest;
import com.amazonaws.services.s3.model.GetObjectRequest;
import com.amazonaws.services.s3.model.ObjectListing;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.S3ObjectSummary;
```

If you are just testing the Ceph Object Storage services, consider using HTTP protocol instead of HTTPS protocol.

First, import the `ClientConfiguration` and `Protocol` classes.

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.Protocol;
```

Then, define the client configuration, and add the client configuration as an argument for the S3 client.

```
AWSCredentials credentials = new BasicAWSCredentials(accessKey, secretKey);

ClientConfiguration clientConfig = new ClientConfiguration();
clientConfig.setProtocol(Protocol.HTTP);

AmazonS3 conn = new AmazonS3Client(credentials, clientConfig);
conn.setEndpoint("endpoint.com");
```

Creating a Connection

This creates a connection so that you can interact with the server.

```
String accessKey = "insert your access key here!";
String secretKey = "insert your secret key here!";

AWSCredentials credentials = new BasicAWSCredentials(accessKey, secretKey);
AmazonS3 conn = new AmazonS3Client(credentials);
conn.setEndpoint("objects.dreamhost.com");
```

Listing Owned Buckets

This gets a list of Buckets that you own. This also prints out the bucket name and creation date of each bucket.

```
List<Bucket> buckets = conn.listBuckets();
for (Bucket bucket : buckets) {
        System.out.println(bucket.getName() + "\t" +
                StringUtils.fromDate(bucket.getCreationDate()));
}
```

The output will look something like this:

```
mahbuckat1   2011-04-21T18:05:39.000Z
mahbuckat2   2011-04-21T18:05:48.000Z
mahbuckat3   2011-04-21T18:07:18.000Z
```

Creating a Bucket

This creates a new bucket called `my-new-bucket`

```
Bucket bucket = conn.createBucket("my-new-bucket");
```

Listing a Bucket's Content

This gets a list of objects in the bucket. This also prints out each object's name, the file size, and last modified date.

```
ObjectListing objects = conn.listObjects(bucket.getName());
do {
        for (S3ObjectSummary objectSummary : objects.getObjectSummaries()) {
                System.out.println(objectSummary.getKey() + "\t" +
                        ObjectSummary.getSize() + "\t" +
                        StringUtils.fromDate(objectSummary.getLastModified()));
        }
        objects = conn.listNextBatchOfObjects(objects);
} while (objects.isTruncated());
```

The output will look something like this:

```
myphoto1.jpg 251262  2011-08-08T21:35:48.000Z
myphoto2.jpg 262518  2011-08-08T21:38:01.000Z
```

Deleting a Bucket

Note: The Bucket must be empty! Otherwise it won't work!

```
conn.deleteBucket(bucket.getName());
```

Forced Delete for Non-empty Buckets

Attention: not available

Creating an Object

This creates a file `hello.txt` with the string `"Hello World!"`

```
ByteArrayInputStream input = new ByteArrayInputStream("Hello World!".getBytes());
conn.putObject(bucket.getName(), "hello.txt", input, new ObjectMetadata());
```

Change an Object's ACL

This makes the object `hello.txt` to be publicly readable, and `secret_plans.txt` to be private.

```
conn.setObjectAcl(bucket.getName(), "hello.txt", CannedAccessControlList.PublicRead);
conn.setObjectAcl(bucket.getName(), "secret_plans.txt", CannedAccessControlList.Private);
```

Download an Object (to a file)

This downloads the object `perl_poetry.pdf` and saves it in `/home/larry/documents`

```
conn.getObject(
        new GetObjectRequest(bucket.getName(), "perl_poetry.pdf"),
        new File("/home/larry/documents/perl_poetry.pdf")
);
```

Delete an Object

This deletes the object `goodbye.txt`

```
conn.deleteObject(bucket.getName(), "goodbye.txt");
```

Generate Object Download URLs (signed and unsigned)

This generates an unsigned download URL for `hello.txt`. This works because we made `hello.txt` public by setting the ACL above. This then generates a signed download URL for `secret_plans.txt` that will work for 1 hour. Signed download URLs will work for the time period even if the object is private (when the time period is up, the URL will stop working).

Note: The java library does not have a method for generating unsigned URLs, so the example below just generates a signed URL.

```
GeneratePresignedUrlRequest request = new GeneratePresignedUrlRequest(bucket.getName(), "secret_plans
System.out.println(conn.generatePresignedUrl(request));
```

The output will look something like this:

```
https://my-bucket-name.objects.dreamhost.com/secret_plans.txt?Signature=XXXXXXXXXXXXXXXXXXXXXXXXXXX&
```

Perl S3 Examples

Creating a Connection

This creates a connection so that you can interact with the server.

```
use Amazon::S3;
my $access_key = 'put your access key here!';
my $secret_key = 'put your secret key here!';

my $conn = Amazon::S3->new({
        aws_access_key_id     => $access_key,
        aws_secret_access_key => $secret_key,
        host                  => 'objects.dreamhost.com',
        secure                => 1,
        retry                 => 1,
});
```

Listing Owned Buckets

This gets a list of Amazon::S3::Bucket objects that you own. We'll also print out the bucket name and creation date of each bucket.

```
my @buckets = @{$conn->buckets->{buckets} || []};
foreach my $bucket (@buckets) {
        print $bucket->bucket . "\t" . $bucket->creation_date . "\n";
}
```

The output will look something like this:

```
mahbuckat1   2011-04-21T18:05:39.000Z
mahbuckat2   2011-04-21T18:05:48.000Z
mahbuckat3   2011-04-21T18:07:18.000Z
```

Creating a Bucket

This creates a new bucket called `my-new-bucket`

```
my $bucket = $conn->add_bucket({ bucket => 'my-new-bucket' });
```

Listing a Bucket's Content

This gets a list of hashes with info about each object in the bucket. We'll also print out each object's name, the file size, and last modified date.

```
my @keys = @{$bucket->list_all->{keys} || []};
foreach my $key (@keys) {
        print "$key->{key}\t$key->{size}\t$key->{last_modified}\n";
}
```

The output will look something like this:

```
myphoto1.jpg 251262  2011-08-08T21:35:48.000Z
myphoto2.jpg 262518  2011-08-08T21:38:01.000Z
```

Deleting a Bucket

Note: The Bucket must be empty! Otherwise it won't work!

```
$conn->delete_bucket($bucket);
```

Forced Delete for Non-empty Buckets

Attention: not available in the Amazon::S3 perl module

Creating an Object

This creates a file `hello.txt` with the string `"Hello World!"`

```
$bucket->add_key(
        'hello.txt', 'Hello World!',
        { content_type => 'text/plain' },
);
```

Change an Object's ACL

This makes the object `hello.txt` to be publicly readable and `secret_plans.txt` to be private.

```
$bucket->set_acl({
        key       => 'hello.txt',
        acl_short => 'public-read',
});
$bucket->set_acl({
        key       => 'secret_plans.txt',
        acl_short => 'private',
});
```

Download an Object (to a file)

This downloads the object `perl_poetry.pdf` and saves it in `/home/larry/documents/`

```
$bucket->get_key_filename('perl_poetry.pdf', undef,
        '/home/larry/documents/perl_poetry.pdf');
```

Delete an Object

This deletes the object `goodbye.txt`

```
$bucket->delete_key('goodbye.txt');
```

Generate Object Download URLs (signed and unsigned)

This generates an unsigned download URL for `hello.txt`. This works because we made `hello.txt` public by setting the ACL above. Then this generates a signed download URL for `secret_plans.txt` that will work for 1 hour. Signed download URLs will work for the time period even if the object is private (when the time period is up, the URL will stop working).

Note: The Amazon::S3 module does not have a way to generate download URLs, so we're going to be using another module instead. Unfortunately, most modules for generating these URLs assume that you are using Amazon, so we've had to go with using a more obscure module, Muck::FS::S3. This should be the same as Amazon's sample S3 perl module, but this sample module is not in CPAN. So, you can either use CPAN to install Muck::FS::S3, or install Amazon's sample S3 module manually. If you go the manual route, you can remove `Muck::FS::` from the example below.

```
use Muck::FS::S3::QueryStringAuthGenerator;
my $generator = Muck::FS::S3::QueryStringAuthGenerator->new(
        $access_key,
        $secret_key,
        0, # 0 means use 'http'. set this to 1 for 'https'
        'objects.dreamhost.com',
);

my $hello_url = $generator->make_bare_url($bucket->bucket, 'hello.txt');
print $hello_url . "\n";

$generator->expires_in(3600); # 1 hour = 3600 seconds
my $plans_url = $generator->get($bucket->bucket, 'secret_plans.txt');
print $plans_url . "\n";
```

The output will look something like this:

```
http://objects.dreamhost.com:80/my-bucket-name/hello.txt
http://objects.dreamhost.com:80/my-bucket-name/secret_plans.txt?Signature=XXXXXXXXXXXXXXXXXXXXXXXXXX
```

PHP S3 Examples

Creating a Connection

This creates a connection so that you can interact with the server.

```
<?php
define('AWS_KEY', 'place access key here');
define('AWS_SECRET_KEY', 'place secret key here');
define('AWS_CANONICAL_ID', 'your DHO Username');
define('AWS_CANONICAL_NAME', 'Also your DHO Username!');
$HOST = 'objects.dreamhost.com';

// require the amazon sdk for php library
require_once 'AWSSDKforPHP/sdk.class.php';

// Instantiate the S3 class and point it at the desired host
$Connection = new AmazonS3(array(
        'key' => AWS_KEY,
        'secret' => AWS_SECRET_KEY,
        'canonical_id' => AWS_CANONICAL_ID,
        'canonical_name' => AWS_CANONICAL_NAME,
));
$Connection->set_hostname($HOST);
$Connection->allow_hostname_override(false);

// Set the S3 class to use objects.dreamhost.com/bucket
// instead of bucket.objects.dreamhost.com
$Connection->enable_path_style();
```

Listing Owned Buckets

This gets a list of CFSimpleXML objects representing buckets that you own. This also prints out the bucket name and creation date of each bucket.

```
<?php
$ListResponse = $Connection->list_buckets();
$Buckets = $ListResponse->body->Buckets->Bucket;
foreach ($Buckets as $Bucket) {
        echo $Bucket->Name . "\t" . $Bucket->CreationDate . "\n";
}
```

The output will look something like this:

```
mahbuckat1   2011-04-21T18:05:39.000Z
mahbuckat2   2011-04-21T18:05:48.000Z
mahbuckat3   2011-04-21T18:07:18.000Z
```

Creating a Bucket

This creates a new bucket called `my-new-bucket` and returns a `CFResponse` object.

Note: This command requires a region as the second argument, so we use `AmazonS3::REGION_US_E1`, because this constant is `''`

```
<?php
$Connection->create_bucket('my-new-bucket', AmazonS3::REGION_US_E1);
```

List a Bucket's Content

This gets an array of `CFSimpleXML` objects representing the objects in the bucket. This then prints out each object's name, the file size, and last modified date.

```
<?php
$ObjectsListResponse = $Connection->list_objects($bucketname);
$Objects = $ObjectsListResponse->body->Contents;
foreach ($Objects as $Object) {
        echo $Object->Key . "\t" . $Object->Size . "\t" . $Object->LastModified . "\n";
}
```

Note: If there are more than 1000 objects in this bucket, you need to check $ObjectListResponse->body->isTruncated and run again with the name of the last key listed. Keep doing this until isTruncated is not true.

The output will look something like this if the bucket has some files:

```
myphoto1.jpg 251262  2011-08-08T21:35:48.000Z
myphoto2.jpg 262518  2011-08-08T21:38:01.000Z
```

Deleting a Bucket

This deletes the bucket called `my-old-bucket` and returns a `CFResponse` object

Note: The Bucket must be empty! Otherwise it won't work!

```php
<?php
$Connection->delete_bucket('my-old-bucket');
```

Forced Delte for Non-empty Buckets

This will delete the bucket even if it is not empty.

```php
<?php
$Connection->delete_bucket('my-old-bucket', 1);
```

Creating an Object

This creates an object `hello.txt` with the string `"Hello World!"`

```php
<?php
$Connection->create_object('my-bucket-name', 'hello.txt', array(
        'body' => "Hello World!",
));
```

Change an Object's ACL

This makes the object `hello.txt` to be publicly readable and `secret_plans.txt` to be private.

```php
<?php
$Connection->set_object_acl('my-bucket-name', 'hello.txt', AmazonS3::ACL_PUBLIC);
$Connection->set_object_acl('my-bucket-name', 'secret_plans.txt', AmazonS3::ACL_PRIVATE);
```

Delete an Object

This deletes the object `goodbye.txt`

```php
<?php
$Connection->delete_object('my-bucket-name', 'goodbye.txt');
```

Download an Object (to a file)

This downloads the object `poetry.pdf` and saves it in `/home/larry/documents/`

```php
<?php
$FileHandle = fopen('/home/larry/documents/poetry.pdf', 'w+');
$Connection->get_object('my-bucket-name', 'poetry.pdf', array(
        'fileDownload' => $FileHandle,
));
```

Generate Object Download URLs (signed and unsigned)

This generates an unsigned download URL for `hello.txt`. This works because we made `hello.txt` public by setting the ACL above. This then generates a signed download URL for `secret_plans.txt` that will work for 1 hour. Signed download URLs will work for the time period even if the object is private (when the time period is up, the URL will stop working).

```
<?php
my $plans_url = $Connection->get_object_url('my-bucket-name', 'hello.txt');
echo $plans_url . "\n";
my $secret_url = $Connection->get_object_url('my-bucket-name', 'secret_plans.txt', '1 hour');
echo $secret_url . "\n";
```

The output of this will look something like:

```
http://objects.dreamhost.com/my-bucket-name/hello.txt
http://objects.dreamhost.com/my-bucket-name/secret_plans.txt?Signature=XXXXXXXXXXXXXXXXXXXXXXXXXXX&E
```

Python S3 Examples

Creating a Connection

This creates a connection so that you can interact with the server.

```
import boto
import boto.s3.connection
access_key = 'put your access key here!'
secret_key = 'put your secret key here!'

conn = boto.connect_s3(
        aws_access_key_id = access_key,
        aws_secret_access_key = secret_key,
        host = 'objects.dreamhost.com',
        #is_secure=False,               # uncomment if you are not using ssl
        calling_format = boto.s3.connection.OrdinaryCallingFormat(),
        )
```

Listing Owned Buckets

This gets a list of Buckets that you own. This also prints out the bucket name and creation date of each bucket.

```
for bucket in conn.get_all_buckets():
        print "{name}\t{created}".format(
                name = bucket.name,
                created = bucket.creation_date,
        )
```

The output will look something like this:

```
mahbuckat1   2011-04-21T18:05:39.000Z
mahbuckat2   2011-04-21T18:05:48.000Z
mahbuckat3   2011-04-21T18:07:18.000Z
```

Creating a Bucket

This creates a new bucket called `my-new-bucket`

```
bucket = conn.create_bucket('my-new-bucket')
```

Listing a Bucket's Content

This gets a list of objects in the bucket. This also prints out each object's name, the file size, and last modified date.

```
for key in bucket.list():
        print "{name}\t{size}\t{modified}".format(
                name = key.name,
                size = key.size,
                modified = key.last_modified,
                )
```

The output will look something like this:

```
myphoto1.jpg 251262  2011-08-08T21:35:48.000Z
myphoto2.jpg 262518  2011-08-08T21:38:01.000Z
```

Deleting a Bucket

Note: The Bucket must be empty! Otherwise it won't work!

```
conn.delete_bucket(bucket.name)
```

Forced Delete for Non-empty Buckets

Attention: not available in python

Creating an Object

This creates a file `hello.txt` with the string `"Hello World!"`

```
key = bucket.new_key('hello.txt')
key.set_contents_from_string('Hello World!')
```

Change an Object's ACL

This makes the object `hello.txt` to be publicly readable, and `secret_plans.txt` to be private.

```
hello_key = bucket.get_key('hello.txt')
hello_key.set_canned_acl('public-read')
plans_key = bucket.get_key('secret_plans.txt')
plans_key.set_canned_acl('private')
```

Download an Object (to a file)

This downloads the object `perl_poetry.pdf` and saves it in `/home/larry/documents/`

```
key = bucket.get_key('perl_poetry.pdf')
key.get_contents_to_filename('/home/larry/documents/perl_poetry.pdf')
```

Delete an Object

This deletes the object `goodbye.txt`

```
bucket.delete_key('goodbye.txt')
```

Generate Object Download URLs (signed and unsigned)

This generates an unsigned download URL for `hello.txt`. This works because we made `hello.txt` public by setting the ACL above. This then generates a signed download URL for `secret_plans.txt` that will work for 1 hour. Signed download URLs will work for the time period even if the object is private (when the time period is up, the URL will stop working).

```
hello_key = bucket.get_key('hello.txt')
hello_url = hello_key.generate_url(0, query_auth=False, force_http=True)
print hello_url

plans_key = bucket.get_key('secret_plans.txt')
plans_url = plans_key.generate_url(3600, query_auth=True, force_http=True)
print plans_url
```

The output of this will look something like:

```
http://objects.dreamhost.com/my-bucket-name/hello.txt
http://objects.dreamhost.com/my-bucket-name/secret_plans.txt?Signature=XXXXXXXXXXXXXXXXXXXXXXXXXXX&E
```

Ruby S3 Examples

Creating a Connection

This creates a connection so that you can interact with the server.

```
AWS::S3::Base.establish_connection!(
        :server            => 'objects.dreamhost.com',
        :use_ssl           => true,
        :access_key_id     => 'my-access-key',
        :secret_access_key => 'my-secret-key'
)
```

Listing Owned Buckets

This gets a list of AWS::S3::Bucket objects that you own. This also prints out the bucket name and creation date of each bucket.

```
AWS::S3::Service.buckets.each do |bucket|
        puts "#{bucket.name}\t#{bucket.creation_date}"
end
```

The output will look something like this:

```
mahbuckat1   2011-04-21T18:05:39.000Z
mahbuckat2   2011-04-21T18:05:48.000Z
mahbuckat3   2011-04-21T18:07:18.000Z
```

Creating a Bucket

This creates a new bucket called `my-new-bucket`

```
AWS::S3::Bucket.create('my-new-bucket')
```

Listing a Bucket's Content

This gets a list of hashes with the contents of each object This also prints out each object's name, the file size, and last modified date.

```
new_bucket = AWS::S3::Bucket.find('my-new-bucket')
new_bucket.each do |object|
        puts "#{object.key}\t#{object.about['content-length']}\t#{object.about['last-modified']}"
end
```

The output will look something like this if the bucket has some files:

```
myphoto1.jpg 251262  2011-08-08T21:35:48.000Z
myphoto2.jpg 262518  2011-08-08T21:38:01.000Z
```

Deleting a Bucket

Note: The Bucket must be empty! Otherwise it won't work!

```
AWS::S3::Bucket.delete('my-new-bucket')
```

Forced Delete for Non-empty Buckets

```
AWS::S3::Bucket.delete('my-new-bucket', :force => true)
```

Creating an Object

This creates a file `hello.txt` with the string `"Hello World!"`

```
AWS::S3::S3Object.store(
        'hello.txt',
        'Hello World!',
        'my-new-bucket',
        :content_type => 'text/plain'
)
```

Change an Object's ACL

This makes the object `hello.txt` to be publicly readable, and `secret_plans.txt` to be private.

```
policy = AWS::S3::S3Object.acl('hello.txt', 'my-new-bucket')
policy.grants = [ AWS::S3::ACL::Grant.grant(:public_read) ]
AWS::S3::S3Object.acl('hello.txt', 'my-new-bucket', policy)
```

```
policy = AWS::S3::S3Object.acl('secret_plans.txt', 'my-new-bucket')
policy.grants = []
AWS::S3::S3Object.acl('secret_plans.txt', 'my-new-bucket', policy)
```

Download an Object (to a file)

This downloads the object `poetry.pdf` and saves it in `/home/larry/documents/`

```
open('/home/larry/documents/poetry.pdf', 'w') do |file|
        AWS::S3::S3Object.stream('poetry.pdf', 'my-new-bucket') do |chunk|
                file.write(chunk)
        end
end
```

Delete an Object

This deletes the object `goodbye.txt`

```
AWS::S3::S3Object.delete('goodbye.txt', 'my-new-bucket')
```

Generate Object Download URLs (signed and unsigned)

This generates an unsigned download URL for `hello.txt`. This works because we made `hello.txt` public by setting the ACL above. This then generates a signed download URL for `secret_plans.txt` that will work for 1 hour. Signed download URLs will work for the time period even if the object is private (when the time period is up, the URL will stop working).

```
puts AWS::S3::S3Object.url_for(
        'hello.txt',
        'my-new-bucket',
        :authenticated => false
)

puts AWS::S3::S3Object.url_for(
        'secret_plans.txt',
        'my-new-bucket',
        :expires_in => 60 * 60
)
```

The output of this will look something like:

```
http://objects.dreamhost.com/my-bucket-name/hello.txt
http://objects.dreamhost.com/my-bucket-name/secret_plans.txt?Signature=XXXXXXXXXXXXXXXXXXXXXXXXXXX&E
```

7.7.2 Features Support

The following table describes the support status for current Amazon S3 functional features:

Feature	Status	Remarks
List Buckets	Supported	
Delete Bucket	Supported	
Create Bucket	Supported	Different set of canned ACLs
Bucket Lifecycle	Not Supported	
Policy (Buckets, Objects)	Not Supported	ACLs are supported
Bucket Website	Not Supported	
Bucket ACLs (Get, Put)	Supported	Different set of canned ACLs
Bucket Location	Supported	
Bucket Notification	Not Supported	
Bucket Object Versions	Not Supported	
Get Bucket Info (HEAD)	Supported	
Bucket Request Payment	Not Supported	
Put Object	Supported	
Delete Object	Supported	
Get Object	Supported	
Object ACLs (Get, Put)	Supported	
Get Object Info (HEAD)	Supported	
POST Object	Supported	
Copy Object	Supported	
Multipart Uploads	Supported	(missing Copy Part)

7.7.3 Unsupported Header Fields

The following common request header fields are not supported:

Name	Type
x-amz-security-token	Request
Server	Response
x-amz-delete-marker	Response
x-amz-id-2	Response
x-amz-request-id	Response
x-amz-version-id	Response

7.8 Ceph Object Gateway Swift API

Ceph supports a RESTful API that is compatible with the basic data access model of the Swift API.

7.8.1 API

Authentication

Swift API requests that require authentication must contain an `X-Storage-Token` authentication token in the request header. The token may be retrieved from RADOS Gateway, or from another authenticator. To obtain a token from RADOS Gateway, you must create a user. For example:

```
sudo radosgw-admin user create --uid="{username}" --display-name="{Display Name}"
```

For details on RADOS Gateway administration, see radosgw-admin.

Auth Get

To authenticate a user, make a request containing an `X-Auth-User` and a `X-Auth-Key` in the header.

Syntax

```
GET /auth HTTP/1.1
Host: swift.radosgwhost.com
X-Auth-User: johndoe
X-Auth-Key: R7UUOLFDI2ZI9PRCQ53K
```

Request Headers `X-Auth-User`

Description The key RADOS GW username to authenticate.

Type String

Required Yes

`X-Auth-Key`

Description The key associated to a RADOS GW username.

Type String

Required Yes

Response Headers The response from the server should include an `X-Auth-Token` value. The response may also contain a `X-Storage-Url` that provides the `{api version}/{account}` prefix that is specified in other requests throughout the API documentation.

`X-Storage-Token`

Description The authorization token for the `X-Auth-User` specified in the request.

Type String

`X-Storage-Url`

Description The URL and `{api version}/{account}` path for the user.

Type String

A typical response looks like this:

```
HTTP/1.1 204 No Content
Date: Mon, 16 Jul 2012 11:05:33 GMT
Server: swift
X-Storage-Url: https://swift.radosgwhost.com/v1/ACCT-12345
X-Auth-Token: UOlCCC8TahFKlWuv9DB09TWHF0nDjpPElha0kAa
Content-Length: 0
Content-Type: text/plain; charset=UTF-8
```

Service Operations

To retrieve data about our Swift-compatible service, you may execute `GET` requests using the `X-Storage-Url` value retrieved during authentication.

List Containers

A GET request that specifies the API version and the account will return a list of containers for a particular user account. Since the request returns a particular user's containers, the request requires an authentication token. The request cannot be made anonymously.

Syntax

```
GET /{api version}/{account} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

Request Parameters `limit`

Description Limits the number of results to the specified value.

Type Integer

Required No

`format`

Description Defines the format of the result.

Type String

Valid Values `json | xml`

Required No

`marker`

Description Returns a list of results greater than the marker value.

Type String

Required No

Response Entities The response contains a list of containers, or returns with an HTTP 204 response code

`account`

Description A list for account information.

Type Container

`container`

Description The list of containers.

Type Container

`name`

Description The name of a container.

Type String

`bytes`

Description The size of the container.

Type Integer

Container Operations

A container is a mechanism for storing data objects. An account may have many containers, but container names must be unique. This API enables a client to create a container, set access controls and metadata, retrieve a container's contents, and delete a container. Since this API makes requests related to information in a particular user's account, all requests in this API must be authenticated unless a container's access control is deliberately made publicly accessible (i.e., allows anonymous requests).

Note: The Amazon S3 API uses the term 'bucket' to describe a data container. When you hear someone refer to a 'bucket' within the Swift API, the term 'bucket' may be construed as the equivalent of the term 'container.'

One facet of object storage is that it does not support hierarchical paths or directories. Instead, it supports one level consisting of one or more containers, where each container may have objects. The RADOS Gateway's Swift-compatible API supports the notion of 'pseudo-hierarchical containers,' which is a means of using object naming to emulate a container (or directory) hierarchy without actually implementing one in the storage system. You may name objects with pseudo-hierarchical names (e.g., photos/buildings/empire-state.jpg), but container names cannot contain a forward slash (`/`) character.

Create a Container

To create a new container, make a `PUT` request with the API version, account, and the name of the new container. The container name must be unique, must not contain a forward-slash (/) character, and should be less than 256 bytes. You may include access control headers and metadata headers in the request. The operation is idempotent; that is, if you make a request to create a container that already exists, it will return with a HTTP 202 return code, but will not create another container.

Syntax

```
PUT /{api version}/{account}/{container} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
X-Container-Read: {comma-separated-uids}
X-Container-Write: {comma-separated-uids}
X-Container-Meta-{key}: {value}
```

Headers `X-Container-Read`

Description The user IDs with read permissions for the container.

Type Comma-separated string values of user IDs.

Required No

`X-Container-Write`

Description The user IDs with write permissions for the container.

Type Comma-separated string values of user IDs.

Required No

`X-Container-Meta-{key}`

Description A user-defined meta data key that takes an arbitrary string value.

Type String

Required No

HTTP Response If a container with the same name already exists, and the user is the container owner then the operation will succeed. Otherwise the operation will fail.

`409`

Description The container already exists under a different user's ownership.

Status Code `BucketAlreadyExists`

List a Container's Objects

To list the objects within a container, make a `GET` request with the with the API version, account, and the name of the container. You can specify query parameters to filter the full list, or leave out the parameters to return a list of the first 10,000 object names stored in the container.

Syntax

```
GET /{api version}/{container} HTTP/1.1
     Host: {fqdn}
     X-Auth-Token: {auth-token}
```

Parameters `format`

Description Defines the format of the result.

Type String

Valid Values `json|xml`

Required No

`prefix`

Description Limits the result set to objects beginning with the specified prefix.

Type String

Required No

`marker`

Description Returns a list of results greater than the marker value.

Type String

Required No

`limit`

Description Limits the number of results to the specified value.

Type Integer

Valid Range 0 - 10,000

Required No

`delimiter`

Description The delimiter between the prefix and the rest of the object name.

Type String

Required No

`path`

Description The pseudo-hierarchical path of the objects.

Type String

Required No

Response Entities `container`

Description The container.

Type Container

`object`

Description An object within the container.

Type Container

`name`

Description The name of an object within the container.

Type String

`hash`

Description A hash code of the object's contents.

Type String

`last_modified`

Description The last time the object's contents were modified.

Type Date

`content_type`

Description The type of content within the object.

Type String

Update a Container's ACLs

When a user creates a container, the user has read and write access to the container by default. To allow other users to read a container's contents or write to a container, you must specifically enable the user. You may also specify `*` in the `X-Container-Read` or `X-Container-Write` settings, which effectively enables all users to either read from or write to the container. Setting `*` makes the container public. That is it enables anonymous users to either read from or write to the container.

Syntax

```
POST /{api version}/{account}/{container} HTTP/1.1
Host: {fqdn}
     X-Auth-Token: {auth-token}
     X-Container-Read: *
     X-Container-Write: {uid1}, {uid2}, {uid3}
```

Request Headers `X-Container-Read`

Description The user IDs with read permissions for the container.

Type Comma-separated string values of user IDs.

Required No

`X-Container-Write`

Description The user IDs with write permissions for the container.

Type Comma-separated string values of user IDs.

Required No

Add/Update Container Metadata

To add metadata to a container, make a `POST` request with the API version, account, and container name. You must have write permissions on the container to add or update metadata.

Syntax

```
POST /{api version}/{account}/{container} HTTP/1.1
Host: {fqdn}
     X-Auth-Token: {auth-token}
     X-Container-Meta-Color: red
     X-Container-Meta-Taste: salty
```

Request Headers `X-Container-Meta-{key}`

Description A user-defined meta data key that takes an arbitrary string value.

Type String

Required No

Delete a Container

To delete a container, make a `DELETE` request with the API version, account, and the name of the container. The container must be empty. If you'd like to check if the container is empty, execute a `HEAD` request against the container. Once you've successfully removed the container, you'll be able to reuse the container name.

Syntax

```
DELETE /{api version}/{account}/{container} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

HTTP Response `204`

Description The container was removed.

Status Code `NoContent`

Object Operations

An object is a container for storing data and metadata. A container may have many objects, but the object names must be unique. This API enables a client to create an object, set access controls and metadata, retrieve an object's data and metadata, and delete an object. Since this API makes requests related to information in a particular user's account, all requests in this API must be authenticated unless the container or object's access control is deliberately made publicly accessible (i.e., allows anonymous requests).

Create/Update an Object

To create a new object, make a `PUT` request with the API version, account, container name and the name of the new object. You must have write permission on the container to create or update an object. The object name must be unique within the container. The `PUT` request is not idempotent, so if you do not use a unique name, the request will update the object. However, you may use pseudo-hierarchical syntax in your object name to distinguish it from another object of the same name if it is under a different pseudo-hierarchical directory. You may include access control headers and metadata headers in the request.

Syntax

```
PUT /{api version}/{account}/{container}/{object} HTTP/1.1
     Host: {fqdn}
     X-Auth-Token: {auth-token}
```

Request Headers `ETag`

Description An MD5 hash of the object's contents. Recommended.

Type String

Required No

`Content-Type`

Description The type of content the object contains.

Type String

Required No

`Transfer-Encoding`

Description Indicates whether the object is part of a larger aggregate object.

Type String

Valid Values `chunked`

Required No

Copy an Object

Copying an object allows you to make a server-side copy of an object, so that you don't have to download it and upload it under another container/name. To copy the contents of one object to another object, you may make either a `PUT` request or a `COPY` request with the API version, account, and the container name. For a `PUT` request, use the destination container and object name in the request, and the source container and object in the request header. For a `Copy` request, use the source container and object in the request, and the destination container and object in the request header. You must have write permission on the container to copy an object. The destination object name must be unique within the container. The request is not idempotent, so if you do not use a unique name, the request will update the destination object. However, you may use pseudo-hierarchical syntax in your object name to distinguish the destination object from the source object of the same name if it is under a different pseudo-hierarchical directory. You may include access control headers and metadata headers in the request.

Syntax

```
PUT /{api version}/{account}/{dest-container}/{dest-object} HTTP/1.1
X-Copy-From: {source-container}/{source-object}
Host: {fqdn}
X-Auth-Token: {auth-token}
```

or alternatively:

```
COPY /{api version}/{account}/{source-container}/{source-object} HTTP/1.1
Destination: {dest-container}/{dest-object}
```

Request Headers `X-Copy-From`

Description Used with a `PUT` request to define the source container/object path.

Type String

Required Yes, if using `PUT`

`Destination`

Description Used with a `COPY` request to define the destination container/object path.

Type String

Required Yes, if using `COPY`

`If-Modified-Since`

Description Only copies if modified since the date/time of the source object's `last_modified` attribute.

Type Date

Required No

`If-Unmodified-Since`

Description Only copies if not modified since the date/time of the source object's `last_modified` attribute.

Type Date

Required No

`Copy-If-Match`

Description Copies only if the ETag in the request matches the source object's ETag.

Type ETag.

Required No

`Copy-If-None-Match`

Description Copies only if the ETag in the request does not match the source object's ETag.

Type ETag.

Required No

Delete an Object

To delete an object, make a `DELETE` request with the API version, account, container and object name. You must have write permissions on the container to delete an object within it. Once you've successfully deleted the object, you'll be able to reuse the object name.

Syntax

```
DELETE /{api version}/{account}/{container}/{object} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

Get an Object

To retrieve an object, make a `GET` request with the API version, account, container and object name. You must have read permissions on the container to retrieve an object within it.

Syntax

```
GET /{api version}/{account}/{container}/{object} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

Request Headers `range`

Description To retrieve a subset of an object's contents, you may specify a byte range.

Type Date

Required No

`If-Modified-Since`

Description Only copies if modified since the date/time of the source object's `last_modified` attribute.

Type Date

Required No

`If-Unmodified-Since`

Description Only copies if not modified since the date/time of the source object's `last_modified` attribute.

Type Date

Required No

`Copy-If-Match`

Description Copies only if the ETag in the request matches the source object's ETag.

Type ETag.

Required No

`Copy-If-None-Match`

Description Copies only if the ETag in the request does not match the source object's ETag.

Type ETag.

Required No

Response Headers `Content-Range`

Description The range of the subset of object contents. Returned only if the range header field was specified in the request

Get Object Metadata

To retrieve an object's metadata, make a `HEAD` request with the API version, account, container and object name. You must have read permissions on the container to retrieve metadata from an object within the container. This request returns the same header information as the request for the object itself, but it does not return the object's data.

Syntax

```
HEAD /{api version}/{account}/{container}/{object} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

Add/Update Object Metadata

To add metadata to an object, make a `POST` request with the API version, account, container and object name. You must have write permissions on the parent container to add or update metadata.

Syntax

```
POST /{api version}/{account}/{container}/{object} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

Request Headers `X-Object-Meta-{key}`

Description A user-defined meta data key that takes an arbitrary string value.

Type String

Required No

Temp URL Operations

To allow temporary access (for eg for *GET* requests) to objects without the need to share credentials, temp url functionality is supported by swift endpoint of radosgw. For this functionality, initially the value of *X-Account-Meta-Temp-URL-Key* and optionally *X-Account-Meta-Temp-URL-Key-2* should be set. The Temp URL functionality relies on a HMAC-SHA1 signature against these secret keys.

POST Temp-URL Keys

A `POST` request to the swift account with the required Key will set the secret temp url key for the account against which temporary url access can be provided to accounts. Up to two keys are supported, and signatures are checked against both the keys, if present, so that keys can be rotated without invalidating the temporary urls.

Syntax

```
POST /{api version}/{account} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

Request Headers `X-Account-Meta-Temp-URL-Key`

Description A user-defined key that takes an arbitrary string value.

Type String

Required Yes

`X-Account-Meta-Temp-URL-Key-2`

Description A user-defined key that takes an arbitrary string value.

Type String

Required No

GET Temp-URL Objects

Temporary URL uses a cryptographic HMAC-SHA1 signature, which includes the following elements:

1. The value of the Request method, "GET" for instance
2. The expiry time, in format of seconds since the epoch, ie Unix time
3. The request path starting from "v1" onwards

The above items are normalized with newlines appended between them, and a HMAC is generated using the SHA-1 hashing algorithm against one of the Temp URL Keys posted earlier.

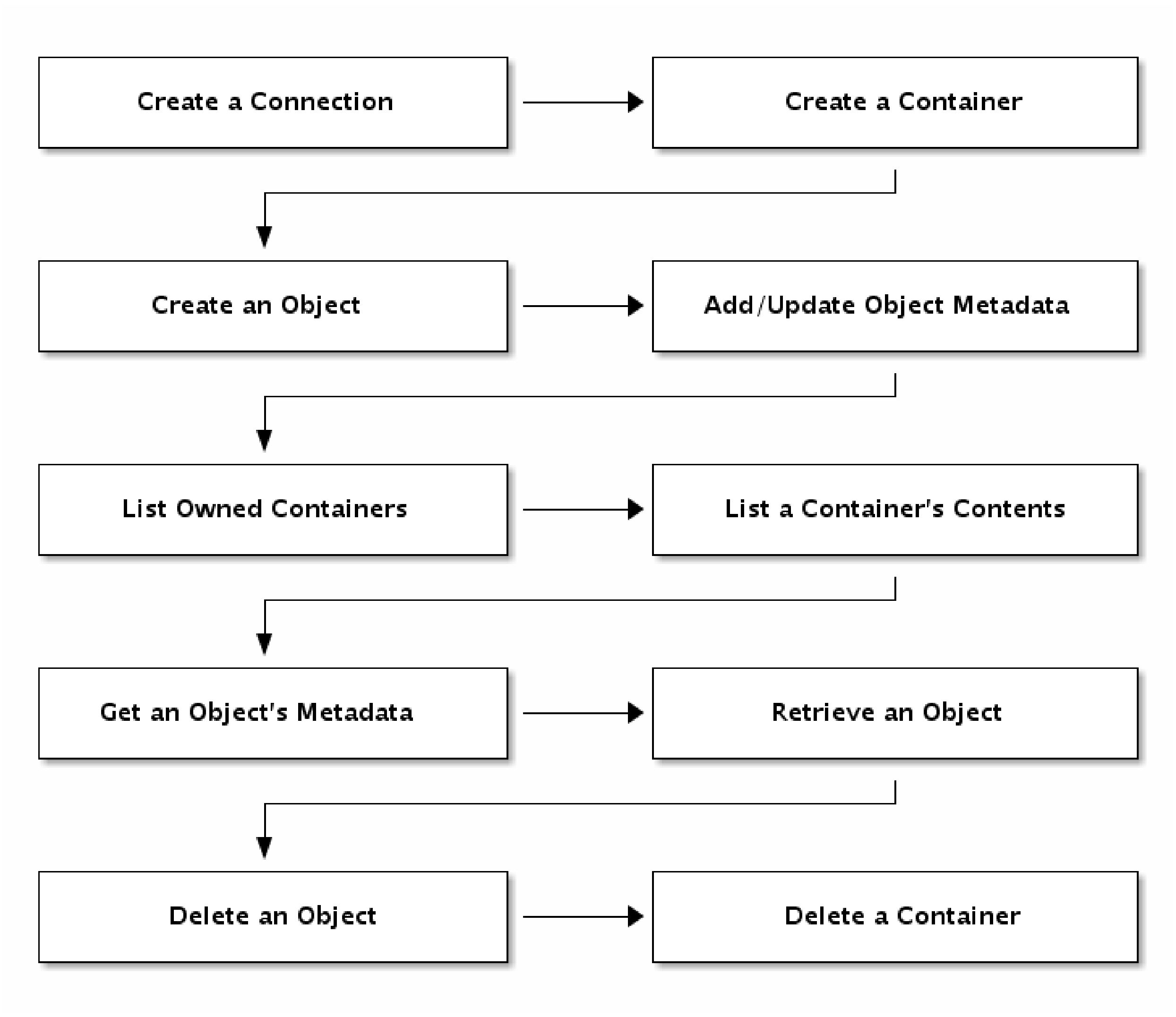
A sample python script to demonstrate the above is given below:

```
import hmac
from hashlib import sha1
from time import time

method = 'GET'
host = 'https://objectstore.example.com'
duration_in_seconds = 300  # Duration for which the url is valid
expires = int(time() + duration_in_seconds)
path = '/v1/your-bucket/your-object'
key = 'secret'
hmac_body = '%s\n%s\n%s' % (method, expires, path)
hmac_body = hmac.new(key, hmac_body, sha1).hexdigest()
sig = hmac.new(key, hmac_body, sha1).hexdigest()
rest_uri = "{host}{path}?temp_url_sig={sig}&temp_url_expires={expires}".format(
             host=host, path=path, sig=sig, expires=expires)
print rest_uri

# Example Output
# https://objectstore.example.com/v1/your-bucket/your-object?temp_url_sig=ff4657876227fc6025f04fcf1e
```

Tutorial

The Swift-compatible API tutorials follow a simple container-based object lifecycle. The first step requires you to setup a connection between your client and the RADOS Gateway server. Then, you may follow a natural container and object lifecycle, including adding and retrieving object metadata. See example code for the following languages:

- Java
- Python
- Ruby



Java Swift Examples

Setup

The following examples may require some or all of the following Java classes to be imported:

```
import java.io.File;
import java.util.List;
import java.util.Map;
import com.rackspacecloud.client.cloudfiles.FilesClient;
import com.rackspacecloud.client.cloudfiles.FilesConstants;
import com.rackspacecloud.client.cloudfiles.FilesContainer;
import com.rackspacecloud.client.cloudfiles.FilesContainerExistsException;
import com.rackspacecloud.client.cloudfiles.FilesObject;
import com.rackspacecloud.client.cloudfiles.FilesObjectMetaData;
```

Create a Connection

This creates a connection so that you can interact with the server:

```
String username = "USERNAME";
String password = "PASSWORD";
String authUrl  = "https://objects.dreamhost.com/auth";

FilesClient client = new FilesClient(username, password, authUrl);
if (!client.login()) {
        throw new RuntimeException("Failed to log in");
}
```

Create a Container

This creates a new container called `my-new-container`:

```
client.createContainer("my-new-container");
```

Create an Object

This creates an object `foo.txt` from the file named `foo.txt` in the container `my-new-container`:

```
File file = new File("foo.txt");
String mimeType = FilesConstants.getMimetype("txt");
client.storeObject("my-new-container", file, mimeType);
```

Add/Update Object Metadata

This adds the metadata key-value pair `key:value` to the object named `foo.txt` in the container `my-new-container`:

```
FilesObjectMetaData metaData = client.getObjectMetaData("my-new-container", "foo.txt");
metaData.addMetaData("key", "value");

Map<String, String> metamap = metaData.getMetaData();
client.updateObjectMetadata("my-new-container", "foo.txt", metamap);
```

List Owned Containers

This gets a list of Containers that you own. This also prints out the container name.

```
List<FilesContainer> containers = client.listContainers();
for (FilesContainer container : containers) {
        System.out.println("  " + container.getName());
}
```

The output will look something like this:

```
mahbuckat1
mahbuckat2
mahbuckat3
```

List a Container's Content

This gets a list of objects in the container `my-new-container`; and, it also prints out each object's name, the file size, and last modified date:

```
List<FilesObject> objects = client.listObjects("my-new-container");
for (FilesObject object : objects) {
        System.out.println("  " + object.getName());
}
```

The output will look something like this:

```
myphoto1.jpg
myphoto2.jpg
```

Retrieve an Object's Metadata

This retrieves metadata and gets the MIME type for an object named `foo.txt` in a container named `my-new-container`:

```
FilesObjectMetaData metaData =  client.getObjectMetaData("my-new-container", "foo.txt");
String mimeType = metaData.getMimeType();
```

Retrieve an Object

This downloads the object `foo.txt` in the container `my-new-container` and saves it in `./outfile.txt`:

```
FilesObject obj;
File outfile = new File("outfile.txt");

List<FilesObject> objects = client.listObjects("my-new-container");
for (FilesObject object : objects) {
        String name = object.getName();
        if (name.equals("foo.txt")) {
                obj = object;
                obj.writeObjectToFile(outfile);
        }
}
```

Delete an Object

This deletes the object `goodbye.txt` in the container "my-new-container":

```
client.deleteObject("my-new-container", "goodbye.txt");
```

Delete a Container

This deletes a container named "my-new-container":

```
client.deleteContainer("my-new-container");
```

Note: The container must be empty! Otherwise it won't work!

Python Swift Examples

Create a Connection

This creates a connection so that you can interact with the server:

```
import swiftclient
user = 'account_name:username'
key = 'your_api_key'

conn = swiftclient.Connection(
        user=user,
        key=key,
        authurl='https://objects.dreamhost.com/auth',
)
```

Create a Container

This creates a new container called `my-new-container`:

```
container_name = 'my-new-container'
conn.put_container(container_name)
```

Create an Object

This creates a file `hello.txt` from the file named `my_hello.txt`:

```
with open('hello.txt', 'r') as hello_file:
        conn.put_object(container_name, 'hello.txt',
                                        contents= hello_file.read(),
                                        content_type='text/plain')
```

List Owned Containers

This gets a list of containers that you own, and prints out the container name:

```
for container in conn.get_account()[1]:
        print container['name']
```

The output will look something like this:

```
mahbuckat1
mahbuckat2
mahbuckat3
```

List a Container's Content

This gets a list of objects in the container, and prints out each object's name, the file size, and last modified date:

```
for data in conn.get_container(container_name)[1]:
        print '{0}\t{1}\t{2}'.format(data['name'], data['bytes'], data['last_modified'])
```

The output will look something like this:

```
myphoto1.jpg 251262  2011-08-08T21:35:48.000Z
myphoto2.jpg 262518  2011-08-08T21:38:01.000Z
```

Retrieve an Object

This downloads the object `hello.txt` and saves it in `./my_hello.txt`:

```
obj_tuple = conn.get_object(container_name, 'hello.txt')
with open('my_hello.txt', 'w') as my_hello:
        my_hello.write(obj_tuple[1])
```

Delete an Object

This deletes the object `goodbye.txt`:

```
conn.delete_object(container_name, 'hello.txt')
```

Delete a Container

Note: The container must be empty! Otherwise the request won't work!

```
conn.delete_container(container_name)
```

Ruby Swift Examples

Create a Connection

This creates a connection so that you can interact with the server:

```
require 'cloudfiles'
username = 'account_name:user_name'
api_key  = 'your_secret_key'

conn = CloudFiles::Connection.new(
        :username => username,
        :api_key  => api_key,
        :auth_url => 'http://objects.dreamhost.com/auth'
)
```

Create a Container

This creates a new container called `my-new-container`

```
container = conn.create_container('my-new-container')
```

Create an Object

This creates a file `hello.txt` from the file named `my_hello.txt`

```
obj = container.create_object('hello.txt')
obj.load_from_filename('./my_hello.txt')
obj.content_type = 'text/plain'
```

List Owned Containers

This gets a list of Containers that you own, and also prints out the container name:

```
conn.containers.each do |container|
        puts container
end
```

The output will look something like this:

```
mahbuckat1
mahbuckat2
mahbuckat3
```

List a Container's Contents

This gets a list of objects in the container, and prints out each object's name, the file size, and last modified date:

```
require 'date'  # not necessary in the next version

container.objects_detail.each do |name, data|
        puts "#{name}\t#{data[:bytes]}\t#{data[:last_modified]}"
end
```

The output will look something like this:

```
myphoto1.jpg 251262  2011-08-08T21:35:48.000Z
myphoto2.jpg 262518  2011-08-08T21:38:01.000Z
```

Retrieve an Object

This downloads the object `hello.txt` and saves it in `./my_hello.txt`:

```
obj = container.object('hello.txt')
obj.save_to_filename('./my_hello.txt')
```

Delete an Object

This deletes the object `goodbye.txt`:

```
container.delete_object('goodbye.txt')
```

Delete a Container

Note: The container must be empty! Otherwise the request won't work!

```
container.delete_container('my-new-container')
```

7.8.2 Features Support

The following table describes the support status for current Swift functional features:

Feature	Status	Remarks
Authentication	Supported	
Get Account Metadata	Supported	No custom metadata
Swift ACLs	Supported	Supports a subset of Swift ACLs
List Containers	Supported	
Delete Container	Supported	
Create Container	Supported	
Get Container Metadata	Supported	
Update Container Metadata	Supported	
Delete Container Metadata	Supported	
List Objects	Supported	
Static Website	Not Supported	
Create Object	Supported	
Create Large Object	Supported	
Delete Object	Supported	
Get Object	Supported	
Copy Object	Supported	
Get Object Metadata	Supported	
Update Object Metadata	Supported	
Expiring Objects	Not Supported	
Object Versioning	Not Supported	
CORS	Not Supported	

7.9 Admin Operations

An admin API request will be done on a URI that starts with the configurable 'admin' resource entry point. Authorization for the admin API duplicates the S3 authorization mechanism. Some operations require that the user holds special administrative capabilities. The response entity type (XML or JSON) may be specified as the 'format' option in the request and defaults to JSON if not specified.

7.9.1 Get Usage

Request bandwidth usage information.

caps usage=read

Syntax

```
GET /{admin}/usage?format=json HTTP/1.1
Host: {fqdn}
```

Request Parameters

`uid`

Description The user for which the information is requested. If not specified will apply to all users.

Type String

Example `foo_user`

Required No

`start`

Description Date and (optional) time that specifies the start time of the requested data.

Type String

Example `2012-09-25 16:00:00`

Required No

`end`

Description Date and (optional) time that specifies the end time of the requested data (non-inclusive).

Type String

Example `2012-09-25 16:00:00`

Required No

`show-entries`

Description Specifies whether data entries should be returned.

Type Boolean

Example True [False]

Required No

`show-summary`

Description Specifies whether data summary should be returned.

Type Boolean

Example True [False]

Required No

Response Entities

If successful, the response contains the requested information.

`usage`

Description A container for the usage information.

Type Container

`entries`

Description A container for the usage entries information.

Type Container

`user`

Description A container for the user data information.

Type Container

`owner`

Description The name of the user that owns the buckets.

Type String

`bucket`

Description The bucket name.

Type String

`time`

Description Time lower bound for which data is being specified (rounded to the beginning of the first relevant hour).

Type String

`epoch`

Description The time specified in seconds since 1/1/1970.

Type String

`categories`

Description A container for stats categories.

Type Container

`entry`

Description A container for stats entry.

Type Container

`category`

Description Name of request category for which the stats are provided.

Type String

`bytes_sent`

Description Number of bytes sent by the RADOS Gateway.

Type Integer

`bytes_received`

Description Number of bytes received by the RADOS Gateway.

Type Integer

`ops`

Description Number of operations.

Type Integer

`successful_ops`

Description Number of successful operations.

Type Integer

`summary`

Description A container for stats summary.

Type Container

`total`

Description A container for stats summary aggregated total.

Type Container

Special Error Responses

TBD.

7.9.2 Trim Usage

Remove usage information. With no dates specified, removes all usage information.

caps usage=write

Syntax

```
DELETE /{admin}/usage?format=json HTTP/1.1
Host: {fqdn}
```

Request Parameters

`uid`

Description The user for which the information is requested. If not specified will apply to all users.

Type String

Example `foo_user`

Required No

`start`

Description Date and (optional) time that specifies the start time of the requested data.

Type String

Example `2012-09-25 16:00:00`

Required No

`end`

Description Date and (optional) time that specifies the end time of the requested data (none inclusive).

Type String

Example `2012-09-25 16:00:00`

Required No

`remove-all`

Description Required when uid is not specified, in order to acknowledge multi user data removal.

Type Boolean

Example True [False]

Required No

Special Error Responses

TBD.

7.9.3 Get User Info

Get user information. If no user is specified returns the list of all users along with suspension information.

caps users=read

Syntax

```
GET /{admin}/user?format=json HTTP/1.1
Host: {fqdn}
```

Request Parameters

`uid`

Description The user for which the information is requested.

Type String

Example `foo_user`

Required No

Response Entities

If successful, the response contains the user information.

`user`

Description A container for the user data information.

Type Container

`user_id`

Description The user id.

Type String

Parent `user`

`display_name`

Description Display name for the user.

Type String

Parent `user`

`suspended`

Description True if the user is suspended.

Type Boolean

Parent `user`

`max_buckets`

Description The maximum number of buckets to be owned by the user.

Type Integer

Parent `user`

`subusers`

Description Subusers associated with this user account.

Type Container

Parent `user`

`keys`

Description S3 keys associated with this user account.

Type Container

Parent `user`

`swift_keys`

Description Swift keys associated with this user account.

Type Container

Parent `user`

`caps`

Description User capabilities.

Type Container

Parent `user`

Special Error Responses

None.

7.9.4 Create User

Create a new user. By Default, a S3 key pair will be created automatically and returned in the response. If only one of `access-key` or `secret-key` is provided, the omitted key will be automatically generated. By default, a generated key is added to the keyring without replacing an existing key pair. If `access-key` is specified and refers to an existing key owned by the user then it will be modified.

caps users=write

Syntax

```
PUT /{admin}/user?format=json HTTP/1.1
Host: {fqdn}
```

Request Parameters

`uid`

Description The user ID to be created.

Type String

Example `foo_user`

Required Yes

`display-name`

Description The display name of the user to be created.

Type String

Example `foo user`

Required Yes

`email`

Description The email address associated with the user.

Type String

Example `foo@bar.com`

Required No

`key-type`

Description Key type to be generated, options are: swift, s3 (default).

Type String

Example `s3` [`s3`]

Required No

`access-key`

Description Specify access key.

Type String

Example `ABCD0EF12GHIJ2K34LMN`

Required No

`secret-key`

Description Specify secret key.

Type String

Example `0AbCDEFg1h2i34JklM5nop6QrSTUV+WxyzaBC7D8`

Required No

`user-caps`

Description User capabilities.

Type String

Example `usage=read, write; users=read`

Required No

`generate-key`

Description Generate a new key pair and add to the existing keyring.

Type Boolean

Example True [True]

Required No

`max-buckets`

Description Specify the maximum number of buckets the user can own.

Type Integer

Example 500 [1000]

Required No

`suspended`

Description Specify whether the user should be suspended.

Type Boolean

Example False [False]

Required No

Response Entities

If successful, the response contains the user information.

`user`

Description A container for the user data information.

Type Container

`user_id`

Description The user id.

Type String

Parent `user`

`display_name`

Description Display name for the user.

Type String

Parent `user`

`suspended`

Description True if the user is suspended.

Type Boolean

Parent `user`

`max_buckets`

Description The maximum number of buckets to be owned by the user.

Type Integer

Parent `user`

`subusers`

Description Subusers associated with this user account.

Type Container

Parent `user`

`keys`

Description S3 keys associated with this user account.

Type Container

Parent `user`

`swift_keys`

Description Swift keys associated with this user account.

Type Container

Parent `user`

`caps`

Description User capabilities.

Type Container

Parent `user`

Special Error Responses

`UserExists`

Description Attempt to create existing user.

Code 409 Conflict

`InvalidAccessKey`

Description Invalid access key specified.

Code 400 Bad Request

`InvalidKeyType`

Description Invalid key type specified.

Code 400 Bad Request

`InvalidSecretKey`

Description Invalid secret key specified.

Code 400 Bad Request

`InvalidKeyType`

Description Invalid key type specified.

Code 400 Bad Request

`KeyExists`

Description Provided access key exists and belongs to another user.

Code 409 Conflict

`EmailExists`

Description Provided email address exists.

Code 409 Conflict

`InvalidCap`

Description Attempt to grant invalid admin capability.

Code 400 Bad Request

7.9.5 Modify User

Modify a user.

caps users=write

Syntax

```
POST /{admin}/user?format=json HTTP/1.1
Host: {fqdn}
```

Request Parameters

`uid`

Description The user ID to be modified.

Type String

Example `foo_user`

Required Yes

`display-name`

Description The display name of the user to be modified.

Type String

Example `foo user`

Required No

`email`

Description The email address to be associated with the user.

Type String

Example `foo@bar.com`

Required No

`generate-key`

Description Generate a new key pair and add to the existing keyring.

Type Boolean

Example True [False]

Required No

`access-key`

Description Specify access key.

Type String

Example `ABCD0EF12GHIJ2K34LMN`

Required No

`secret-key`

Description Specify secret key.

Type String

Example `0AbCDEFg1h2i34JklM5nop6QrSTUV+WxyzaBC7D8`

Required No

`key-type`

Description Key type to be generated, options are: swift, s3 (default).

Type String

Example `s3`

Required No

`user-caps`

Description User capabilities.

Type String

Example `usage=read, write; users=read`

Required No

`max-buckets`

Description Specify the maximum number of buckets the user can own.

Type Integer

Example 500 [1000]

Required No

`suspended`

Description Specify whether the user should be suspended.

Type Boolean

Example False [False]

Required No

Response Entities

If successful, the response contains the user information.

`user`

Description A container for the user data information.

Type Container

`user_id`

Description The user id.

Type String

Parent `user`

`display_name`

Description Display name for the user.

Type String

Parent `user`

`suspended`

Description True if the user is suspended.

Type Boolean

Parent `user`

`max_buckets`

Description The maximum number of buckets to be owned by the user.

Type Integer

Parent `user`

`subusers`

Description Subusers associated with this user account.

Type Container

Parent `user`

`keys`

Description S3 keys associated with this user account.

Type Container

Parent `user`

`swift_keys`

Description Swift keys associated with this user account.

Type Container

Parent `user`

`caps`

Description User capabilities.

Type Container

Parent `user`

Special Error Responses

`InvalidAccessKey`

Description Invalid access key specified.

Code 400 Bad Request

`InvalidKeyType`

Description Invalid key type specified.

Code 400 Bad Request

`InvalidSecretKey`

Description Invalid secret key specified.

Code 400 Bad Request

`KeyExists`

Description Provided access key exists and belongs to another user.

Code 409 Conflict

`EmailExists`

Description Provided email address exists.

Code 409 Conflict

`InvalidCap`

Description Attempt to grant invalid admin capability.

Code 400 Bad Request

7.9.6 Remove User

Remove an existing user.

caps users=write

Syntax

```
DELETE /{admin}/user?format=json HTTP/1.1
Host: {fqdn}
```

Request Parameters

`uid`

Description The user ID to be removed.

Type String

Example `foo_user`

Required Yes.

`purge-data`

Description When specified the buckets and objects belonging to the user will also be removed.

Type Boolean

Example True

Required No

Response Entities

None

Special Error Responses

None.

7.9.7 Create Subuser

Create a new subuser (primarily useful for clients using the Swift API). Note that either `gen-subuser` or `subuser` is required for a valid request. Note that in general for a subuser to be useful, it must be granted permissions by specifying `access`. As with user creation if `subuser` is specified without `secret`, then a secret key will be automatically generated.

caps users=write

Syntax

```
PUT /{admin}/user?subuser&format=json HTTP/1.1
Host {fqdn}
```

Request Parameters

`uid`

Description The user ID under which a subuser is to be created.

Type String

Example `foo_user`

Required Yes

`subuser`

Description Specify the subuser ID to be created.

Type String

Example `sub_foo`

Required No

`secret-key`

Description Specify secret key.

Type String

Example `0AbCDEFg1h2i34JklM5nop6QrSTUV+WxyzaBC7D8`

Required No

`key-type`

Description Key type to be generated, options are: swift (default), s3.

Type String

Example `swift` [`swift`]

Required No

`access`

Description Set access permissions for sub-user, should be one of `read, write, readwrite, full`.

Type String

Example `read`

Required No

`generate-secret`

Description Generate the secret key.

Type Boolean

Example True [False]

Required No

Response Entities

If successful, the response contains the subuser information.

`subusers`

Description Subusers associated with the user account.

Type Container

`id`

Description Subuser id.

Type String

Parent `subusers`

`permissions`

Description Subuser access to user account.

Type String

Parent `subusers`

Special Error Responses

`SubuserExists`

Description Specified subuser exists.

Code 409 Conflict

`InvalidKeyType`

Description Invalid key type specified.

Code 400 Bad Request

`InvalidSecretKey`

Description Invalid secret key specified.

Code 400 Bad Request

`InvalidAccess`

Description Invalid subuser access specified.

Code 400 Bad Request

7.9.8 Modify Subuser

Modify an existing subuser

caps users=write

Syntax

```
POST /{admin}/user?subuser&format=json HTTP/1.1
Host {fqdn}
```

Request Parameters

`uid`

Description The user ID under which the subuser is to be modified.

Type String

Example `foo_user`

Required Yes

`subuser`

Description The subuser ID to be modified.

Type String

Example `sub_foo`

Required Yes

`generate-secret`

Description Generate a new secret key for the subuser, replacing the existing key.

Type Boolean

Example True [False]

Required No

`secret`

Description Specify secret key.

Type String

Example `0AbCDEFg1h2i34JklM5nop6QrSTUV+WxyzaBC7D8`

Required No

`key-type`

Description Key type to be generated, options are: swift (default), s3 .

Type String

Example `swift` [`swift`]

Required No

`access`

Description Set access permissions for sub-user, should be one of `read, write, readwrite, full`.

Type String

Example `read`

Required No

Response Entities

If successful, the response contains the subuser information.

`subusers`

Description Subusers associated with the user account.

Type Container

`id`

Description Subuser id.

Type String

Parent `subusers`

`permissions`

Description Subuser access to user account.

Type String

Parent `subusers`

Special Error Responses

`InvalidKeyType`

Description Invalid key type specified.

Code 400 Bad Request

`InvalidSecretKey`

Description Invalid secret key specified.

Code 400 Bad Request

`InvalidAccess`

Description Invalid subuser access specified.

Code 400 Bad Request

7.9.9 Remove Subuser

Remove an existing subuser

caps users=write

Syntax

```
DELETE /{admin}/user?subuser&format=json HTTP/1.1
Host {fqdn}
```

Request Parameters

`uid`

Description The user ID under which the subuser is to be removed.

Type String

Example `foo_user`

Required Yes

`subuser`

Description The subuser ID to be removed.

Type String

Example `sub_foo`

Required Yes

`purge-keys`

Description Remove keys belonging to the subuser.

Type Boolean

Example True [True]

Required No

Response Entities

None.

Special Error Responses

None.

7.9.10 Create Key

Create a new key. If a `subuser` is specified then by default created keys will be swift type. If only one of `access-key` or `secret-key` is provided the committed key will be automatically generated, that is if only `secret-key` is specified then `access-key` will be automatically generated. By default, a generated key is added to the keyring without replacing an existing key pair. If `access-key` is specified and refers to an existing key owned by the user then it will be modified. The response is a container listing all keys of the same type as the key created. Note that when creating a swift key, specifying the option `access-key` will have no effect. Additionally, only one swift key may be held by each user or subuser.

caps users=write

Syntax

```
PUT /{admin}/user?key&format=json HTTP/1.1
Host {fqdn}
```

Request Parameters

`uid`

Description The user ID to receive the new key.

Type String

Example `foo_user`

Required Yes

`subuser`

Description The subuser ID to receive the new key.

Type String

Example `sub_foo`

Required No

`key-type`

Description Key type to be generated, options are: swift, s3 (default).

Type String

Example `s3` [`s3`]

Required No

`access-key`

Description Specify the access key.

Type String

Example `AB01C2D3EF45G6H7IJ8K`

Required No

`secret-key`

Description Specify the secret key.

Type String

Example `0ab/CdeFGhij1klmnopqRSTUv1WxyZabcDEFgHij`

Required No

`generate-key`

Description Generate a new key pair and add to the existing keyring.

Type Boolean

Example True [`True`]

Required No

Response Entities

`keys`

Description Keys of type created associated with this user account.

Type Container

`user`

Description The user account associated with the key.

Type String

Parent `keys`

`access-key`

Description The access key.

Type String

Parent `keys`

`secret-key`

Description The secret key

Type String

Parent `keys`

Special Error Responses

`InvalidAccessKey`

Description Invalid access key specified.

Code 400 Bad Request

`InvalidKeyType`

Description Invalid key type specified.

Code 400 Bad Request

`InvalidSecretKey`

Description Invalid secret key specified.

Code 400 Bad Request

`InvalidKeyType`

Description Invalid key type specified.

Code 400 Bad Request

`KeyExists`

Description Provided access key exists and belongs to another user.

Code 409 Conflict

7.9.11 Remove Key

Remove an existing key.

caps users=write

Syntax

```
DELETE /{admin}/user?key&format=json HTTP/1.1
Host {fqdn}
```

Request Parameters

`access-key`

Description The S3 access key belonging to the S3 key pair to remove.

Type String

Example `AB01C2D3EF45G6H7IJ8K`

Required Yes

`uid`

Description The user to remove the key from.

Type String

Example `foo_user`

Required No

`subuser`

Description The subuser to remove the key from.

Type String

Example `sub_foo`

Required No

`key-type`

Description Key type to be removed, options are: swift, s3. NOTE: Required to remove swift key.

Type String

Example `swift`

Required No

Special Error Responses

None.

Response Entities

None.

7.9.12 Get Bucket Info

Get information about a subset of the existing buckets. If `uid` is specified without `bucket` then all buckets beloning to the user will be returned. If `bucket` alone is specified, information for that particular bucket will be retrieved.

caps buckets=read

Syntax

```
GET /{admin}/bucket?format=json HTTP/1.1
Host {fqdn}
```

Request Parameters

`bucket`

Description The bucket to return info on.

Type String

Example `foo_bucket`

Required No

`uid`

Description The user to retrieve bucket information for.

Type String

Example `foo_user`

Required No

`stats`

Description Return bucket statistics.

Type Boolean

Example True [False]

Required No

Response Entities

If successful the request returns a buckets container containing the desired bucket information.

`stats`

Description Per bucket information.

Type Container

`buckets`

Description Contains a list of one or more bucket containers.

Type Container

`bucket`

Description Container for single bucket information.

Type Container

Parent `buckets`

`name`

Description The name of the bucket.

Type String

Parent `bucket`

`pool`

Description The pool the bucket is stored in.

Type String

Parent `bucket`

`id`

Description The unique bucket id.

Type String

Parent `bucket`

`marker`

Description Internal bucket tag.

Type String

Parent `bucket`

`owner`

Description The user id of the bucket owner.

Type String

Parent `bucket`

`usage`

Description Storage usage information.

Type Container

Parent `bucket`

`index`

Description Status of bucket index.

Type String

Parent `bucket`

Special Error Responses

`IndexRepairFailed`

Description Bucket index repair failed.

Code 409 Conflict

7.9.13 Check Bucket Index

Check the index of an existing bucket. NOTE: to check multipart object accounting with `check-objects`, `fix` must be set to True.

caps buckets=write

Syntax

```
GET /{admin}/bucket?index&format=json HTTP/1.1
Host {fqdn}
```

Request Parameters

`bucket`

Description The bucket to return info on.

Type String

Example `foo_bucket`

Required Yes

`check-objects`

Description Check multipart object accounting.

Type Boolean

Example True [False]

Required No

`fix`

Description Also fix the bucket index when checking.

Type Boolean

Example False [False]

Required No

Response Entities

`index`

Description Status of bucket index.

Type String

Special Error Responses

`IndexRepairFailed`

Description Bucket index repair failed.

Code 409 Conflict

7.9.14 Remove Bucket

Delete an existing bucket.

caps buckets=write

Syntax

```
DELETE /{admin}/bucket?format=json HTTP/1.1
Host {fqdn}
```

Request Parameters

`bucket`

Description The bucket to remove.

Type String

Example `foo_bucket`

Required Yes

`purge-objects`

Description Remove a buckets objects before deletion.

Type Boolean

Example True [False]

Required No

Response Entities

None.

Special Error Responses

`BucketNotEmpty`

  Description Attempted to delete non-empty bucket.

  Code 409 Conflict

`ObjectRemovalFailed`

  Description Unable to remove objects.

  Code 409 Conflict

7.9.15 Unlink Bucket

Unlink a bucket from a specified user. Primarily useful for changing bucket ownership.

  caps buckets=write

Syntax

```
POST /{admin}/bucket?format=json HTTP/1.1
Host {fqdn}
```

Request Parameters

`bucket`

  Description The bucket to unlink.

  Type String

  Example `foo_bucket`

  Required Yes

`uid`

  Description The user ID to unlink the bucket from.

  Type String

  Example `foo_user`

  Required Yes

Response Entities

None.

Special Error Responses

`BucketUnlinkFailed`

  Description Unable to unlink bucket from specified user.

  Code 409 Conflict

7.9.16 Link Bucket

Link a bucket to a specified user, unlinking the bucket from any previous user.

caps buckets=write

Syntax

```
PUT /{admin}/bucket?format=json HTTP/1.1
Host {fqdn}
```

Request Parameters

`bucket`

Description The bucket to unlink.

Type String

Example `foo_bucket`

Required Yes

`uid`

Description The user ID to link the bucket to.

Type String

Example `foo_user`

Required Yes

Response Entities

`bucket`

Description Container for single bucket information.

Type Container

`name`

Description The name of the bucket.

Type String

Parent `bucket`

`pool`

Description The pool the bucket is stored in.

Type String

Parent `bucket`

`id`

Description The unique bucket id.

Type String

Parent `bucket`

`marker`

Description Internal bucket tag.

Type String

Parent `bucket`

`owner`

Description The user id of the bucket owner.

Type String

Parent `bucket`

`usage`

Description Storage usage information.

Type Container

Parent `bucket`

`index`

Description Status of bucket index.

Type String

Parent `bucket`

Special Error Responses

`BucketUnlinkFailed`

Description Unable to unlink bucket from specified user.

Code 409 Conflict

`BucketLinkFailed`

Description Unable to link bucket to specified user.

Code 409 Conflict

7.9.17 Remove Object

Remove an existing object. NOTE: Does not require owner to be non-suspended.

caps buckets=write

Syntax

```
DELETE /{admin}/bucket?object&format=json HTTP/1.1
Host {fqdn}
```

Request Parameters

`bucket`

> **Description** The bucket containing the object to be removed.
>
> **Type** String
>
> **Example** `foo_bucket`
>
> **Required** Yes

`object`

> **Description** The object to remove.
>
> **Type** String
>
> **Example** `foo.txt`
>
> **Required** Yes

Response Entities

None.

Special Error Responses

`NoSuchObject`

> **Description** Specified object does not exist.
>
> **Code** 404 Not Found

`ObjectRemovalFailed`

> **Description** Unable to remove objects.
>
> **Code** 409 Conflict

7.9.18 Get Bucket or Object Policy

Read the policy of an object or bucket.

> **caps** buckets=read

Syntax

```
GET /{admin}/bucket?policy&format=json HTTP/1.1
Host {fqdn}
```

Request Parameters

`bucket`

> **Description** The bucket to read the policy from.
>
> **Type** String

Example `foo_bucket`

Required Yes

`object`

Description The object to read the policy from.

Type String

Example `foo.txt`

Required No

Response Entities

If successful, returns the object or bucket policy

`policy`

Description Access control policy.

Type Container

Special Error Responses

`IncompleteBody`

Description Either bucket was not specified for a bucket policy request or bucket and object were not specified for an object policy request.

Code 400 Bad Request

7.9.19 Add A User Capability

Add an administrative capability to a specified user.

caps users=write

Syntax

```
PUT /{admin}/user?caps&format=json HTTP/1.1
Host {fqdn}
```

Request Parameters

`uid`

Description The user ID to add an administrative capability to.

Type String

Example `foo_user`

Required Yes

`user-caps`

Description The administrative capability to add to the user.

Type String

Example `usage=read, write`

Required Yes

Response Entities

If successful, the response contains the user's capabilities.

`user`

Description A container for the user data information.

Type Container

Parent `user`

`user_id`

Description The user id.

Type String

Parent `user`

`caps`

Description User capabilities.

Type Container

Parent `user`

Special Error Responses

`InvalidCap`

Description Attempt to grant invalid admin capability.

Code 400 Bad Request

Example Request

```
PUT /{admin}/user?caps&format=json HTTP/1.1
Host: {fqdn}
Content-Type: text/plain
Authorization: {your-authorization-token}

usage=read
```

7.9.20 Remove A User Capability

Remove an administrative capability from a specified user.

caps users=write

Syntax

```
DELETE /{admin}/user?caps&format=json HTTP/1.1
Host {fqdn}
```

Request Parameters

`uid`

Description The user ID to remove an administrative capability from.

Type String

Example `foo_user`

Required Yes

`user-caps`

Description The administrative capabilities to remove from the user.

Type String

Example `usage=read, write`

Required Yes

Response Entities

If successful, the response contains the user's capabilities.

`user`

Description A container for the user data information.

Type Container

Parent `user`

`user_id`

Description The user id.

Type String

Parent `user`

`caps`

Description User capabilities.

Type Container

Parent `user`

Special Error Responses

`InvalidCap`

Description Attempt to remove an invalid admin capability.

Code 400 Bad Request

`NoSuchCap`

Description User does not possess specified capability.

Code 404 Not Found

Special Error Responses

None.

7.9.21 Quotas

The Admin Operations API enables you to set quotas on users and on bucket owned by users. See Quota Management for additional details. Quotas include the maximum number of objects in a bucket and the maximum storage size in megabytes.

To view quotas, the user must have a `users=read` capability. To set, modify or disable a quota, the user must have `users=write` capability. See the Admin Guide for details.

Valid parameters for quotas include:

- **Bucket:** The `bucket` option allows you to specify a quota for buckets owned by a user.
- **Maximum Objects:** The `max-objects` setting allows you to specify the maximum number of objects. A negative value disables this setting.
- **Maximum Size:** The `max-size` option allows you to specify a quota for the maximum number of bytes. A negative value disables this setting.
- **Quota Scope:** The `quota-scope` option sets the scope for the quota. The options are `bucket` and `user`.

Get User Quota

To get a quota, the user must have `users` capability set with `read` permission.

```
GET /admin/user?quota&uid=<uid>&quota-type=user
```

Set User Quota

To set a quota, the user must have `users` capability set with `write` permission.

```
PUT /admin/user?quota&uid=<uid>&quota-type=user
```

The content must include a JSON representation of the quota settings as encoded in the corresponding read operation.

Get Bucket Quota

To get a quota, the user must have `users` capability set with `read` permission.

```
GET /admin/user?quota&uid=<uid>&quota-type=bucket
```

Set Bucket Quota

To set a quota, the user must have `users` capability set with `write` permission.

```
PUT /admin/user?quota&uid=<uid>&quota-type=bucket
```

The content must include a JSON representation of the quota settings as encoded in the corresponding read operation.

7.9.22 Standard Error Responses

`AccessDenied`

Description Access denied.

Code 403 Forbidden

`InternalError`

Description Internal server error.

Code 500 Internal Server Error

`NoSuchUser`

Description User does not exist.

Code 404 Not Found

`NoSuchBucket`

Description Bucket does not exist.

Code 404 Not Found

`NoSuchKey`

Description No such access key.

Code 404 Not Found

7.10 Integrating with OpenStack Keystone

It is possible to integrate the Ceph Object Gateway with Keystone, the OpenStack identity service. This sets up the gateway to accept Keystone as the users authority. A user that Keystone authorizes to access the gateway will also be automatically created on the Ceph Object Gateway (if didn't exist beforehand). A token that Keystone validates will be considered as valid by the gateway.

The following configuration options are available for Keystone integration:

```
[client.radosgw.gateway]
rgw keystone url = {keystone server url:keystone server admin port}
rgw keystone admin token = {keystone admin token}
rgw keystone accepted roles = {accepted user roles}
rgw keystone token cache size = {number of tokens to cache}
rgw keystone revocation interval = {number of seconds before checking revoked tickets}
rgw s3 auth use keystone = true
nss db path = {path to nss db}
```

A Ceph Object Gateway user is mapped into a Keystone `tenant`. A Keystone user has different roles assigned to it on possibly more than a single tenant. When the Ceph Object Gateway gets the ticket, it looks at the tenant, and the user roles that are assigned to that ticket, and accepts/rejects the request according to the `rgw keystone accepted roles` configurable.

Keystone itself needs to be configured to point to the Ceph Object Gateway as an object-storage endpoint:

```
keystone service-create --name swift --type object-store
keystone endpoint-create --service-id <id> --publicurl http://radosgw.example.com/swift/v1 \
        --internalurl http://radosgw.example.com/swift/v1 --adminurl http://radosgw.example.com/swift
```

The keystone URL is the Keystone admin RESTful API URL. The admin token is the token that is configured internally in Keystone for admin requests.

The Ceph Object Gateway will query Keystone periodically for a list of revoked tokens. These requests are encoded and signed. Also, Keystone may be configured to provide self-signed tokens, which are also encoded and signed. The gateway needs to be able to decode and verify these signed messages, and the process requires that the gateway be set up appropriately. Currently, the Ceph Object Gateway will only be able to perform the procedure if it was compiled with `--with-nss`. Configuring the Ceph Object Gateway to work with Keystone also requires converting the OpenSSL certificates that Keystone uses for creating the requests to the nss db format, for example:

```
mkdir /var/ceph/nss

openssl x509 -in /etc/keystone/ssl/certs/ca.pem -pubkey | \
        certutil -d /var/ceph/nss -A -n ca -t "TCu,Cu,Tuw"
openssl x509 -in /etc/keystone/ssl/certs/signing_cert.pem -pubkey | \
        certutil -A -d /var/ceph/nss -n signing_cert -t "P,P,P"
```

7.11 Troubleshooting

7.11.1 The Gateway Won't Start

If you cannot start the gateway (i.e., there is no existing `pid`), check to see if there is an existing `.asok` file from another user. If an `.asok` file from another user exists and there is no running `pid`, remove the `.asok` file and try to start the process again.

This may occur when you start the process as a `root` user and the startup script is trying to start the process as a `www-data` or `apache` user and an existing `.asok` is preventing the script from starting the daemon.

The radosgw init script (/etc/init.d/radosgw) also has a verbose argument that can provide some insight as to what could be the issue:

> /etc/init.d/radosgw start -v

or

> /etc/init.d radosgw start –verbose

7.11.2 HTTP Request Errors

Examining the access and error logs for the web server itself is probably the first step in identifying what is going on. If there is a 500 error, that usually indicates a problem communicating with the `radosgw` daemon. Ensure the daemon is running, its socket path is configured, and that the web server is looking for it in the proper location.

7.11.3 Crashed `radosgw` process

If the `radosgw` process dies, you will normally see a 500 error from the web server (apache, nginx, etc.). In that situation, simply restarting radosgw will restore service.

To diagnose the cause of the crash, check the log in `/var/log/ceph` and/or the core file (if one was generated).

7.11.4 Blocked `radosgw` Requests

If some (or all) radosgw requests appear to be blocked, you can get some insight into the internal state of the `radosgw` daemon via its admin socket. By default, there will be a socket configured to reside in `/var/run/ceph`, and the daemon can be queried with:

```
ceph --admin-daemon /var/run/ceph/client.rgw help

help                   list available commands
objecter_requests      show in-progress osd requests
perfcounters_dump      dump perfcounters value
perfcounters_schema    dump perfcounters schema
version                get protocol version
```

Of particular interest:

```
ceph --admin-daemon /var/run/ceph/client.rgw objecter_requests
...
```

will dump information about current in-progress requests with the RADOS cluster. This allows one to identify if any requests are blocked by a non-responsive OSD. For example, one might see:

```
{ "ops": [
      { "tid": 1858,
        "pg": "2.d2041a48",
        "osd": 1,
        "last_sent": "2012-03-08 14:56:37.949872",
        "attempts": 1,
        "object_id": "fatty_25647_object1857",
        "object_locator": "@2",
        "snapid": "head",
        "snap_context": "0=[]",
        "mtime": "2012-03-08 14:56:37.949813",
        "osd_ops": [
              "write 0~4096"]},
      { "tid": 1873,
        "pg": "2.695e9f8e",
        "osd": 1,
        "last_sent": "2012-03-08 14:56:37.970615",
        "attempts": 1,
        "object_id": "fatty_25647_object1872",
        "object_locator": "@2",
        "snapid": "head",
        "snap_context": "0=[]",
        "mtime": "2012-03-08 14:56:37.970555",
        "osd_ops": [
              "write 0~4096"]}],
"linger_ops": [],
"pool_ops": [],
"pool_stat_ops": [],
"statfs_ops": []}
```

In this dump, two requests are in progress. The `last_sent` field is the time the RADOS request was sent. If this is a while ago, it suggests that the OSD is not responding. For example, for request 1858, you could check the OSD status with:

```
ceph pg map 2.d2041a48

osdmap e9 pg 2.d2041a48 (2.0) -> up [1,0] acting [1,0]
```

This tells us to look at `osd.1`, the primary copy for this PG:

```
ceph --admin-daemon /var/run/ceph/osd.1.asok
{ "num_ops": 651,
 "ops": [
       { "description": "osd_op(client.4124.0:1858 fatty_25647_object1857 [write 0~4096] 2.d2041a48)"
         "received_at": "1331247573.344650",
         "age": "25.606449",
         "flag_point": "waiting for sub ops",
         "client_info": { "client": "client.4124",
             "tid": 1858}},
...
```

The `flag_point` field indicates that the OSD is currently waiting for replicas to respond, in this case `osd.0`.

7.11.5 Java S3 API Troubleshooting

Peer Not Authenticated

You may receive an error that looks like this:

```
[java] INFO: Unable to execute HTTP request: peer not authenticated
```

The Java SDK for S3 requires a valid certificate from a recognized certificate authority, because it uses HTTPS by default. If you are just testing the Ceph Object Storage services, you can resolve this problem in a few ways:

1. Prepend the IP address or hostname with `http://`. For example, change this:

```
    conn.setEndpoint("myserver");
```

   To:

```
    conn.setEndpoint("http://myserver")
```

2. After setting your credentials, add a client configuration and set the protocol to `Protocol.HTTP`.

```
    AWSCredentials credentials = new BasicAWSCredentials(accessKey, secretKey);

    ClientConfiguration clientConfig = new ClientConfiguration();
    clientConfig.setProtocol(Protocol.HTTP);

    AmazonS3 conn = new AmazonS3Client(credentials, clientConfig);
```

405 MethodNotAllowed

If you receive an 405 error, check to see if you have the S3 subdomain set up correctly. You will need to have a wild card setting in your DNS record for subdomain functionality to work properly.

Also, check to ensure that the default site is disabled.

```
[java] Exception in thread "main" Status Code: 405, AWS Service: Amazon S3, AWS Request ID: null, AWS
```

7.12 radosgw – rados REST gateway

7.12.1 Synopsis

radosgw

7.12.2 Description

radosgw is an HTTP REST gateway for the RADOS object store, a part of the Ceph distributed storage system. It is implemented as a FastCGI module using libfcgi, and can be used in conjunction with any FastCGI capable web server.

7.12.3 Options

-c ceph.conf, **--conf**=ceph.conf
Use ceph.conf configuration file instead of the default /etc/ceph/ceph.conf to determine monitor addresses during startup.

-m monaddress[:port]
Connect to specified monitor (instead of looking through ceph.conf).

-i ID, **--id** ID
Set the ID portion of name for radosgw

-n TYPE.ID, **--name** TYPE.ID
Set the rados user name for the gateway (eg. client.radosgw.gateway)

--cluster NAME
Set the cluster name (default: ceph)

-d
Run in foreground, log to stderr

-f
Run in foreground, log to usual location

--rgw-socket-path=path
Specify a unix domain socket path.

--rgw-region=region
The region where radosgw runs

--rgw-zone=zone
The zone where radosgw runs

7.12.4 Configuration

Earlier RADOS Gateway had to be configured with `Apache` and `mod_fastcgi`. Now, `mod_proxy_fcgi` module is used instead of `mod_fastcgi`. `mod_proxy_fcgi` works differently than a traditional FastCGI module. This module requires the service of `mod_proxy` which provides support for the FastCGI protocol. So, to be able to handle FastCGI protocol, both `mod_proxy` and `mod_proxy_fcgi` have to be present in the server. Unlike `mod_fastcgi`, `mod_proxy_fcgi` cannot start the application process. Some platforms have `fcgistarter` for that purpose. However, external launching of application or process management may be available in the FastCGI application framework in use.

`Apache` can be configured in a way that enables `mod_proxy_fcgi` to be used with localhost tcp or through unix domain socket. `mod_proxy_fcgi` that doesn't support unix domain socket such as the ones in Apache 2.2 and earlier versions of Apache 2.4, needs to be configured for use with localhost tcp. Later versions of Apache like Apache 2.4.9 or later support unix domain socket and as such they allow for the configuration with unix domain socket instead of localhost tcp.

The following steps show the configuration in Ceph's configuration file i.e, `/etc/ceph/ceph.conf` and the gateway configuration file i.e, `/etc/httpd/conf.d/rgw.conf` (RPM-based distros) or `/etc/apache2/conf-available/rgw.conf` (Debian-based distros) with localhost tcp and through unix domain socket:

1. For distros with Apache 2.2 and early versions of Apache 2.4 that use localhost TCP and do not support Unix Domain Socket, append the following contents to `/etc/ceph/ceph.conf`:

```
[client.radosgw.gateway]
host = {hostname}
keyring = /etc/ceph/ceph.client.radosgw.keyring
rgw socket path = ""
log file = /var/log/radosgw/client.radosgw.gateway.log
rgw frontends = fastcgi socket_port=9000 socket_host=0.0.0.0
rgw print continue = false
```

2. Add the following content in the gateway configuration file:

   For Debian/Ubuntu add in `/etc/apache2/conf-available/rgw.conf`:

```
<VirtualHost *:80>
ServerName localhost
DocumentRoot /var/www/html

ErrorLog /var/log/apache2/rgw_error.log
CustomLog /var/log/apache2/rgw_access.log combined

# LogLevel debug

RewriteEngine On

RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization},L]

SetEnv proxy-nokeepalive 1

ProxyPass / fcgi://localhost:9000/

</VirtualHost>
```

   For CentOS/RHEL add in `/etc/httpd/conf.d/rgw.conf`:

```
<VirtualHost *:80>
ServerName localhost
DocumentRoot /var/www/html

ErrorLog /var/log/httpd/rgw_error.log
CustomLog /var/log/httpd/rgw_access.log combined

# LogLevel debug

RewriteEngine On
```

```
    RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization},L]

    SetEnv proxy-nokeepalive 1

    ProxyPass / fcgi://localhost:9000/

    </VirtualHost>
```

3. For distros with Apache 2.4.9 or later that support Unix Domain Socket, append the following configuration to `/etc/ceph/ceph.conf`:

```
[client.radosgw.gateway]
host = {hostname}
keyring = /etc/ceph/ceph.client.radosgw.keyring
rgw socket path = /var/run/ceph/ceph.radosgw.gateway.fastcgi.sock
log file = /var/log/radosgw/client.radosgw.gateway.log
rgw print continue = false
```

4. Add the following content in the gateway configuration file:

   For CentOS/RHEL add in `/etc/httpd/conf.d/rgw.conf`:

```
<VirtualHost *:80>
ServerName localhost
DocumentRoot /var/www/html

ErrorLog /var/log/httpd/rgw_error.log
CustomLog /var/log/httpd/rgw_access.log combined

# LogLevel debug

RewriteEngine On

RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization},L]

SetEnv proxy-nokeepalive 1

ProxyPass / unix:///var/run/ceph/ceph.radosgw.gateway.fastcgi.sock|fcgi://localhost:9000/

</VirtualHost>
```

   The latest version of Ubuntu i.e, 14.04 ships with `Apache 2.4.7` that does not have Unix Domain Socket support in it and as such it has to be configured with localhost tcp. The Unix Domain Socket support is available in `Apache 2.4.9` and later versions. A bug has been filed to backport the UDS support to `Apache 2.4.7` for `Ubuntu 14.04`. See: https://bugs.launchpad.net/ubuntu/+source/apache2/+bug/1411030

5. Generate a key for radosgw to use for authentication with the cluster.

```
ceph-authtool -C -n client.radosgw.gateway --gen-key /etc/ceph/keyring.radosgw.gateway
ceph-authtool -n client.radosgw.gateway --cap mon 'allow rw' --cap osd 'allow rwx' /etc/ceph/key
```

6. Add the key to the auth entries.

```
ceph auth add client.radosgw.gateway --in-file=keyring.radosgw.gateway
```

7. Start Apache and radosgw.

   Debian/Ubuntu:

```
    sudo /etc/init.d/apache2 start
    sudo /etc/init.d/radosgw start
```

CentOS/RHEL:

```
    sudo apachectl start
    sudo /etc/init.d/ceph-radosgw start
```

7.12.5 Usage Logging

radosgw maintains an asynchronous usage log. It accumulates statistics about user operations and flushes it periodically. The logs can be accessed and managed through **radosgw-admin**.

The information that is being logged contains total data transfer, total operations, and total successful operations. The data is being accounted in an hourly resolution under the bucket owner, unless the operation was done on the service (e.g., when listing a bucket) in which case it is accounted under the operating user.

Following is an example configuration:

```
[client.radosgw.gateway]
    rgw enable usage log = true
    rgw usage log tick interval = 30
    rgw usage log flush threshold = 1024
    rgw usage max shards = 32
    rgw usage max user shards = 1
```

The total number of shards determines how many total objects hold the usage log information. The per-user number of shards specify how many objects hold usage information for a single user. The tick interval configures the number of seconds between log flushes, and the flush threshold specify how many entries can be kept before resorting to synchronous flush.

7.12.6 Availability

radosgw is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

7.12.7 See also

ceph(8) radosgw-admin(8)

7.13 radosgw-admin – rados REST gateway user administration utility

7.13.1 Synopsis

radosgw-admin *command* [*options* ...]

7.13.2 Description

radosgw-admin is a RADOS gateway user administration utility. It allows creating and modifying users.

7.13.3 Commands

`radosgw-admin` utility uses many commands for administration purpose which are as follows:

`user create` Create a new user.

`user modify` Modify a user.

`user info` Display information of a user, and any potentially available subusers and keys.

`user rm` Remove a user.

`user suspend` Suspend a user.

`user enable` Re-enable user after suspension.

`user check` Check user info.

`user stats` Show user stats as accounted by quota subsystem.

`caps add` Add user capabilities.

`caps rm` Remove user capabilities.

`subuser create` Create a new subuser (primarily useful for clients using the Swift API).

`subuser modify` Modify a subuser.

`subuser rm` Remove a subuser.

`key create` Create access key.

`key rm` Remove access key.

`bucket list` List all buckets.

`bucket link` Link bucket to specified user.

`bucket unlink` Unlink bucket from specified user.

`bucket stats` Returns bucket statistics.

`bucket rm` Remove a bucket.

`bucket check` Check bucket index.

`object rm` Remove an object.

`object unlink` Unlink object from bucket index.

`quota set` Set quota params.

`quota enable` Enable quota.

`quota disable` Disable quota.

`region get` Show region info.

`regions list` List all regions set on this cluster.

`region set` Set region info (requires infile).

`region default` Set default region.

`region-map get` Show region-map.

`region-map set` Set region-map (requires infile).

`zone get` Show zone cluster params.

`zone set` Set zone cluster params (requires infile).

zone list List all zones set on this cluster.

pool add Add an existing pool for data placement.

pool rm Remove an existing pool from data placement set.

pools list List placement active set.

policy Display bucket/object policy.

log list List log objects.

log show Dump a log from specific object or (bucket + date + bucket-id).

log rm Remove log object.

usage show Show the usage information (with optional user and date range).

usage trim Trim usage information (with optional user and date range).

temp remove Remove temporary objects that were created up to specified date (and optional time).

gc list Dump expired garbage collection objects (specify –include-all to list all entries, including unexpired).

gc process Manually process garbage.

metadata get Get metadata info.

metadata put Put metadata info.

metadata rm Remove metadata info.

metadata list List metadata info.

mdlog list List metadata log.

mdlog trim Trim metadata log.

bilog list List bucket index log.

bilog trim Trim bucket index log (use start-marker, end-marker).

datalog list List data log.

datalog trim Trim data log.

opstate list List stateful operations entries (use client_id, op_id, object).

opstate set Set state on an entry (use client_id, op_id, object, state).

opstate renew Renew state on an entry (use client_id, op_id, object).

opstate rm Remove entry (use client_id, op_id, object).

replicalog get Get replica metadata log entry.

replicalog delete Delete replica metadata log entry.

7.13.4 Options

-c `ceph.conf`, **--conf**=`ceph.conf`
: Use `ceph.conf` configuration file instead of the default `/etc/ceph/ceph.conf` to determine monitor addresses during startup.

-m `monaddress[:port]`
: Connect to specified monitor (instead of looking through ceph.conf).

--uid=uid
The radosgw user ID.

--subuser=<name>
Name of the subuser.

--email=email
The e-mail address of the user.

--display-name=name
Configure the display name of the user.

--access-key=<key>
S3 access key.

--gen-access-key
Generate random access key (for S3).

--secret=secret
The secret associated with a given key.

--gen-secret
Generate random secret key.

--key-type=<type>
key type, options are: swift, S3.

--temp-url-key[-2]=<key>
Temporary url key.

--system
Set the system flag on the user.

--bucket=bucket
Specify the bucket name.

--object=object
Specify the object name.

--date=yyyy-mm-dd
The date needed for some commands.

--start-date=yyyy-mm-dd
The start date needed for some commands.

--end-date=yyyy-mm-dd
The end date needed for some commands.

--shard-id=<shard-id>
Optional for mdlog list. Required for `mdlog trim`, `replica mdlog get/delete`, `replica datalog get/delete`.

--auth-uid=auid
The librados auid.

--purge-data
Remove user data before user removal.

--purge-keys
When specified, subuser removal will also purge all the subuser keys.

--purge-objects
Remove all objects before bucket removal.

--lazy-remove
Defer removal of object tail.

--metadata-key=<key>
Key to retrieve metadata from with `metadata get`.

--rgw-region=<region>
Region in which radosgw is running.

--rgw-zone=<zone>
Zone in which radosgw is running.

--fix
Besides checking bucket index, will also fix it.

--check-objects
bucket check: Rebuilds bucket index according to actual objects state.

--format=<format>
Specify output format for certain operations: xml, json.

--sync-stats
Option to 'user stats', update user stats with current stats reported by user's buckets indexes.

--show-log-entries=<flag>
Enable/disable dump of log entries on log show.

--show-log-sum=<flag>
Enable/disable dump of log summation on log show.

--skip-zero-entries
Log show only dumps entries that don't have zero value in one of the numeric field.

--infile
Specify a file to read in when setting data.

--state=<state string>
Specify a state for the opstate set command.

--replica-log-type
Replica log type (metadata, data, bucket), required for replica log operations.

--categories=<list>
Comma separated list of categories, used in usage show.

--caps=<caps>
List of caps (e.g., "usage=read, write; user=read".

--yes-i-really-mean-it
Required for certain operations.

7.13.5 Quota Options

--max-objects
Specify max objects (negative value to disable).

--max-size
Specify max size (in bytes, negative value to disable).

--quota-scope
Scope of quota (bucket, user).

7.13.6 Examples

Generate a new user:

```
$ radosgw-admin user create --display-name="johnny rotten" --uid=johnny
{ "user_id": "johnny",
  "rados_uid": 0,
  "display_name": "johnny rotten",
  "email": "",
  "suspended": 0,
  "subusers": [],
  "keys": [
        { "user": "johnny",
          "access_key": "TCICW53D9BQ2VGC46I44",
          "secret_key": "tfm9aHMI8X76L3UdgE+ZQaJag1vJQmE6HDb5Lbrz"}],
  "swift_keys": []}
```

Remove a user:

```
$ radosgw-admin user rm --uid=johnny
```

Remove a user and all associated buckets with their contents:

```
$ radosgw-admin user rm --uid=johnny --purge-data
```

Remove a bucket:

```
$ radosgw-admin bucket unlink --bucket=foo
```

Show the logs of a bucket from April 1st, 2012:

```
$ radosgw-admin log show --bucket=foo --date=2012-04-01
```

Show usage information for user from March 1st to (but not including) April 1st, 2012:

```
$ radosgw-admin usage show --uid=johnny \
              --start-date=2012-03-01 --end-date=2012-04-01
```

Show only summary of usage information for all users:

```
$ radosgw-admin usage show --show-log-entries=false
```

Trim usage information for user until March 1st, 2012:

```
$ radosgw-admin usage trim --uid=johnny --end-date=2012-04-01
```

7.13.7 Availability

radosgw-admin is part of Ceph, a massively scalable, open-source, distributed storage system. Please refer to the Ceph documentation at http://ceph.com/docs for more information.

7.13.8 See also

ceph(8) radosgw(8)

CHAPTER

EIGHT

API DOCUMENTATION

8.1 Ceph Storage Cluster APIs

See Ceph Storage Cluster APIs.

8.2 Ceph Filesystem APIs

See libcephfs (javadoc).

8.3 Ceph Block Device APIs

See librbdpy.

8.4 Calamari APIs

See Calamari API.

8.5 Ceph Object Store APIs

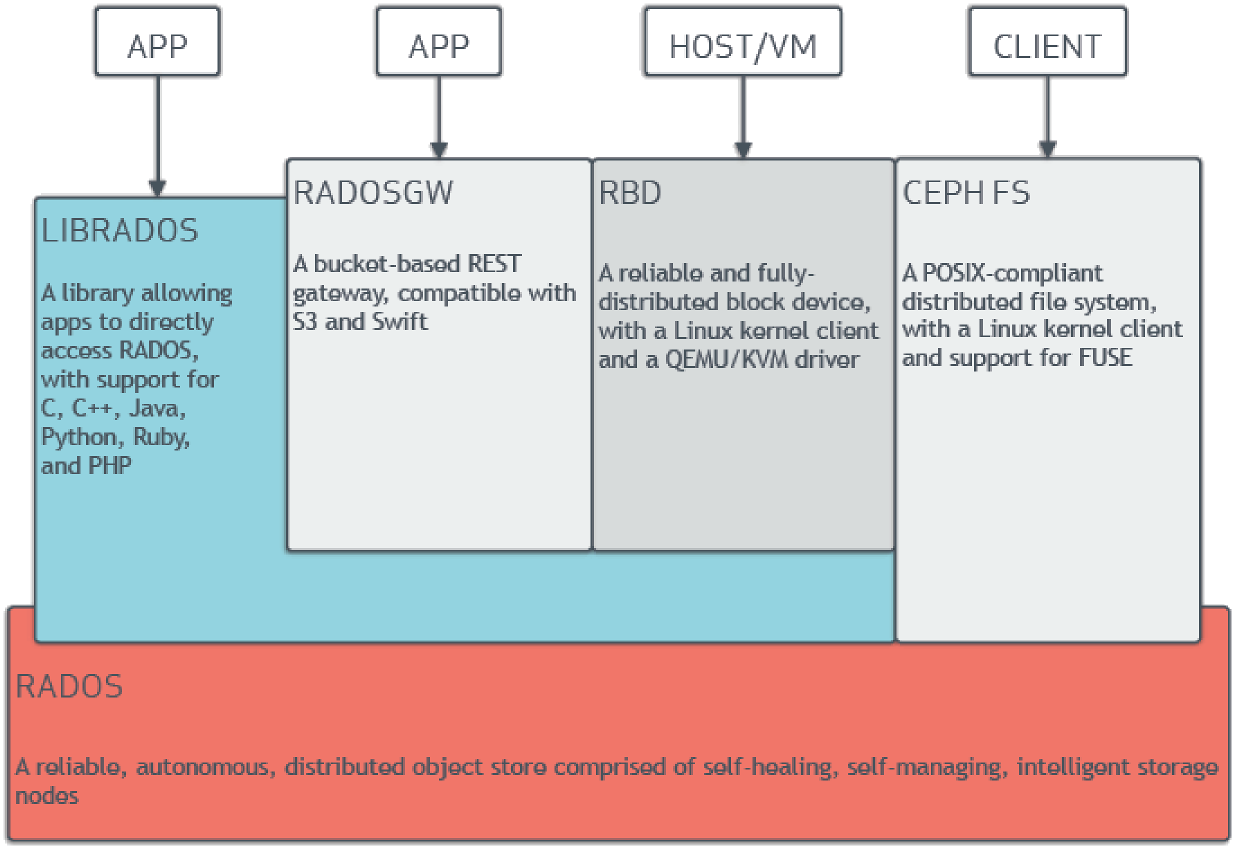
- See S3-compatible API.
- See Swift-compatible API.
- See Admin Ops API.

CHAPTER

NINE

ARCHITECTURE

Ceph uniquely delivers **object, block, and file storage** in one unified system. Ceph is highly reliable, easy to manage, and free. The power of Ceph can transform your company's IT infrastructure and your ability to manage vast amounts of data. Ceph delivers extraordinary scalability–thousands of clients accessing petabytes to exabytes of data. A *Ceph Node* leverages commodity hardware and intelligent daemons, and a *Ceph Storage Cluster* accommodates large numbers of nodes, which communicate with each other to replicate and redistribute data dynamically.

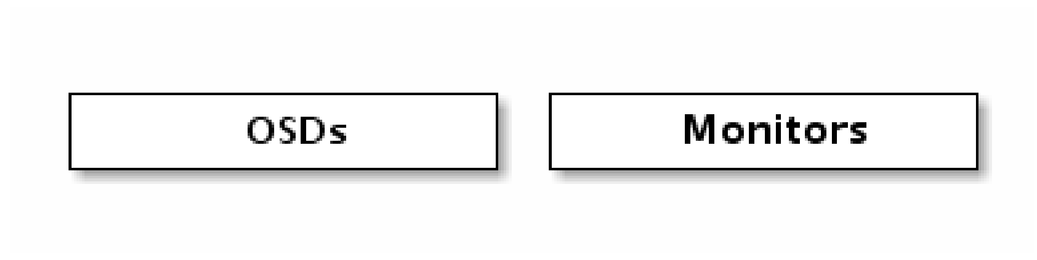

APP
APP
HOST/VM
CLIENT
LIBRADOS
A library allowing apps to directly access RADOS, with support for C, C++, Java, Python, Ruby, and PHP
RADOSGW
A bucket-based REST gateway, compatible with S3 and Swift
RBD
A reliable and fully-distributed block device, with a Linux kernel client and a QEMU/KVM driver
CEPH FS
A POSIX-compliant distributed file system, with a Linux kernel client and support for FUSE
RADOS
A reliable, autonomous, distributed object store comprised of self-healing, self-managing, intelligent storage nodes

9.1 The Ceph Storage Cluster

Ceph provides an infinitely scalable *Ceph Storage Cluster* based upon RADOS, which you can read about in RADOS - A Scalable, Reliable Storage Service for Petabyte-scale Storage Clusters.

A Ceph Storage Cluster consists of two types of daemons:

- *Ceph Monitor*
- *Ceph OSD Daemon*



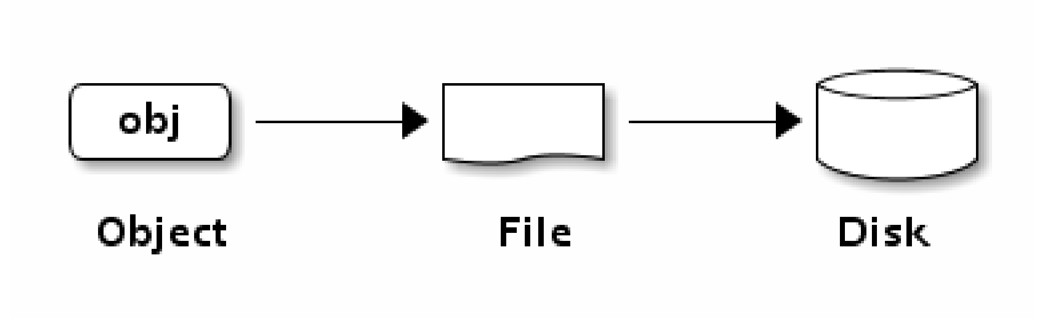
A Ceph Monitor maintains a master copy of the cluster map. A cluster of Ceph monitors ensures high availability should a monitor daemon fail. Storage cluster clients retrieve a copy of the cluster map from the Ceph Monitor.

A Ceph OSD Daemon checks its own state and the state of other OSDs and reports back to monitors.

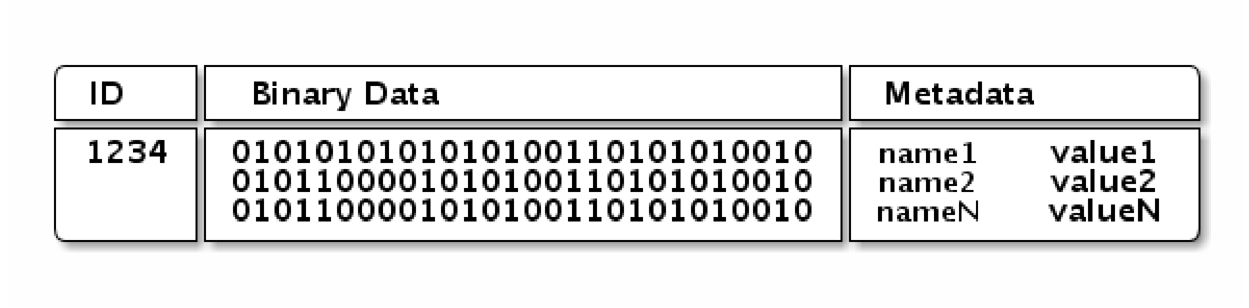
Storage cluster clients and each *Ceph OSD Daemon* use the CRUSH algorithm to efficiently compute information about data location, instead of having to depend on a central lookup table. Ceph's high-level features include providing a native interface to the Ceph Storage Cluster via `librados`, and a number of service interfaces built on top of `librados`.

9.1.1 Storing Data

The Ceph Storage Cluster receives data from *Ceph Clients*–whether it comes through a *Ceph Block Device*, *Ceph Object Storage*, the *Ceph Filesystem* or a custom implementation you create using `librados`–and it stores the data as objects. Each object corresponds to a file in a filesystem, which is stored on an *Object Storage Device*. Ceph OSD Daemons handle the read/write operations on the storage disks.



Ceph OSD Daemons store all data as objects in a flat namespace (e.g., no hierarchy of directories). An object has an identifier, binary data, and metadata consisting of a set of name/value pairs. The semantics are completely up to *Ceph Clients*. For example, CephFS uses metadata to store file attributes such as the file owner, created date, last modified date, and so forth.



ID	Binary Data	Metadata
1234	0101010101010100110101010010 0101100001010100110101010010 0101100001010100110101010010	name1 value1 name2 value2 nameN valueN

Note: An object ID is unique across the entire cluster, not just the local filesystem.

9.1.2 Scalability and High Availability

In traditional architectures, clients talk to a centralized component (e.g., a gateway, broker, API, facade, etc.), which acts as a single point of entry to a complex subsystem. This imposes a limit to both performance and scalability, while introducing a single point of failure (i.e., if the centralized component goes down, the whole system goes down, too).

Ceph eliminates the centralized gateway to enable clients to interact with Ceph OSD Daemons directly. Ceph OSD Daemons create object replicas on other Ceph Nodes to ensure data safety and high availability. Ceph also uses a cluster of monitors to ensure high availability. To eliminate centralization, Ceph uses an algorithm called CRUSH.

CRUSH Introduction

Ceph Clients and Ceph OSD Daemons both use the CRUSH algorithm to efficiently compute information about object location, instead of having to depend on a central lookup table. CRUSH provides a better data management mechanism compared to older approaches, and enables massive scale by cleanly distributing the work to all the clients and OSD daemons in the cluster. CRUSH uses intelligent data replication to ensure resiliency, which is better suited to hyper-scale storage. The following sections provide additional details on how CRUSH works. For a detailed discussion of CRUSH, see CRUSH - Controlled, Scalable, Decentralized Placement of Replicated Data.

Cluster Map

Ceph depends upon Ceph Clients and Ceph OSD Daemons having knowledge of the cluster topology, which is inclusive of 5 maps collectively referred to as the "Cluster Map":

1. **The Monitor Map:** Contains the cluster `fsid`, the position, name address and port of each monitor. It also indicates the current epoch, when the map was created, and the last time it changed. To view a monitor map, execute `ceph mon dump`.
2. **The OSD Map:** Contains the cluster `fsid`, when the map was created and last modified, a list of pools, replica sizes, PG numbers, a list of OSDs and their status (e.g., `up`, `in`). To view an OSD map, execute `ceph osd dump`.
3. **The PG Map:** Contains the PG version, its time stamp, the last OSD map epoch, the full ratios, and details on each placement group such as the PG ID, the *Up Set*, the *Acting Set*, the state of the PG (e.g., `active + clean`), and data usage statistics for each pool.
4. **The CRUSH Map:** Contains a list of storage devices, the failure domain hierarchy (e.g., device, host, rack, row, room, etc.), and rules for traversing the hierarchy when storing data. To view a CRUSH map, execute `ceph osd getcrushmap -o {filename}`; then, decompile it by executing `crushtool -d {comp-crushmap-filename} -o {decomp-crushmap-filename}`. You can view the decompiled map in a text editor or with `cat`.
5. **The MDS Map:** Contains the current MDS map epoch, when the map was created, and the last time it changed. It also contains the pool for storing metadata, a list of metadata servers, and which metadata servers are `up` and `in`. To view an MDS map, execute `ceph mds dump`.

Each map maintains an iterative history of its operating state changes. Ceph Monitors maintain a master copy of the cluster map including the cluster members, state, changes, and the overall health of the Ceph Storage Cluster.

High Availability Monitors

Before Ceph Clients can read or write data, they must contact a Ceph Monitor to obtain the most recent copy of the cluster map. A Ceph Storage Cluster can operate with a single monitor; however, this introduces a single point of failure (i.e., if the monitor goes down, Ceph Clients cannot read or write data).

For added reliability and fault tolerance, Ceph supports a cluster of monitors. In a cluster of monitors, latency and other faults can cause one or more monitors to fall behind the current state of the cluster. For this reason, Ceph must have agreement among various monitor instances regarding the state of the cluster. Ceph always uses a majority of monitors (e.g., 1, 2:3, 3:5, 4:6, etc.) and the Paxos algorithm to establish a consensus among the monitors about the current state of the cluster.

For details on configuring monitors, see the Monitor Config Reference.

High Availability Authentication

To identify users and protect against man-in-the-middle attacks, Ceph provides its `cephx` authentication system to authenticate users and daemons.

Note: The `cephx` protocol does not address data encryption in transport (e.g., SSL/TLS) or encryption at rest.

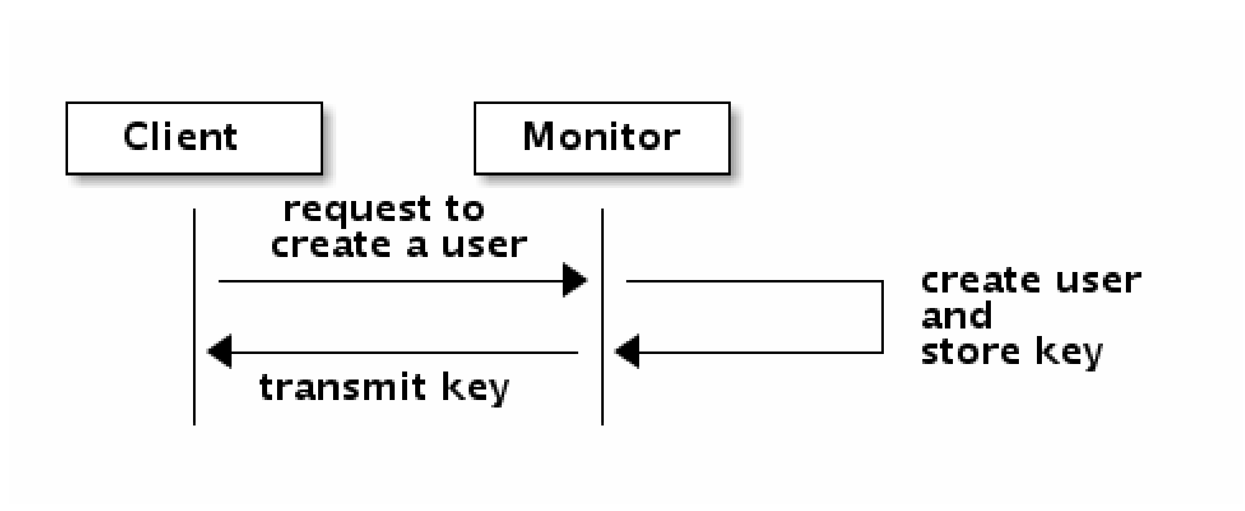
Cephx uses shared secret keys for authentication, meaning both the client and the monitor cluster have a copy of the client's secret key. The authentication protocol is such that both parties are able to prove to each other they have a copy of the key without actually revealing it. This provides mutual authentication, which means the cluster is sure the user possesses the secret key, and the user is sure that the cluster has a copy of the secret key.

A key scalability feature of Ceph is to avoid a centralized interface to the Ceph object store, which means that Ceph clients must be able to interact with OSDs directly. To protect data, Ceph provides its `cephx` authentication system, which authenticates users operating Ceph clients. The `cephx` protocol operates in a manner with behavior similar to Kerberos.

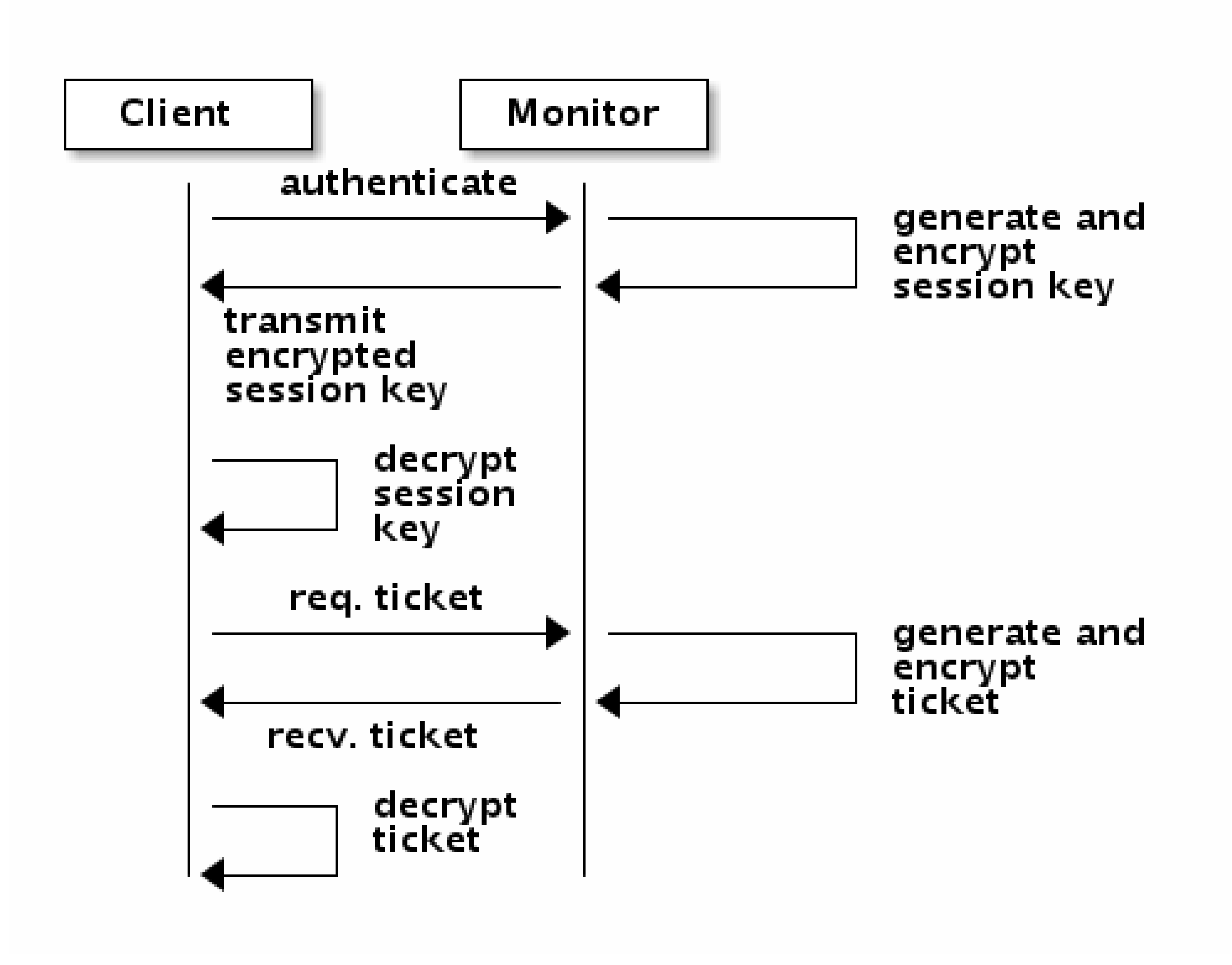
A user/actor invokes a Ceph client to contact a monitor. Unlike Kerberos, each monitor can authenticate users and distribute keys, so there is no single point of failure or bottleneck when using `cephx`. The monitor returns an authentication data structure similar to a Kerberos ticket that contains a session key for use in obtaining Ceph services. This session key is itself encrypted with the user's permanent secret key, so that only the user can request services from the Ceph monitor(s). The client then uses the session key to request its desired services from the monitor, and the monitor provides the client with a ticket that will authenticate the client to the OSDs that actually handle data. Ceph monitors and OSDs share a secret, so the client can use the ticket provided by the monitor with any OSD or metadata server in the cluster. Like Kerberos, `cephx` tickets expire, so an attacker cannot use an expired ticket or session key obtained surreptitiously. This form of authentication will prevent attackers with access to the communications medium from either creating bogus messages under another user's identity or altering another user's legitimate messages, as long as the user's secret key is not divulged before it expires.

To use `cephx`, an administrator must set up users first. In the following diagram, the `client.admin` user invokes `ceph auth get-or-create-key` from the command line to generate a username and secret key. Ceph's `auth` subsystem generates the username and key, stores a copy with the monitor(s) and transmits the user's secret back to the `client.admin` user. This means that the client and the monitor share a secret key.

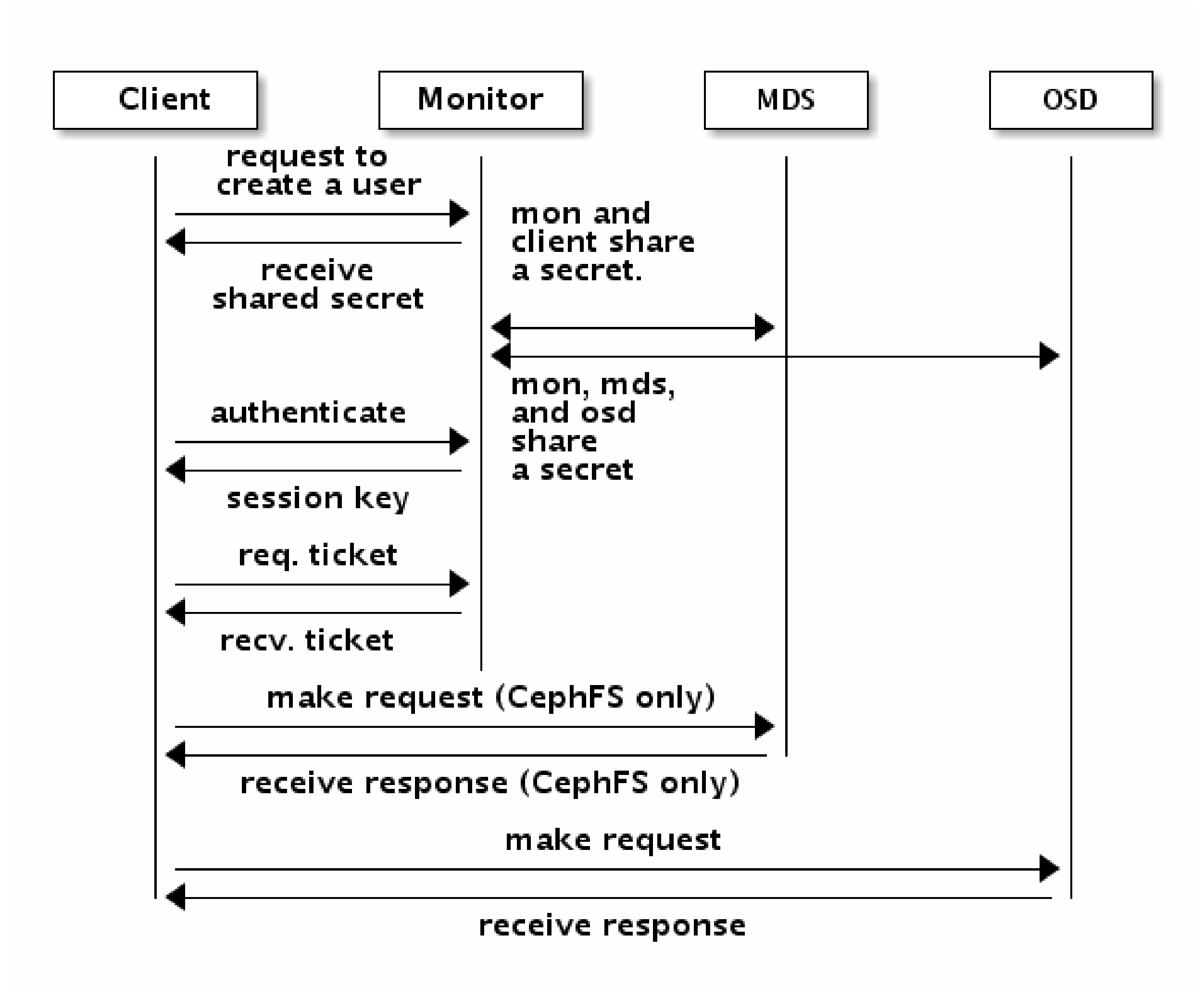
Note: The `client.admin` user must provide the user ID and secret key to the user in a secure manner.



To authenticate with the monitor, the client passes in the user name to the monitor, and the monitor generates a session key and encrypts it with the secret key associated to the user name. Then, the monitor transmits the encrypted ticket back to the client. The client then decrypts the payload with the shared secret key to retrieve the session key. The session key identifies the user for the current session. The client then requests a ticket on behalf of the user signed by the session key. The monitor generates a ticket, encrypts it with the user's secret key and transmits it back to the client. The client decrypts the ticket and uses it to sign requests to OSDs and metadata servers throughout the cluster.


Client
Monitor
authenticate
generate and encrypt session key
transmit encrypted session key
decrypt session key
req. ticket
generate and encrypt ticket
recv. ticket
decrypt ticket

The `cephx` protocol authenticates ongoing communications between the client machine and the Ceph servers. Each message sent between a client and server, subsequent to the initial authentication, is signed using a ticket that the monitors, OSDs and metadata servers can verify with their shared secret.



The protection offered by this authentication is between the Ceph client and the Ceph server hosts. The authentication is not extended beyond the Ceph client. If the user accesses the Ceph client from a remote host, Ceph authentication is not applied to the connection between the user's host and the client host.

For configuration details, see Cephx Config Guide. For user management details, see User Management.

Smart Daemons Enable Hyperscale

In many clustered architectures, the primary purpose of cluster membership is so that a centralized interface knows which nodes it can access. Then the centralized interface provides services to the client through a double dispatch– which is a **huge** bottleneck at the petabyte-to-exabyte scale.

Ceph eliminates the bottleneck: Ceph's OSD Daemons AND Ceph Clients are cluster aware. Like Ceph clients, each Ceph OSD Daemon knows about other Ceph OSD Daemons in the cluster. This enables Ceph OSD Daemons to interact directly with other Ceph OSD Daemons and Ceph monitors. Additionally, it enables Ceph Clients to interact directly with Ceph OSD Daemons.

The ability of Ceph Clients, Ceph Monitors and Ceph OSD Daemons to interact with each other means that Ceph OSD

Daemons can utilize the CPU and RAM of the Ceph nodes to easily perform tasks that would bog down a centralized server. The ability to leverage this computing power leads to several major benefits:

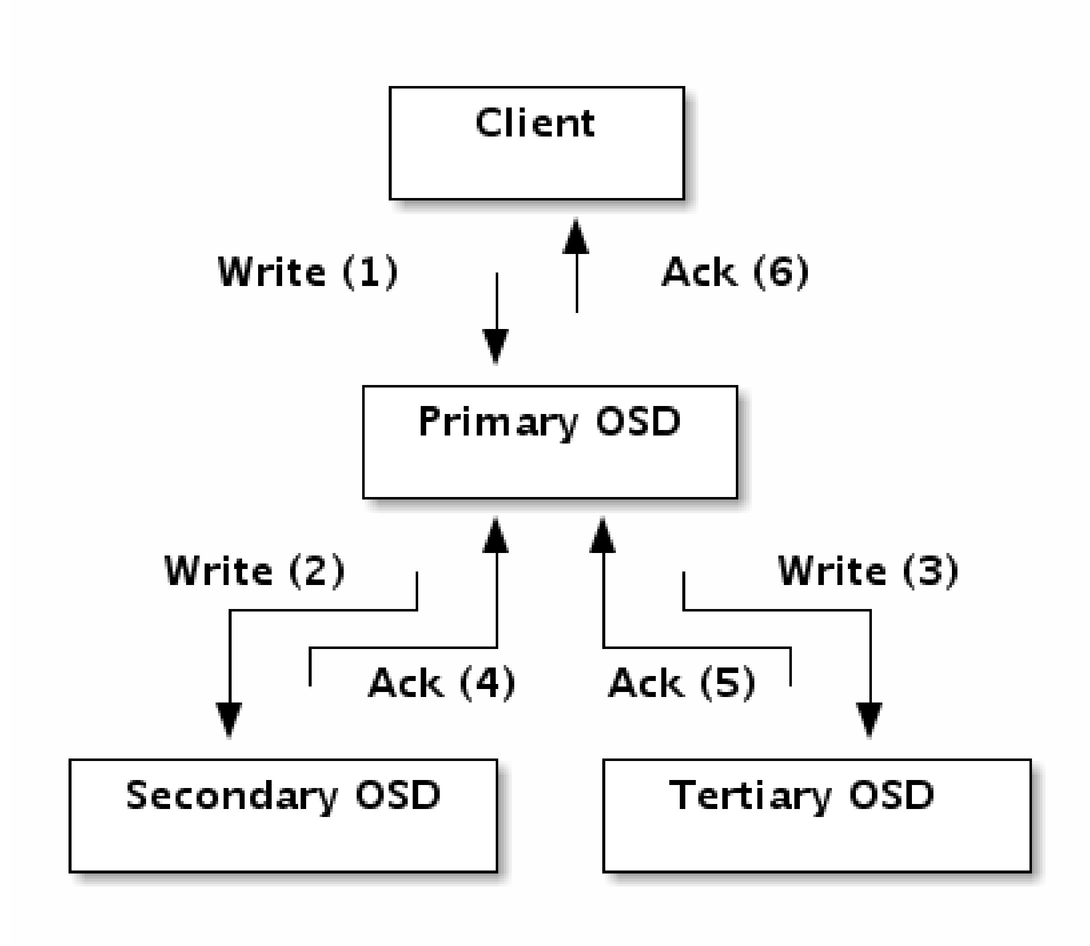
1. **OSDs Service Clients Directly:** Since any network device has a limit to the number of concurrent connections it can support, a centralized system has a low physical limit at high scales. By enabling Ceph Clients to contact Ceph OSD Daemons directly, Ceph increases both performance and total system capacity simultaneously, while removing a single point of failure. Ceph Clients can maintain a session when they need to, and with a particular Ceph OSD Daemon instead of a centralized server.

2. **OSD Membership and Status**: Ceph OSD Daemons join a cluster and report on their status. At the lowest level, the Ceph OSD Daemon status is `up` or `down` reflecting whether or not it is running and able to service Ceph Client requests. If a Ceph OSD Daemon is `down` and `in` the Ceph Storage Cluster, this status may indicate the failure of the Ceph OSD Daemon. If a Ceph OSD Daemon is not running (e.g., it crashes), the Ceph OSD Daemon cannot notify the Ceph Monitor that it is `down`. The Ceph Monitor can ping a Ceph OSD Daemon periodically to ensure that it is running. However, Ceph also empowers Ceph OSD Daemons to determine if a neighboring OSD is `down`, to update the cluster map and to report it to the Ceph monitor(s). This means that Ceph monitors can remain light weight processes. See Monitoring OSDs and Heartbeats for additional details.

3. **Data Scrubbing:** As part of maintaining data consistency and cleanliness, Ceph OSD Daemons can scrub objects within placement groups. That is, Ceph OSD Daemons can compare object metadata in one placement group with its replicas in placement groups stored on other OSDs. Scrubbing (usually performed daily) catches bugs or filesystem errors. Ceph OSD Daemons also perform deeper scrubbing by comparing data in objects bit-for-bit. Deep scrubbing (usually performed weekly) finds bad sectors on a drive that weren't apparent in a light scrub. See Data Scrubbing for details on configuring scrubbing.

4. **Replication:** Like Ceph Clients, Ceph OSD Daemons use the CRUSH algorithm, but the Ceph OSD Daemon uses it to compute where replicas of objects should be stored (and for rebalancing). In a typical write scenario, a client uses the CRUSH algorithm to compute where to store an object, maps the object to a pool and placement group, then looks at the CRUSH map to identify the primary OSD for the placement group.

   The client writes the object to the identified placement group in the primary OSD. Then, the primary OSD with its own copy of the CRUSH map identifies the secondary and tertiary OSDs for replication purposes, and replicates the object to the appropriate placement groups in the secondary and tertiary OSDs (as many OSDs as additional replicas), and responds to the client once it has confirmed the object was stored successfully.



With the ability to perform data replication, Ceph OSD Daemons relieve Ceph clients from that duty, while ensuring high data availability and data safety.

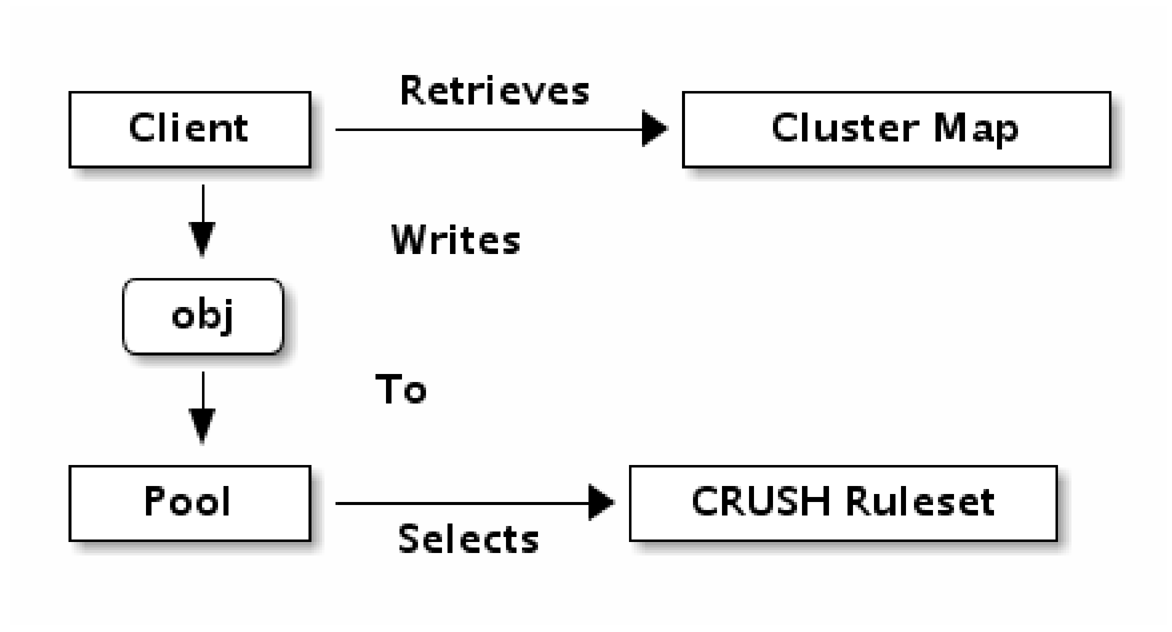
9.1.3 Dynamic Cluster Management

In the *Scalability and High Availability* section, we explained how Ceph uses CRUSH, cluster awareness and intelligent daemons to scale and maintain high availability. Key to Ceph's design is the autonomous, self-healing, and intelligent Ceph OSD Daemon. Let's take a deeper look at how CRUSH works to enable modern cloud storage infrastructures to place data, rebalance the cluster and recover from faults dynamically.

About Pools

The Ceph storage system supports the notion of 'Pools', which are logical partitions for storing objects.

Ceph Clients retrieve a *Cluster Map* from a Ceph Monitor, and write objects to pools. The pool's `size` or number of replicas, the CRUSH ruleset and the number of placement groups determine how Ceph will place the data.



Pools set at least the following parameters:

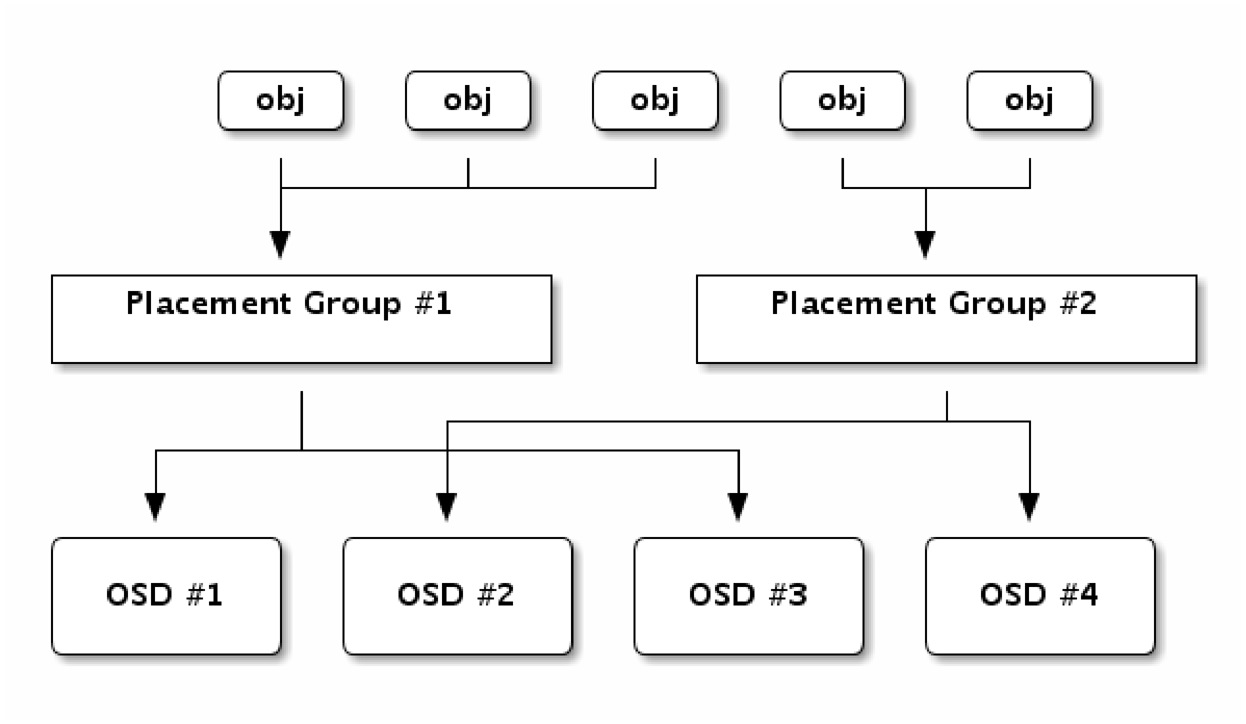
- Ownership/Access to Objects
- The Number of Placement Groups, and
- The CRUSH Ruleset to Use.

See Set Pool Values for details.

Mapping PGs to OSDs

Each pool has a number of placement groups. CRUSH maps PGs to OSDs dynamically. When a Ceph Client stores objects, CRUSH will map each object to a placement group.

Mapping objects to placement groups creates a layer of indirection between the Ceph OSD Daemon and the Ceph Client. The Ceph Storage Cluster must be able to grow (or shrink) and rebalance where it stores objects dynamically. If the Ceph Client "knew" which Ceph OSD Daemon had which object, that would create a tight coupling between the Ceph Client and the Ceph OSD Daemon. Instead, the CRUSH algorithm maps each object to a placement group and then maps each placement group to one or more Ceph OSD Daemons. This layer of indirection allows Ceph to rebalance dynamically when new Ceph OSD Daemons and the underlying OSD devices come online. The following diagram depicts how CRUSH maps objects to placement groups, and placement groups to OSDs.



With a copy of the cluster map and the CRUSH algorithm, the client can compute exactly which OSD to use when reading or writing a particular object.

Calculating PG IDs

When a Ceph Client binds to a Ceph Monitor, it retrieves the latest copy of the *Cluster Map*. With the cluster map, the client knows about all of the monitors, OSDs, and metadata servers in the cluster. **However, it doesn't know anything about object locations.**

Object locations get computed.

The only input required by the client is the object ID and the pool. It's simple: Ceph stores data in named pools (e.g., "liverpool"). When a client wants to store a named object (e.g., "john," "paul," "george," "ringo", etc.) it calculates a placement group using the object name, a hash code, the number of PGs in the pool and the pool name. Ceph clients use the following steps to compute PG IDs.

1. The client inputs the pool ID and the object ID. (e.g., pool = "liverpool" and object-id = "john")
2. Ceph takes the object ID and hashes it.
3. Ceph calculates the hash modulo the number of PGs. (e.g., `58`) to get a PG ID.
4. Ceph gets the pool ID given the pool name (e.g., "liverpool" = `4`)
5. Ceph prepends the pool ID to the PG ID (e.g., `4.58`).

Computing object locations is much faster than performing object location query over a chatty session. The CRUSH algorithm allows a client to compute where objects *should* be stored, and enables the client to contact the primary OSD to store or retrieve the objects.

Peering and Sets

In previous sections, we noted that Ceph OSD Daemons check each others heartbeats and report back to the Ceph Monitor. Another thing Ceph OSD daemons do is called 'peering', which is the process of bringing all of the OSDs that store a Placement Group (PG) into agreement about the state of all of the objects (and their metadata) in that PG. In fact, Ceph OSD Daemons Report Peering Failure to the Ceph Monitors. Peering issues usually resolve themselves; however, if the problem persists, you may need to refer to the Troubleshooting Peering Failure section.

Note: Agreeing on the state does not mean that the PGs have the latest contents.

The Ceph Storage Cluster was designed to store at least two copies of an object (i.e., `size = 2`), which is the minimum requirement for data safety. For high availability, a Ceph Storage Cluster should store more than two copies of an object (e.g., `size = 3` and `min size = 2`) so that it can continue to run in a `degraded` state while maintaining data safety.

Referring back to the diagram in *Smart Daemons Enable Hyperscale*, we do not name the Ceph OSD Daemons specifically (e.g., `osd.0`, `osd.1`, etc.), but rather refer to them as *Primary*, *Secondary*, and so forth. By convention, the *Primary* is the first OSD in the *Acting Set*, and is responsible for coordinating the peering process for each placement group where it acts as the *Primary*, and is the **ONLY** OSD that that will accept client-initiated writes to objects for a given placement group where it acts as the *Primary*.

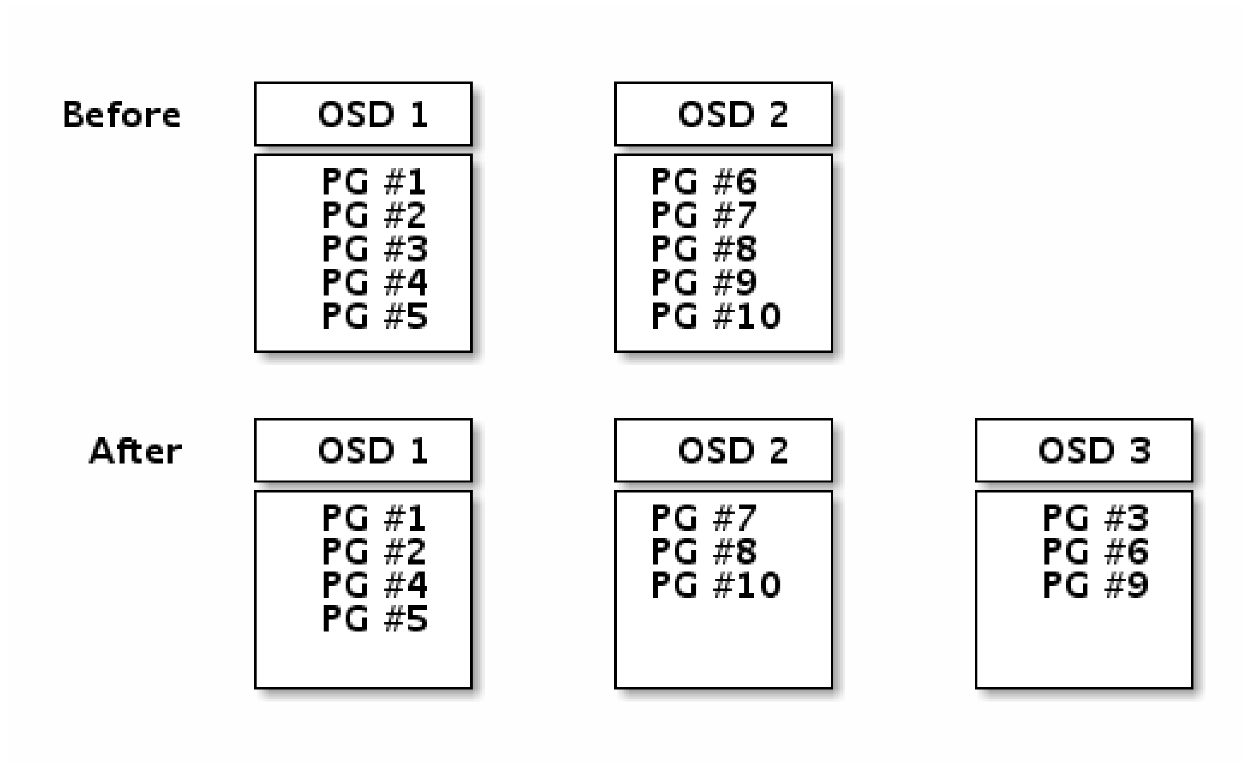
When a series of OSDs are responsible for a placement group, that series of OSDs, we refer to them as an *Acting Set*. An *Acting Set* may refer to the Ceph OSD Daemons that are currently responsible for the placement group, or the Ceph OSD Daemons that were responsible for a particular placement group as of some epoch.

The Ceph OSD daemons that are part of an *Acting Set* may not always be `up`. When an OSD in the *Acting Set* is `up`, it is part of the *Up Set*. The *Up Set* is an important distinction, because Ceph can remap PGs to other Ceph OSD Daemons when an OSD fails.

Note: In an *Acting Set* for a PG containing `osd.25`, `osd.32` and `osd.61`, the first OSD, `osd.25`, is the *Primary*. If that OSD fails, the Secondary, `osd.32`, becomes the *Primary*, and `osd.25` will be removed from the *Up Set*.

Rebalancing

When you add a Ceph OSD Daemon to a Ceph Storage Cluster, the cluster map gets updated with the new OSD. Referring back to *Calculating PG IDs*, this changes the cluster map. Consequently, it changes object placement, because it changes an input for the calculations. The following diagram depicts the rebalancing process (albeit rather crudely, since it is substantially less impactful with large clusters) where some, but not all of the PGs migrate from existing OSDs (OSD 1, and OSD 2) to the new OSD (OSD 3). Even when rebalancing, CRUSH is stable. Many of the placement groups remain in their original configuration, and each OSD gets some added capacity, so there are no load spikes on the new OSD after rebalancing is complete.



Data Consistency

As part of maintaining data consistency and cleanliness, Ceph OSDs can also scrub objects within placement groups. That is, Ceph OSDs can compare object metadata in one placement group with its replicas in placement groups stored in other OSDs. Scrubbing (usually performed daily) catches OSD bugs or filesystem errors. OSDs can also perform deeper scrubbing by comparing data in objects bit-for-bit. Deep scrubbing (usually performed weekly) finds bad sectors on a disk that weren't apparent in a light scrub.

See Data Scrubbing for details on configuring scrubbing.

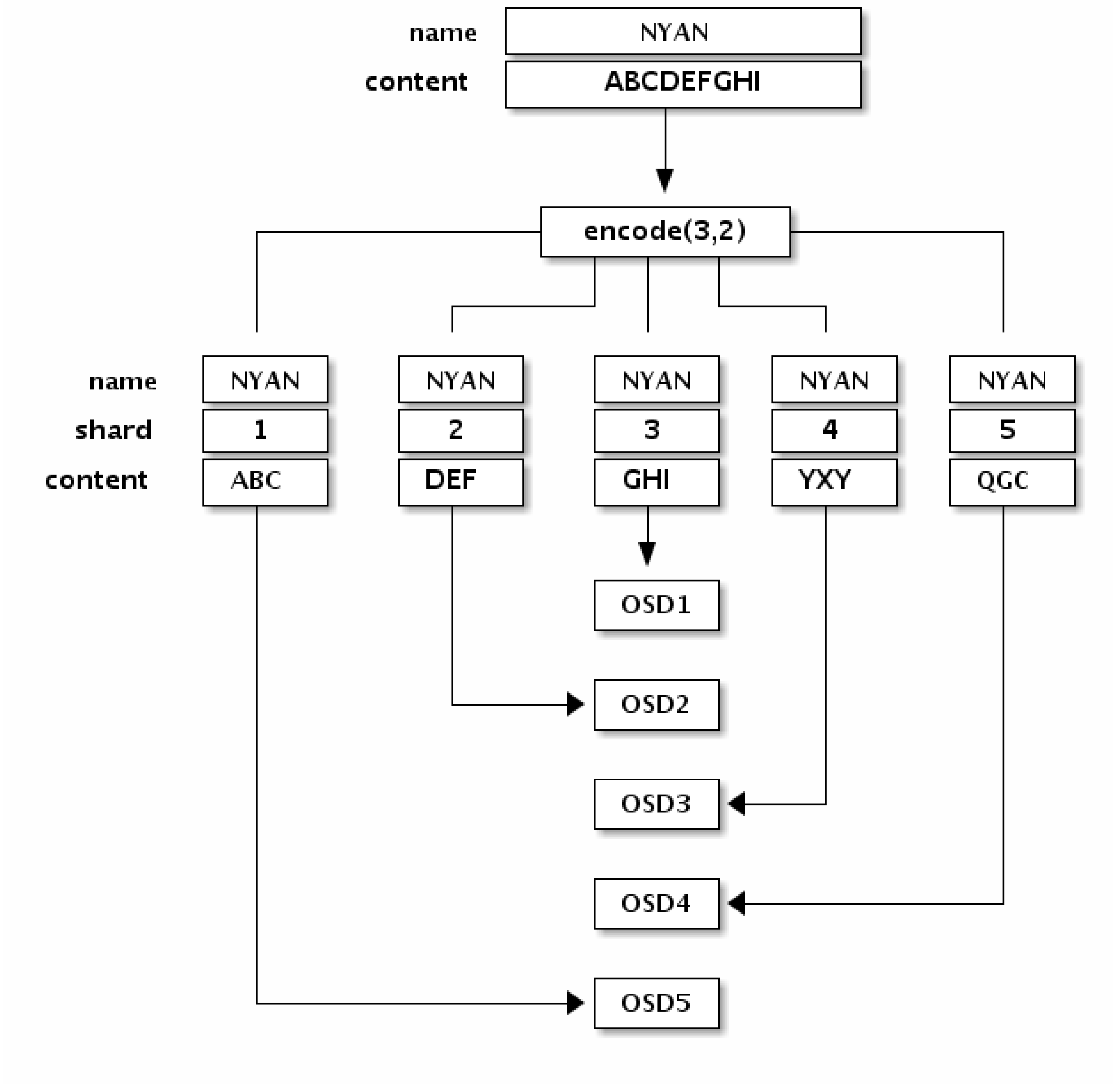
9.1.4 Erasure Coding

An erasure coded pool stores each object as `K+M` chunks. It is divided into `K` data chunks and `M` coding chunks. The pool is configured to have a size of `K+M` so that each chunk is stored in an OSD in the acting set. The rank of the chunk is stored as an attribute of the object.

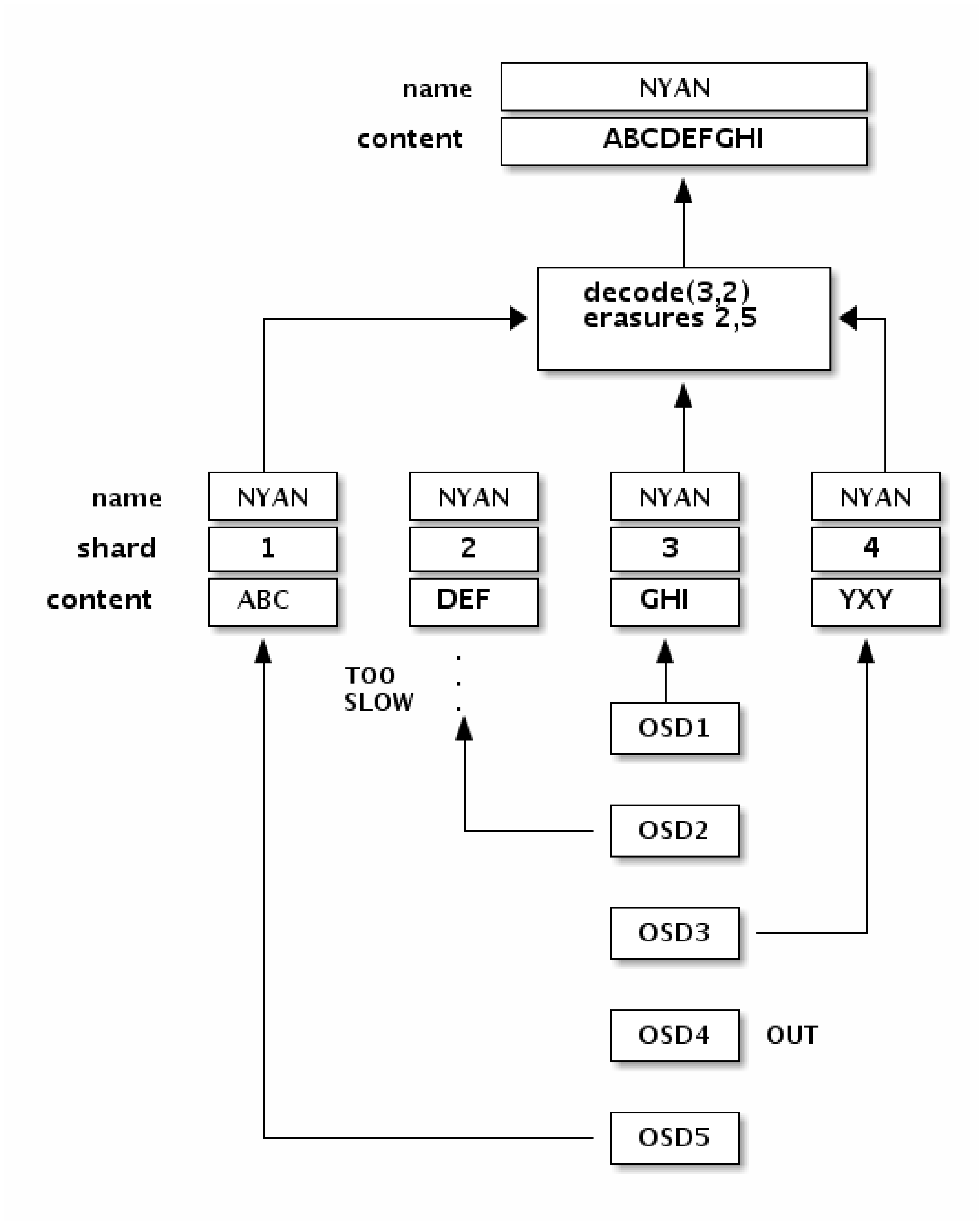
For instance an erasure coded pool is created to use five OSDs (`K+M = 5`) and sustain the loss of two of them (`M = 2`).

Reading and Writing Encoded Chunks

When the object **NYAN** containing `ABCDEFGHI` is written to the pool, the erasure encoding function splits the content into three data chunks simply by dividing the content in three: the first contains `ABC`, the second `DEF` and the last `GHI`. The content will be padded if the content length is not a multiple of `K`. The function also creates two coding chunks: the fourth with `YXY` and the fifth with `GQC`. Each chunk is stored in an OSD in the acting set. The chunks are stored in objects that have the same name (**NYAN**) but reside on different OSDs. The order in which the chunks were created must be preserved and is stored as an attribute of the object (`shard_t`), in addition to its name. Chunk 1 contains `ABC` and is stored on **OSD5** while chunk 4 contains `YXY` and is stored on **OSD3**.



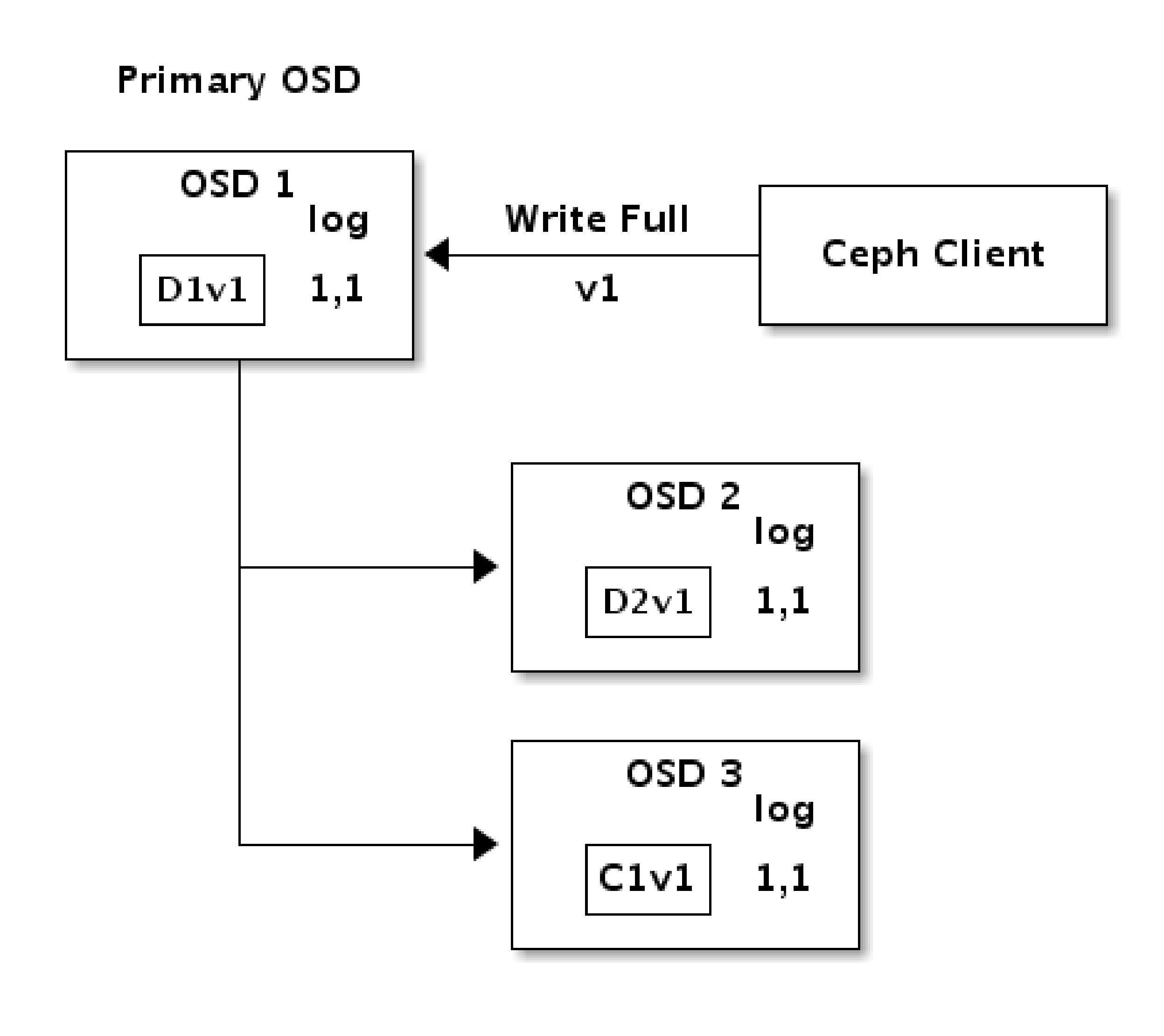
When the object **NYAN** is read from the erasure coded pool, the decoding function reads three chunks: chunk 1 containing `ABC`, chunk 3 containing `GHI` and chunk 4 containing `YXY`. Then, it rebuilds the original content of the object `ABCDEFGHI`. The decoding function is informed that the chunks 2 and 5 are missing (they are called 'erasures'). The chunk 5 could not be read because the **OSD4** is out. The decoding function can be called as soon as three chunks are read: **OSD2** was the slowest and its chunk was not taken into account.


name
NYAN
content
ABCDEFGHI
decode(3,2)
erasures 2,5
name
shard
content
NYAN
1
ABC
NYAN
2
DEF
NYAN
3
GHI
NYAN
4
YXY
TOO
SLOW
OSD1
OSD2
OSD3
OSD4
OUT
OSD5

Interrupted Full Writes

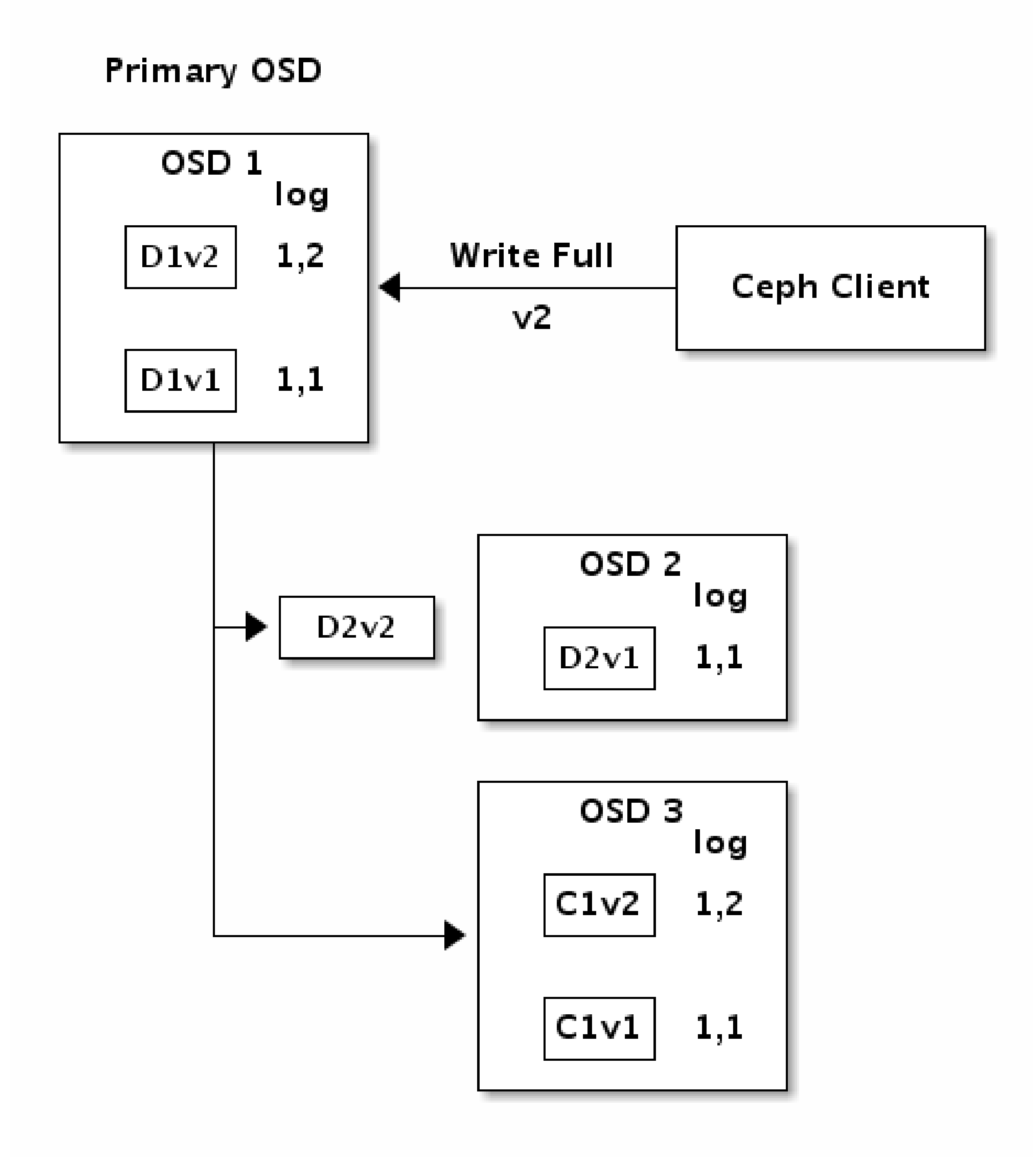
In an erasure coded pool, the primary OSD in the up set receives all write operations. It is responsible for encoding the payload into `K+M` chunks and sends them to the other OSDs. It is also responsible for maintaining an authoritative version of the placement group logs.

In the following diagram, an erasure coded placement group has been created with `K = 2 + M = 1` and is supported by three OSDs, two for `K` and one for `M`. The acting set of the placement group is made of **OSD 1**, **OSD 2** and **OSD 3**. An object has been encoded and stored in the OSDs : the chunk `D1v1` (i.e. Data chunk number 1, version 1) is on **OSD 1**, `D2v1` on **OSD 2** and `C1v1` (i.e. Coding chunk number 1, version 1) on **OSD 3**. The placement group logs on each OSD are identical (i.e. `1,1` for epoch 1, version 1).

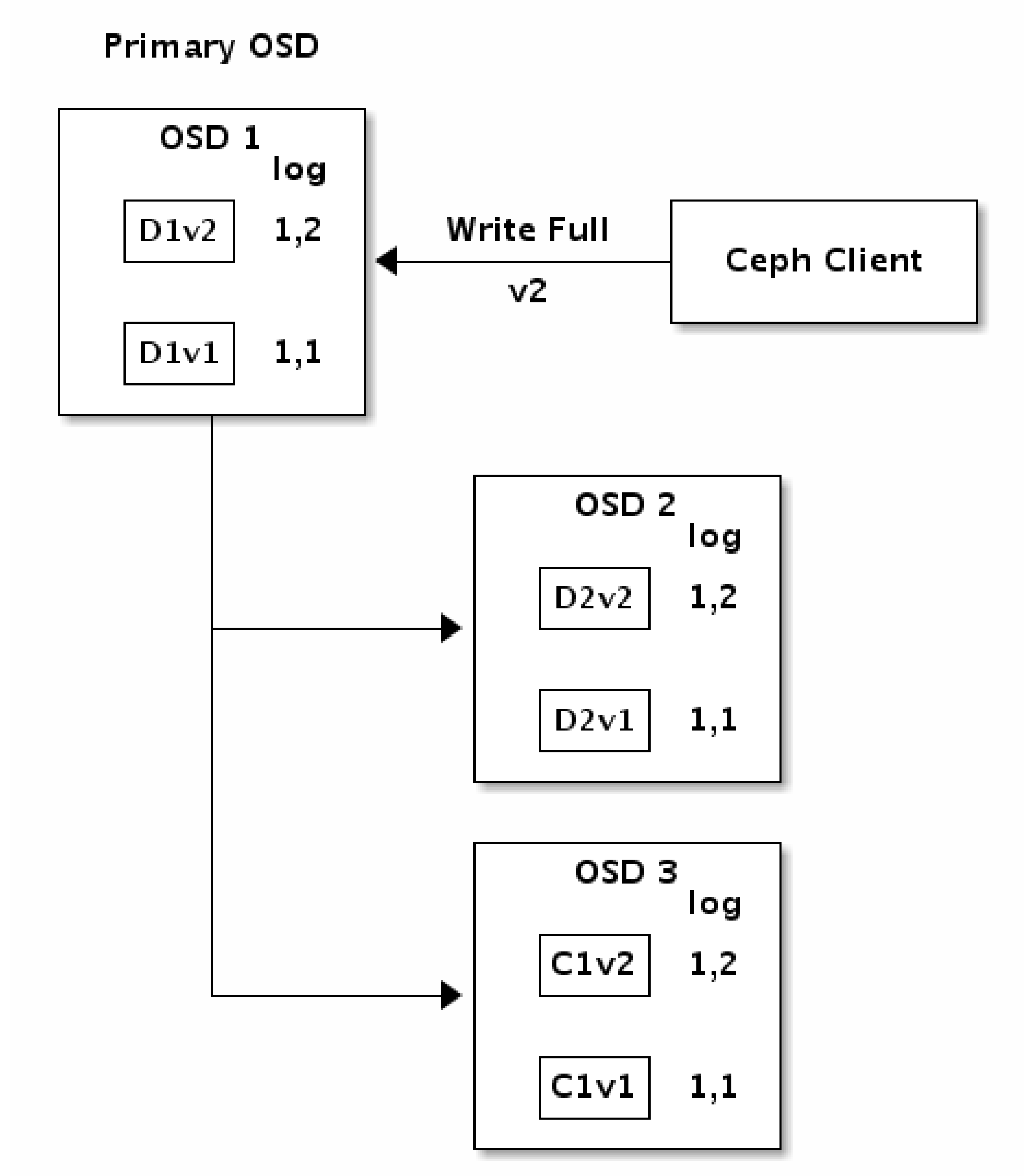


OSD 1 is the primary and receives a **WRITE FULL** from a client, which means the payload is to replace the object entirely instead of overwriting a portion of it. Version 2 (v2) of the object is created to override version 1 (v1). **OSD 1** encodes the payload into three chunks: `D1v2` (i.e. Data chunk number 1 version 2) will be on **OSD 1**, `D2v2` on **OSD 2** and `C1v2` (i.e. Coding chunk number 1 version 2) on **OSD 3**. Each chunk is sent to the target OSD, including the primary OSD which is responsible for storing chunks in addition to handling write operations and maintaining an authoritative version of the placement group logs. When an OSD receives the message instructing it to write the chunk, it also creates a new entry in the placement group logs to reflect the change. For instance, as soon as **OSD 3** stores `C1v2`, it adds the entry `1,2` (i.e. epoch 1, version 2) to its logs. Because the OSDs work asynchronously,

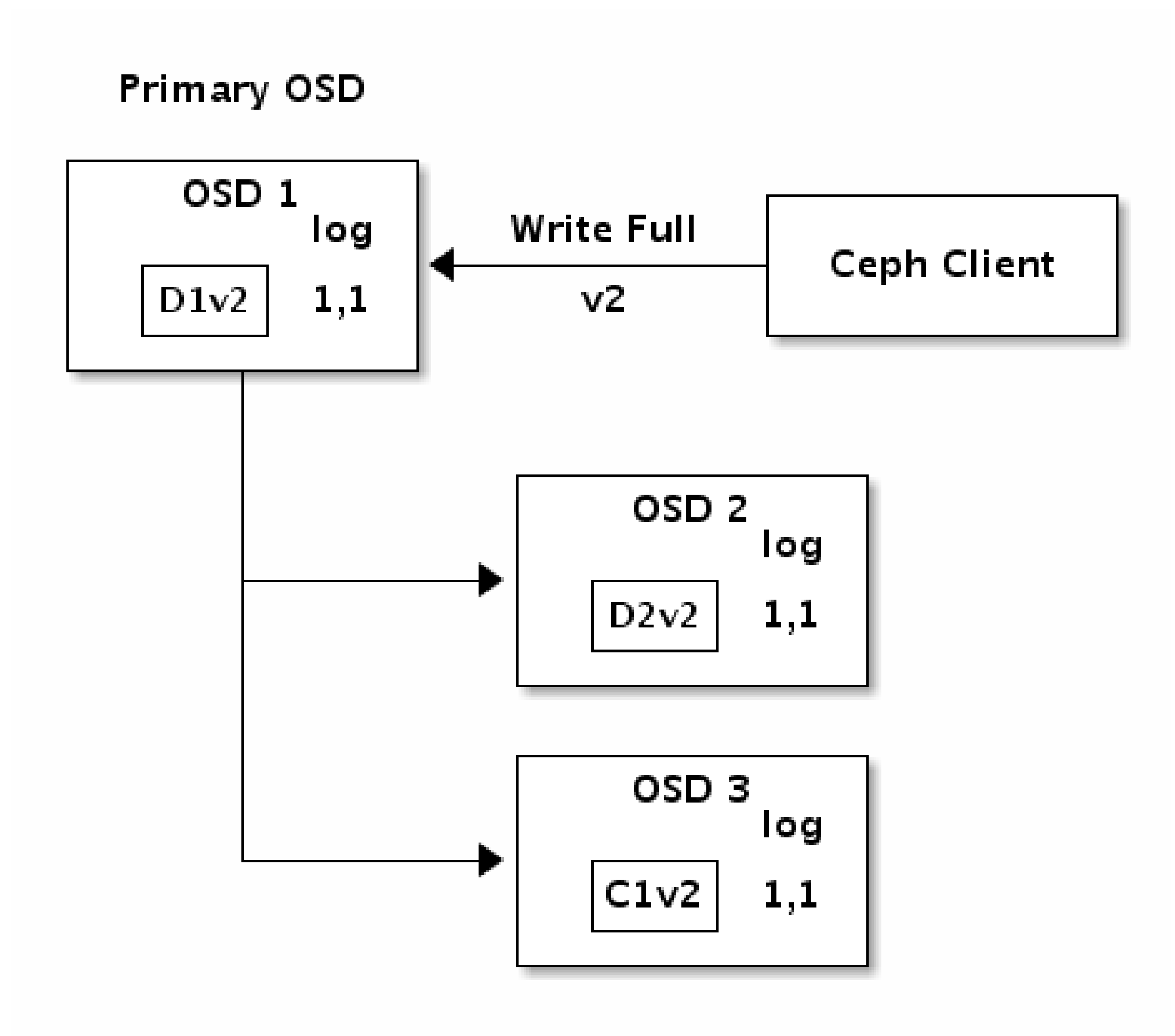
some chunks may still be in flight (such as `D2v2`) while others are acknowledged and on disk (such as `C1v1` and `D1v1`).



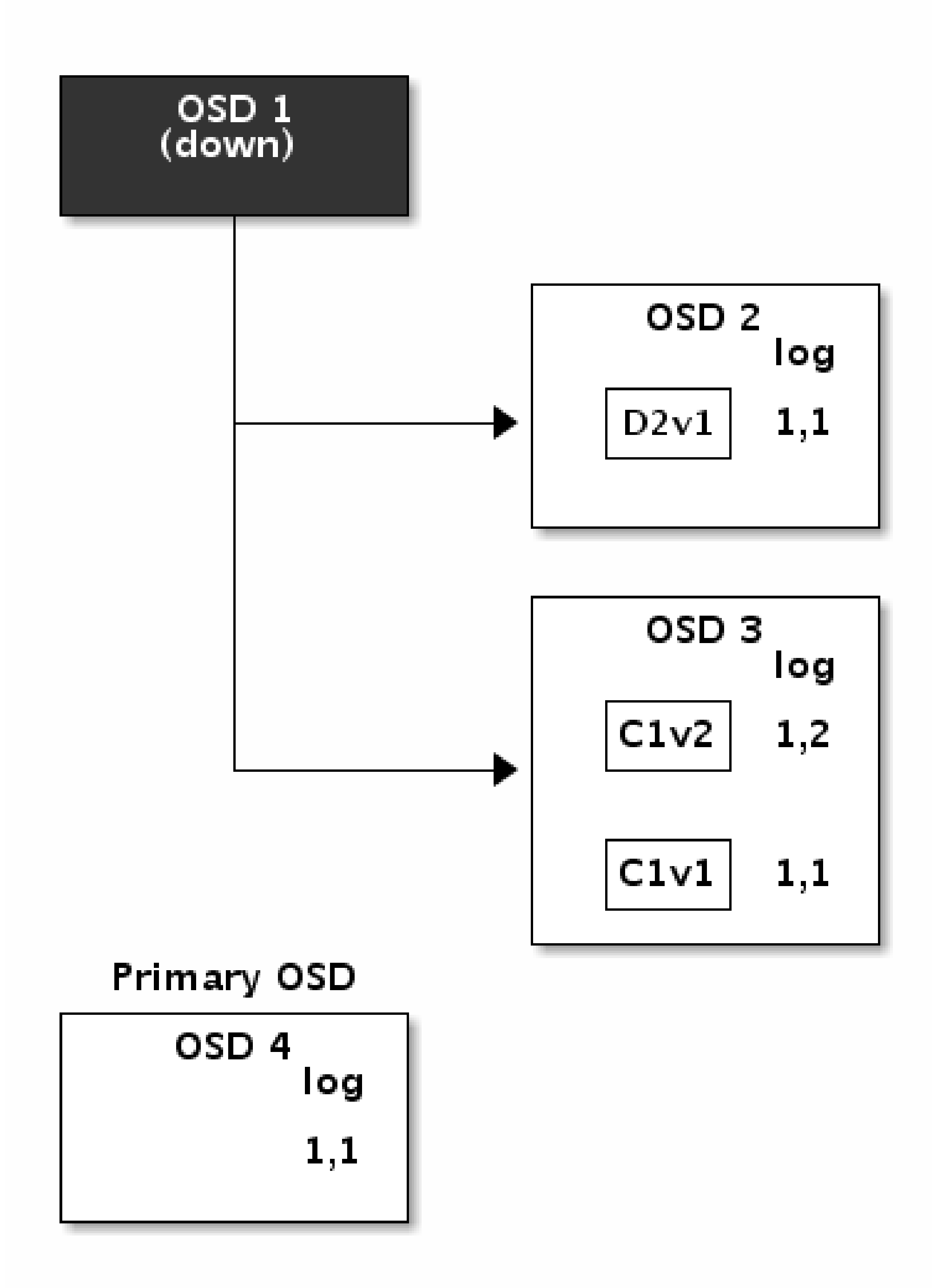
If all goes well, the chunks are acknowledged on each OSD in the acting set and the logs' `last_complete` pointer can move from `1,1` to `1,2`.



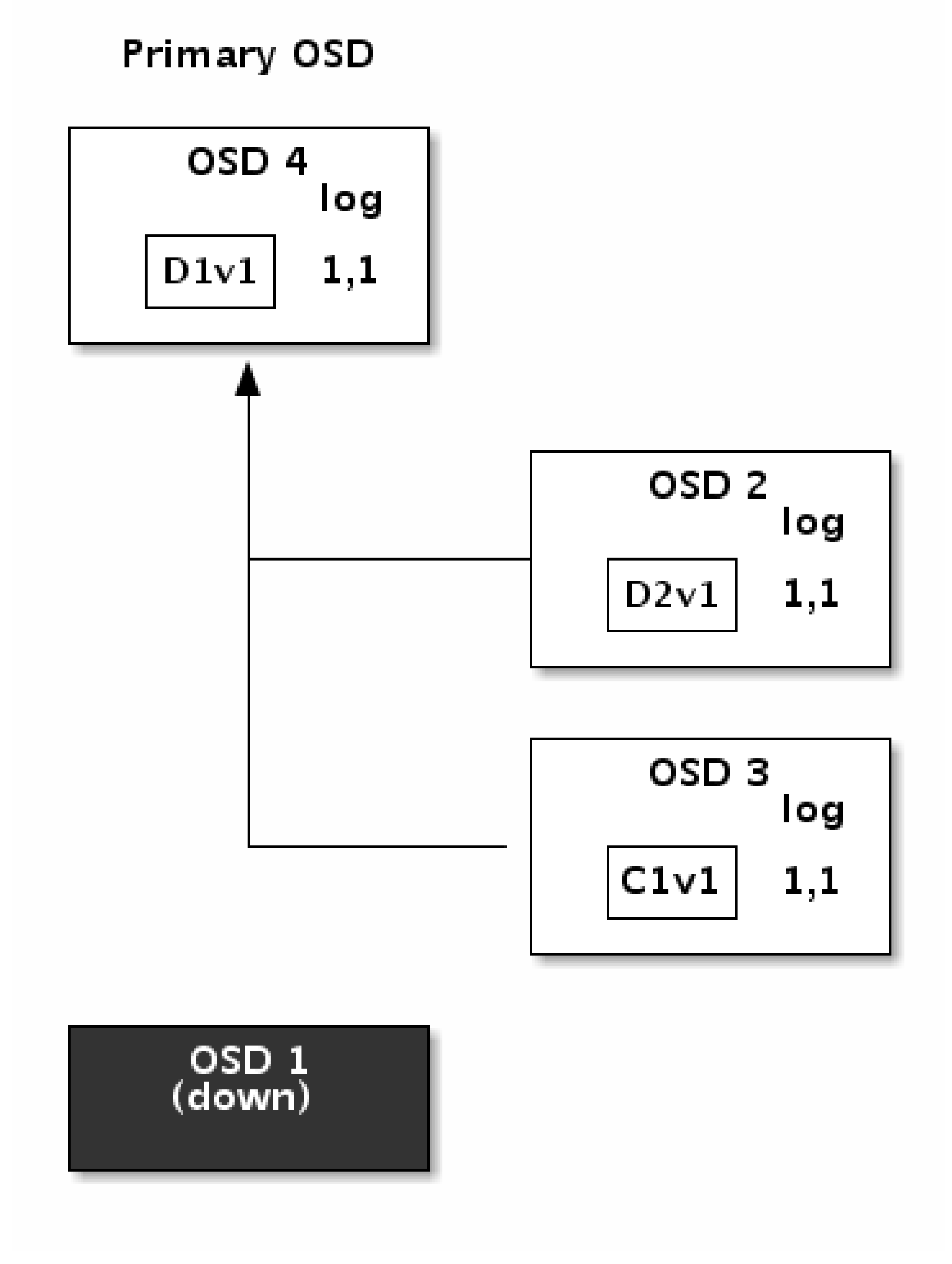
Finally, the files used to store the chunks of the previous version of the object can be removed: `D1v1` on **OSD 1**, `D2v1` on **OSD 2** and `C1v1` on **OSD 3**.



But accidents happen. If **OSD 1** goes down while `D2v2` is still in flight, the object's version 2 is partially written: **OSD 3** has one chunk but that is not enough to recover. It lost two chunks: `D1v2` and `D2v2` and the erasure coding parameters `K = 2,M = 1` require that at least two chunks are available to rebuild the third. **OSD 4** becomes the new primary and finds that the `last_complete` log entry (i.e., all objects before this entry were known to be available on all OSDs in the previous acting set) is `1,1` and that will be the head of the new authoritative log.



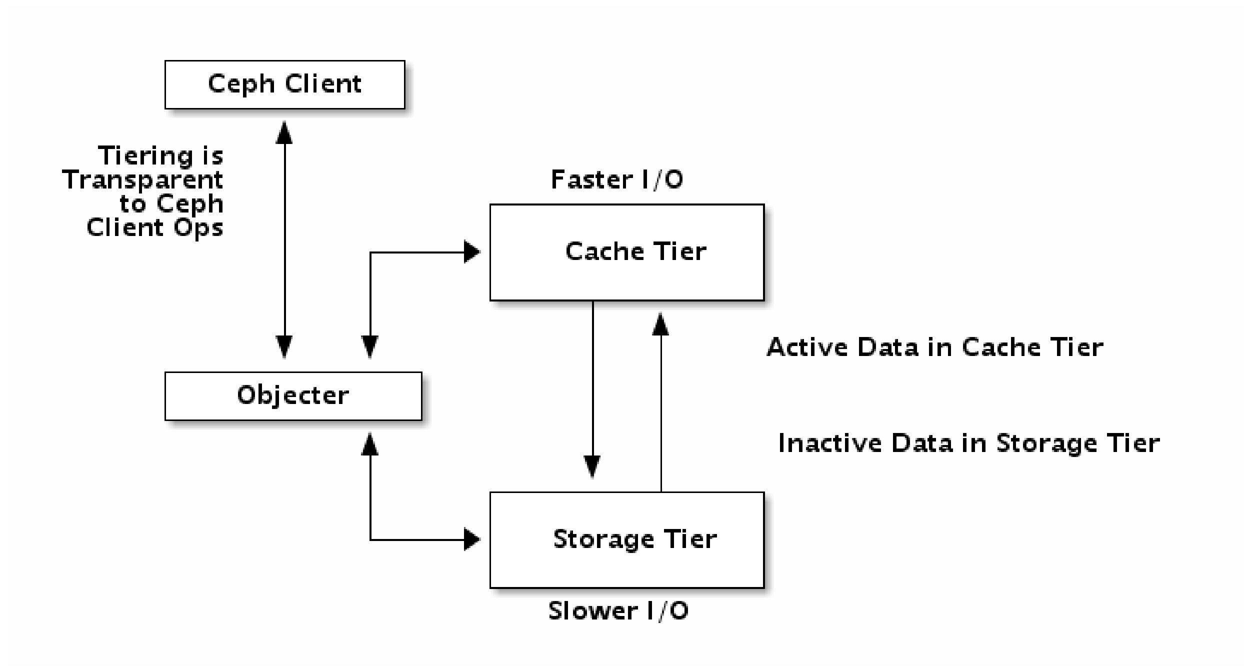
The log entry 1,2 found on **OSD 3** is divergent from the new authoritative log provided by **OSD 4**: it is discarded and the file containing the `C1v2` chunk is removed. The `D1v1` chunk is rebuilt with the `decode` function of the erasure coding library during scrubbing and stored on the new primary **OSD 4**.



See Erasure Code Notes for additional details.

9.1.5 Cache Tiering

A cache tier provides Ceph Clients with better I/O performance for a subset of the data stored in a backing storage tier. Cache tiering involves creating a pool of relatively fast/expensive storage devices (e.g., solid state drives) configured to act as a cache tier, and a backing pool of either erasure-coded or relatively slower/cheaper devices configured to act as an economical storage tier. The Ceph objecter handles where to place the objects and the tiering agent determines when to flush objects from the cache to the backing storage tier. So the cache tier and the backing storage tier are completely transparent to Ceph clients.


Ceph Client
Tiering is Transparent to Ceph Client Ops
Faster I/O
Cache Tier
Active Data in Cache Tier
Objecter
Inactive Data in Storage Tier
Storage Tier
Slower I/O

See Cache Tiering for additional details.

9.1.6 Extending Ceph

You can extend Ceph by creating shared object classes called 'Ceph Classes'. Ceph loads `.so` classes stored in the `osd class dir` directory dynamically (i.e., `$libdir/rados-classes` by default). When you implement a class, you can create new object methods that have the ability to call the native methods in the Ceph Object Store, or other class methods you incorporate via libraries or create yourself.

On writes, Ceph Classes can call native or class methods, perform any series of operations on the inbound data and generate a resulting write transaction that Ceph will apply atomically.

On reads, Ceph Classes can call native or class methods, perform any series of operations on the outbound data and return the data to the client.

Ceph Class Example

A Ceph class for a content management system that presents pictures of a particular size and aspect ratio could take an inbound bitmap image, crop it to a particular aspect ratio, resize it and embed an invisible copyright or watermark to help protect the intellectual property; then, save the resulting bitmap image to the object store.

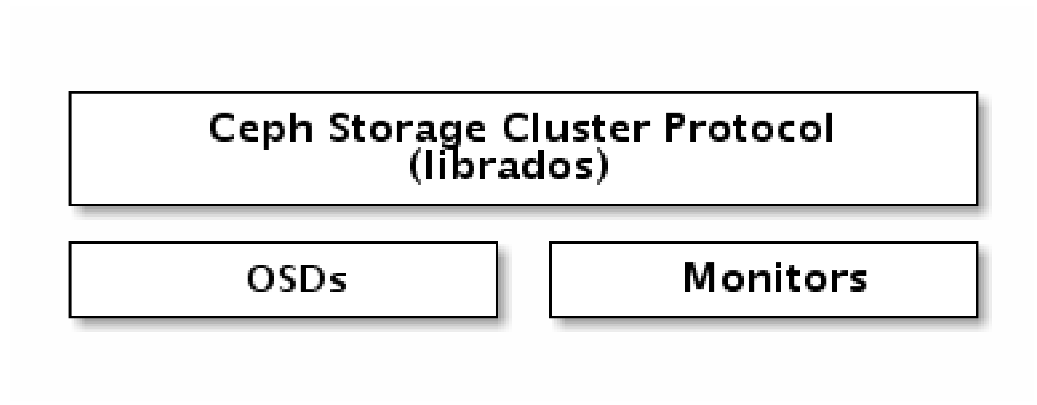
See `src/objclass/objclass.h`, `src/fooclass.cc` and `src/barclass` for exemplary implementations.

9.1.7 Summary

Ceph Storage Clusters are dynamic–like a living organism. Whereas, many storage appliances do not fully utilize the CPU and RAM of a typical commodity server, Ceph does. From heartbeats, to peering, to rebalancing the cluster or recovering from faults, Ceph offloads work from clients (and from a centralized gateway which doesn't exist in the Ceph architecture) and uses the computing power of the OSDs to perform the work. When referring to Hardware Recommendations and the Network Config Reference, be cognizant of the foregoing concepts to understand how Ceph utilizes computing resources.

9.2 Ceph Protocol

Ceph Clients use the native protocol for interacting with the Ceph Storage Cluster. Ceph packages this functionality into the `librados` library so that you can create your own custom Ceph Clients. The following diagram depicts the basic architecture.



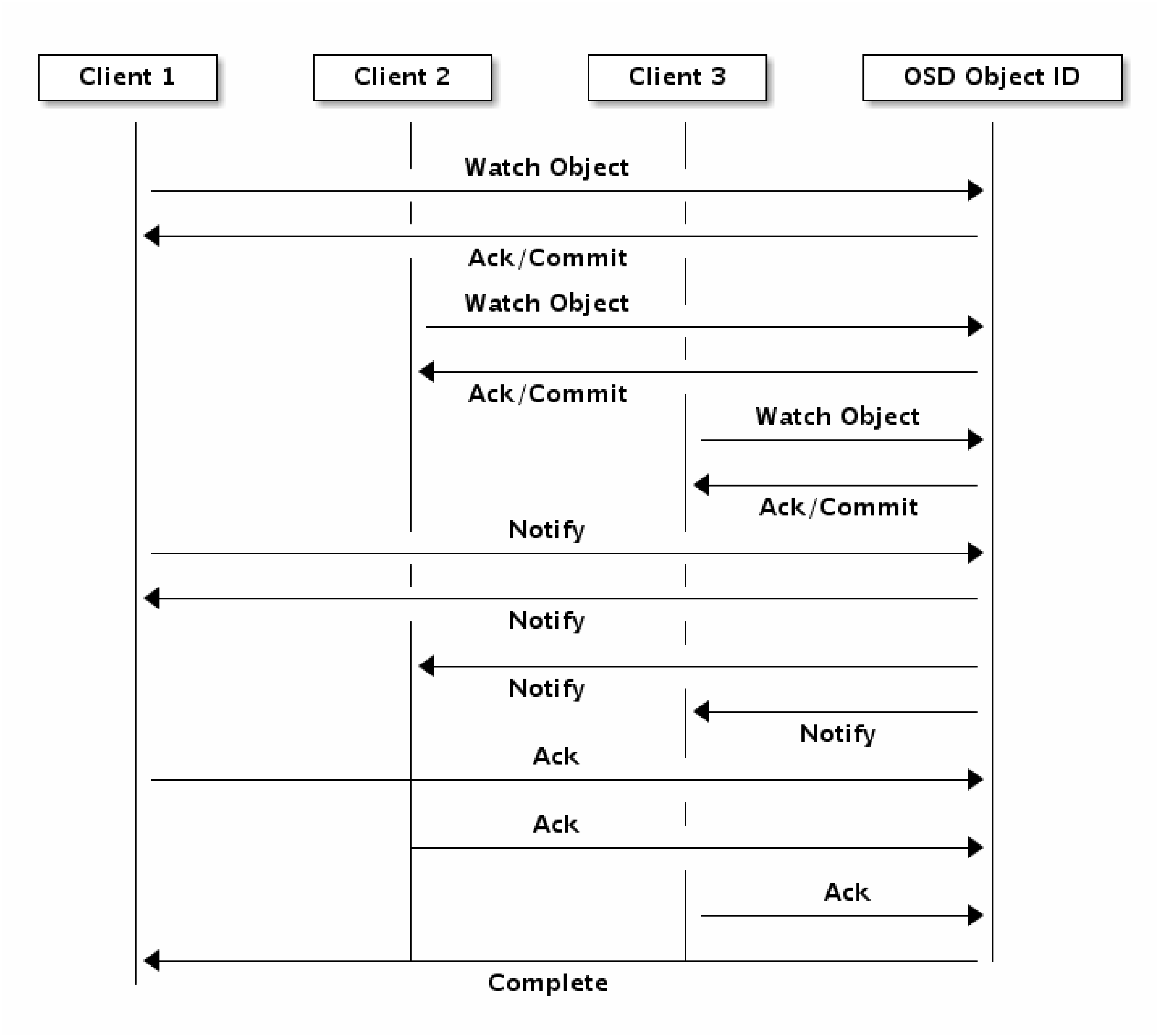
9.2.1 Native Protocol and `librados`

Modern applications need a simple object storage interface with asynchronous communication capability. The Ceph Storage Cluster provides a simple object storage interface with asynchronous communication capability. The interface provides direct, parallel access to objects throughout the cluster.

- Pool Operations
- Snapshots and Copy-on-write Cloning
- Read/Write Objects - Create or Remove - Entire Object or Byte Range - Append or Truncate
- Create/Set/Get/Remove XATTRs
- Create/Set/Get/Remove Key/Value Pairs
- Compound operations and dual-ack semantics
- Object Classes

9.2.2 Object Watch/Notify

A client can register a persistent interest with an object and keep a session to the primary OSD open. The client can send a notification message and payload to all watchers and receive notification when the watchers receive the notification. This enables a client to use any object a synchronization/communication channel.



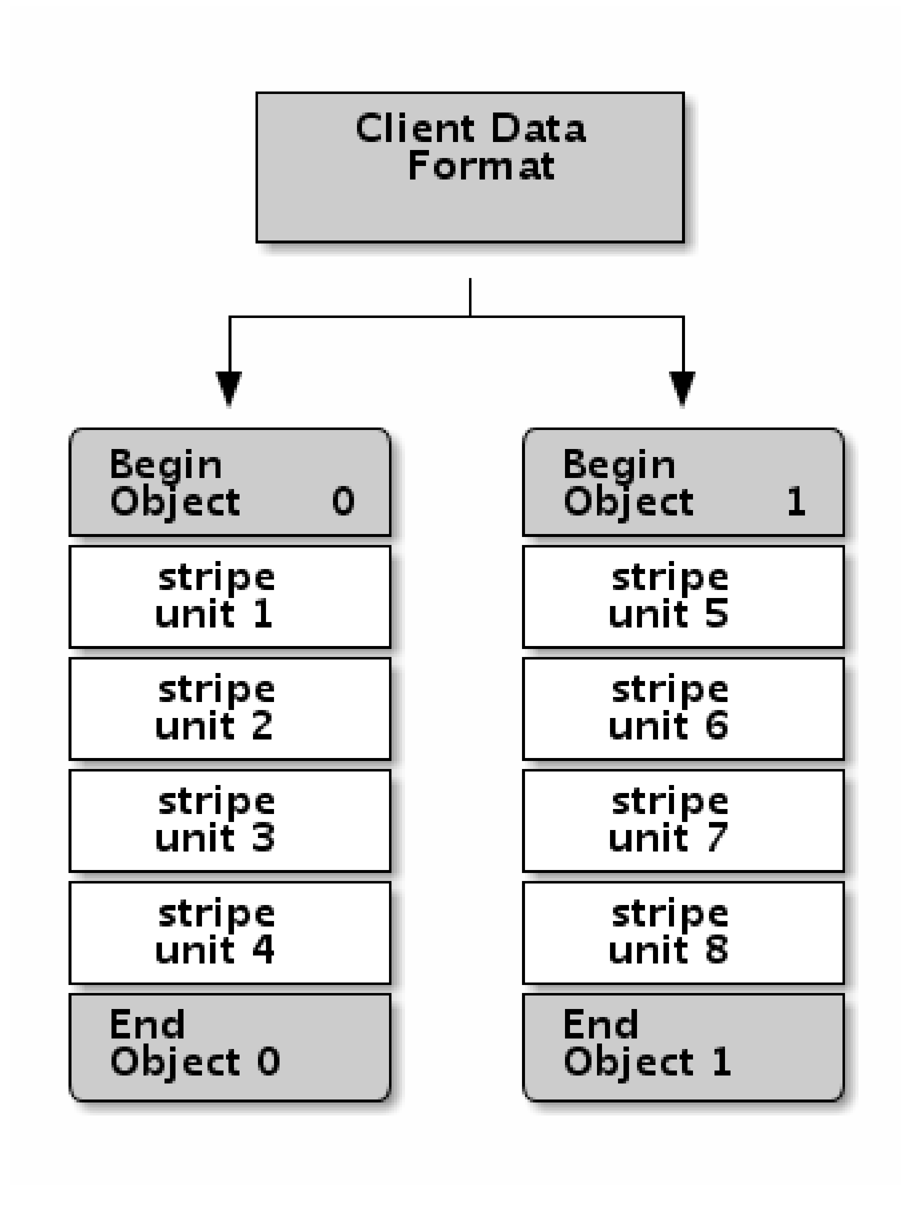
9.2.3 Data Striping

Storage devices have throughput limitations, which impact performance and scalability. So storage systems often support striping–storing sequential pieces of information across across multiple storage devices–to increase throughput and performance. The most common form of data striping comes from RAID. The RAID type most similar to Ceph's striping is RAID 0, or a 'striped volume.' Ceph's striping offers the throughput of RAID 0 striping, the reliability of n-way RAID mirroring and faster recovery.

Ceph provides three types of clients: Ceph Block Device, Ceph Filesystem, and Ceph Object Storage. A Ceph Client converts its data from the representation format it provides to its users (a block device image, RESTful objects, CephFS filesystem directories) into objects for storage in the Ceph Storage Cluster.

Tip: The objects Ceph stores in the Ceph Storage Cluster are not striped. Ceph Object Storage, Ceph Block Device, and the Ceph Filesystem stripe their data over multiple Ceph Storage Cluster objects. Ceph Clients that write directly to the Ceph Storage Cluster via `librados` must perform the striping (and parallel I/O) for themselves to obtain these benefits.

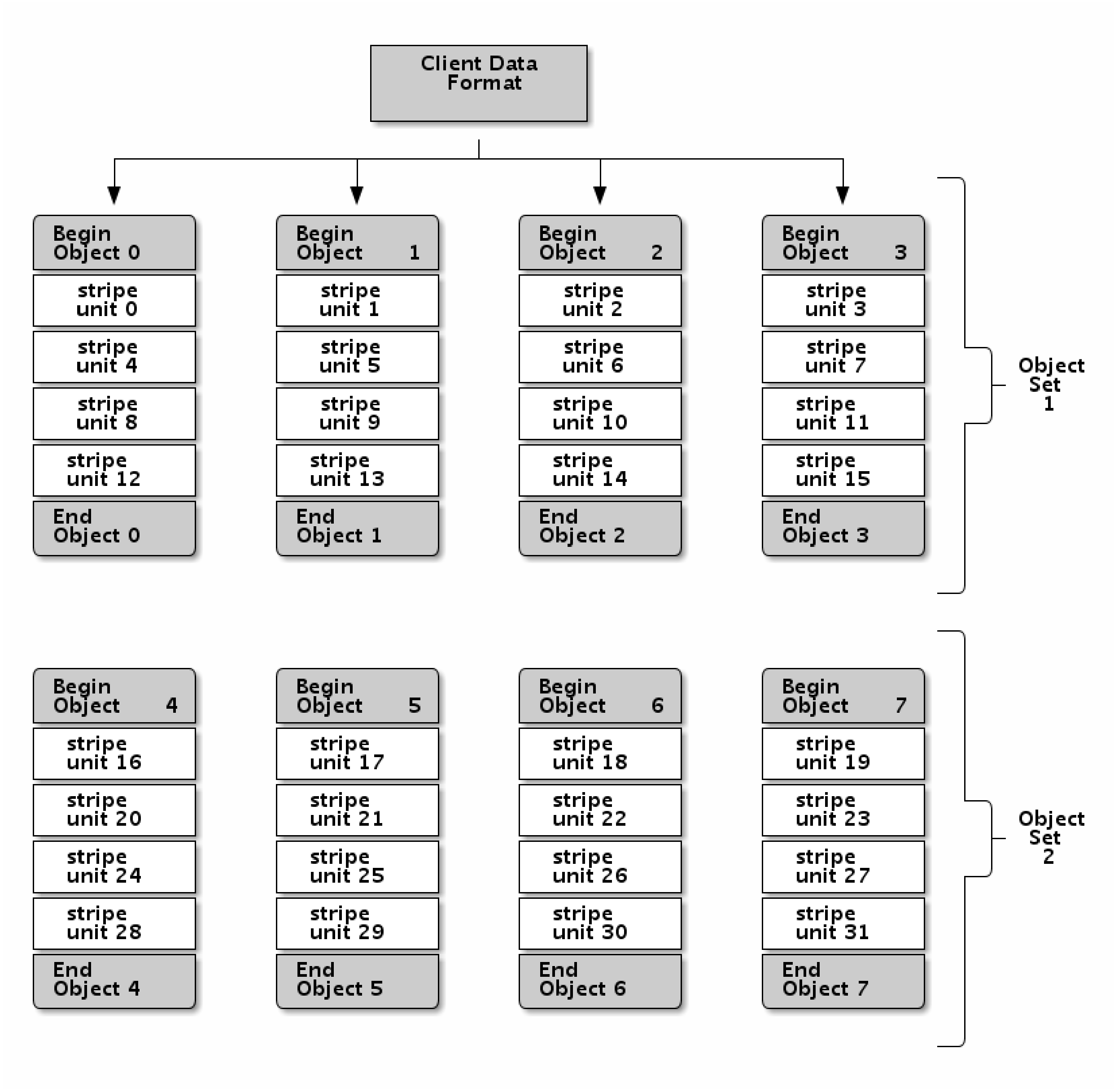
The simplest Ceph striping format involves a stripe count of 1 object. Ceph Clients write stripe units to a Ceph Storage Cluster object until the object is at its maximum capacity, and then create another object for additional stripes of data. The simplest form of striping may be sufficient for small block device images, S3 or Swift objects and CephFS files. However, this simple form doesn't take maximum advantage of Ceph's ability to distribute data across placement groups, and consequently doesn't improve performance very much. The following diagram depicts the simplest form of striping:



If you anticipate large images sizes, large S3 or Swift objects (e.g., video), or large CephFS directories, you may see considerable read/write performance improvements by striping client data over multiple objects within an object set. Significant write performance occurs when the client writes the stripe units to their corresponding objects in parallel. Since objects get mapped to different placement groups and further mapped to different OSDs, each write occurs in parallel at the maximum write speed. A write to a single disk would be limited by the head movement (e.g. 6ms per seek) and bandwidth of that one device (e.g. 100MB/s). By spreading that write over multiple objects (which map to different placement groups and OSDs) Ceph can reduce the number of seeks per drive and combine the throughput of multiple drives to achieve much faster write (or read) speeds.

Note: Striping is independent of object replicas. Since CRUSH replicates objects across OSDs, stripes get replicated automatically.

In the following diagram, client data gets striped across an object set (`object set 1` in the following diagram) consisting of 4 objects, where the first stripe unit is `stripe unit 0` in `object 0`, and the fourth stripe unit is `stripe unit 3` in `object 3`. After writing the fourth stripe, the client determines if the object set is full. If the object set is not full, the client begins writing a stripe to the first object again (`object 0` in the following diagram). If the object set is full, the client creates a new object set (`object set 2` in the following diagram), and begins writing to the first stripe (`stripe unit 16`) in the first object in the new object set (`object 4` in the diagram below).



Three important variables determine how Ceph stripes data:

- **Object Size:** Objects in the Ceph Storage Cluster have a maximum configurable size (e.g., 2MB, 4MB, etc.). The object size should be large enough to accommodate many stripe units, and should be a multiple of the stripe unit.

- **Stripe Width:** Stripes have a configurable unit size (e.g., 64kb). The Ceph Client divides the data it will write to objects into equally sized stripe units, except for the last stripe unit. A stripe width, should be a fraction of the Object Size so that an object may contain many stripe units.
- **Stripe Count:** The Ceph Client writes a sequence of stripe units over a series of objects determined by the stripe count. The series of objects is called an object set. After the Ceph Client writes to the last object in the object set, it returns to the first object in the object set.

Important: Test the performance of your striping configuration before putting your cluster into production. You CANNOT change these striping parameters after you stripe the data and write it to objects.

Once the Ceph Client has striped data to stripe units and mapped the stripe units to objects, Ceph's CRUSH algorithm maps the objects to placement groups, and the placement groups to Ceph OSD Daemons before the objects are stored as files on a storage disk.

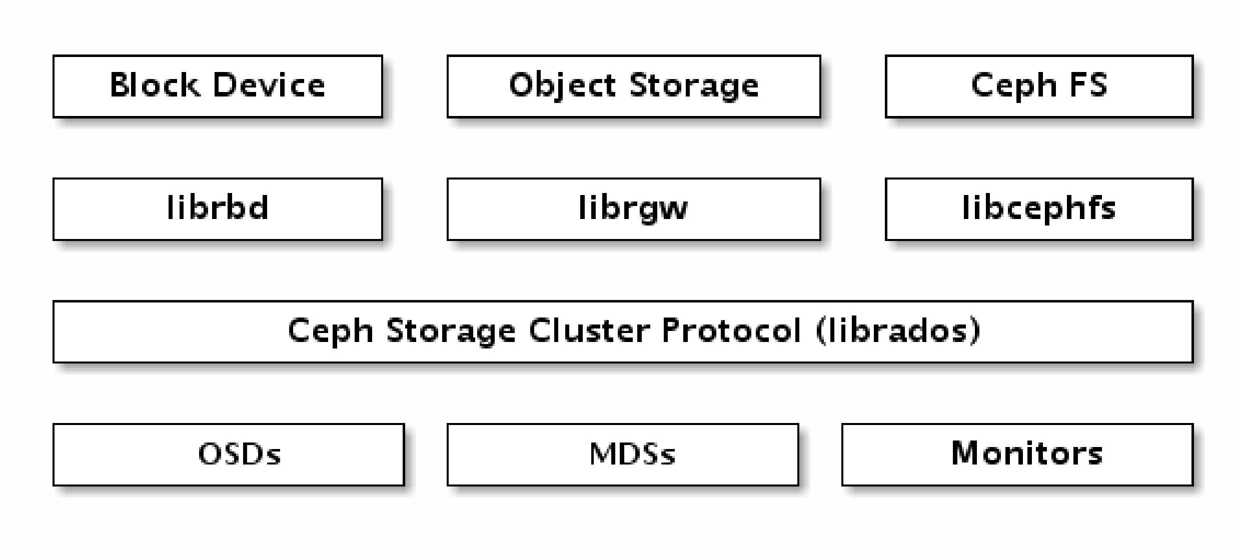
Note: Since a client writes to a single pool, all data striped into objects get mapped to placement groups in the same pool. So they use the same CRUSH map and the same access controls.

9.3 Ceph Clients

Ceph Clients include a number of service interfaces. These include:

- **Block Devices:** The *Ceph Block Device* (a.k.a., RBD) service provides resizable, thin-provisioned block devices with snapshotting and cloning. Ceph stripes a block device across the cluster for high performance. Ceph supports both kernel objects (KO) and a QEMU hypervisor that uses `librbd` directly–avoiding the kernel object overhead for virtualized systems.
- **Object Storage:** The *Ceph Object Storage* (a.k.a., RGW) service provides RESTful APIs with interfaces that are compatible with Amazon S3 and OpenStack Swift.
- **Filesystem**: The *Ceph Filesystem* (CephFS) service provides a POSIX compliant filesystem usable with `mount` or as a filesytem in user space (FUSE).

Ceph can run additional instances of OSDs, MDSs, and monitors for scalability and high availability. The following diagram depicts the high-level architecture.



9.3.1 Ceph Object Storage

The Ceph Object Storage daemon, `radosgw`, is a FastCGI service that provides a RESTful HTTP API to store objects and metadata. It layers on top of the Ceph Storage Cluster with its own data formats, and maintains its own user database, authentication, and access control. The RADOS Gateway uses a unified namespace, which means you can use either the OpenStack Swift-compatible API or the Amazon S3-compatible API. For example, you can write data using the S3-compatible API with one application and then read data using the Swift-compatible API with another application.

S3/Swift Objects and Store Cluster Objects Compared

Ceph's Object Storage uses the term *object* to describe the data it stores. S3 and Swift objects are not the same as the objects that Ceph writes to the Ceph Storage Cluster. Ceph Object Storage objects are mapped to Ceph Storage Cluster objects. The S3 and Swift objects do not necessarily correspond in a 1:1 manner with an object stored in the storage cluster. It is possible for an S3 or Swift object to map to multiple Ceph objects.

See Ceph Object Storage for details.

9.3.2 Ceph Block Device

A Ceph Block Device stripes a block device image over multiple objects in the Ceph Storage Cluster, where each object gets mapped to a placement group and distributed, and the placement groups are spread across separate `ceph-osd` daemons throughout the cluster.

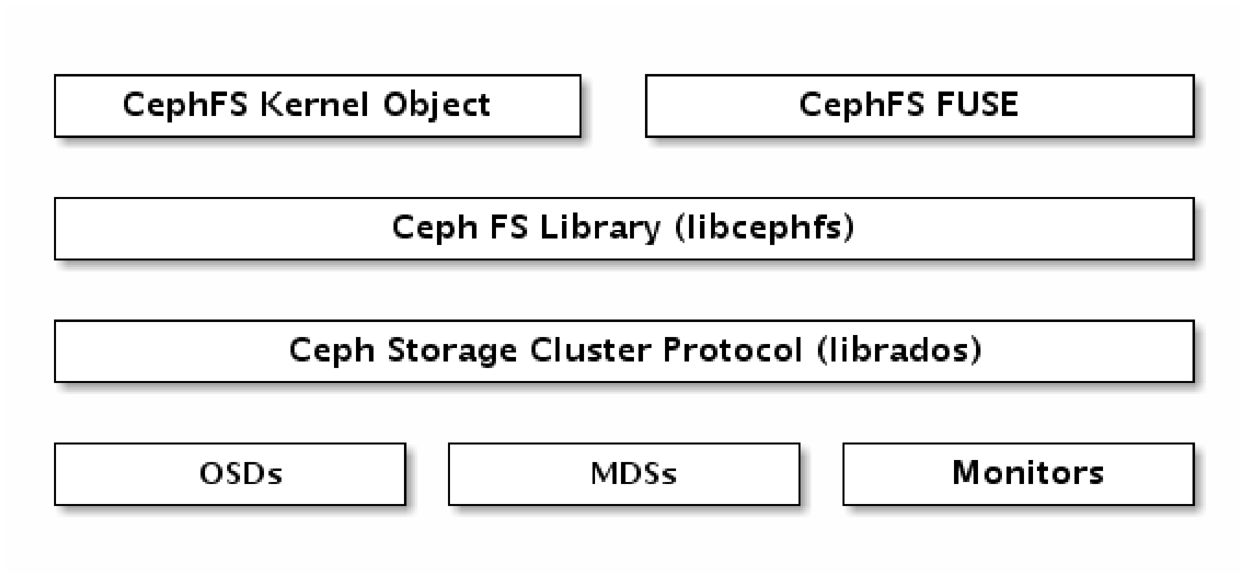
Important: Striping allows RBD block devices to perform better than a single server could!

Thin-provisioned snapshottable Ceph Block Devices are an attractive option for virtualization and cloud computing. In virtual machine scenarios, people typically deploy a Ceph Block Device with the `rbd` network storage driver in Qemu/KVM, where the host machine uses `librbd` to provide a block device service to the guest. Many cloud computing stacks use `libvirt` to integrate with hypervisors. You can use thin-provisioned Ceph Block Devices with Qemu and `libvirt` to support OpenStack and CloudStack among other solutions.

While we do not provide `librbd` support with other hypervisors at this time, you may also use Ceph Block Device kernel objects to provide a block device to a client. Other virtualization technologies such as Xen can access the Ceph Block Device kernel object(s). This is done with the command-line tool `rbd`.

9.3.3 Ceph Filesystem

The Ceph Filesystem (Ceph FS) provides a POSIX-compliant filesystem as a service that is layered on top of the object-based Ceph Storage Cluster. Ceph FS files get mapped to objects that Ceph stores in the Ceph Storage Cluster. Ceph Clients mount a CephFS filesystem as a kernel object or as a Filesystem in User Space (FUSE).


CephFS Kernel Object
CephFS FUSE
Ceph FS Library (libcephfs)
Ceph Storage Cluster Protocol (librados)
OSDs
MDSs
Monitors

The Ceph Filesystem service includes the Ceph Metadata Server (MDS) deployed with the Ceph Storage cluster. The purpose of the MDS is to store all the filesystem metadata (directories, file ownership, access modes, etc) in high-availability Ceph Metadata Servers where the metadata resides in memory. The reason for the MDS (a daemon called `ceph-mds`) is that simple filesystem operations like listing a directory or changing a directory (`ls`, `cd`) would tax the Ceph OSD Daemons unnecessarily. So separating the metadata from the data means that the Ceph Filesystem can provide high performance services without taxing the Ceph Storage Cluster.

Ceph FS separates the metadata from the data, storing the metadata in the MDS, and storing the file data in one or more objects in the Ceph Storage Cluster. The Ceph filesystem aims for POSIX compatibility. `ceph-mds` can run as a single process, or it can be distributed out to multiple physical machines, either for high availability or for scalability.

- **High Availability**: The extra `ceph-mds` instances can be *standby*, ready to take over the duties of any failed `ceph-mds` that was *active*. This is easy because all the data, including the journal, is stored on RADOS. The transition is triggered automatically by `ceph-mon`.
- **Scalability**: Multiple `ceph-mds` instances can be *active*, and they will split the directory tree into subtrees (and shards of a single busy directory), effectively balancing the load amongst all *active* servers.

Combinations of *standby* and *active* etc are possible, for example running 3 *active* `ceph-mds` instances for scaling, and one *standby* instance for high availability.

CHAPTER TEN

INTERNAL DEVELOPER DOCUMENTATION

Note: If you're looking for how to use Ceph as a library from your own software, please see API Documentation.

You can start a development mode Ceph cluster, after compiling the source, with:

```
cd src
install -d -m0755 out dev/osd0
./vstart.sh -n -x -l
# check that it's there
./ceph health
```

Todo

vstart is woefully undocumented and full of sharp sticks to poke yourself with.

Mailing list

The official development email list is `ceph-devel@vger.kernel.org`. Subscribe by sending a message to `majordomo@vger.kernel.org` with the line:

```
subscribe ceph-devel
```

in the body of the message.

Contents

10.1 Cache pool

10.1.1 Purpose

Use a pool of fast storage devices (probably SSDs) and use it as a cache for an existing larger pool.

Use a replicated pool as a front-end to service most I/O, and destage cold data to a separate erasure coded pool that does not currently (and cannot efficiently) handle the workload.

We should be able to create and add a cache pool to an existing pool of data, and later remove it, without disrupting service or migrating data around.

10.1.2 Use cases

Read-write pool, writeback

We have an existing data pool and put a fast cache pool “in front” of it. Writes will go to the cache pool and immediately ack. We flush them back to the data pool based on the defined policy.

Read-only pool, weak consistency

We have an existing data pool and add one or more read-only cache pools. We copy data to the cache pool(s) on read. Writes are forwarded to the original data pool. Stale data is expired from the cache pools based on the defined policy.

This is likely only useful for specific applications with specific data access patterns. It may be a match for rgw, for example.

10.1.3 Interface

Set up a read/write cache pool foo-hot for pool foo:

```
ceph osd tier add foo foo-hot
ceph osd tier cache-mode foo-hot writeback
```

Direct all traffic for foo to foo-hot:

```
ceph osd tier set-overlay foo foo-hot
```

Set the target size and enable the tiering agent for foo-hot:

```
ceph osd pool set foo-hot hit_set_type bloom
ceph osd pool set foo-hot hit_set_count 1
ceph osd pool set foo-hot hit_set_period 3600   # 1 hour
ceph osd pool set foo-hot target_max_bytes 1000000000000  # 1 TB
```

Drain the cache in preparation for turning it off:

```
ceph osd tier cache-mode foo-hot forward
rados -p foo-hot cache-flush-evict-all
```

When cache pool is finally empty, disable it:

```
ceph osd tier remove-overlay foo
ceph osd tier remove foo foo-hot
```

Read-only pools with lazy consistency:

```
ceph osd tier add foo foo-east
ceph osd tier cache-mode foo-east readonly
ceph osd tier add foo foo-west
ceph osd tier cache-mode foo-west readonly
```

10.1.4 Tiering agent

The tiering policy is defined as properties on the cache pool itself.

HitSet metadata

First, the agent requires HitSet information to be tracked on the cache pool in order to determine which objects in the pool are being accessed. This is enabled with:

```
ceph osd pool set foo-hot hit_set_type bloom
ceph osd pool set foo-hot hit_set_count 1
ceph osd pool set foo-hot hit_set_period 3600   # 1 hour
```

The supported HitSet types include 'bloom' (a bloom filter, the default), 'explicit_hash', and 'explicit_object'. The latter two explicitly enumerate accessed objects and are less memory efficient. They are there primarily for debugging and to demonstrate pluggability for the infrastructure. For the bloom filter type, you can additionally define the false positive probability for the bloom filter (default is 0.05):

```
ceph osd pool set foo-hot hit_set_fpp 0.15
```

The hit_set_count and hit_set_period define how much time each HitSet should cover, and how many such HitSets to store. Binning accesses over time allows Ceph to independently determine whether an object was accessed at least once and whether it was accessed more than once over some time period ("age" vs "temperature"). Note that the longer the period and the higher the count the more RAM will be consumed by the ceph-osd process. In particular, when the agent is active to flush or evict cache objects, all hit_set_count HitSets are loaded into RAM.

Currently there is minimal benefit for hit_set_count > 1 since the agent does not yet act intelligently on that information.

Cache mode

The most important policy is the cache mode:

> ceph osd pool set foo-hot cache-mode writeback

The supported modes are 'none', 'writeback', 'forward', and 'readonly'. Most installations want 'writeback', which will write into the cache tier and only later flush updates back to the base tier. Similarly, any object that is read will be promoted into the cache tier.

The 'forward' mode is intended for when the cache is being disabled and needs to be drained. No new objects will be promoted or written to the cache pool unless they are already present. A background operation can then do something like:

```
rados -p foo-hot cache-try-flush-evict-all
rados -p foo-hot cache-flush-evict-all
```

to force all data to be flushed back to the base tier.

The 'readonly' mode is intended for read-only workloads that do not require consistency to be enforced by the storage system. Writes will be forwarded to the base tier, but objects that are read will get promoted to the cache. No attempt is made by Ceph to ensure that the contents of the cache tier(s) are consistent in the presence of object updates.

Cache sizing

The agent performs two basic functions: flushing (writing 'dirty' cache objects back to the base tier) and evicting (removing cold and clean objects from the cache).

The thresholds at which Ceph will flush or evict objects is specified relative to a 'target size' of the pool. For example:

```
ceph osd pool set foo-hot cache_target_dirty_ratio .4
ceph osd pool set foo-hot cache_target_full_ratio .8
```

will begin flushing dirty objects when 40% of the pool is dirty and begin evicting clean objects when we reach 80% of the target size.

The target size can be specified either in terms of objects or bytes:

```
ceph osd pool set foo-hot target_max_bytes 1000000000000  # 1 TB
ceph osd pool set foo-hot target_max_objects 1000000       # 1 million objects
```

Note that if both limits are specified, Ceph will begin flushing or evicting when either threshold is triggered.

Other tunables

You can specify a minimum object age before a recently updated object is flushed to the base tier:

```
ceph osd pool set foo-hot cache_min_flush_age 600   # 10 minutes
```

You can specify the minimum age of an object before it will be evicted from the cache tier:

```
ceph osd pool set foo-hot cache_min_evict_age 1800   # 30 minutes
```

10.2 A Detailed Description of the Cephx Authentication Protocol

Peter Reiher 7/13/12

This document provides deeper detail on the Cephx authorization protocol whose high level flow is described in the memo by Yehuda (12/19/09). Because this memo discusses details of routines called and variables used, it represents a snapshot. The code might be changed subsequent to the creation of this document, and the document is not likely to be updated in lockstep. With luck, code comments will indicate major changes in the way the protocol is implemented.

10.2.1 Introduction

The basic idea of the protocol is based on Kerberos. A client wishes to obtain something from a server. The server will only offer the requested service to authorized clients. Rather than requiring each server to deal with authentication and authorization issues, the system uses an authorization server. Thus, the client must first communicate with the authorization server to authenticate itself and to obtain credentials that will grant it access to the service it wants.

Authorization is not the same as authentication. Authentication provides evidence that some party is who it claims to be. Authorization provides evidence that a particular party is allowed to do something. Generally, secure authorization implies secure authentication (since without authentication, you may authorize something for an imposter), but the reverse is not necessarily true. One can authenticate without authorizing. The purpose of this protocol is to authorize.

The basic approach is to use symmetric cryptography throughout. Each client C has its own secret key, known only to itself and the authorization server A. Each server S has its own secret key, known only to itself and the authorization server A. Authorization information will be passed in tickets, encrypted with the secret key of the entity that offers the service. There will be a ticket that A gives to C, which permits C to ask A for other tickets. This ticket will be encrypted with A's key, since A is the one who needs to check it. There will later be tickets that A issues that allow C to communicate with S to ask for service. These tickets will be encrypted with S's key, since S needs to check them. Since we wish to provide security of the communications, as well, session keys are set up along with the tickets. Currently, those session keys are only used for authentication purposes during this protocol and the handshake between the client C and the server S, when the client provides its service ticket. They could be used for authentication or secrecy throughout, with some changes to the system.

Several parties need to prove something to each other if this protocol is to achieve its desired security effects.

1. The client C must prove to the authenticator A that it really is C. Since everything is being done via messages, the client must also prove that the message proving authenticity is fresh, and is not being replayed by an attacker.

2. The authenticator A must prove to client C that it really is the authenticator. Again, proof that replay is not occurring is also required.

3. A and C must securely share a session key to be used for distribution of later authorization material between them. Again, no replay is allowable, and the key must be known only to A and C.

4. A must receive evidence from C that allows A to look up C's authorized operations with server S.

5. C must receive a ticket from A that will prove to S that C can perform its authorized operations. This ticket must be usable only by C.

6. C must receive from A a session key to protect the communications between C and S. The session key must be fresh and not the result of a replay.

10.2.2 Getting Started With Authorization

When the client first needs to get service, it contacts the monitor. At the moment, it has no tickets. Therefore, it uses the "unknown" protocol to talk to the monitor. This protocol is specified as `CEPH_AUTH_UNKNOWN`. The monitor also takes on the authentication server role, A. The remainder of the communications will use the cephx protocol (most of whose code will be found in files in `auth/cephx`). This protocol is responsible for creating and communicating the tickets spoken of above.

Currently, this document does not follow the pre-cephx protocol flow. It starts up at the point where the client has contacted the server and is ready to start the cephx protocol itself.

Once we are in the cephx protocol, we can get the tickets. First, C needs a ticket that allows secure communications with A. This ticket can then be used to obtain other tickets. This is phase I of the protocol, and consists of a send from C to A and a response from A to C. Then, C needs a ticket to allow it to talk to S to get services. This is phase II of the protocol, and consists of a send from C to A and a response from A to C.

10.2.3 Phase I:

The client is set up to know that it needs certain things, using a variable called `need`, which is part of the `AuthClientHandler` class, which the `CephxClientHandler` inherits from. At this point, one thing that's encoded in the `need` variable is `CEPH_ENTITY_TYPE_AUTH`, indicating that we need to start the authentication protocol from scratch. Since we're always talking to the same authorization server, if we've gone through this step of the protocol before (and the resulting ticket/session hasn't timed out), we can skip this step and just ask for client tickets. But it must be done initially, and we'll assume that we are in that state.

The message C sends to A in phase I is build in `CephxClientHandler::build_request()` (in `auth/cephx/CephxClientHandler.cc`). This routine is used for more than one purpose. In this case, we first call `validate_tickets()` (from routine `CephXTicektManager::validate_tickets()` which lives in `auth/cephx/CephxProtocol.h`). This code runs through the list of possible tickets to determine what we need, setting values in the `need` flag as necessary. Then we call `ticket.get_handler()`. This routine (in `CephxProtocol.h`) finds a ticket of the specified type (a ticket to perform authorization) in the ticket map, creates a ticket handler object for it, and puts the handler into the right place in the map. Then we hit specialized code to deal with individual cases. The case here is when we still need to authenticate to A (the `if (need & CEPH_ENTITY_TYPE_AUTH)` branch).

We now create a message of type `CEPH_AUTH_UNKNOWN`. We need to authenticate this message with C's secret key, so we fetch that from the local key repository. (It's called a key server in the code, but it's not really a separate machine or processing entity. It's more like the place where locally used keys are kept.) We create a random challenge, whose purpose is to prevent replays. We encrypt that challenge. We already have a server challenge (a similar set of random bytes, but created by the server and sent to the client) from our pre-cephx stage. We take both challenges and our secret key and produce a combined encrypted challenge value, which goes into `req.key`.

If we have an old ticket, we store it in `req.old_ticket`. We're about to get a new one.

The entire `req` structure, including the old ticket and the cryptographic hash of the two challenges, gets put into the message. Then we return from this function, and the message is sent.

We now switch over to the authenticator side, A. The server receives the message that was sent, of type `CEPH_AUTH_UNKNOWN`. The message gets handled in `prep_auth()`, in `mon/AuthMonitor.cc`, which calls `handle_request()` is `CephxServiceHandler.cc` to do most of the work. This routine, also, handles multiple cases.

The control flow is determined by the `request_type` in the `cephx_header` associated with the message. Our case here is `CEPH_AUTH_UNKNOWN`. We need the secret key A shares with C, so we call `get_secret()` from out local key repository to get it. We should have set up a server challenge already with this client, so we make sure we really do have one. (This variable is specific to a `CephxServiceHandler`, so there is a different one for each such structure we create, presumably one per client A is dealing with.) If there is no challenge, we'll need to start over, since we need to check the client's crypto hash, which depends on a server challenge, in part.

We now call the same routine the client used to calculate the hash, based on the same values: the client challenge (which is in the incoming message), the server challenge (which we saved), and the client's key (which we just obtained). We check to see if the client sent the same thing we expected. If so, we know we're talking to the right client. We know the session is fresh, because it used the challenge we sent it to calculate its crypto hash. So we can give it an authentication ticket.

We fetch C's `eauth` structure. This contains an ID, a key, and a set of caps (capabilities).

The client sent us its old ticket in the message, if it had one. If so, we set a flag, `should_enc_ticket`, to true and set the global ID to the global ID in that old ticket. If the attempt to decode its old ticket fails (most probably because it didn't have one), `should_enc_ticket` remains false. Now we set up the new ticket, filling in timestamps, the name of C, the global ID provided in the method call (unless there was an old ticket), and his `auid`, obtained from the `eauth` structure obtained above. We need a new session key to help the client communicate securely with us, not using its permanent key. We set the service ID to `CEPH_ENTITY_TYPE_AUTH`, which will tell the client C what to do with the message we send it. We build a cephx response header and call `cephx_build_service_ticket_reply()`.

`cephx_build_service_ticket_reply()` is in `auth/cephx/CephxProtocol.cc`. This routine will build up the response message. Much of it copies data from its parameters to a message structure. Part of that information (the session key and the validity period) gets encrypted with C's permanent key. If the `should_encrypt_ticket` flag is set, encrypt it using the old ticket's key. Otherwise, there was no old ticket key, so the new ticket is not encrypted. (It is, of course, already encrypted with A's permanent key.) Presumably the point of this second encryption is to expose less material encrypted with permanent keys.

Then we call the key server's `get_service_caps()` routine on the entity name, with a flag `CEPH_ENTITY_TYPE_MON`, and capabilities, which will be filled in by this routine. The use of that constant flag means we're going to get the client's caps for A, not for some other data server. The ticket here is to access the authorizer A, not the service S. The result of this call is that the caps variable (a parameter to the routine we're in) is filled in with the monitor capabilities that will allow C to access A's authorization services.

`handle_request()` itself does not send the response message. It builds up the `result_bl`, which basically holds that message's contents, and the capabilities structure, but it doesn't send the message. We go back to `prep_auth()`, in `mon/AuthMonitor.cc`, for that. This routine does some fiddling around with the caps structure that just got filled in. There's a global ID that comes up as a result of this fiddling that is put into the reply message. The reply message is built here (mostly from the `response_bl` buffer) and sent off.

This completes Phase I of the protocol. At this point, C has authenticated itself to A, and A has generated a new session key and ticket allowing C to obtain server tickets from A.

10.2.4 Phase II

This phase starts when C receives the message from A containing a new ticket and session key. The goal of this phase is to provide A with a session key and ticket allowing it to communicate with S.

The message A sent to C is dispatched to `build_request()` in `CephxClientHandler.cc`, the same routine that was used early in Phase I to build the first message in the protocol. This time, when `validate_tickets()` is called, the `need` variable will not contain `CEPH_ENTITY_TYPE_AUTH`, so a different branch through the bulk of the routine will be used. This is the branch indicated by `if (need)`. We have a ticket for the authorizer, but we still need service tickets.

We must send another message to A to obtain the tickets (and session key) for the server S. We set the `request_type` of the message to `CEPHX_GET_PRINCIPAL_SESSION_KEY` and call `ticket_handler.build_authorizer()` to obtain an authorizer. This routine is in `CephxProtocol.cc`. We set the key for this authorizer to be the session key we just got from A,and create a new nonce. We put the global ID, the service ID, and the ticket into a message buffer that is part of the authorizer. Then we create a new `CephXAuthorize` structure. The nonce we just created goes there. We encrypt this `CephXAuthorize` structure with the current session key and stuff it into the authorizer's buffer. We return the authorizer.

Back in `build_request()`, we take the part of the authorizer that was just built (its buffer, not the session key or anything else) and shove it into the buffer we're creating for the message that will go to A. Then we delete the authorizer. We put the requirements for what we want in `req.keys`, and we put `req` into the buffer. Then we return, and the message gets sent.

The authorizer A receives this message which is of type `CEPHX_GET_PRINCIPAL_SESSION_KEY`. The message gets handled in `prep_auth()`, in `mon/AuthMonitor.cc`, which again calls `handle_request()` in `CephxServiceHandler.cc` to do most of the work.

In this case, `handle_request()` will take the `CEPHX_GET_PRINCIPAL_SESSION_KEY` case. It will call `cephx_verify_authorizer()` in `CephxProtocol.cc`. Here, we will grab a bunch of data out of the input buffer, including the global and service IDs and the ticket for A. The ticket contains a `secret_id`, indicating which key is being used for it. If the secret ID pulled out of the ticket was -1, the ticket does not specify which secret key A should use. In this case, A should use the key for the specific entity that C wants to contact, rather than a rotating key shared by all server entities of the same type. To get that key, A must consult the key repository to find the right key. Otherwise, there's already a structure obtained from the key repository to hold the necessary secret. Server secrets rotate on a time expiration basis (key rotation is not covered in this document), so run through that structure to find its current secret. Either way, A now knows the secret key used to create this ticket. Now decrypt the encrypted part of the ticket, using this key. It should be a ticket for A.

The ticket also contains a session key that C should have used to encrypt other parts of this message. Use that session key to decrypt the rest of the message.

Create a `CephXAuthorizeReply` to hold our reply. Extract the nonce (which was in the stuff we just decrypted), add 1 to it, and put the result in the reply. Encrypt the reply and put it in the buffer provided in the call to `cephx_verify_authorizer()` and return to `handle`'`_request()`. This will be used to prove to C that A (rather than an attacker) created this response.

Having verified that the message is valid and from C, now we need to build it a ticket for S. We need to know what S it wants to communicate with and what services it wants. Pull the ticket request that describes those things out of its message. Now run through the ticket request to see what it wanted. (He could potentially be asking for multiple different services in the same request, but we will assume it's just one, for this discussion.) Once we know which service ID it's after, call `build_session_auth_info()`.

`build_session_auth_info()` is in `CephxKeyServer.cc`. It checks to see if the secret for the `service_ID` of S is available and puts it into the subfield of one of the parameters, and calls the similarly named `_build_session_auth_info()`, located in the same file. This routine loads up the new `auth_info` structure with the ID of S, a ticket, and some timestamps for that ticket. It generates a new session key and puts it in the structure.

It then calls `get_caps()` to fill in the `info.ticket` caps field. `get_caps()` is also in `CephxKeyServer.cc`. It fills the `caps_info` structure it is provided with caps for S allowed to C.

Once `build_session_auth_info()` returns, A has a list of the capabilities allowed to C for S. We put a validity period based on the current TTL for this context into the info structure, and put it into the `info_vec` structure we are preparing in response to the message.

Now call `build_cephx_response_header()`, also in `CephxServiceHandler.cc`. Fill in the `request_type`, which is `CEPHX_GET_PRINCIPAL_SESSION_KEY`, a status of 0, and the result buffer.

Now call `cephx_build_service_ticket_reply()`, which is in `CephxProtocol.cc`. The same routine was used towards the end of A's handling of its response in phase I. Here, the session key (now a session key to talk to S, not A) and the validity period for that key will be encrypted with the existing session key shared between C and A. The `should_encrypt_ticket` parameter is false here, and no key is provided for that encryption. The ticket in question, destined for S once C sends it there, is already encrypted with S's secret. So, essentially, this routine will put ID information, the encrypted session key, and the ticket allowing C to talk to S into the buffer to be sent to C.

After this routine returns, we exit from `handle_request()`, going back to `prep_auth()` and ultimately to the underlying message send code.

The client receives this message. The nonce is checked as the message passes through `Pipe::connect()`, which is in `msg/SimpleMessager.cc`. In a lengthy `while(1)` loop in the middle of this routine, it gets an authorizer. If the get was successful, eventually it will call `verify_reply()`, which checks the nonce. `connect()` never explicitly checks to see if it got an authorizer, which would suggest that failure to provide an authorizer would allow an attacker to skip checking of the nonce. However, in many places, if there is no authorizer, important connection fields will get set to zero, which will ultimately cause the connection to fail to provide data. It would be worth testing, but it looks like failure to provide an authorizer, which contains the nonce, would not be helpful to an attacker.

The message eventually makes its way through to `handle_response()`, in `CephxClientHandler.cc`. In this routine, we call `get_handler()` to get a ticket handler to hold the ticket we have just received. This routine is embedded in the definition for a `CephXTicketManager` structure. It takes a type (`CEPH_ENTITY_TYPE_AUTH`, in this case) and looks through the `tickets_map` to find that type. There should be one, and it should have the session key of the session between C and A in its entry. This key will be used to decrypt the information provided by A, particularly the new session key allowing C to talk to S.

We then call `verify_service_ticket_reply()`, in `CephxProtocol.cc`. This routine needs to determine if the ticket is OK and also obtain the session key associated with this ticket. It decrypts the encrypted portion of the message buffer, using the session key shared with A. This ticket was not encrypted (well, not twice - tickets are always encrypted, but sometimes double encrypted, which this one isn't). So it can be stored in a service ticket buffer directly. We now grab the ticket out of that buffer.

The stuff we decrypted with the session key shared between C and A included the new session key. That's our current session key for this ticket, so set it. Check validity and set the expiration times. Now return true, if we got this far.

Back in `handle_response()`, we now call `validate_tickets()` to adjust what we think we need, since we now have a ticket we didn't have before. If we've taken care of everything we need, we'll return 0.

This ends phase II of the protocol. We have now successfully set up a ticket and session key for client C to talk to server S. S will know that C is who it claims to be, since A will verify it. C will know it is S it's talking to, again because A verified it. The only copies of the session key for C and S to communicate were sent encrypted under the permanent keys of C and S, respectively, so no other party (excepting A, who is trusted by all) knows that session key. The ticket will securely indicate to S what C is allowed to do, attested to by A. The nonces passed back and forth between A and C ensure that they have not been subject to a replay attack. C has not yet actually talked to S, but it is ready to.

Much of the security here falls apart if one of the permanent keys is compromised. Compromise of C's key means that the attacker can pose as C and obtain all of C's privileges, and can eavesdrop on C's legitimate conversations. He can also pretend to be A, but only in conversations with C. Since it does not (by hypothesis) have keys for any services, he cannot generate any new tickets for services, though it can replay old tickets and session keys until S's permanent key is changed or the old tickets time out.

Compromise of S's key means that the attacker can pose as S to anyone, and can eavesdrop on any user's conversation with S. Unless some client's key is also compromised, the attacker cannot generate new fake client tickets for S, since doing so requires it to authenticate himself as A, using the client key it doesn't know.

10.3 Configuration Management System

The configuration management system exists to provide every daemon with the proper configuration information. The configuration can be viewed as a set of key-value pairs.

How can the configuration be set? Well, there are several sources:

- the ceph configuration file, usually named ceph.conf
- **command line arguments::** –debug-ms=1 –debug-pg=10 etc.
- arguments injected at runtime by using injectargs

10.3.1 The Configuration File

Most configuration settings originate in the Ceph configuration file.

How do we find the configuration file? Well, in order, we check:

- the default locations
- the environment variable CEPH_CONF
- the command line argument -c

Each stanza of the configuration file describes the key-value pairs that will be in effect for a particular subset of the daemons. The "global" stanza applies to everything. The "mon", "osd", and "mds" stanzas specify settings to take effect for all monitors, all OSDs, and all mds servers, respectively. A stanza of the form mon.$name, osd.$name, or mds.$name gives settings for the monitor, OSD, or MDS of that name, respectively. Configuration values that appear later in the file win over earlier ones.

A sample configuration file can be found in src/sample.ceph.conf.

10.3.2 Metavariables

The configuration system allows any configuration value to be substituted into another value using the `$varname` syntax, similar to how bash shell expansion works.

A few additional special metavariables are also defined:

- $host: expands to the current hostname
- $type: expands to one of "mds", "osd", "mon", or "client"
- $id: expands to the daemon identifier. For `osd.0`, this would be `0`; for `mds.a`, it would be `a`; for `client.admin`, it would be `admin`.
- $num: same as $id
- $name: expands to $type.$id

10.3.3 Reading configuration values

There are two ways for Ceph code to get configuration values. One way is to read it directly from a variable named "g_conf," or equivalently, "g_ceph_ctx->_conf." The other is to register an observer that will be called every time the relevant configuration values changes. This observer will be called soon after the initial configuration is read, and every time after that when one of the relevant values changes. Each observer tracks a set of keys and is invoked only when one of the relevant keys changes.

The interface to implement is found in common/config_obs.h.

The observer method should be preferred in new code because

- It is more flexible, allowing the code to do whatever reinitialization needs to be done to implement the new configuration value.
- It is the only way to create a std::string configuration variable that can be changed by injectargs.
- Even for int-valued configuration options, changing the values in one thread while another thread is reading them can lead to subtle and impossible-to-diagnose bugs.

For these reasons, reading directly from g_conf should be considered deprecated and not done in new code. Do not ever alter g_conf.

10.3.4 Changing configuration values

Configuration values can be changed by calling g_conf->set_val. After changing the configuration, you should call g_conf->apply_changes to re-run all the affected configuration observers. For convenience, you can call g_conf->set_val_or_die to make a configuration change which you think should never fail.

Injectargs, parse_argv, and parse_env are three other functions which modify the configuration. Just like with set_val, you should call apply_changes after calling these functions to make sure your changes get applied.

10.4 CephContext

A CephContext represents a single view of the Ceph cluster. It comes complete with a configuration, a set of performance counters (PerfCounters), and a heartbeat map. You can find more information about CephContext in src/common/ceph_context.h.

Generally, you will have only one CephContext in your application, called g_ceph_context. However, in library code, it is possible that the library user will initialize multiple CephContexts. For example, this would happen if he called rados_create more than once.

A ceph context is required to issue log messages. Why is this? Well, without the CephContext, we would not know which log messages were disabled and which were enabled. The dout() macro implicitly references g_ceph_context, so it can't be used in library code. It is fine to use dout and derr in daemons, but in library code, you must use ldout and lderr, and pass in your own CephContext object. The compiler will enforce this restriction.

10.5 Corpus structure

ceph.git/ceph-object-corpus is a submodule.:

```
bin/   # misc scripts
archive/$version/objects/$type/$hash  # a sample of encoded objects from a specific version
```

You can also mark known or deliberate incompatibilities between versions with:

```
archive/$version/forward_incompat/$type
```

The presence of a file indicates that new versions of code cannot decode old objects across that $version (this is normally the case).

10.5.1 How to generate an object corpus

We can generate an object corpus for a particular version of ceph like so.

1. Checkout a clean repo (best not to do this where you normally work):

```
git clone ceph.git
cd ceph
git submodule update --init --recursive
```

2. Build with flag to dump objects to /tmp/foo:

```
rm -rf /tmp/foo ; mkdir /tmp/foo
./do_autogen.sh -e /tmp/foo
make
```

3. Start via vstart:

```
cd src
MON=3 OSD=3 MDS=3 RGW=1 ./vstart.sh -n -x
```

4. Use as much functionality of the cluster as you can, to exercise as many object encoder methods as possible:

```
./rados -p rbd bench 10 write -b 123
./ceph osd out 0
./init-ceph stop osd.1
for f in ../qa/workunits/cls/*.sh ; do PATH=".:$PATH" $f ; done
../qa/workunits/rados/test.sh
./ceph_test_librbd
./ceph_test_libcephfs
./init-ceph restart mds.a
```

Do some more stuff with rgw if you know how.

1. Stop:

```
./stop.sh
```

2. Import the corpus (this will take a few minutes):

```
test/encoding/import.sh /tmp/foo `./ceph-dencoder version` ../ceph-object-corpus/archive
test/encoding/import-generated.sh ../ceph-object-corpus/archive
```

3. Prune it! There will be a bazillion copies of various objects, and we only want a representative sample.:

```
pushd ../ceph-object-corpus
bin/prune-archive.sh
popd
```

4. Verify the tests pass:

```
make check-local
```

5. Commit it to the corpus repo and push:

```
     pushd ../ceph-object-corpus
     git checkout -b wip-new
     git add archive/`../src/ceph-dencoder version`
     git commit -m `../src/ceph-dencoder version`
     git remote add cc ceph.com:/git/ceph-object-corpus.git
     git push cc wip-new
     popd
```

6. Go test it out:

```
     cd my/regular/tree
     cd ceph-object-corpus
     git fetch origin
     git checkout wip-new
     cd ../src
     make check-local
```

7. If everything looks good, update the submodule master branch, and commit the submodule in ceph.git.

10.6 Installing Oprofile

The easiest way to profile Ceph's CPU consumption is to use the oprofile system-wide profiler.

10.6.1 Installation

If you are using a Debian/Ubuntu distribution, you can install `oprofile` by executing the following:

```
sudo apt-get install oprofile oprofile-gui
```

10.6.2 Compiling Ceph for Profiling

To compile Ceph for profiling, first clean everything.

```
make distclean
```

Then, export the following settings so that you can see callgraph output.

```
export CFLAGS="-fno=omit-frame-pointer -O2 -g"
```

Finally, compile Ceph.

```
./autogen.sh
./configure
make
```

You can use `make -j` to execute multiple jobs depending upon your system. For example:

```
make -j4
```

10.6.3 Ceph Configuration

Ensure that you disable `lockdep`. Consider setting logging to levels appropriate for a production cluster. See Ceph Logging and Debugging for details.

See the CPU Profiling section of the RADOS Troubleshooting documentation for details on using Oprofile.

10.7 CephFS delayed deletion

When you delete a file, the data is not immediately removed. Each object in the file needs to be removed independently, and sending `size_of_file / stripe_size * replication_count` messages would slow the client down too much, and use a too much of the clients bandwidth. Additionally, snapshots may mean some objects should not be deleted.

Instead, the file is marked as deleted on the MDS, and deleted lazily.

10.8 Deploying a development cluster

In order to develop on ceph, a Ceph utility, *vstart.sh*, allows you to deploy fake local cluster for development purpose.

10.8.1 Usage

It allows to deploy a fake local cluster on your machine for development purpose. It starts mon, osd and/or mds, or all of them if not specified.

To start your development cluster, type the following:

```
vstart.sh [OPTIONS]... [mon] [osd] [mds]
```

In order to stop the cluster, you can type:

```
./stop.sh
```

10.8.2 Options

-i `ip_address`
: Bind to the specified *ip_address* instead of guessing and resolve from hostname.

-k
: Keep old configuration files instead of overwritting theses.

-l, --localhost
: Use localhost instead of hostanme.

-m `ip[:port]`
: Specifies monitor *ip* address and *port*.

-n, --new
: Create a new cluster.

-o `config`
: Add *config* to all sections in the ceph configuration.

-r
: Start radosgw (ceph needs to be compiled with –radosgw), create an apache2 configuration file, and start apache2 with it (needs apache2 with mod_fastcgi) on port starting from 8000.

--nodaemon
: Use ceph-run as wrapper for mon/osd/mds.

--smallmds
: Configure mds with small limit cache size.

-x
Enable Cephx (on by default).

-X
Disable Cephx.

-d, --debug
Launch in debug mode

--valgrind[_{osd,mds,mon}] 'valgrind_toolname [args...]'
Launch the osd/mds/mon/all the ceph binaries using valgrind with the specified tool and arguments.

10.8.3 Environment variables

{OSD,MDS,MON,RGW}

Theses environment variables will contains the number of instances of the desired ceph process you want to start.

Example:

```
OSD=3 MON=3 RGW=1 vstart.sh
```

10.9 Development workflows

This page explains the workflows a developer is expected to follow to implement the goals that are part of the Ceph lifecycle. It does not go into technical details and is designed to provide a high level view instead. Each chapter is about a given goal such as `Merging bug fixes or features` or `Publishing point releases and backporting`.

A key aspect of all workflows is that none of them blocks another. For instance, a bug fix can be backported and merged to a stable branch while the next point release is being published. For that specific example to work, a branch should be created to avoid any interference. In practice it is not necessary for Ceph because:

- there are few people involved
- the frequency of backports is not too high
- the reviewers know a release is being published are unlikely to merge anything that may cause issues

This ad-hoc approach implies the workflows are changed on a regular basis to adapt. For instance, `quality engineers` were not involved in the workflow to publish dumpling point releases. The number of commits being backported to firefly made it impractical for developers tasked to write code or fix bugs to also run and verify the full suite of integration tests. Inserting `quality engineers` makes it possible for someone to participate in the workflow by analyzing test results.

The workflows are not enforced when they impose an overhead that does not make sense. For instance, if the release notes for a point release were not written prior to checking all integration tests, they can be commited to the stable branch and the result sent for publication without going through another run of integraiton tests.

10.9.1 Lifecycle

```
Ceph                hammer                                 infernalis
Developer            CDS                                      CDS
Summit                 |                                        |
                       |                                        |
development            |                                        |
release          |  v0.88  v0.89  v0.90   ...        |  v0.95
               --v--^----^--v---^------^--v-     ---v----^----^---  2015
                 |         |            |           |
stable         giant       |            |         hammer
release        v0.87       |            |         v0.94
                           |            |
point                    firefly      dumpling
release                  v0.80.8      v0.67.12
```

Four times a year, the development roadmap is discussed online during the Ceph Developer Summit. A new stable release (argonaut, cuttlefish, dumpling, emperor, firefly, giant, hammer ...) is published at the same frequency. Every other release is a long time support (dumpling, firefly, hammer, ...) which means point releases are published more often. In 2014 point releases (i.e. dumpling 0.67.11, dumpling 0.67.12 ...) are published up to eighteen months after the publication of a stable release. Once or twice a month, a development release is published.

10.9.2 Merging bug fixes or features

The development branch is `master` and the workflow followed by all developers can be summarized as follows:

- The developer prepares a series of commits
- The developer submits the series of commits via a pull request
- A reviewer is assigned the pull request
- When the pull request looks good to the reviewer, it is merged into an integration branch by the tester
- After a successfull run of integration tests, the pull request is merged by the tester

The `developer` is the author of a series of commits. The `reviewer` is responsible for providing feedback to the developer on a regular basis and the developer is invited to ping the reviewer if nothing happened after a week. After the `reviewer` is satisfied with the pull request, (s)he passes it to the `tester`. The `tester` is responsible for running teuthology integration tests on the pull request. If nothing happens within a month the reviewer is invited to ping the tester.

10.9.3 Resolving bug reports and implementing features

All bug reports and feature requests are in the issue tracker and the workflow can be summarized as follows:

- The reporter creates the issue with priority `Normal`
- A developer may pick the issue right away
- During a bi-weekly bug scrub, the team goes over all new issue and assign them a priority
- The bugs with higher priority are worked on first

Each `team` is responsible for a project:

- rgw lead is Yehuda Sadeh
- CephFS lead is Gregory Farnum
- rados lead is Samuel Just
- rbd lead is Josh Durgin

The `developer` assigned to an issue is responsible for it. The status of an open issue can be:

- `New`: it is unclear if the issue needs work.
- `Verified`: the bug can be reproduced or showed up multiple times

- `In Progress`: the developer is working on it this week
- `Pending Backport`: the fix needs to be backported to the stable releases listed in the backport field

10.9.4 Running and interpreting teuthology integration tests

The Sepia community test lab runs teuthology integration tests and the results are posted on pulpito and the ceph-qa mailing list.

- The quality engineer analyzes the integration job failure
- If the cause is environmental (e.g. network connectivity), an issue is created in the sepia lab project
- If the bug is known a pulpito URL to failed job is added to the issue
- If the bug is new, an issue is created

The `quality engineer` is either a developer or a member of the QE team. There is at least one integration test suite per project:

- rgw suite
- CephFS suite
- rados suite
- rbd suite

and a many others such as

- upgrade suites
- power-cyle suite
- ...

10.9.5 Preparing a new release

A release is prepared in a dedicated branch, different from the `master` branch.

- For a stable releases it is the branch matching the release code name (dumpling, firefly, etc.)
- For a development release it is the `next` branch

The workflow expected of all developers to stabilize the release candidate is the same as the normal development workflow with the following differences:

- The pull requests must target the stable branch or next instead of master
- The reviewer rejects pull requests that are not bug fixes
- The issues matching a teuthology integration test failure is set with severity `Critical` if it must be fixed before the release

10.9.6 Cutting a new stable release

A new stable release can be cut when:

- all bugs with severity `Critical` are fixed
- integration and upgrade tests run successfully

Publishing a new stable release implies a risk of regression or discovering new bugs during the upgrade, no matter how carefully it is tested. The decision to cut a release must take this into account: it may not be wise to publish a stable release that only fixes a few minor bugs. For instance if only one commit has been backported to a stable release that is not a LTS, it is better to wait until there are more.

When a stable release reaches its end of life, it may be safer to recommend an upgrade to the next long term support release instead of proposing a new point release to fix a problem. For instance, the Dumpling v0.67.11 release has bugs related to backfilling which have been fixed in Firefly v0.80.x. A backport fixing these backfilling bugs has been tested in the draft point release Dumpling v0.67.12 but they are large enough to introduce a risk of regression. As Dumpling is approaching its end of life, users suffering from this bug can upgrade to Firefly to fix it. Unless users manifest themselves and ask for Dumpling v0.67.12, this draft release may never be published.

- The `Ceph lead` decides a new stable release must be published
- The `release master` gets approval from all leads
- The `release master` writes and commits the release notes
- The `release master` informs the `quality engineer` that the branch is ready for testing
- The `quality engineer` runs additional integration tests
- If the `quality engineer` discovers new bugs with severity `Critical`, the relase goes back to being prepared, it was not ready after all
- The `quality engineer` informs the `publisher` that the branch is ready for release
- The `publisher` creates the packages and sets the release tag

The person responsible for each role is:

- Sage Weil is the `Ceph lead`
- Sage Weil is the `release master` for major stable releases (Firefly 0.80, Giant 0.87 etc.)
- Loic Dachary is the `release master` for stable point releases (Dumpling 0.68.12, Giant 0.87.1 etc.)
- Yuri Weinstein is the `quality engineer`
- Alfredo Deza is the `publisher`

10.9.7 Cutting a new development release

The publication workflow of a development release is the same as preparing a new release and cutting it, with the following differences:

- The `next` branch is reset to the tip of `master` after publication
- The `quality engineer` is not required to run additional tests, the `release master` directly informs the `publisher` that the release is ready to be published.

10.9.8 Publishing point releases and backporting

The publication workflow of the point releases is the same as preparing a new release and cutting it, with the following differences:

- The `backport` field of each issue contains the code name of the stable release
- All commits are cherry-picked with `git cherry-pick -x` to reference the original commit

10.10 Differences from POSIX

Todo

delete http://ceph.com/wiki/Differences_from_POSIX

Ceph does have a few places where it diverges from strict POSIX semantics for various reasons:

- Sparse files propagate incorrectly to tools like df. They will only use up the required space, but in df will increase the "used" space by the full file size. We do this because actually keeping track of the space a large, sparse file uses is very expensive.
- In shared simultaneous writer situations, a write that crosses object boundaries is not necessarily atomic. This means that you could have writer A write "aa|aa" and writer B write "bb|bb" simultaneously (where | is the object boundary), and end up with "aa|bb" rather than the proper "aa|aa" or "bb|bb".

10.11 Documenting Ceph

10.11.1 Code Documentation

C and C++ can be documented with Doxygen, using the subset of Doxygen markup supported by Breathe.

The general format for function documentation is:

```
/**
 * Short description
 *
 * Detailed description when necessary
 *
 * preconditons, postconditions, warnings, bugs or other notes
 *
 * parameter reference
 * return value (if non-void)
 */
```

This should be in the header where the function is declared, and functions should be grouped into logical categories. The librados C API provides a complete example. It is pulled into Sphinx by librados.rst, which is rendered at Librados (C).

10.11.2 Drawing diagrams

Graphviz

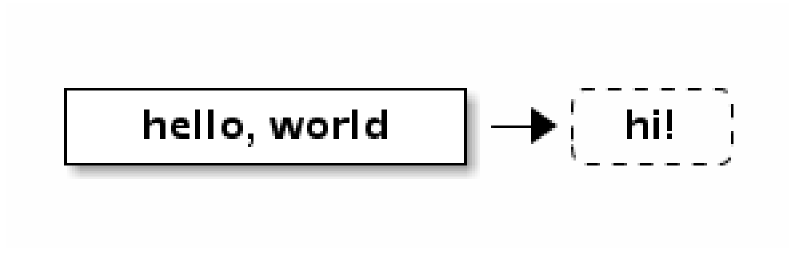
You can use Graphviz, as explained in the Graphviz extension documentation.

Most of the time, you'll want to put the actual DOT source in a separate file, like this:

```
.. graphviz:: myfile.dot
```

Ditaa

You can use Ditaa:



Blockdiag

If a use arises, we can integrate Blockdiag. It is a Graphviz-style declarative language for drawing things, and includes:

- block diagrams: boxes and arrows (automatic layout, as opposed to Ditaa)
- sequence diagrams: timelines and messages between them
- activity diagrams: subsystems and activities in them
- network diagrams: hosts, LANs, IP addresses etc (with Cisco icons if wanted)

Inkscape

You can use Inkscape to generate scalable vector graphics. http://inkscape.org for restructedText documents.

If you generate diagrams with Inkscape, you should commit both the Scalable Vector Graphics (SVG) file and export a Portable Network Graphic (PNG) file. Reference the PNG file.

By committing the SVG file, others will be able to update the SVG diagrams using Inkscape.

HTML5 will support SVG inline.

10.12 Erasure Coded pool

10.12.1 Purpose

Erasure-coded pools require less storage space compared to replicated pools. The erasure-coding support has higher computational requirements and only supports a subset of the operations allowed on an object (for instance, partial write is not supported).

10.12.2 Use cases

Cold storage

An erasure-coded pool is created to store a large number of 1GB objects (imaging, genomics, etc.) and 10% of them are read per month. New objects are added every day and the objects are not modified after being written. On average there is one write for 10,000 reads.

A replicated pool is created and set as a cache tier for the erasure coded pool. An agent demotes objects (i.e. moves them from the replicated pool to the erasure-coded pool) if they have not been accessed in a week.

The erasure-coded pool crush ruleset targets hardware designed for cold storage with high latency and slow access time. The replicated pool crush ruleset targets faster hardware to provide better response times.

Cheap multidatacenter storage

Ten datacenters are connected with dedicated network links. Each datacenter contains the same amount of storage with no power-supply backup and no air-cooling system.

An erasure-coded pool is created with a crush map ruleset that will ensure no data loss if at most three datacenters fail simultaneously. The overhead is 50% with erasure code configured to split data in six (k=6) and create three coding chunks (m=3). With replication the overhead would be 400% (four replicas).

10.12.3 Interface

Set up an erasure-coded pool:

```
$ ceph osd pool create ecpool 12 12 erasure
```

Set up an erasure-coded pool and the associated crush ruleset:

```
$ ceph osd crush rule create-erasure ecruleset
$ ceph osd pool create ecpool 12 12 erasure \
    default ecruleset
```

Set the ruleset failure domain to osd (instead of the host which is the default):

```
$ ceph osd erasure-code-profile set myprofile \
    ruleset-failure-domain=osd
$ ceph osd erasure-code-profile get myprofile
k=2
m=1
plugin=jerasure
technique=reed_sol_van
ruleset-failure-domain=osd
$ ceph osd pool create ecpool 12 12 erasure myprofile
```

Control the parameters of the erasure code plugin:

```
$ ceph osd erasure-code-profile set myprofile \
    k=3 m=1
$ ceph osd erasure-code-profile get myprofile
k=3
m=1
plugin=jerasure
technique=reed_sol_van
$ ceph osd pool create ecpool 12 12 erasure \
    myprofile
```

Choose an alternate erasure code plugin:

```
$ ceph osd erasure-code-profile set myprofile \
    plugin=example technique=xor
$ ceph osd erasure-code-profile get myprofile
k=2
m=1
plugin=example
technique=xor
$ ceph osd pool create ecpool 12 12 erasure \
    myprofile
```

Display the default erasure code profile:

```
$ ceph osd erasure-code-profile ls
default
$ ceph osd erasure-code-profile get default
k=2
m=1
plugin=jerasure
technique=reed_sol_van
```

Create a profile to set the data to be distributed on six OSDs (k+m=6) and sustain the loss of three OSDs (m=3) without losing data:

```
$ ceph osd erasure-code-profile set myprofile k=3 m=3
$ ceph osd erasure-code-profile get myprofile
k=3
m=3
plugin=jerasure
technique=reed_sol_van
$ ceph osd erasure-code-profile ls
default
myprofile
```

Remove a profile that is no longer in use (otherwise it will fail with EBUSY):

```
$ ceph osd erasure-code-profile ls
default
myprofile
$ ceph osd erasure-code-profile rm myprofile
$ ceph osd erasure-code-profile ls
default
```

Set the ruleset to take ssd (instead of default):

```
$ ceph osd erasure-code-profile set myprofile \
    ruleset-root=ssd
$ ceph osd erasure-code-profile get myprofile
k=2
m=1
plugin=jerasure
technique=reed_sol_van
ruleset-root=ssd
```

10.13 File striping

The text below describes how files from Ceph file system clients are stored across objects stored in RADOS.

10.13.1 ceph_file_layout

Ceph distributes (stripes) the data for a given file across a number of underlying objects. The way file data is mapped to those objects is defined by the ceph_file_layout structure. The data distribution is a modified RAID 0, where data is striped across a set of objects up to a (per-file) fixed size, at which point another set of objects holds the file's data. The second set also holds no more than the fixed amount of data, and then another set is used, and so on.

Defining some terminology will go a long way toward explaining the way file data is laid out across Ceph objects.

- **file** A collection of contiguous data, named from the perspective of the Ceph client (i.e., a file on a Linux system using Ceph storage). The data for a file is divided into fixed-size "stripe units," which are stored in ceph "objects."
- **stripe unit** The size (in bytes) of a block of data used in the RAID 0 distribution of a file. All stripe units for a file have equal size. The last stripe unit is typically incomplete–i.e. it represents the data at the end of the file as well as unused "space" beyond it up to the end of the fixed stripe unit size.
- **stripe count** The number of consecutive stripe units that constitute a RAID 0 "stripe" of file data.
- **stripe** A contiguous range of file data, RAID 0 striped across "stripe count" objects in fixed-size "stripe unit" blocks.
- **object** A collection of data maintained by Ceph storage. Objects are used to hold portions of Ceph client files.
- **object set** A set of objects that together represent a contiguous portion of a file.

Three fields in the ceph_file_layout structure define this mapping:

```
u32 fl_stripe_unit;
u32 fl_stripe_count;
u32 fl_object_size;
```

(They are actually maintained in their on-disk format, __le32.)

The role of the first two fields should be clear from the definitions above.

The third field is the maximum size (in bytes) of an object used to back file data. The object size is a multiple of the stripe unit.

A file's data is blocked into stripe units, and consecutive stripe units are stored on objects in an object set. The number of objects in a set is the same as the stripe count. No object storing file data will exceed the file's designated object size, so after some fixed number of complete stripes, a new object set is used to store subsequent file data.

Note that by default, Ceph uses a simple striping strategy in which object_size equals stripe_unit and stripe_count is 1. This simply puts one stripe_unit in each object.

Here's a more complex example:

```
file size = 1 trillion = 1000000000000 bytes

fl_stripe_unit = 64KB = 65536 bytes
fl_stripe_count = 5 stripe units per stripe
fl_object_size = 64GB = 68719476736 bytes
```

This means:

```
file stripe size = 64KB * 5 = 320KB = 327680 bytes
each object holds 64GB / 64KB = 1048576 stripe units
file object set size = 64GB * 5 = 320GB = 343597383680 bytes
    (also 1048576 stripe units * 327680 bytes per stripe unit)
```

So the file's 1 trillion bytes can be divided into complete object sets, then complete stripes, then complete stripe units, and finally a single incomplete stripe unit:

```
- 1 trillion bytes / 320GB per object set = 2 complete object sets
    (with 312805232640 bytes remaining)
- 312805232640 bytes / 320KB per stripe = 954605 complete stripes
    (with 266240 bytes remaining)
- 266240 bytes / 64KB per stripe unit = 4 complete stripe units
    (with 4096 bytes remaining)
- and the final incomplete stripe unit holds those 4096 bytes.
```

The ASCII art below attempts to capture this:

```
   _________   _________   _________   _________   _________
  /object  0\ /object  1\ /object  2\ /object  3\ /object  4\
  +=========+ +=========+ +=========+ +=========+ +=========+
  |  stripe | |  stripe | |  stripe | |  stripe | |  stripe |
o |   unit  | |   unit  | |   unit  | |   unit  | |   unit  | stripe 0
b |    0    | |    1    | |    2    | |    3    | |    4    |
j |---------| |---------| |---------| |---------| |---------|
e |  stripe | |  stripe | |  stripe | |  stripe | |  stripe |
c |   unit  | |   unit  | |   unit  | |   unit  | |   unit  | stripe 1
t |    5    | |    6    | |    7    | |    8    | |    9    |
  |---------| |---------| |---------| |---------| |---------|
s |    .    | |    .    | |    .    | |    .    | |    .    |
e      .           .           .           .           .
t |    .    | |    .    | |    .    | |    .    | |    .    |
  |---------| |---------| |---------| |---------| |---------|
0 |  stripe | |  stripe | |  stripe | |  stripe | |  stripe | stripe
  |   unit  | |   unit  | |   unit  | |   unit  | |   unit  | 1048575
  | 5242875 | | 5242876 | | 5242877 | | 5242878 | | 5242879 |
  \=========/ \=========/ \=========/ \=========/ \=========/


   _________   _________   _________   _________   _________
  /object  5\ /object  6\ /object  7\ /object  8\ /object  9\
  +=========+ +=========+ +=========+ +=========+ +=========+
  |  stripe | |  stripe | |  stripe | |  stripe | |  stripe | stripe
o |   unit  | |   unit  | |   unit  | |   unit  | |   unit  | 1048576
b | 5242880 | | 5242881 | | 5242882 | | 5242883 | | 5242884 |
j |---------| |---------| |---------| |---------| |---------|
e |  stripe | |  stripe | |  stripe | |  stripe | |  stripe | stripe
c |   unit  | |   unit  | |   unit  | |   unit  | |   unit  | 1048577
t | 5242885 | | 5242886 | | 5242887 | | 5242888 | | 5242889 |
  |---------| |---------| |---------| |---------| |---------|
s |    .    | |    .    | |    .    | |    .    | |    .    |
e      .           .           .           .           .
t |    .    | |    .    | |    .    | |    .    | |    .    |
  |---------| |---------| |---------| |---------| |---------|
1 |  stripe | |  stripe | |  stripe | |  stripe | |  stripe | stripe
  |   unit  | |   unit  | |   unit  | |   unit  | |   unit  | 2097151
  | 10485755| | 10485756| | 10485757| | 10485758| | 10485759|
  \=========/ \=========/ \=========/ \=========/ \=========/


   _________   _________   _________   _________   _________
  /object 10\ /object 11\ /object 12\ /object 13\ /object 14\
  +=========+ +=========+ +=========+ +=========+ +=========+
  |  stripe | |  stripe | |  stripe | |  stripe | |  stripe | stripe
o |   unit  | |   unit  | |   unit  | |   unit  | |   unit  | 2097152
b | 10485760| | 10485761| | 10485762| | 10485763| | 10485764|
j |---------| |---------| |---------| |---------| |---------|
e |  stripe | |  stripe | |  stripe | |  stripe | |  stripe | stripe
c |   unit  | |   unit  | |   unit  | |   unit  | |   unit  | 2097153
t | 10485765| | 10485766| | 10485767| | 10485768| | 10485769|
  |---------| |---------| |---------| |---------| |---------|
s |    .    | |    .    | |    .    | |    .    | |    .    |
e      .           .           .           .           .
t |    .    | |    .    | |    .    | |    .    | |    .    |
  |---------| |---------| |---------| |---------| |---------|
2 |  stripe | |  stripe | |  stripe | |  stripe | |  stripe | stripe
  |   unit  | |   unit  | |   unit  | |   unit  | |   unit  | 3051756
  | 15258780| | 15258781| | 15258782| | 15258783| | 15258784|
  |---------| |---------| |---------| |---------| |---------|
  |  stripe | |  stripe | |  stripe | |  stripe | | (partial| (partial
  |   unit  | |   unit  | |   unit  | |   unit  | |  stripe | stripe
  | 15258785| | 15258786| | 15258787| | 15258788| |  unit)  | 3051757)
  \=========/ \=========/ \=========/ \=========/ \=========/
```

10.14 Filestore filesystem compatilibity

http://marc.info/?l=ceph-devel&m=131942130322957&w=2

Although running on ext4, xfs, or whatever other non-btrfs you want mostly works, there are a few important remaining issues:

10.14.1 OSD journal replay of non-idempotent transactions

Resolved with full sync but not ideal. See http://tracker.newdream.net/issues/213

On non-btrfs backends, the Ceph OSDs use a write-ahead journal. After restart, the OSD does not know exactly which transactions in the journal may have already been committed to disk, and may reapply a transaction again during replay. For most operations (write, delete, truncate) this is fine.

Some operations, though, are non-idempotent. The simplest example is CLONE, which copies (efficiently, on btrfs) data from one object to another. If the source object is modified, the osd restarts, and then the clone is replayed, the target will get incorrect (newer) data. For example,

- clone A -> B
- modify A
- <osd crash, replay from 1>

B will get new instead of old contents.

(This doesn't happen on btrfs because the snapshots allow us to replay from a known consistent point in time.)

Possibilities:

- full sync after any non-idempotent operation
- re-evaluate the lower level interface based on needs from higher levels, construct only safe operations, be very careful; brittle
- use xattrs to add sequence numbers to objects:
  - on non-btrfs, we set a xattr on every modified object with the op_seq, the unique sequence number for the transaction.
  - for any (potentially) non-idempotent operation, we fsync() before continuing to the next transaction, to ensure that xattr hits disk.
  - on replay, we skip a transaction if the xattr indicates we already performed this transaction.

  Because every 'transaction' only modifies on a single object (file), this ought to work. It'll make things like clone slow, but let's face it: they're already slow on non-btrfs file systems because they actually copy the data (instead of duplicating the extent refs in btrfs). And it should make the full ObjectStore interface safe, without upper layers having to worry about the kinds and orders of transactions they perform.

10.15 Building Ceph Documentation

Ceph utilizes Python's Sphinx documentation tool. For details on the Sphinx documentation tool, refer to The Sphinx Documentation Tool.

To build the Ceph documentation set, you must:

1. Clone the Ceph repository
2. Install the required tools
3. Build the documents

10.15.1 Clone the Ceph Repository

To clone the Ceph repository, you must have `git` installed on your local host. To install `git`, execute:

```
sudo apt-get install git
```

To clone the Ceph repository, execute:

```
git clone git://github.com/ceph/ceph
```

You should have a full copy of the Ceph repository.

10.15.2 Install the Required Tools

If you do not have Sphinx and its dependencies installed, a list of dependencies will appear in the output. Install the dependencies on your system, and then execute the build.

To run Sphinx, at least the following are required:

- `python-dev`
- `python-pip`
- `python-virtualenv`
- `libxml2-dev`
- `libxslt-dev`
- `doxygen`
- `ditaa`
- `graphviz`

Execute `apt-get install` for each dependency that isn't installed on your host.:

```
sudo apt-get install python-dev python-pip python-virtualenv libxml2-dev libxslt-dev doxygen ditaa gr
```

10.15.3 Build the Documents

Once you have installed all the dependencies, execute the build:

```
cd ceph
admin/build-doc
```

Once you build the documentation set, you may navigate to the source directory to view it:

```
cd build-doc/output
```

There should be an `html` directory and a `man` directory containing documentation in HTML and manpage formats respectively.

10.16 Kernel client troubleshooting (FS)

If there is an issue with the cephfs kernel client, the most important thing is figuring out whether the problem is with the client or the MDS. Generally, this is easy to work out. If the kernel client broke directly, there will be output in dmesg. Collect it and any appropriate kernel state. If the problem is with the MDS, there will be hung requests that the client is waiting on. Look in `/sys/kernel/debug/ceph/*/` and cat the `mdsc` file to get a listing of requests in progress. If one of them remains there, the MDS has probably "forgotten" it. We can get hints about what's going on by dumping the MDS cache: ceph mds tell 0 dumpcache /tmp/dump.txt

And if high logging levels are set on the MDS, that will almost certainly hold the information we need to diagnose and solve the issue.

10.17 Library architecture

Ceph is structured into libraries which are built and then combined together to make executables and other libraries.

- libcommon: a collection of utilities which are available to nearly every ceph library and executable. In general, libcommon should not contain global variables, because it is intended to be linked into libraries such as libcephfs.so.
- libglobal: a collection of utilities focused on the needs of Ceph daemon programs. In here you will find pidfile management functions, signal handlers, and so forth.

Todo

document other libraries

10.18 Debug logs

The main debugging tool for Ceph is the dout and derr logging functions. Collectively, these are referred to as "dout logging."

Dout has several log faculties, which can be set at various log levels using the configuration management system. So it is possible to enable debugging just for the messenger, by setting debug_ms to 10, for example.

Dout is implemented mainly in common/DoutStreambuf.cc

The dout macro avoids even generating log messages which are not going to be used, by enclosing them in an "if" statement. What this means is that if you have the debug level set at 0, and you run this code:

```
dout(20) << "myfoo() = " << myfoo() << dendl;
```

myfoo() will not be called here.

Unfortunately, the performance of debug logging is relatively low. This is because there is a single, process-wide mutex which every debug output statement takes, and every debug output statement leads to a write() system call or a call to syslog(). There is also a computational overhead to using C++ streams to consider. So you will need to be parsimonious in your logging to get the best performance.

Sometimes, enabling logging can hide race conditions and other bugs by changing the timing of events. Keep this in mind when debugging.

10.18.1 Performance counters

Ceph daemons use performance counters to track key statistics like number of inodes pinned. Performance counters are essentially sets of integers and floats which can be set, incremented, and read using the PerfCounters API.

A PerfCounters object is usually associated with a single subsystem. It contains multiple counters. This object is thread-safe because it is protected by an internal mutex. You can create multiple PerfCounters objects.

Currently, three types of performance counters are supported: u64 counters, float counters, and long-run floating-point average counters. These are created by PerfCountersBuilder::add_u64, PerfCountersBuilder::add_fl, and PerfCountersBuilder::add_fl_avg, respectively. u64 and float counters simply provide a single value which can be updated, incremented, and read atomically. floating-pointer average counters provide two values: the current total, and the number of times the total has been changed. This is intended to provide a long-run average value.

Performance counter information can be read in JSON format from the administrative socket (admin_sock). This is implemented as a UNIX domain socket. The Ceph performance counter plugin for collectd shows an example of how to access this information. Another example can be found in the unit tests for the administrative sockets.

10.19 Monitor bootstrap

Terminology:

- `cluster`: a set of monitors
- `quorum`: an active set of monitors consisting of a majority of the cluster

In order to initialize a new monitor, it must always be fed:

1. a logical name
2. secret keys
3. a cluster fsid (uuid)

In addition, a monitor needs to know two things:

1. what address to bind to
2. who its peers are (if any)

There are a range of ways to do both.

10.19.1 Logical id

The logical id should be unique across the cluster. It will be appended to `mon.` to logically describe the monitor in the Ceph cluster. For example, if the logical id is `foo`, the monitor's name will be `mon.foo`.

For most users, there is no more than one monitor per host, which makes the short hostname logical choice.

10.19.2 Secret keys

The mon. secret key is stored a keyring file in the mon data directory. It can be generated with a command like:

```
ceph-authtool --create-keyring /path/to/keyring --gen-key -n mon.
```

When creating a new monitor cluster, the keyring should also contain a client.admin key that can be used to administer the system:

```
ceph-authtool /path/to/keyring --gen-key -n client.admin --set-uid=0 --cap mon 'allow *' --cap osd 'a
```

The resulting keyring is fed to ceph-mon --mkfs with the --keyring <keyring> command-line argument.

10.19.3 Cluster fsid

The cluster fsid is a normal uuid, like that generated by the uuidgen command. It can be provided to the monitor in two ways:

1. via the --fsid <uuid> command-line argument (or config file option)
2. via a monmap provided to the new monitor via the --monmap <path> command-line argument.

10.19.4 Monitor address

The monitor address can be provided in several ways.

1. via the --public-addr <ip[:port]> command-line option (or config file option)
2. via the --public-network <cidr> command-line option (or config file option)
3. via the monmap provided via --monmap <path>, if it includes a monitor with our name
4. via the bootstrap monmap (provided via --inject-monmap <path> or generated from --mon-host <list>) if it includes a monitor with no name (noname-<something>) and an address configured on the local host.

10.19.5 Peers

The monitor peers are provided in several ways:

1. via the initial monmap, provided via --monmap <filename>
2. via the bootstrap monmap generated from --mon-host <list>
3. via the bootstrap monmap generated from [mon.*] sections with mon addr in the config file
4. dynamically via the admin socket

However, these methods are not completely interchangeable because of the complexity of creating a new monitor cluster without danger of races.

10.19.6 Cluster creation

There are three basic approaches to creating a cluster:

1. Create a new cluster by specifying the monitor names and addresses ahead of time.

2. Create a new cluster by specifying the monitor names ahead of time, and dynamically setting the addresses as `ceph-mon` daemons configure themselves.
3. Create a new cluster by specifying the monitor addresses ahead of time.

Names and addresses

Generate a monmap using `monmaptool` with the names and addresses of the initial monitors. The generated monmap will also include a cluster fsid. Feed that monmap to each monitor daemon:

```
ceph-mon --mkfs -i <name> --monmap <initial_monmap> --keyring <initial_keyring>
```

When the daemons start, they will know exactly who they and their peers are.

Addresses only

The initial monitor addresses can be specified with the `mon host` configuration value, either via a config file or the command-line argument. This method has the advantage that a single global config file for the cluster can have a line like:

```
mon host = a.foo.com, b.foo.com, c.foo.com
```

and will also serve to inform any ceph clients or daemons who the monitors are.

The `ceph-mon` daemons will need to be fed the initial keyring and cluster fsid to initialize themselves:

> ceph-mon –mkfs -i <name> –fsid <uuid> –keyring <initial_keyring>

When the daemons first start up, they will share their names with each other and form a new cluster.

Names only

In dynamic “cloud” environments, the cluster creator may not (yet) know what the addresses of the monitors are going to be. Instead, they may want machines to configure and start themselves in parallel and, as they come up, form a new cluster on their own. The problem is that the monitor cluster relies on strict majorities to keep itself consistent, and in order to “create” a new cluster, it needs to know what the *initial* set of monitors will be.

This can be done with the `mon initial members` config option, which should list the ids of the initial monitors that are allowed to create the cluster:

```
mon initial members = foo, bar, baz
```

The monitors can then be initialized by providing the other pieces of information (they keyring, cluster fsid, and a way of determining their own address). For example:

```
ceph-mon --mkfs -i <name> --mon-initial-hosts 'foo,bar,baz' --keyring <initial_keyring> --public-addr
```

When these daemons are started, they will know their own address, but not their peers. They can learn those addresses via the admin socket:

```
ceph --admin-daemon /var/run/ceph/mon.<id>.asok add_bootstrap_peer_hint <peer ip>
```

Once they learn enough of their peers from the initial member set, they will be able to create the cluster.

10.19.7 Cluster expansion

Cluster expansion is slightly less demanding than creation, because the creation of the initial quorum is not an issue and there is no worry about creating separately independent clusters.

New nodes can be forced to join an existing cluster in two ways:

1. by providing no initial monitor peers addresses, and feeding them dynamically.
2. by specifying the `mon initial members` config option to prevent the new nodes from forming a new, independent cluster, and feeding some existing monitors via any available method.

Initially peerless expansion

Create a new monitor and give it no peer addresses other than it's own. For example:

```
ceph-mon --mkfs -i <myid> --fsid <fsid> --keyring <mon secret key> --public-addr <ip>
```

Once the daemon starts, you can give it one or more peer addresses to join with:

```
ceph --admin-daemon /var/run/ceph/mon.<id>.asok add_bootstrap_peer_hint <peer ip>
```

This monitor will never participate in cluster creation; it can only join an existing cluster.

Expanding with initial members

You can feed the new monitor some peer addresses initially and avoid badness by also setting `mon initial members`. For example:

```
ceph-mon --mkfs -i <myid> --fsid <fsid> --keyring <mon secret key> --public-addr <ip> --mon-host foo,
```

When the daemon is started, `mon initial members` must be set via the command line or config file:

```
ceph-mon -i <myid> --mon-initial-members foo,bar,baz
```

to prevent any risk of split-brain.

10.20 Network Encoding

This describes the encoding used to serialize data. It doesn't cover specific objects/messages but focuses on the base types.

The types are not self documenting in any way. They can not be decoded unless you know what they are.

10.20.1 Conventions

Integers

The integer types used will be named `{signed}{size}{endian}`. For example `u16le` is an unsigned 16 bit integer encoded in little endian byte order while `s64be` is a signed 64 bit integer in big endian. Additionally `u8` and `s8` will represent signed and unsigned bytes respectively. Signed integers use two's complement encoding.

Complex Types

This document will use a c-like syntax for describing structures. The structure represents the data that will go over the wire. There will be no padding between the elements and the elements will be sent in the order they appear. For example:

```
struct foo {
        u8    tag;
        u32le data;
}
```

When encoding the values `0x05` and `0x12345678` respectively will appear on the wire as `05 78 56 34 12`.

Variable Arrays

Unlike c, length arrays can be used anywhere in structures and will be inline in the protocol. Furthermore the length may be described using an earlier item in the structure.

```
struct blob {
        u32le size;
        u8    data[size];
        u32le checksum;
}
```

This structure is encoded as a 32 bit size, followed by `size` data bytes, then a 32 bit checksum.

Primitive Aliases

These types are just aliases for primitive types.

```
// From /src/include/types.h

typedef u32le epoch_t;
typedef u32le ceph_seq_t;
typedef u64le ceph_tid_t;
typedef u64le version_t;
```

10.20.2 Structures

These are the way structures are encoded. Note that these structures don't actually exist in the source but are the way that different types are encoded.

Optional

Optionals are represented as a presence byte, followed by the item if it exists.

```
struct ceph_optional<T> {
        u8 present;
        T  element[present? 1 : 0]; // Only if present is non-zero.
}
```

Optionals are used to encode `boost::optional`.

Pair

Pairs are simply the first item followed by the second.

```
struct ceph_pair<A,B> {
        A a;
        B b;
}
```

Pairs are used to encode `std::pair`.

Triple

Triples are simply the tree elements one after another.

```
struct ceph_triple<A,B,C> {
        A a;
        B b;
        C c;
}
```

Triples are used to encode `ceph::triple`.

List

Lists are represented as an element count followed by that many elements.

```
struct ceph_list<T> {
        u32le length;
        T     elements[length];
}
```

Note: The size of the elements in the list are not necessarily uniform.

Lists are used to encode `std::list`, `std::vector`, `std::deque`, `std::set` and `ceph::unordered_set`.

Blob

A Blob is simply a list of bytes.

```
struct ceph_string {
        ceph_list<u8>;
}

// AKA

struct ceph_string {
        u32le size;
        u8    data[size];
}
```

Blobs are used to encode `std::string`, `const char *` and `bufferlist`.

Note: The content of a Blob is arbratrary binary data.

Map

Maps are a list of pairs.

```
struct ceph_map<K,V> {
        ceph_list<ceph_pair<K,V>>;
}

// AKA

struct ceph_map<K,V> {
        u32le length;
        ceph_pair<K,V> entries[length];
}
```

Maps are used to encode `std::map`, `std::multimap` and `ceph::unordered_map`.

10.20.3 Complex Types

These aren't hard to find in the source but the common ones are listed here for convenience.

utime_t

```
// From /src/include/utime.h
struct utime_t {
        u32le tv_sec;  // Seconds since epoch.
        u32le tv_nsec; // Nanoseconds since the last second.
}
```

ceph_entity_name

```
// From /src/include/msgr.h
struct ceph_entity_name {
        u8    type; // CEPH_ENTITY_TYPE_*
        u64le num;
}

// CEPH_ENTITY_TYPE_* defined in /src/include/msgr.h
```

10.21 Network Protocol

This file describes the network protocol used by Ceph. In order to understand the way the structures are defined it is recommended to read the introduction of Network Encoding first.

10.21.1 Hello

The protocol starts with a handshake that confirms that both nodes are talking ceph and shares some basic information.

Banner

The first action is the server sending banner to the client. The banner is defined in CEPH_BANNER from src/include/msgr.h. This is followed by the server's then client's address each encoded as a entity_addr_t.

Once the client verifies that the servers banner matches its own it replies with its banner and its address.

Connect

Once the banners have been verified and the addresses exchanged the connection negotiation begins. First the client sends a ceph_msg_connect structure with its information.

```
// From src/include/msgr.h
struct ceph_msg_connect {
        u64le features;            // Supported features (CEPH_FEATURE_*)
        u32le host_type;           // CEPH_ENTITY_TYPE_*
        u32le global_seq;          // Number of connections initiated by this host.
        u32le connect_seq;         // Number of connections initiated in this session.
        u32le protocol_version;
        u32le authorizer_protocol;
        u32le authorizer_len;
        u8    flags;               // CEPH_MSG_CONNECT_*
        u8    authorizer[authorizer_len];
}
```

Connect Reply

Once the connect has been sent the connection has effectively been opened, however the first message the server sends must be a connect reply message.

```
struct ceph_msg_connect_reply {
        u8    tag; // Tag indicating response code.
        u64le features;
        u32le global_seq;
        u32le connect_seq;
        u32le protocol_version;
        u32le authorizer_len;
        u8    flags;
        u8    authorizer[authorizer_len];
}
```

10.21.2 MSGR Protocol

This is a low level protocol over which messages are delivered. The messages at this level consist of a tag byte, identifying the type of message, followed by the message data.

```
// Virtual structure.
struct {
        u8 tag; // CEPH_MSGR_TAG_*
        u8 data[]; // Length depends on tag and data.
}
```

The length of data is determined by the tag byte and depending on the message type via information in the data array itself.

Note: There is no way to determine the length of the message if you do not understand the type of message.

The message tags are defined in `src/include/msgr.h` and the current ones are listed below along with the data they include. Note that the defined structures don't exist in the source and are merely for representing the protocol.

CEPH_MSGR_TAG_CLOSE (0x06)

```
struct ceph_msgr_close {
        u8 tag = 0x06;
        u8 data[0]; // No data.
}
```

The close message indicates that the connection is being closed.

CEPH_MSGR_TAG_MSG (0x07)

```
struct ceph_msgr_msg {
        u8 tag = 0x07;
        ceph_msg_header header;
        u8 front [header.front_len ];
        u8 middle[header.middle_len];
        u8 data  [header.data_len  ];
        ceph_msg_footer footer;
}

// From src/include/msgr.h
struct ceph_msg_header {
        u64le seq;       // Sequence number.
        u64le tid;       // Transaction ID.
        u16le type;      // Message type (CEPH_MSG_* or MSG_*).
        u16le priority;  // Priority (higher is more important).
        u16le version;   // Version of message encoding.

        u32le front_len;  // The size of the front section.
        u32le middle_len; // The size of the middle section.
        u32le data_len;   // The size of the data section.
        u16le data_off;   // The way data should be aligned by the reciever.

        ceph_entity_name src; // Information about the sender.

        u16le compat_version; // Oldest compatible encoding version.
        u16le reserved;       // Unused.
        u32le crc;            // CRC of header.
}

// From src/include/msgr.h
struct ceph_msg_footer {
        u32le front_crc;  // Checksums of the various sections.
        u32le middle_crc; //
        u32le data_crc;   //
        u64le sig; // Crypographic signature.
        u8    flags;
}
```

Messages are the business logic of Ceph. They are what is used to send data and requests between nodes. The message header contains the length of the message so unknown messages can be handled gracefully.

There are two names for the message type constants `CEPH_MSG_*` and `MSG_*`. The only difference between the two is that the first are considered "public" while the second is for internal use only. There is no protocol-level difference.

CEPH_MSGR_TAG_ACK (0x08)

```
struct ceph_msgr_ack {
        u8    tag = 0x08;
        u64le seq; // The sequence number of the message being acknowledged.
}
```

CEPH_MSGR_TAG_KEEPALIVE (0x09)

```
struct ceph_msgr_keepalive {
        u8 tag = 0x09;
        u8 data[0]; // No data.
}
```

CEPH_MSGR_TAG_KEEPALIVE2 (0x04)

```
struct ceph_msgr_keepalive2 {
        u8      tag = 0x0E;
        utime_t timestamp;
}
```

CEPH_MSGR_TAG_KEEPALIVE2_ACK (0x05)

```
struct ceph_msgr_keepalive2_ack {
        u8      tag = 0x0F;
        utime_t timestamp;
}
```

10.22 Object Store Architecture Overview

Todo

write more here

10.23 OSD class path issues

```
2011-12-05 17:41:00.994075 7ffe8b5c3760 librbd: failed to assign a block name for image
create error: error 5: Input/output error
```

This usually happens because your OSDs can't find `cls_rbd.so`. They search for it in `osd_class_dir`, which may not be set correctly by default (http://tracker.newdream.net/issues/1722).

Most likely it's looking in `/usr/lib/rados-classes` instead of `/usr/lib64/rados-classes` - change `osd_class_dir` in your `ceph.conf` and restart the OSDs to fix it.

10.24 Peering

10.24.1 Concepts

Peering the process of bringing all of the OSDs that store a Placement Group (PG) into agreement about the state of all of the objects (and their metadata) in that PG. Note that agreeing on the state does not mean that they all have the latest contents.

Acting set the ordered list of OSDs who are (or were as of some epoch) responsible for a particular PG.

Up set the ordered list of OSDs responsible for a particular PG for a particular epoch according to CRUSH. Normally this is the same as the *acting set*, except when the *acting set* has been explicitly overridden via *PG temp* in the OSDMap.

PG temp a temporary placement group acting set used while backfilling the primary osd. Let say acting is [0,1,2] and we are active+clean. Something happens and acting is now [3,1,2]. osd 3 is empty and can't serve reads although it is the primary. osd.3 will see that and request a *PG temp* of [1,2,3] to the monitors using a MOSDPGTemp message so that osd.1 temporarily becomes the primary. It will select osd.3 as a backfill peer and continue to serve reads and writes while osd.3 is backfilled. When backfilling is complete, *PG temp* is discarded and the acting set changes back to [3,1,2] and osd.3 becomes the primary.

current interval* or *past interval a sequence of OSD map epochs during which the *acting set* and *up set* for particular PG do not change

primary the (by convention first) member of the *acting set*, who is responsible for coordination peering, and is the only OSD that will accept client initiated writes to objects in a placement group.

replica a non-primary OSD in the *acting set* for a placement group (and who has been recognized as such and *activated* by the primary).

stray an OSD who is not a member of the current *acting set*, but has not yet been told that it can delete its copies of a particular placement group.

recovery ensuring that copies of all of the objects in a PG are on all of the OSDs in the *acting set*. Once *peering* has been performed, the primary can start accepting write operations, and *recovery* can proceed in the background.

***PG info* basic metadata about the PG's creation epoch, the version** for the most recent write to the PG, *last epoch started*, *last epoch clean*, and the beginning of the *current interval*. Any inter-OSD communication about PGs includes the *PG info*, such that any OSD that knows a PG exists (or once existed) also has a lower bound on *last epoch clean* or *last epoch started*.

PG log a list of recent updates made to objects in a PG. Note that these logs can be truncated after all OSDs in the *acting set* have acknowledged up to a certain point.

missing set Each OSD notes update log entries and if they imply updates to the contents of an object, adds that object to a list of needed updates. This list is called the *missing set* for that <OSD,PG>.

Authoritative History a complete, and fully ordered set of operations that, if performed, would bring an OSD's copy of a Placement Group up to date.

epoch a (monotonically increasing) OSD map version number

last epoch start the last epoch at which all nodes in the *acting set* for a particular placement group agreed on an *authoritative history*. At this point, *peering* is deemed to have been successful.

up_thru before a primary can successfully complete the *peering* process, it must inform a monitor that is alive through the current OSD map epoch by having the monitor set its *up_thru* in the osd map. This helps peering ignore previous *acting sets* for which peering never completed after certain sequences of failures, such as the second interval below:

- *acting set* = [A,B]
- *acting set* = [A]
- *acting set* = [] very shortly after (e.g., simultaneous failure, but staggered detection)
- *acting set* = [B] (B restarts, A does not)

last epoch clean the last epoch at which all nodes in the *acting set* for a particular placement group were completely up to date (both PG logs and object contents). At this point, *recovery* is deemed to have been completed.

10.24.2 Description of the Peering Process

The *Golden Rule* is that no write operation to any PG is acknowledged to a client until it has been persisted by all members of the *acting set* for that PG. This means that if we can communicate with at least one member of each *acting set* since the last successful *peering*, someone will have a record of every (acknowledged) operation since the last successful *peering*. This means that it should be possible for the current primary to construct and disseminate a new *authoritative history*.

It is also important to appreciate the role of the OSD map (list of all known OSDs and their states, as well as some information about the placement groups) in the *peering* process:

> When OSDs go up or down (or get added or removed) this has the potential to affect the *active sets* of many placement groups.
>
> Before a primary successfully completes the *peering* process, the OSD map must reflect that the OSD was alive and well as of the first epoch in the *current interval*.
>
> Changes can only be made after successful *peering*.

Thus, a new primary can use the latest OSD map along with a recent history of past maps to generate a set of *past intervals* to determine which OSDs must be consulted before we can successfully *peer*. The set of past intervals is bounded by *last epoch started*, the most recent *past interval* for which we know *peering* completed. The process by with an OSD discovers a PG exists in the first place is by exchanging *PG info* messages, so the OSD always has some lower bound on *last epoch started*.

The high level process is for the current PG primary to:

1. get a recent OSD map (to identify the members of the all interesting *acting sets*, and confirm that we are still the primary).
2. generate a list of *past intervals* since *last epoch started*. Consider the subset of those for which *up_thru* was greater than the first interval epoch by the last interval epoch's OSD map; that is, the subset for which *peering* could have completed before the *acting set* changed to another set of OSDs.

   Successful *peering* will require that we be able to contact at least one OSD from each of *past interval*'s *acting set*.
3. ask every node in that list for its *PG info*, which includes the most recent write made to the PG, and a value for *last epoch started*. If we learn about a *last epoch started* that is newer than our own, we can prune older *past intervals* and reduce the peer OSDs we need to contact.

5. if anyone else has (in its PG log) operations that I do not have, instruct them to send me the missing log entries so that the primary's *PG log* is up to date (includes the newest write)..

5. for each member of the current *acting set*:

   (a) ask it for copies of all PG log entries since *last epoch start* so that I can verify that they agree with mine (or know what objects I will be telling it to delete).

   If the cluster failed before an operation was persisted by all members of the *acting set*, and the subsequent *peering* did not remember that operation, and a node that did remember that operation later rejoined, its logs would record a different (divergent) history than the *authoritative history* that was reconstructed in the *peering* after the failure.

   Since the *divergent* events were not recorded in other logs from that *acting set*, they were not acknowledged to the client, and there is no harm in discarding them (so that all OSDs agree on the *authoritative history*). But, we will have to instruct any OSD that stores data from a divergent update to delete the affected (and now deemed to be apocryphal) objects.

   (b) ask it for its *missing set* (object updates recorded in its PG log, but for which it does not have the new data). This is the list of objects that must be fully replicated before we can accept writes.

6. at this point, the primary's PG log contains an *authoritative history* of the placement group, and the OSD now has sufficient information to bring any other OSD in the *acting set* up to date.

7. if the primary's *up_thru* value in the current OSD map is not greater than or equal to the first epoch in the *current interval*, send a request to the monitor to update it, and wait until receive an updated OSD map that reflects the change.

8. for each member of the current *acting set*:

   (a) send them log updates to bring their PG logs into agreement with my own (*authoritative history*) ... which may involve deciding to delete divergent objects.

   (b) await acknowledgment that they have persisted the PG log entries.

9. at this point all OSDs in the *acting set* agree on all of the meta-data, and would (in any future *peering*) return identical accounts of all updates.

   (a) start accepting client write operations (because we have unanimous agreement on the state of the objects into which those updates are being accepted). Note, however, that if a client tries to write to an object it will be promoted to the front of the recovery queue, and the write willy be applied after it is fully replicated to the current *acting set*.

   (b) update the *last epoch started* value in our local *PG info*, and instruct other *active set* OSDs to do the same.

   (c) start pulling object data updates that other OSDs have, but I do not. We may need to query OSDs from additional *past intervals* prior to *last epoch started* (the last time *peering* completed) and following *last epoch clean* (the last epoch that recovery completed) in order to find copies of all objects.

   (d) start pushing object data updates to other OSDs that do not yet have them.

   We push these updates from the primary (rather than having the replicas pull them) because this allows the primary to ensure that a replica has the current contents before sending it an update write. It also makes it possible for a single read (from the primary) to be used to write the data to multiple replicas. If each replica did its own pulls, the data might have to be read multiple times.

10. once all replicas store the all copies of all objects (that existed prior to the start of this epoch) we can update *last epoch clean* in the *PG info*, and we can dismiss all of the *stray* replicas, allowing them to delete their copies of objects for which they are no longer in the *acting set*.

    We could not dismiss the *strays* prior to this because it was possible that one of those *strays* might hold the sole surviving copy of an old object (all of whose copies disappeared before they could be replicated on members of the current *acting set*).

10.24.3 State Model

10.25 Perf counters

The perf counters provide generic internal infrastructure for gauges and counters. The counted values can be both integer and float. There is also an "average" type (normally float) that combines a sum and num counter which can be divided to provide an average.

The intention is that this data will be collected and aggregated by a tool like `collectd` or `statsd` and fed into a tool like `graphite` for graphing and analysis.

10.25.1 Access

The perf counter data is accessed via the admin socket. For example:

```
ceph --admin-daemon /var/run/ceph/ceph-osd.0.asok perf schema
ceph --admin-daemon /var/run/ceph/ceph-osd.0.asok perf dump
```

10.25.2 Collections

The values are grouped into named collections, normally representing a subsystem or an instance of a subsystem. For example, the internal `throttle` mechanism reports statistics on how it is throttling, and each instance is named something like:

```
throttle-msgr_dispatch_throttler-hbserver
throttle-msgr_dispatch_throttler-client
throttle-filestore_bytes
...
```

10.25.3 Schema

The `perf schema` command dumps a json description of which values are available, and what their type is. Each named value as a `type` bitfield, with the following bits defined.

bit	meaning
1	floating point value
2	unsigned 64-bit integer value
4	average (sum + count pair)
8	counter (vs gauge)

Every value with have either bit 1 or 2 set to indicate the type (float or integer). If bit 8 is set (counter), the reader may want to subtract off the previously read value to get the delta during the previous interval.

If bit 4 is set (average), there will be two values to read, a sum and a count. If it is a counter, the average for the previous interval would be sum delta (since the previous read) divided by the count delta. Alternatively, dividing the values outright would provide the lifetime average value. Normally these are used to measure latencies (number of requests and a sum of request latencies), and the average for the previous interval is what is interesting.

Here is an example of the schema output:

```
{
  "throttle-msgr_dispatch_throttler-hbserver" : {
     "get_or_fail_fail" : {
        "type" : 10
     },
     "get_sum" : {
        "type" : 10
     },
     "max" : {
        "type" : 10
     },
     "put" : {
        "type" : 10
     },
     "val" : {
        "type" : 10
     },
     "take" : {
        "type" : 10
     },
     "get_or_fail_success" : {
        "type" : 10
     },
     "wait" : {
        "type" : 5
     },
     "get" : {
        "type" : 10
     },
     "take_sum" : {
        "type" : 10
     },
     "put_sum" : {
        "type" : 10
     }
  },
  "throttle-msgr_dispatch_throttler-client" : {
     "get_or_fail_fail" : {
        "type" : 10
     },
     "get_sum" : {
        "type" : 10
     },
     "max" : {
        "type" : 10
     },
     "put" : {
        "type" : 10
     },
     "val" : {
        "type" : 10
     },
      "take" : {
         "type" : 10
      },
      "get_or_fail_success" : {
         "type" : 10
      },
      "wait" : {
         "type" : 5
      },
      "get" : {
         "type" : 10
      },
      "take_sum" : {
         "type" : 10
      },
      "put_sum" : {
         "type" : 10
      }
   }
}
```

10.25.4 Dump

The actual dump is similar to the schema, except that average values are grouped. For example:

```
{
   "throttle-msgr_dispatch_throttler-hbserver" : {
      "get_or_fail_fail" : 0,
      "get_sum" : 0,
      "max" : 104857600,
      "put" : 0,
      "val" : 0,
      "take" : 0,
      "get_or_fail_success" : 0,
      "wait" : {
         "avgcount" : 0,
         "sum" : 0
      },
      "get" : 0,
      "take_sum" : 0,
      "put_sum" : 0
   },
   "throttle-msgr_dispatch_throttler-client" : {
      "get_or_fail_fail" : 0,
      "get_sum" : 82760,
      "max" : 104857600,
      "put" : 2637,
      "val" : 0,
      "take" : 0,
      "get_or_fail_success" : 0,
      "wait" : {
         "avgcount" : 0,
         "sum" : 0
      },
      "get" : 2637,
      "take_sum" : 0,
      "put_sum" : 82760
```

```
  }
}
```

10.26 PG (Placement Group) notes

Miscellaneous copy-pastes from emails, when this gets cleaned up it should move out of /dev.

10.26.1 Overview

PG = "placement group". When placing data in the cluster, objects are mapped into PGs, and those PGs are mapped onto OSDs. We use the indirection so that we can group objects, which reduces the amount of per-object metadata we need to keep track of and processes we need to run (it would be prohibitively expensive to track eg the placement history on a per-object basis). Increasing the number of PGs can reduce the variance in per-OSD load across your cluster, but each PG requires a bit more CPU and memory on the OSDs that are storing it. We try and ballpark it at 100 PGs/OSD, although it can vary widely without ill effects depending on your cluster. You hit a bug in how we calculate the initial PG number from a cluster description.

There are a couple of different categories of PGs; the 6 that exist (in the original emailer's `ceph -s` output) are "local" PGs which are tied to a specific OSD. However, those aren't actually used in a standard Ceph configuration.

10.26.2 Mapping algorithm (simplified)

> How does the Object->PG mapping look like, do you map more than one object on
> one PG, or do you sometimes map an object to more than one PG? How about the
> mapping of PGs to OSDs, does one PG belong to exactly one OSD?
>
> Does one PG represent a fixed amount of storage space?

Many objects map to one PG.

Each object maps to exactly one PG.

One PG maps to a single list of OSDs, where the first one in the list is the primary and the rest are replicas.

Many PGs can map to one OSD.

A PG represents nothing but a grouping of objects; you configure the number of PGs you want (see http://ceph.com/wiki/Changing_the_number_of_PGs), number of OSDs * 100 is a good starting point, and all of your stored objects are pseudo-randomly evenly distributed to the PGs. So a PG explicitly does NOT represent a fixed amount of storage; it represents 1/pg_num 'th of the storage you happen to have on your OSDs.

Ignoring the finer points of CRUSH and custom placement, it goes something like this in pseudocode:

```
locator = object_name
obj_hash = hash(locator)
pg = obj_hash % num_pg
OSDs_for_pg = crush(pg)  # returns a list of OSDs
primary = osds_for_pg[0]
replicas = osds_for_pg[1:]
```

If you want to understand the crush() part in the above, imagine a perfectly spherical datacenter in a vacuum ;) that is, if all OSDs have weight 1.0, and there is no topology to the data center (all OSDs are on the top level), and you use defaults, etc, it simplifies to consistent hashing; you can think of it as:

```
def crush(pg):
   all_osds = ['osd.0', 'osd.1', 'osd.2', ...]
   result = []
   # size is the number of copies; primary+replicas
   while len(result) < size:
       r = hash(pg)
       chosen = all_osds[ r % len(all_osds) ]
       if chosen in result:
           # OSD can be picked only once
           continue
       result.append(chosen)
   return result
```

10.26.3 User-visible PG States

Todo

diagram of states and how they can overlap

creating the PG is still being created

active requests to the PG will be processed

clean all objects in the PG are replicated the correct number of times

down a replica with necessary data is down, so the pg is offline

replay the PG is waiting for clients to replay operations after an OSD crashed

splitting the PG is being split into multiple PGs (not functional as of 2012-02)

scrubbing the PG is being checked for inconsistencies

degraded some objects in the PG are not replicated enough times yet

inconsistent replicas of the PG are not consistent (e.g. objects are the wrong size, objects are missing from one replica *after* recovery finished, etc.)

peering the PG is undergoing the Peering process

repair the PG is being checked and any inconsistencies found will be repaired (if possible)

recovering objects are being migrated/synchronized with replicas

recovery_wait the PG is waiting for the local/remote recovery reservations

backfill a special case of recovery, in which the entire contents of the PG are scanned and synchronized, instead of inferring what needs to be transferred from the PG logs of recent operations

backfill-wait the PG is waiting in line to start backfill

backfill_toofull backfill reservation rejected, OSD too full

incomplete a pg is missing a necessary period of history from its log. If you see this state, report a bug, and try to start any failed OSDs that may contain the needed information.

stale the PG is in an unknown state - the monitors have not received an update for it since the PG mapping changed.

remapped the PG is temporarily mapped to a different set of OSDs from what CRUSH specified

10.27 Developer Guide (Quick)

This guide will describe how to build and test Ceph for development.

10.27.1 Development

The `run-make-check.sh` script will install Ceph dependencies, compile everything in debug mode and run a number of tests to verify the result behaves as expected.

```
$ ./run-make-check.sh
```

10.27.2 Running a development deployment

Ceph contains a script called `vstart.sh` which allows developers to quickly test their code using a simple deployment on your development system. Once the build finishes successfully, start the ceph deployment using the following command:

```
$ cd src
$ ./vstart.sh -d -n -x
```

You can also configure `vstart.sh` to use only one monitor and one metadata server by using the following:

```
$ MON=1 MDS=1 ./vstart.sh -d -n -x
```

The system creates three pools on startup: *cephfs_data*, *cephfs_metadata*, and *rbd*. Let's get some stats on the current pools:

```
$ ./ceph osd pool stats
*** DEVELOPER MODE: setting PATH, PYTHONPATH and LD_LIBRARY_PATH ***
pool rbd id 0
  nothing is going on

pool cephfs_data id 1
  nothing is going on

pool cephfs_metadata id 2
  nothing is going on

$ ./ceph osd pool stats cephfs_data
*** DEVELOPER MODE: setting PATH, PYTHONPATH and LD_LIBRARY_PATH ***
pool cephfs_data id 1
  nothing is going on

$ ./rados df
pool name       category                 KB      objects       clones     degraded      unfound
rbd             -                         0            0            0            0           0
cephfs_data     -                         0            0            0            0           0
cephfs_metadata -                         2           20            0           40           0
  total used        12771536           20
  total avail     3697045460
  total space     3709816996
```

Make a pool and run some benchmarks against it:

```
$ ./rados mkpool mypool
$ ./rados -p mypool bench 10 write -b 123
```

Place a file into the new pool:

```
$ ./rados -p mypool put objectone <somefile>
$ ./rados -p mypool put objecttwo <anotherfile>
```

List the objects in the pool:

```
$ ./rados -p mypool ls
```

Once you are done, type the following to stop the development ceph deployment:

```
$ ./stop.sh
```

10.27.3 Running a RadosGW development environment

Add the `-r` to vstart.sh to enable the RadosGW

```
$ cd src
$ ./vstart.sh -d -n -x -r
```

You can now use the swift python client to communicate with the RadosGW.

```
$ swift -A http://localhost:8000/auth -U tester:testing -K asdf list
$ swift -A http://localhost:8000/auth -U tester:testing -K asdf upload mycontainer ceph
$ swift -A http://localhost:8000/auth -U tester:testing -K asdf list
```

10.27.4 Run unit tests

The tests are located in *src/tests*. To run them type:

```
$ make check
```

10.28 RBD Incremental Backup

This is a simple streaming file format for representing a diff between two snapshots (or a snapshot and the head) of an RBD image.

10.28.1 Header

"rbd diff v1\n"

10.28.2 Metadata records

Every record has a one byte "tag" that identifies the record type, followed by some other data.

Metadata records come in the first part of the image. Order is not important, as long as all the metadata records come before the data records.

From snap

- u8: 'f'
- le32: snap name length
- snap name

To snap

- u8: 't'
- le32: snap name length
- snap name

Size

- u8: 's'
- u64: (ending) image size

10.28.3 Data Records

These records come in the second part of the sequence.

Updated data

- u8: 'w'
- le64: offset
- le64: length
- length bytes of actual data

Zero data

- u8: 'z'
- le64: offset
- le64: length

10.28.4 Final Record

End

- u8: 'e'

10.29 RBD Layering

RBD layering refers to the creation of copy-on-write clones of block devices. This allows for fast image creation, for example to clone a golden master image of a virtual machine into a new instance. To simplify the semantics, you can only create a clone of a snapshot - snapshots are always read-only, so the rest of the image is unaffected, and there's no possibility of writing to them accidentally.

From a user's perspective, a clone is just like any other rbd image. You can take snapshots of them, read/write them, resize them, etc. There are no restrictions on clones from a user's viewpoint.

Note: the terms *child* and *parent* below mean an rbd image created by cloning, and the rbd image snapshot a child was cloned from.

10.29.1 Command line interface

Before cloning a snapshot, you must mark it as protected, to prevent it from being deleted while child images refer to it:

```
$ rbd snap protect pool/image@snap
```

Then you can perform the clone:

```
$ rbd clone [--parent] pool/parent@snap [--image] pool2/child1
```

You can create a clone with different object sizes from the parent:

```
$ rbd clone --order 25 pool/parent@snap pool2/child2
```

To delete the parent, you must first mark it unprotected, which checks that there are no children left:

```
$ rbd snap unprotect pool/image@snap
Cannot unprotect: Still in use by pool2/image2
$ rbd children pool/image@snap
pool2/child1
pool2/child2
$ rbd flatten pool2/child1
$ rbd rm pool2/child2
$ rbd snap rm pool/image@snap
Cannot remove a protected snapshot: pool/image@snap
$ rbd snap unprotect pool/image@snap
```

Then the snapshot can be deleted like normal:

```
$ rbd snap rm pool/image@snap
```

10.29.2 Implementation

Data Flow

In the initial implementation, called 'trivial layering', there will be no tracking of which objects exist in a clone. A read that hits a non-existent object will attempt to read from the parent snapshot, and this will continue recursively until an object exists or an image with no parent is found. This is done through the normal read path from the parent, so differing object sizes between parents and children do not matter.

Before a write to an object is performed, the object is checked for existence. If it doesn't exist, a copy-up operation is performed, which means reading the relevant range of data from the parent snapshot and writing it (plus the original write) to the child image. To prevent races with multiple writes trying to copy-up the same object, this copy-up operation will include an atomic create. If the atomic create fails, the original write is done instead. This copy-up operation is implemented as a class method so that extra metadata can be stored by it in the future. In trivial layering, the copy-up operation copies the entire range needed to the child object (that is, the full size of the child object). A future optimization could make this copy-up more fine-grained.

Another future optimization could be storing a bitmap of which objects actually exist in a child. This would obviate the check for existence before each write, and let reads go directly to the parent if needed.

These optimizations are discussed in:

http://marc.info/?l=ceph-devel&m=129867273303846

Parent/Child relationships

Children store a reference to their parent in their header, as a tuple of (pool id, image id, snapshot id). This is enough information to open the parent and read from it.

In addition to knowing which parent a given image has, we want to be able to tell if a protected snapshot still has children. This is accomplished with a new per-pool object, *rbd_children*, which maps (parent pool id, parent image id, parent snapshot id) to a list of child image ids. This is stored in the same pool as the child image because the client creating a clone already has read/write access to everything in this pool, but may not have write access to the parent's pool. This lets a client with read-only access to one pool clone a snapshot from that pool into a pool they have full access to. It increases the cost of unprotecting an image, since this needs to check for children in every pool, but this is a rare operation. It would likely only be done before removing old images, which is already much more expensive because it involves deleting every data object in the image.

Protection

Internally, protection_state is a field in the header object that can be in three states. "protected", "unprotected", and "unprotecting". The first two are set as the result of "rbd protect/unprotect". The "unprotecting" state is set while the "rbd unprotect" command checks for any child images. Only snapshots in the "protected" state may be cloned, so the "unprotected" state prevents a race like:

1. A: walk through all pools, look for clones, find none
2. B: create a clone
3. A: unprotect parent
4. A: rbd snap rm pool/parent@snap

Resizing

Resizing an rbd image is like truncating a sparse file. New space is treated as zeroes, and shrinking an rbd image deletes the contents beyond the old bounds. This means that if you have a 10G image full of data, and you resize it down to 5G and then up to 10G again, the last 5G is treated as zeroes (and any objects that held that data were removed when the image was shrunk).

Layering complicates this because the absence of an object no longer implies it should be treated as zeroes - if the object is part of a clone, it may mean that some data needs to be read from the parent.

To preserve the resizing behavior for clones, we need to keep track of which objects could be stored in the parent. We can track this as the amount of overlap the child has with the parent, since resizing only changes the end of an image. When a child is created, its overlap is the size of the parent snapshot. On each subsequent resize, the overlap is *min(overlap, new_size)*. That is, shrinking the image may shrinks the overlap, but increasing the image's size does not change the overlap.

Objects that do not exist past the overlap are treated as zeroes. Objects that do not exist before that point fall back to reading from the parent.

Since this overlap changes over time, we store it as part of the metadata for a snapshot as well.

Renaming

Currently the rbd header object (that stores all the metadata about an image) is named after the name of the image. This makes renaming disrupt clients who have the image open (such as children reading from a parent). To avoid this, we can name the header object by the id of the image, which does not change. That is, the name of the header object could be *rbd_header.$id*, where $id is a unique id for the image in the pool.

When a client opens an image, all it knows is the name. There is already a per-pool *rbd_directory* object that maps image names to ids, but if we relied on it to get the id, we could not open any images in that pool if that single object was unavailable. To avoid this dependency, we can store the id of an image in an object called *rbd_id.$image_name*, where $image_name is the name of the image. The per-pool *rbd_directory* object is still useful for listing all images in a pool, however.

10.29.3 Header changes

The header needs a few new fields:

- int64_t parent_pool_id
- string parent_image_id
- uint64_t parent_snap_id
- uint64_t overlap (how much of the image may be referring to the parent)

These are stored in a "parent" key, which is only present if the image has a parent.

cls_rbd

Some new methods are needed:

```
/***************** methods on the rbd header *********************/
/**
 * Sets the parent and overlap keys.
 * Fails if any of these keys exist, since the image already
 * had a parent.
 */
set_parent(uint64_t pool_id, string image_id, uint64_t snap_id)

/**
 * returns the parent pool id, image id, snap id, and overlap, or -ENOENT
 * if parent_pool_id does not exist or is -1
 */
get_parent(uint64_t snapid)

/**
 * Removes the parent key
 */
remove_parent() // after all parent data is copied to the child

/*************** methods on the rbd_children object *****************/
```

```
add_child(uint64_t parent_pool_id, string parent_image_id,
          uint64_t parent_snap_id, string image_id);
remove_child(uint64_t parent_pool_id, string parent_image_id,
             uint64_t parent_snap_id, string image_id);
/**
 * List ids of a given parent
 */
get_children(uint64_t parent_pool_id, string parent_image_id,
             uint64_t parent_snap_id, uint64_t max_return,
             string start);
/**
 * list parent
 */
get_parents(uint64_t max_return, uint64_t start_pool_id,
            string start_image_id, string start_snap_id);


/************ methods on the rbd_id.$image_name object **************/

set_id(string id)
get_id()

/************** methods on the rbd_directory object *****************/

dir_get_id(string name);
dir_get_name(string id);
dir_list(string start_after, uint64_t max_return);
dir_add_image(string name, string id);
dir_remove_image(string name, string id);
dir_rename_image(string src, string dest, string id);
```

Two existing methods will change if the image supports layering:

```
snapshot_add - stores current overlap and has_parent with
               other snapshot metadata (images that don't have
               layering enabled aren't affected)

set_size     - will adjust the parent overlap down as needed.
```

librbd

Opening a child image opens its parent (and this will continue recursively as needed). This means that an ImageCtx will contain a pointer to the parent image context. Differing object sizes won’t matter, since reading from the parent will go through the parent image context.

Discard will need to change for layered images so that it only truncates objects, and does not remove them. If we removed objects, we could not tell if we needed to read them from the parent.

A new clone method will be added, which takes the same arguments as create except size (size of the parent image is used).

Instead of expanding the rbd_info struct, we will break the metadata retrieval into several API calls. Right now, the only users of rbd_stat() other than ‘rbd info’ only use it to retrieve image size.

10.30 Ceph Release Process

10.30.1 1. Build environment

There are multiple build environments, debian based packages are built via pbuilder for multiple distributions. The build hosts are listed in the `deb_hosts` file, and the list of distributions are in `deb_dist`. All distributions are build on each of the build hosts. Currently there is 1 64 bit and 1 32 bit build host.

The RPM based packages are built natively, so one distribution per build host. The list of hosts is found in `rpm_hosts`.

Prior to building, it's necessary to update the pbuilder seed tarballs:

```
./update_all_pbuilders.sh
```

10.30.2 2. Setup keyring for signing packages

```
export GNUPGHOME=<path to keyring dir>

# verify it's accessible
gpg --list-keys
```

The release key should be present:

```
pub   4096R/17ED316D 2012-05-20
uid   Ceph Release Key <sage@newdream.net>
```

10.30.3 3. Set up build area

Clone the ceph and ceph-build source trees:

```
git clone http://github.com/ceph/ceph.git
git clone http://github.com/ceph/ceph-build.git
```

In the ceph source directory, checkout next branch (for point releases use the {codename} branch):

```
git checkout next
```

Checkout the submodules:

```
git submodule update --force --init --recursive
```

10.30.4 4. Update Build version numbers

Substitute the ceph release number where indicated below by the string `0.xx`.

Edit configure.ac and update the version number. Example diff:

```
-AC_INIT([ceph], [0.54], [ceph-devel@vger.kernel.org])
+AC_INIT([ceph], [0.55], [ceph-devel@vger.kernel.org])
```

Update the version number in the debian change log:

```
DEBEMAIL user@host dch -v 0.xx-1
```

Commit the changes:

```
git commit -a
```

Tag the release:

```
../ceph-build/tag-release v0.xx
```

10.30.5 5. Create Makefiles

The actual configure options used to build packages are in the `ceph.spec.in` and `debian/rules` files. At this point we just need to create a Makefile.:

```
./do_autogen.sh
```

10.30.6 6. Run the release scripts

This creates tarballs and copies them, with other needed files to the build hosts listed in deb_hosts and rpm_hosts, runs a local build script, then rsyncs the results back to the specified release directory.:

```
../ceph-build/do_release.sh /tmp/release
```

10.30.7 7. Create RPM Repo

Copy the rpms to the destination repo:

```
mkdir /tmp/rpm-repo
../ceph-build/push_to_rpm_repo.sh /tmp/release /tmp/rpm-repo 0.xx
```

Next add any additional rpms to the repo that are needed such as leveldb and and ceph-deploy. See RPM Backports section

Finally, sign the rpms and build the repo indexes:

```
../ceph-build/sign_and_index_rpm_repo.sh /tmp/release /tmp/rpm-repo 0.xx
```

10.30.8 8. Create Debian repo

The key-id used below is the id of the ceph release key from step 2:

```
mkdir /tmp/debian-repo
../ceph-build/gen_reprepro_conf.sh /tmp/debian-repo key-id
../ceph-build/push_to_deb_repo.sh /tmp/release /tmp/debian-repo 0.xx main
```

Next add any addition debian packages that are needed such as leveldb and ceph-deploy. See the Debian Backports section below.

Debian packages are signed when added to the repo, so no further action is needed.

10.30.9 9. Push repos to ceph.org

For a development release:

```
rcp ceph-0.xx.tar.bz2 ceph-0.xx.tar.gz \
     ceph_site@ceph.com:ceph.com/downloads/.
rsync -av /tmp/rpm-repo/0.xx/ ceph_site@ceph.com:ceph.com/rpm-testing
rsync -av /tmp/debian-repo/ ceph_site@ceph.com:ceph.com/debian-testing
```

For a stable release, replace {CODENAME} with the release codename (e.g., `argonaut` or `bobtail`):

```
rcp ceph-0.xx.tar.bz2 \
    ceph_site@ceph.com:ceph.com/downloads/ceph-0.xx.tar.bz2
rcp ceph-0.xx.tar.gz  \
    ceph_site@ceph.com:ceph.com/downloads/ceph-0.xx.tar.gz
rsync -av /tmp/rpm-repo/0.xx/ ceph_site@ceph.com:ceph.com/rpm-{CODENAME}
rsync -auv /tmp/debian-repo/ ceph_site@ceph.com:ceph.com/debian-{CODENAME}
```

10.30.10 10. Update Git

Point release

For point releases just push the version number update to the branch and the new tag:

```
git push origin {codename}
git push origin v0.xx
```

Development and Stable releases

For a development release, update tags for `ceph.git`:

```
git push origin v0.xx
git push origin HEAD:last
git checkout master
git merge next
git push origin master
git push origin HEAD:next
```

Similarly, for a development release, for both `teuthology.git` and `ceph-qa-suite.git`:

```
git checkout master
git reset --hard origin/master
git branch -f last origin/next
git push -f origin last
git push -f origin master:next
```

10.31 Notes on Ceph repositories

10.31.1 Special branches

- `master`: current tip (integration branch)
- `next`: pending release (feature frozen, bugfixes only)
- `last`: last/previous release
- `dumpling`, `cuttlefish`, `bobtail`, `argonaut`, etc.: stable release branches
- `dumpling-next`: backports for stable release, pending testing

10.31.2 Rules

The source repos are all on github.

- Any branch pushed to ceph.git will kick off builds that will either run unit tests or generate packages for gitbuilder.ceph.com. Try not to generate unnecessary load. For private, unreviewed work, only push to branches named `wip-*`. This avoids colliding with any special branches.
- Nothing should every reach a special branch unless it has been reviewed.
- Preferred means of review is via github pull requests to capture any review discussion.
- For multi-patch series, the pull request can be merged via github, and a Reviewed-by: ... line added to the merge commit.
- For single- (or few-) patch merges, it is preferable to add the Reviewed-by: directly to the commit so that it is also visible when the patch is cherry-picked for backports.
- All backports should use `git cherry-pick -x` to capture which commit they are cherry-picking from.

10.32 Sepia community test lab

The Ceph community maintains a test lab that is open to active contributors to the Ceph project. Please see the Sepia repository for more information.

10.33 Session Authentication for the Cephx Protocol

Peter Reiher 7/30/12

The original Cephx protocol authenticated the client to the authenticator and set up a session key used to authenticate the client to the server it needs to talk to. It did not, however, authenticate the ongoing messages between the client and server. Based on the fact that they share a secret key, these ongoing session messages can be easily authenticated by using the key to sign the messages.

This document describes changes to the code that allow such ongoing session authentication. The changes allow for future changes that permit other authentication protocols (and the existing null NONE and UNKNOWN protocols) to handle signatures, but the only protocol that actually does signatures, at the time of the writing, is the Cephx protocol.

10.33.1 Introduction

This code comes into play after the Cephx protocol has completed. At this point, the client and server share a secret key. This key will be used for authentication. For other protocols, there may or may not be such a key in place, and perhaps the actual procedures used to perform signing will be different, so the code is written to be general.

The "session" here is represented by an established pipe. For such pipes, there should be a `session\_security` structure attached to the pipe. Whenever a message is to be sent on the pipe, code that handles the signature for this kind of session security will be called. On the other end of the pipe, code that checks this kind of session security's message signatures will be called. Messages that fail the signature check will not be processed further. That implies that the sender had better be in agreement with the receiver on the session security being used, since otherwise messages will be uniformly dropped between them.

The code is also prepared to handle encryption and decryption of session messages, which would add secrecy to the integrity provided by the signatures. No protocol currently implemented encrypts the ongoing session messages, though.

For this functionality to work, several steps are required. First, the sender and receiver must have a successful run of the cephx protocol to establish a shared key. They must store that key somewhere that the pipe can get at later, to permit messages to be signed with it. Sent messages must be signed, and received messages must have their signatures checked.

The signature could be computed in a variety of ways, but currently its size is limited to 64 bits. A message's signature is placed in its footer, in a field called `sig`.

The signature code in Cephx can be turned on and off at runtime, using a Ceph boolean option called `cephx\_sign\_messages`. It is currently set to false, by default, so no messages will be signed. It must be changed to true to cause signatures to be calculated and checked.

10.33.2 Storing the Key

The key is needed to create signatures on the sending end and check signatures on the receiving end. In the future, if asymmetric crypto is an option, it's possible that two keys (a private one for this end of the pipe and a public one for the other end) would need to be stored. At this time, messages going in both directions will be signed with the same key, so only that key needs to be saved.

The key is saved when the pipe is established. On the client side, this happens in `connect()`, which is located in `msg/Pipe.cc`. The key is obtained from a run of the Cephx protocol, which results in a successfully checked authorizer structure. If there is such an authorizer available, the code calls `get\_auth\_session\_handler()` to create a new authentication session handler and stores it in the pipe data structure. On the server side, a similar thing is done in `accept()` after the authorizer provided by the client has been verified.

Once these things are done on either end of the connection, session authentication can start.

These routines (`connect()` and `accept()`) are also used to handle situations where a new session is being set up. At this stage, no authorizer has been created yet, so there's no key. Special cases in the code that calls the signature code skip these calls when the `CEPH\_AUTH\_UNKNOWN` protocol is in use. This protocol label is on the pre-authorizer messages in a session, indicating that negotiation on an authentication protocol is ongoing and thus signature is not possible. There will be a reliable authentication operation later in this session before anything sensitive should be passed, so this is not a security problem.

10.33.3 Signing Messages

Messages are signed in the `write\_message` call located in `msg/Pipe.cc`. The actual signature process is to encrypt the CRCs for the message using the shared key. Thus, we must defer signing until all CRCs have been computed. The header CRC is computed last, so we call `sign\_message()` as soon as we've calculated that CRC.

`sign\_message()` is a virtual function defined in `auth/AuthSessionHandler.h`. Thus, a specific version of it must be written for each authentication protocol supported. Currently, only UNKNOWN, NONE and CEPHX are supported. So there is a separate version of `sign\_message()` in `auth/unknown/AuthUnknownSessionHandler.h`, `auth/none/AuthNoneSessionHandler.h` and `auth/cephx/CephxSessionHandler.cc`. The UNKNOWN and NONE versions simply return 0, indicating success.

The CEPHX version is more extensive. It is found in `auth/cephx/CephxSessionHandler.cc`. The first thing done is to determine if the run time option to handle signatures (see above) is on. If not, the Cephx version of `sign\_message()` simply returns success without actually calculating a signature or inserting it into the message.

If the run time option is enabled, `sign\_message()` copies all of the message's CRCs (one from the header and three from the footer) into a buffer. It calls `encode\_encrypt()` on the buffer, using the key obtained from the pipe's `session\_security` structure. 64 bits of the encrypted result are put into the message footer's signature field and a footer flag is set to indicate that the message was signed. (This flag is a sanity check. It is not regarded as definitive evidence that the message was signed. The presence of a `session\_security` structure at the receiving end requires a signature regardless of the value of this flag.) If this all goes well, `sign\_message()` returns 0. If there is a problem anywhere along the line and no signature was computed, it returns `SESSION\_SIGNATURE\_FAILURE`.

10.33.4 Checking Signatures

The signature is checked by a routine called `check\_message\_signature()`. This is also a virtual function, defined in `auth/AuthSessionHandler.h`. So again there are specific versions for supported authentication protocols, such as UNKNOWN, NONE and CEPHX. Again, the UNKNOWN and NONE versions are stored in `auth/unknown/AuthUnknownSessionHandler.h` and `auth/none/AuthNoneSessionHandler.h`, respectively, and again they simply return 0, indicating success.

The CEPHX version of `check\_message\_signature()` performs a real signature check. This routine (stored in `auth/cephx/CephxSessionHandler.cc`) exits with success if the run time option has disabled signatures. Otherwise, it takes the CRCs from the header and footer, encrypts the result, and compares it to the signature stored in the footer. Since an earlier routine has checked that the CRCs actually match the contents of the message, it is unnecessary to recompute the CRCs on the raw data in the message. The encryption is performed with the same `encode\_encrypt()` routine used on the sending end, using the key stored in the local `session\_security` data structure.

If everything checks out, the CEPHX routine returns 0, indicating succcess. If there is a problem, the routine returns `SESSION\_SIGNATURE\_FAILURE`.

10.33.5 Adding New Session Authentication Methods

For the purpose of session authentication only (not the basic authentication of client and server currently performed by the Cephx protocol), in addition to adding a new protocol, that protocol must have a `sign\_message()` routine and a `check\_message\_signature` routine. These routines will take a message pointer as a parameter and return 0 on success. The procedure used to sign and check will be specific to the new method, but probably there will be a `session\_security` structure attached to the pipe that contains a cryptographic key. This structure will be either an `AuthSessionHandler` (found in `auth/AuthSessionHandler.h`) or a structure derived from that type.

10.33.6 Adding Encryption to Sessions

The existing code is partially, but not fully, set up to allow sessions to have their packets encrypted. Part of adding encryption would be similar to adding a new authentication method. But one would also need to add calls to the encryption and decryption routines in `write\_message()` and `read\_message()`. These calls would probably go near where the current calls for authentication are made. You should consider whether you want to replace the existing calls with something more general that does whatever the chosen form of session security requires, rather than explicitly saying `sign` or `encrypt`.

10.33.7 Session Security Statistics

The existing Cephx authentication code keeps statistics on how many messages were signed, how many message signature were checked, and how many checks succeeded and failed. It is prepared to keep similar statistics on encryption and decryption. These statistics can be accessed through the call `printAuthSessionHandlerStats` in `auth/AuthSessionHandler.cc`.

If new authentication or encryption methods are added, they should include code that keeps these statistics.

10.34 Public OSD Version

We maintain two versions on disk: an eversion_t pg_log.head and a version_t info.user_version. Each object is tagged with both the pg version and user_version it was last modified with. The PG version is modified by manipulating OpContext::at_version and then persisting it to the pg log as transactions, and is incremented in all the places it used to be. The user_version is modified by manipulating the new OpContext::user_at_version and is also persisted via the pg log transactions. user_at_version is modified only in ReplicatedPG::prepare_transaction when the op was a "user modify" (a non-watch write), and the durable user_version is updated according to the following rules: 1) set user_at_version to the maximum of ctx->new_obs.oi.user_version+1 and info.last_user_version+1. 2) set user_at_version to the maximum of itself and ctx->at_version.version. 3) ctx->new_obs.oi.user_version = ctx->user_at_version (to change the object's user_version)

This set of update semantics mean that for traditional pools the user_version will be equal to the past reassert_version, while for caching pools the object and PG user-version will be able to cross pools without making a total mess of things. In order to support old clients, we keep the old reassert_version but rename it to "bad_replay_version"; we fill it in as before: for writes it is set to the at_version (and is the proper replay version); for watches it is set to our user version; for ENOENT replies it is set to the replay version's epoch but the user_version's version. We also now fill in the version_t portion of the bad_replay_version on read ops as well as write ops, which should be fine for all old clients.

For new clients, we prevent them from reading bad_replay_version and add two proper members: user_version and replay_version; user_version is filled in on every operation (reads included) while replay_version is filled in for writes.

The objclass function get_current_version() now always returns the pg->info.last_user_version, which means it is guaranteed to contain the version of the last user update in the PG (including on reads!).

10.35 Wireshark Dissector

Wireshark has support for the Ceph protocol and it will be shipped in the 1.12.1 release.

10.35.1 Using

To use the Wireshark dissector you must build it from git, the process is outlined in great detail in the Building and Installing section of the Wireshark Users Guide.

10.35.2 Developing

The Ceph dissector lives in Wireshark git at `epan/dissectors/packet-ceph.c`. At the top of that file there are some comments explaining how to insert new functionality or to update the encoding of existing types.

Before you start hacking on Wireshark code you should look at the `doc/README.developer` and `doc/README.dissector` documents as they explain the basics of writing dissectors. After reading those two documents you should be prepared to work on the Ceph dissector. The Wireshark developers guide also contains a lot of useful information but it is less directed and is more useful as a reference then an introduction.

10.36 OSD developer documentation

Contents

10.36.1 Backfill Reservation

When a new osd joins a cluster, all pgs containing it must eventually backfill to it. If all of these backfills happen simultaneously, it would put excessive load on the osd. osd_max_backfills limits the number of outgoing or incoming backfills on a single node. The maximum number of outgoing backfills is osd_max_backfills. The maximum number of incoming backfills is osd_max_backfills. Therefore there can be a maximum of osd_max_backfills * 2 simultaneous backfills on one osd.

Each OSDService now has two AsyncReserver instances: one for backfills going from the osd (local_reserver) and one for backfills going to the osd (remote_reserver). An AsyncReserver (common/AsyncReserver.h) manages a queue by priority of waiting items and a set of current reservation holders. When a slot frees up, the AsyncReserver queues the Context* associated with the next item on the highest priority queue in the finisher provided to the constructor.

For a primary to initiate a backfill, it must first obtain a reservation from its own local_reserver. Then, it must obtain a reservation from the backfill target's remote_reserver via a MBackfillReserve message. This process is managed by substates of Active and ReplicaActive (see the substates of Active in PG.h). The reservations are dropped either on the Backfilled event, which is sent on the primary before calling recovery_complete and on the replica on receipt of the BackfillComplete progress message), or upon leaving Active or ReplicaActive.

It's important that we always grab the local reservation before the remote reservation in order to prevent a circular dependency.

We want to minimize the risk of data loss by prioritizing the order in which PGs are recovered. The highest priority is log based recovery (OSD_RECOVERY_PRIORITY_MAX) since this must always complete before backfill can start. The next priority is backfill of degraded PGs and is a function of the degradation. A backfill for a PG missing two replicas will have a priority higher than a backfill for a PG missing one replica. The lowest priority is backfill of non-degraded PGs.

10.36.2 Erasure Coded Placement Groups

Glossary

chunk when the encoding function is called, it returns chunks of the same size. Data chunks which can be concatenated to reconstruct the original object and coding chunks which can be used to rebuild a lost chunk.

chunk rank the index of a chunk when returned by the encoding function. The rank of the first chunk is 0, the rank of the second chunk is 1 etc.

stripe when an object is too large to be encoded with a single call, each set of chunks created by a call to the encoding function is called a stripe.

shard|strip an ordered sequence of chunks of the same rank from the same object. For a given placement group, each OSD contains shards of the same rank. When dealing with objects that are encoded with a single operation, *chunk* is sometime used instead of *shard* because the shard is made of a single chunk. The *chunks* in a *shard* are ordered according to the rank of the stripe they belong to.

K the number of data *chunks*, i.e. the number of *chunks* in which the original object is divided. For instance if $K = 2$ a 10KB object will be divided into K objects of 5KB each.

M the number of coding *chunks*, i.e. the number of additional *chunks* computed by the encoding functions. If there are 2 coding *chunks*, it means 2 OSDs can be out without losing data.

N the number of data *chunks* plus the number of coding *chunks*, i.e. $K+M$.

rate the proportion of the *chunks* that contains useful information, i.e. *K/N*. For instance, for *K* = 9 and *M* = 3 (i.e. *K+M* = *N* = 12) the rate is *K* = 9 / *N* = 12 = 0.75, i.e. 75% of the chunks contain useful information.

The definitions are illustrated as follows (PG stands for placement group):

```
                OSD 40                       OSD 33
      +-------------------------+ +-------------------------+
      |      shard 0 - PG 10    | |      shard 1 - PG 10    |
      |+------ object O -------+| |+------ object O -------+|
      ||+---------------------+|| ||+---------------------+||
stripe|||    chunk  0         ||| |||    chunk  1         ||| ...
  0   |||    stripe 0         ||| |||    stripe 0         |||
      ||+---------------------+|| ||+---------------------+||
      ||+---------------------+|| ||+---------------------+||
stripe|||    chunk  0         ||| |||    chunk  1         ||| ...
  1   |||    stripe 1         ||| |||    stripe 1         |||
      ||+---------------------+|| ||+---------------------+||
      ||+---------------------+|| ||+---------------------+||
stripe|||    chunk  0         ||| |||    chunk  1         ||| ...
  2   |||    stripe 2         ||| |||    stripe 2         |||
      ||+---------------------+|| ||+---------------------+||
      |+-----------------------+| |+-----------------------+|
      |         ...             | |         ...             |
      +-------------------------+ +-------------------------+
```

Table of content

Erasure Code developer notes

Introduction Each chapter of this document explains an aspect of the implementation of the erasure code within Ceph. It is mostly based on examples being explained to demonstrate how things work.

Reading and writing encoded chunks from and to OSDs An erasure coded pool stores each object as K+M chunks. It is divided into K data chunks and M coding chunks. The pool is configured to have a size of K+M so that each chunk is stored in an OSD in the acting set. The rank of the chunk is stored as an attribute of the object.

Let's say an erasure coded pool is created to use five OSDs (K+M = 5) and sustain the loss of two of them (M = 2).

When the object *NYAN* containing *ABCDEFGHI* is written to it, the erasure encoding function splits the content in three data chunks, simply by dividing the content in three : the first contains *ABC*, the second *DEF* and the last *GHI*. The content will be padded if the content length is not a multiple of K. The function also creates two coding chunks : the fourth with *YXY* and the fifth with *GQC*. Each chunk is stored in an OSD in the acting set. The chunks are stored in objects that have the same name (*NYAN*) but reside on different OSDs. The order in which the chunks were created must be preserved and is stored as an attribute of the object (shard_t), in addition to its name. Chunk *1* contains *ABC* and is stored on *OSD5* while chunk *4* contains *YXY* and is stored on *OSD3*.

```
                          +-------------------+
                     name |        NYAN       |
                          +-------------------+
                  content |      ABCDEFGHI    |
                          +--------+----------+
                                   |
                                   |
                                   v
                            +------+------+
            +---------------+ encode(3,2) +-----------+
            |               +--+--+---+---+           |
            |                  |  |   |               |
            |          +-------+  |   +-----+         |
            |          |          |         |         |
         +--v---+   +--v---+   +--v---+  +--v---+  +--v---+
   name  | NYAN |   | NYAN |   | NYAN |  | NYAN |  | NYAN |
         +------+   +------+   +------+  +------+  +------+
  shard  |  1   |   |  2   |   |  3   |  |  4   |  |  5   |
         +------+   +------+   +------+  +------+  +------+
content  | ABC  |   | DEF  |   | GHI  |  | YXY  |  | QGC  |
         +--+---+   +--+---+   +--+---+  +--+---+  +--+---+
            |          |          |         |         |
            |          |          |         |         |
            |          |       +--+---+     |         |
            |          |       | OSD1 |     |         |
            |          |       +------+     |         |
            |          |       +------+     |         |
            |          +------>| OSD2 |     |         |
            |                  +------+     |         |
            |                  +------+     |         |
            |                  | OSD3 |<----+         |
            |                  +------+               |
            |                  +------+               |
            |                  | OSD4 |<--------------+
            |                  +------+
            |                  +------+
            +----------------->| OSD5 |
                               +------+
```

When the object *NYAN* is read from the erasure coded pool, the decoding function reads three chunks : chunk *1* containing *ABC*, chunk *3* containing *GHI* and chunk *4* containing *YXY* and rebuild the original content of the object *ABCDEFGHI*. The decoding function is informed that the chunks *2* and *5* are missing (they are called *erasures*). The chunk *5* could not be read because the *OSD4* is *out*.

The decoding function could be called as soon as three chunks are read : *OSD2* was the slowest and its chunk does not need to be taken into account. This optimization is not implemented in Firefly.

```
                          +-------------------+
                     name |       NYAN        |
                          +-------------------+
                  content |      ABCDEFGHI    |
                          +---------+---------+
                                    ^
                                    |
                                    |
                            +------+------+
                            | decode(3,2) |
                            | erasures 2,5|
            +-------------->|             |
            |               +-------------+
            |                     ^   ^
            |                     |   +-----+
            |                     |         |
         +--+---+   +------+   +--+---+  +--+---+
   name  | NYAN |   | NYAN |   | NYAN |  | NYAN |
         +------+   +------+   +------+  +------+
  shard  |  1   |   |  2   |   |  3   |  |  4   |
         +------+   +------+   +------+  +------+
content  | ABC  |   | DEF  |   | GHI  |  | YXY  |
         +--+---+   +--+---+   +--+---+  +--+---+
            ^          .          ^         ^
            |    TOO   .          |         |
            |    SLOW  .       +--+---+     |
            |          ^       | OSD1 |     |
            |          |       +------+     |
            |          |       +------+     |
            |          +-------| OSD2 |     |
            |                  +------+     |
            |                  +------+     |
            |                  | OSD3 |-----+
            |                  +------+
            |                  +------+
            |                  | OSD4 | OUT
            |                  +------+
            |                  +------+
            +------------------| OSD5 |
                               +------+
```

Erasure code library Using Reed-Solomon, with parameters K+M, object O is encoded by dividing it into chunks O1, O2, ... OM and computing coding chunks P1, P2, ... PK. Any K chunks out of the available K+M chunks can be used to obtain the original object. If data chunk O2 or coding chunk P2 are lost, they can be repaired using any K chunks out of the K+M chunks. If more than M chunks are lost, it is not possible to recover the object.

Reading the original content of object O can be a simple concatenation of O1, O2, ... OM, because the plugins are using systematic codes. Otherwise the chunks must be given to the erasure code library *decode* method to retrieve the content of the object.

Performance depend on the parameters to the encoding functions and is also influenced by the packet sizes used when calling the encoding functions (for Cauchy or Liberation for instance): smaller packets means more calls and more overhead.

Although Reed-Solomon is provided as a default, Ceph uses it via an abstract API designed to allow each pool to choose the plugin that implements it using key=value pairs stored in an erasure code profile.

```
$ ceph osd erasure-code-profile set myprofile \
    ruleset-failure-domain=osd
$ ceph osd erasure-code-profile get myprofile
directory=/usr/lib/ceph/erasure-code
k=2
m=1
plugin=jerasure
technique=reed_sol_van
ruleset-failure-domain=osd
$ ceph osd pool create ecpool 12 12 erasure myprofile
```

The *plugin* is dynamically loaded from *directory* and expected to implement the *int __erasure_code_init(char *plugin_name, char *directory)* function which is responsible for registering an object derived from *ErasureCodePlugin* in the registry. The ErasureCodePluginExample plugin reads:

```
ErasureCodePluginRegistry &instance =
                           ErasureCodePluginRegistry::instance();
instance.add(plugin_name, new ErasureCodePluginExample());
```

The *ErasureCodePlugin* derived object must provide a factory method from which the concrete implementation of the *ErasureCodeInterface* object can be generated. The ErasureCodePluginExample plugin reads:

```
virtual int factory(const map<std::string,std::string> &parameters,
                    ErasureCodeInterfaceRef *erasure_code) {
```

```
  *erasure_code = ErasureCodeInterfaceRef(new ErasureCodeExample(parameters));
  return 0;
}
```

The *parameters* argument is the list of *key=value* pairs that were set in the erasure code profile, before the pool was created.

```
ceph osd erasure-code-profile set myprofile \
   directory=<dir>         \ # mandatory
   plugin=jerasure         \ # mandatory
   m=10                    \ # optional and plugin dependant
   k=3                     \ # optional and plugin dependant
   technique=reed_sol_van  \ # optional and plugin dependant
```

Notes If the objects are large, it may be impractical to encode and decode them in memory. However, when using *RBD* a 1TB device is divided in many individual 4MB objects and *RGW* does the same.

Encoding and decoding is implemented in the OSD. Although it could be implemented client side for read write, the OSD must be able to encode and decode on its own when scrubbing.

jerasure plugin

Introduction The parameters interpreted by the jerasure plugin are:

```
ceph osd erasure-code-profile set myprofile \
   directory=<dir>         \ # plugin directory absolute path
   plugin=jerasure         \ # plugin name (only jerasure)
   k=<k>                   \ # data chunks (default 2)
   m=<m>                   \ # coding chunks (default 2)
   technique=<technique>   \ # coding technique
```

The coding techniques can be chosen among *reed_sol_van*, *reed_sol_r6_op*, *cauchy_orig*, *cauchy_good*, *liberation*, *blaum_roth* and *liber8tion*.

The *src/erasure-code/jerasure* directory contains the implementation. It is a wrapper around the code found at https://github.com/ceph/jerasure and https://github.com/ceph/gf-complete , pinned to the latest stable version in *.gitmodules*. These repositories are copies of the upstream repositories http://jerasure.org/jerasure/jerasure and http://jerasure.org/jerasure/gf-complete . The difference between the two, if any, should match pull requests against upstream.

PG Backend Proposal

NOTE: the last update of this page is dated 2013, before the Firefly release. The details of the implementation may be different.

Motivation The purpose of the PG Backend interface is to abstract over the differences between replication and erasure coding as failure recovery mechanisms.

Much of the existing PG logic, particularly that for dealing with peering, will be common to each. With both schemes, a log of recent operations will be used to direct recovery in the event that an OSD is down or disconnected for a brief period of time. Similarly, in both cases it will be necessary to scan a recovered copy of the PG in order to recover an empty OSD. The PGBackend abstraction must be sufficiently expressive for Replicated and ErasureCoded backends to be treated uniformly in these areas.

However, there are also crucial differences between using replication and erasure coding which PGBackend must abstract over:

1. The current write strategy would not ensure that a particular object could be reconstructed after a failure.
2. Reads on an erasure coded PG require chunks to be read from the replicas as well.
3. Object recovery probably involves recovering the primary and replica missing copies at the same time to avoid performing extra reads of replica shards.
4. Erasure coded PG chunks created for different acting set positions are not interchangeable. In particular, it might make sense for a single OSD to hold more than 1 PG copy for different acting set positions.
5. Selection of a pgtemp for backfill may differ between replicated and erasure coded backends.
6. The set of necessary OSDs from a particular interval required to to continue peering may differ between replicated and erasure coded backends.
7. The selection of the authoritative log may differ between replicated and erasure coded backends.

Client Writes The current PG implementation performs a write by performing the write locally while concurrently directing replicas to perform the same operation. Once all operations are durable, the operation is considered durable. Because these writes may be destructive overwrites, during peering, a log entry on a replica (or the primary) may be found to be divergent if that replica remembers a log event which the authoritative log does not contain. This can happen if only 1 out of 3 replicas persisted an operation, but was not available in the next interval to provide an authoritative log. With replication, we can repair the divergent object as long as at least 1 replica has a current copy of the divergent object. With erasure coding, however, it might be the case that neither the new version of the object nor the old version of the object has enough available chunks to be reconstructed. This problem is much simpler if we arrange for all supported operations to be locally roll-back-able.

- CEPH_OSD_OP_APPEND: We can roll back an append locally by including the previous object size as part of the PG log event.
- CEPH_OSD_OP_DELETE: The possibility of rolling back a delete requires that we retain the deleted object until all replicas have persisted the deletion event. ErasureCoded backend will therefore need to store objects with the version at which they were created included in the key provided to the filestore. Old versions of an object can be pruned when all replicas have committed up to the log event deleting the object.
- CEPH_OSD_OP_(SET|RM)ATTR: If we include the prior value of the attr to be set or removed, we can roll back these operations locally.

Core Changes:

- Current code should be adapted to use and rollback as appropriate APPEND, DELETE, (SET|RM)ATTR log entries.
- The filestore needs to be able to deal with multiply versioned hobjects. This means adapting the filestore internally to use a ghobject which is basically a tuple<hobject_t, gen_t, shard_t>. The gen_t + shard_t need to be included in the on-disk filename. gen_t is a unique object identifier to make sure there are no name collisions when object N is created + deleted + created again. An interface needs to be added to get all versions of a particular hobject_t or the most recently versioned instance of a particular hobject_t.

PGBackend Interfaces:

- PGBackend::perform_write() : It seems simplest to pass the actual ops vector. The reason for providing an async, callback based interface rather than having the PGBackend respond directly is that we might want to use this interface for internal operations like watch/notify expiration or snap trimming which might not necessarily have an external client.

- PGBackend::try_rollback() : Some log entries (all of the ones valid for the Erasure coded backend) will support local rollback. In those cases, PGLog can avoid adding objects to the missing set when identifying divergent objects.

Peering and PG Logs Currently, we select the log with the newest last_update and the longest tail to be the authoritative log. This is fine because we aren't generally able to roll operations on the other replicas forward or backwards, instead relying on our ability to re-replicate divergent objects. With the write approach discussed in the previous section, however, the erasure coded backend will rely on being able to roll back divergent operations since we may not be able to re-replicate divergent objects. Thus, we must choose the *oldest* last_update from the last interval which went active in order to minimize the number of divergent objects.

The difficulty is that the current code assumes that as long as it has an info from at least 1 OSD from the prior interval, it can complete peering. In order to ensure that we do not end up with an unrecoverably divergent object, a K+M erasure coded PG must hear from at least K of the replicas of the last interval to serve writes. This ensures that we will select a last_update old enough to roll back at least K replicas. If a replica with an older last_update comes along later, we will be able to provide at least K chunks of any divergent object.

Core Changes:

- PG::choose_acting(), etc. need to be generalized to use PGBackend to determine the authoritative log.
- PG::RecoveryState::GetInfo needs to use PGBackend to determine whether it has enough infos to continue with authoritative log selection.

PGBackend interfaces:

- have_enough_infos()
- choose_acting()

PGTemp Currently, an OSD is able to request a temp acting set mapping in order to allow an up-to-date OSD to serve requests while a new primary is backfilled (and for other reasons). An erasure coded pg needs to be able to designate a primary for these reasons without putting it in the first position of the acting set. It also needs to be able to leave holes in the requested acting set.

Core Changes:

- OSDMap::pg_to_*_osds needs to separately return a primary. For most cases, this can continue to be acting[0].
- MOSDPGTemp (and related OSD structures) needs to be able to specify a primary as well as an acting set.
- Much of the existing code base assumes that acting[0] is the primary and that all elements of acting are valid. This needs to be cleaned up since the acting set may contain holes.

Client Reads Reads with the replicated strategy can always be satisfied synchronously out of the primary OSD. With an erasure coded strategy, the primary will need to request data from some number of replicas in order to satisfy a read. The perform_read() interface for PGBackend therefore will be async.

PGBackend interfaces:

- perform_read(): as with perform_write() it seems simplest to pass the ops vector. The call to oncomplete will occur once the out_bls have been appropriately filled in.

Distinguished acting set positions With the replicated strategy, all replicas of a PG are interchangeable. With erasure coding, different positions in the acting set have different pieces of the erasure coding scheme and are not interchangeable. Worse, crush might cause chunk 2 to be written to an OSD which happens already to contain an (old) copy of chunk 4. This means that the OSD and PG messages need to work in terms of a type like pair<shard_t, pg_t> in order to distinguish different pg chunks on a single OSD.

Because the mapping of object name to object in the filestore must be 1-to-1, we must ensure that the objects in chunk 2 and the objects in chunk 4 have different names. To that end, the filestore must include the chunk id in the object key.

Core changes:

- The filestore ghobject_t needs to also include a chunk id making it more like tuple<hobject_t, gen_t, shard_t>.
- coll_t needs to include a shard_t.
- The OSD pg_map and similar pg mappings need to work in terms of a spg_t (essentially pair<pg_t, shard_t>). Similarly, pg->pg messages need to include a shard_t
- For client->PG messages, the OSD will need a way to know which PG chunk should get the message since the OSD may contain both a primary and non-primary chunk for the same pg

Object Classes We probably won't support object classes at first on Erasure coded backends.

Scrub We currently have two scrub modes with different default frequencies:

1. [shallow] scrub: compares the set of objects and metadata, but not the contents
2. deep scrub: compares the set of objects, metadata, and a crc32 of the object contents (including omap)

The primary requests a scrubmap from each replica for a particular range of objects. The replica fills out this scrubmap for the range of objects including, if the scrub is deep, a crc32 of the contents of each object. The primary gathers these scrubmaps from each replica and performs a comparison identifying inconsistent objects.

Most of this can work essentially unchanged with erasure coded PG with the caveat that the PGBackend implementation must be in charge of actually doing the scan, and that the PGBackend implementation should be able to attach arbitrary information to allow PGBackend on the primary to scrub PGBackend specific metadata.

The main catch, however, for erasure coded PG is that sending a crc32 of the stored chunk on a replica isn't particularly helpful since the chunks on different replicas presumably store different data. Because we don't support overwrites except via DELETE, however, we have the option of maintaining a crc32 on each chunk through each append. Thus, each replica instead simply computes a crc32 of its own stored chunk and compares it with the locally stored checksum. The replica then reports to the primary whether the checksums match.

PGBackend interfaces:

- scan()
- scrub()
- compare_scrub_maps()

Crush If crush is unable to generate a replacement for a down member of an acting set, the acting set should have a hole at that position rather than shifting the other elements of the acting set out of position.

Core changes:

- Ensure that crush behaves as above for INDEP.

Recovery The logic for recovering an object depends on the backend. With the current replicated strategy, we first pull the object replica to the primary and then concurrently push it out to the replicas. With the erasure coded strategy, we probably want to read the minimum number of replica chunks required to reconstruct the object and push out the replacement chunks concurrently.

Another difference is that objects in erasure coded pg may be unrecoverable without being unfound. The "unfound" concept should probably then be renamed to unrecoverable. Also, the PGBackend implementation will have to be able to direct the search for pg replicas with unrecoverable object chunks and to be able to determine whether a particular object is recoverable.

Core changes:

- s/unfound/unrecoverable

PGBackend interfaces:

- on_local_recover_start
- on_local_recover
- on_global_recover
- on_peer_recover
- begin_peer_recover

Backfill For the most part, backfill itself should behave similarly between replicated and erasure coded pools with a few exceptions:

1. We probably want to be able to backfill multiple OSDs concurrently with an erasure coded pool in order to cut down on the read overhead.
2. We probably want to avoid having to place the backfill peers in the acting set for an erasure coded pg because we might have a good temporary pg chunk for that acting set slot.

For 2, we don't really need to place the backfill peer in the acting set for replicated PGs anyway. For 1, PGBackend::choose_backfill() should determine which OSDs are backfilled in a particular interval.

Core changes:

- Backfill should be capable of handling multiple backfill peers concurrently even for replicated pgs (easier to test for now)
- Backfill peers should not be placed in the acting set.

PGBackend interfaces:

- choose_backfill(): allows the implementation to determine which OSDs should be backfilled in a particular interval.

10.36.3 last_epoch_started

info.last_epoch_started records an activation epoch e for interval i such that all writes commited in i or earlier are reflected in the local info/log and no writes after i are reflected in the local info/log. Since no committed write is ever divergent, even if we get an authoritative log/info with an older info.last_epoch_started, we can leave our info.last_epoch_started alone since no writes could have commited in any intervening interval (See PG::proc_master_log).

info.history.last_epoch_started records a lower bound on the most recent interval in which the pg as a whole went active and accepted writes. On a particular osd, it is also an upper bound on the activation epoch of intervals in which writes in the local pg log occurred (we update it before accepting writes). Because all committed writes are committed by all acting set osds, any non-divergent writes ensure that history.last_epoch_started was recorded by all acting set members in the interval. Once peering has queried one osd from each interval back to some seen history.last_epoch_started, it follows that no interval after the max history.last_epoch_started can have reported writes as committed (since we record it before recording client writes in an interval). Thus, the minimum last_update across all infos with info.last_epoch_started >= MAX(history.last_epoch_started) must be an upper bound on writes reported as committed to the client.

We update info.last_epoch_started with the intial activation message, but we only update history.last_epoch_started after the new info.last_epoch_started is persisted (possibly along with the first write). This ensures that we do not require an osd with the most recent info.last_epoch_started until all acting set osds have recorded it. In find_best_info, we do include info.last_epoch_started values when calculating the max_last_epoch_started_found because we want to avoid designating a log entry divergent which in a prior interval would have been non-divergent. In activate(), we use the peer's last_epoch_started value as a bound on how far back divergent log entries can be found.

10.36.4 Map and PG Message handling

Overview

The OSD handles routing incoming messages to PGs, creating the PG if necessary in some cases.

PG messages generally come in two varieties:

1. Peering Messages
2. Ops/SubOps

There are several ways in which a message might be dropped or delayed. It is important that the message delaying does not result in a violation of certain message ordering requirements on the way to the relevant PG handling logic:

1. Ops referring to the same object must not be reordered.
2. Peering messages must not be reordered.
3. Subops must not be reordered.

MOSDMap

MOSDMap messages may come from either monitors or other OSDs. Upon receipt, the OSD must perform several tasks:

1. Persist the new maps to the filestore. Several PG operations rely on having access to maps dating back to the last time the PG was clean.
2. Update and persist the superblock.
3. Update OSD state related to the current map.
4. Expose new maps to PG processes via *OSDService*.
5. Remove PGs due to pool removal.
6. Queue dummy events to trigger PG map catchup.

Each PG asynchronously catches up to the currently published map during process_peering_events before processing the event. As a result, different PGs may have different views as to the "current" map.

One consequence of this design is that messages containing submessages from multiple PGs (MOSDPGInfo, MOSDPGQuery, MOSDPGNotify) must tag each submessage with the PG's epoch as well as tagging the message as a whole with the OSD's current published epoch.

MOSDPGOp/MOSDPGSubOp

See OSD::dispatch_op, OSD::handle_op, OSD::handle_sub_op

MOSDPGOps are used by clients to initiate rados operations. MOSDSubOps are used between OSDs to coordinate most non peering activities including replicating MOSDPGOp operations.

OSD::require_same_or_newer map checks that the current OSDMap is at least as new as the map epoch indicated on the message. If not, the message is queued in OSD::waiting_for_osdmap via OSD::wait_for_new_map. Note, this cannot violate the above conditions since any two messages will be queued in order of receipt and if a message is received with epoch e0, a later message from the same source must be at epoch at least e0. Note that two PGs from the same OSD count for these purposes as different sources for single PG messages. That is, messages from different PGs may be reordered.

MOSDPGOps follow the following process:

1. OSD::handle_op: validates permissions and crush mapping. discard the request if they are not connected and the client cannot get the reply (See OSD::op_is_discardable) See OSDService::handle_misdirected_op See PG::op_has_sufficient_caps See OSD::require_same_or_newer_map
2. OSD::enqueue_op

MOSDSubOps follow the following process:

1. OSD::handle_sub_op checks that sender is an OSD
2. OSD::enqueue_op

OSD::enqueue_op calls PG::queue_op which checks waiting_for_map before calling OpWQ::queue which adds the op to the queue of the PG responsible for handling it.

OSD::dequeue_op is then eventually called, with a lock on the PG. At this time, the op is passed to PG::do_request, which checks that:

1. the PG map is new enough (PG::must_delay_op)
2. the client requesting the op has enough permissions (PG::op_has_sufficient_caps)
3. the op is not to be discarded (PG::can_discard_{request,op,subop,scan,backfill})
4. the PG is active (PG::flushed boolean)
5. the op is a CEPH_MSG_OSD_OP and the PG is in PG_STATE_ACTIVE state and not in PG_STATE_REPLAY

If these conditions are not met, the op is either discarded or queued for later processing. If all conditions are met, the op is processed according to its type:

1. CEPH_MSG_OSD_OP is handled by PG::do_op
2. MSG_OSD_SUBOP is handled by PG::do_sub_op
3. MSG_OSD_SUBOPREPLY is handled by PG::do_sub_op_reply
4. MSG_OSD_PG_SCAN is handled by PG::do_scan
5. MSG_OSD_PG_BACKFILL is handled by PG::do_backfill

CEPH_MSG_OSD_OP processing

ReplicatedPG::do_op handles CEPH_MSG_OSD_OP op and will queue it

1. in wait_for_all_missing if it is a CEPH_OSD_OP_PGLS for a designated snapid and some object updates are still missing
2. in waiting_for_active if the op may write but the scrubber is working
3. in waiting_for_missing_object if the op requires an object or a snapdir or a specific snap that is still missing
4. in waiting_for_degraded_object if the op may write an object or a snapdir that is degraded, or if another object blocks it ("blocked_by")
5. in waiting_for_backfill_pos if the op requires an object that will be available after the backfill is complete

6. in waiting_for_ack if an ack from another OSD is expected
7. in waiting_for_ondisk if the op is waiting for a write to complete

Peering Messages

See OSD::handle_pg_(notify|info|log|query)

Peering messages are tagged with two epochs:

1. epoch_sent: map epoch at which the message was sent
2. query_epoch: map epoch at which the message triggering the message was sent

These are the same in cases where there was no triggering message. We discard a peering message if the message's query_epoch if the PG in question has entered a new epoch (See PG::old_peering_evt, PG::queue_peering_event). Notifies, infos, notifies, and logs are all handled as PG::RecoveryMachine events and are wrapped by PG::queue_* by PG::CephPeeringEvts, which include the created state machine event along with epoch_sent and query_epoch in order to generically check PG::old_peering_message upon insertion and removal from the queue.

Note, notifies, logs, and infos can trigger the creation of a PG. See OSD::get_or_create_pg.

10.36.5 OSD

Concepts

Messenger See src/msg/Messenger.h

Handles sending and receipt of messages on behalf of the OSD. The OSD uses two messengers:

1. cluster_messenger - handles traffic to other OSDs, monitors
2. client_messenger - handles client traffic

   This division allows the OSD to be configured with different interfaces for client and cluster traffic.

Dispatcher See src/msg/Dispatcher.h

OSD implements the Dispatcher interface. Of particular note is ms_dispatch, which serves as the entry point for messages received via either the client or cluster messenger. Because there are two messengers, ms_dispatch may be called from at least two threads. The osd_lock is always held during ms_dispatch.

WorkQueue See src/common/WorkQueue.h

The WorkQueue class abstracts the process of queueing independent tasks for asynchronous execution. Each OSD process contains workqueues for distinct tasks:

1. OpWQ: handles ops (from clients) and subops (from other OSDs). Runs in the op_tp threadpool.
2. PeeringWQ: handles peering tasks and pg map advancement Runs in the op_tp threadpool. See Peering
3. CommandWQ: handles commands (pg query, etc) Runs in the command_tp threadpool.
4. RecoveryWQ: handles recovery tasks. Runs in the recovery_tp threadpool.
5. SnapTrimWQ: handles snap trimming Runs in the disk_tp threadpool. See SnapTrimmer
6. ScrubWQ: handles primary scrub path Runs in the disk_tp threadpool. See Scrub
7. ScrubFinalizeWQ: handles primary scrub finalize Runs in the disk_tp threadpool. See Scrub

8. RepScrubWQ: handles replica scrub path Runs in the disk_tp threadpool See Scrub
9. RemoveWQ: Asynchronously removes old pg directories Runs in the disk_tp threadpool See PGRemoval

ThreadPool See src/common/WorkQueue.h See also above.

There are 4 OSD threadpools:

1. op_tp: handles ops and subops
2. recovery_tp: handles recovery tasks
3. disk_tp: handles disk intensive tasks
4. command_tp: handles commands

OSDMap See src/osd/OSDMap.h

The crush algorithm takes two inputs: a picture of the cluster with status information about which nodes are up/down and in/out, and the pgid to place. The former is encapsulated by the OSDMap. Maps are numbered by *epoch* (epoch_t). These maps are passed around within the OSD as std::tr1::shared_ptr<const OSDMap>.

See MapHandling

PG See src/osd/PG.* src/osd/ReplicatedPG.*

Objects in rados are hashed into *PGs* and *PGs* are placed via crush onto OSDs. The PG structure is responsible for handling requests pertaining to a particular *PG* as well as for maintaining relevant metadata and controlling recovery.

OSDService See src/osd/OSD.cc OSDService

The OSDService acts as a broker between PG threads and OSD state which allows PGs to perform actions using OSD services such as workqueues and messengers. This is still a work in progress. Future cleanups will focus on moving such state entirely from the OSD into the OSDService.

Overview

See src/ceph_osd.cc

The OSD process represents one leaf device in the crush hierarchy. There might be one OSD process per physical machine, or more than one if, for example, the user configures one OSD instance per disk.

10.36.6 OSD Internals

```
                                                                                                   Me
         |------------------------------------------------------------------------------------------
                                                                     FileStore op_queue throttle (nu
                                                                 |-----------------------------------



                                                                                         |------------


Op: Read Header --DispatchQ--> OSD::_dispatch --OpWQ--> PG::do_request --journalq--> Journal --FileSt
                                              |
SubOp:                                        --Messenger--> ReadHeader --DispatchQ--> OSD::_dispatch
```

10.36.7 PG

Concepts

Peering Interval See PG::start_peering_interval. See PG::acting_up_affected See PG::RecoveryState::Reset

A peering interval is a maximal set of contiguous map epochs in which the up and acting sets did not change. PG::RecoveryMachine represents a transition from one interval to another as passing through RecoveryState::Reset. On PG::RecoveryState::AdvMap PG::acting_up_affected can cause the pg to transition to Reset.

Peering Details and Gotchas

For an overview of peering, see Peering.

- PG::flushed defaults to false and is set to false in PG::start_peering_interval. Upon transitioning to PG::RecoveryState::Started we send a transaction through the pg op sequencer which, upon complete, sends a FlushedEvt which sets flushed to true. The primary cannot go active until this happens (See PG::RecoveryState::WaitFlushedPeering). Replicas can go active but cannot serve ops (writes or reads). This is necessary because we cannot read our ondisk state until unstable transactions from the previous interval have cleared.

10.36.8 PG Removal

See OSD::_remove_pg, OSD::RemoveWQ

There are two ways for a pg to be removed from an OSD:

1. MOSDPGRemove from the primary
2. OSD::advance_map finds that the pool has been removed

In either case, our general strategy for removing the pg is to atomically set the metadata objects (pg->log_oid, pg->biginfo_oid) to backfill and asynronously remove the pg collections. We do not do this inline because scanning the collections to remove the objects is an expensive operation.

OSDService::deleting_pgs tracks all pgs in the process of being deleted. Each DeletingState object in deleting_pgs lives while at least one reference to it remains. Each item in RemoveWQ carries a reference to the DeletingState for the relevant pg such that deleting_pgs.lookup(pgid) will return a null ref only if there are no collections currently being deleted for that pg.

The DeletingState for a pg also carries information about the status of the current deletion and allows the deletion to be cancelled. The possible states are:

1. QUEUED: the PG is in the RemoveWQ
2. CLEARING_DIR: the PG's contents are being removed synchronously
3. DELETING_DIR: the PG's directories and metadata being queued for removal
4. DELETED_DIR: the final removal transaction has been queued

5. CANCELED: the deletion has been canceled

In 1 and 2, the deletion can be canceled. Each state transition method (and check_canceled) returns false if deletion has been canceled and true if the state transition was successful. Similarly, try_stop_deletion() returns true if it succeeds in canceling the deletion. Additionally, try_stop_deletion() in the event that it fails to stop the deletion will not return until the final removal transaction is queued. This ensures that any operations queued after that point will be ordered after the pg deletion.

OSD::_create_lock_pg must handle two cases:

1. Either there is no DeletingStateRef for the pg, or it failed to cancel
2. We succeeded in canceling the deletion.

In case 1., we proceed as if there were no deletion occurring, except that we avoid writing to the PG until the deletion finishes. In case 2., we proceed as in case 1., except that we first mark the PG as backfilling.

Similarly, OSD::osr_registry ensures that the OpSequencers for those pgs can be reused for a new pg if created before the old one is fully removed, ensuring that operations on the new pg are sequenced properly with respect to operations on the old one.

10.36.9 Recovery Reservation

Recovery reservation extends and subsumes backfill reservation. The reservation system from backfill recovery is used for local and remote reservations.

When a PG goes active, first it determines what type of recovery is necessary, if any. It may need log-based recovery, backfill recovery, both, or neither.

In log-based recovery, the primary first acquires a local reservation from the OSDService's local_reserver. Then a MRemoteReservationRequest message is sent to each replica in order of OSD number. These requests will always be granted (i.e., cannot be rejected), but they may take some time to be granted if the remotes have already granted all their remote reservation slots.

After all reservations are acquired, log-based recovery proceeds as it would without the reservation system.

After log-based recovery completes, the primary releases all remote reservations. The local reservation remains held. The primary then determines whether backfill is necessary. If it is not necessary, the primary releases its local reservation and waits in the Recovered state for all OSDs to indicate that they are clean.

If backfill recovery occurs after log-based recovery, the local reservation does not need to be reacquired since it is still held from before. If it occurs immediately after activation (log-based recovery not possible/necessary), the local reservation is acquired according to the typical process.

Once the primary has its local reservation, it requests a remote reservation from the backfill target. This reservation CAN be rejected, for instance if the OSD is too full (osd_backfill_full_ratio config option). If the reservation is rejected, the primary drops its local reservation, waits (osd_backfill_retry_interval), and then retries. It will retry indefinitely.

Once the primary has the local and remote reservations, backfill proceeds as usual. After backfill completes the remote reservation is dropped.

Finally, after backfill (or log-based recovery if backfill was not necessary), the primary drops the local reservation and enters the Recovered state. Once all the PGs have reported they are clean, the primary enters the Clean state and marks itself active+clean.

Things to Note

We always grab the local reservation first, to prevent a circular dependency. We grab remote reservations in order of OSD number for the same reason.

The recovery reservation state chart controls the PG state as reported to the monitor. The state chart can set:

- recovery_wait: waiting for local/remote reservations
- recovering: recoverying
- wait_backfill: waiting for remote backfill reservations
- backfilling: backfilling
- backfill_toofull: backfill reservation rejected, OSD too full

See Also

The Active substate of the automatically generated OSD state diagram.

10.36.10 Scrubbing Behavior Table

Flags	none	noscrub	nodeep_scrub	noscrub/nodeep_scrub
Periodic tick	S	X	S	X
Periodic tick after osd_deep_scrub_interval	D	D	S	X
Initiated scrub	S	S	S	S
Initiated scrub after osd_deep_scrub_interval	D	D	S	S
Initiated deep scrub	D	D	D	D

- X = Do nothing
- S = Do regular scrub
- D = Do deep scrub

State variables

- Periodic tick state is !must_scrub && !must_deep_scrub && !time_for_deep
- Periodic tick after osd_deep_scrub_interval state is !must_scrub && !must_deep_scrub && time_for_deep
- Initiated scrub state is must_scrub && !must_deep_scrub && !time_for_deep
- Initiated scrub after osd_deep_scrub_interval state is must scrub && !must_deep_scrub && time_for_deep
- Initiated deep scrub state is must_scrub && must_deep_scrub

10.36.11 Snaps

Overview

Rados supports two related snapshotting mechanisms:

1. *pool snaps*: snapshots are implicitely applied to all objects in a pool

2. *self managed snaps*: the user must provide the current *SnapContext* on each write.

These two are mutually exclusive, only one or the other can be used on a particular pool.

The *SnapContext* is the set of snapshots currently defined for an object as well as the most recent snapshot (the *seq*) requested from the mon for sequencing purposes (a *SnapContext* with a newer *seq* is considered to be more recent).

The difference between *pool snaps* and *self managed snaps* from the OSD's point of view lies in whether the *SnapContext* comes to the OSD via the client's MOSDOp or via the most recent OSDMap.

See OSD::make_writeable

Ondisk Structures

Each object has in the pg collection a *head* object (or *snapdir*, which we will come to shortly) and possibly a set of *clone* objects. Each hobject_t has a snap field. For the *head* (the only writeable version of an object), the snap field is set to CEPH_NOSNAP. For the *clones*, the snap field is set to the *seq* of the *SnapContext* at their creation. When the OSD services a write, it first checks whether the most recent *clone* is tagged with a snapid prior to the most recent snap represented in the *SnapContext*. If so, at least one snapshot has occurred between the time of the write and the time of the last clone. Therefore, prior to performing the mutation, the OSD creates a new clone for servicing reads on snaps between the snapid of the last clone and the most recent snapid.

The *head* object contains a *SnapSet* encoded in an attribute, which tracks

1. The full set of snaps defined for the object
2. The full set of clones which currently exist
3. Overlapping intervals between clones for tracking space usage
4. Clone size

If the *head* is deleted while there are still clones, a *snapdir* object is created instead to house the *SnapSet*.

Additionally, the *object_info_t* on each clone includes a vector of snaps for which clone is defined.

Snap Removal

To remove a snapshot, a request is made to the *Monitor* cluster to add the snapshot id to the list of purged snaps (or to remove it from the set of pool snaps in the case of *pool snaps*). In either case, the *PG* adds the snap to its *snap_trimq* for trimming.

A clone can be removed when all of its snaps have been removed. In order to determine which clones might need to be removed upon snap removal, we maintain a mapping from snap to *hobject_t* using the *SnapMapper*.

See ReplicatedPG::SnapTrimmer, SnapMapper

This trimming is performed asynchronously by the snap_trim_wq while the pg is clean and not scrubbing.

1. The next snap in PG::snap_trimq is selected for trimming
2. We determine the next object for trimming out of PG::snap_mapper. For each object, we create a log entry and repop updating the object info and the snap set (including adjusting the overlaps).
3. We also locally update our *SnapMapper* instance with the object's new snaps.
4. The log entry containing the modification of the object also contains the new set of snaps, which the replica uses to update its own *SnapMapper* instance.
5. The primary shares the info with the replica, which persists the new set of purged_snaps along with the rest of the info.

Recovery

Because the trim operations are implemented using repops and log entries, normal pg peering and recovery maintain the snap trimmer operations with the caveat that push and removal operations need to update the local *SnapMapper* instance. If the purged_snaps update is lost, we merely retrim a now empty snap.

SnapMapper

SnapMapper is implemented on top of map_cacher<string, bufferlist>, which provides an interface over a backing store such as the filesystem with async transactions. While transactions are incomplete, the map_cacher instance buffers unstable keys allowing consistent access without having to flush the filestore. *SnapMapper* provides two mappings:

1. hobject_t -> set<snapid_t>: stores the set of snaps for each clone object
2. snapid_t -> hobject_t: stores the set of hobjects with a the snapshot as one of its snaps

Assumption: there are lots of hobjects and relatively few snaps. The first encoding has a stringification of the object as the key and an encoding of the set of snaps as a value. The second mapping, because there might be many hobjects for a single snap, is stored as a collection of keys of the form stringify(snap)_stringify(object) such that stringify(snap) is constant length. These keys have a bufferlist encoding pair<snapid, hobject_t> as a value. Thus, creating or trimming a single object does not involve reading all objects for any snap. Additionally, upon construction, the *SnapMapper* is provided with a mask for filtering identifying the objects in the single SnapMapper keyspace belonging to that pg.

Split

The snapid_t -> hobject_t key entries are arranged such that for any pg, up to 8 prefixes need to be checked to determine all hobjects in a particular snap for a particular pg. Upon split, the prefixes to check on the parent are adjusted such that only the objects remaining in the pg will be visible. The children will immediately have the correct mapping.

10.36.12 Watch Notify

See librados for the watch/notify interface.

Overview

The object_info (See osd/osd_types.h) tracks the set of watchers for a particular object persistently in the object_info_t::watchers map. In order to track notify progress, we also maintain some ephemeral structures associated with the ObjectContext.

Each Watch has an associated Watch object (See osd/Watch.h). The ObjectContext for a watched object will have a (strong) reference to one Watch object per watch, and each Watch object holds a reference to the corresponding ObjectContext. This circular reference is deliberate and is broken when the Watch state is discarded on a new peering interval or removed upon timeout expiration or an unwatch operation.

A watch tracks the associated connection via a strong ConnectionRef Watch::conn. The associated connection has a WatchConState stashed in the OSD::Session for tracking associated Watches in order to be able to notify them upon ms_handle_reset() (via WatchConState::reset()).

Each Watch object tracks the set of currently un-acked notifies. start_notify() on a Watch object adds a reference to a new in-progress Notify to the Watch and either:

- if the Watch is *connected*, sends a Notify message to the client
- if the Watch is *unconnected*, does nothing.

When the Watch becomes connected (in ReplicatedPG::do_osd_op_effects), Notifies are resent to all remaining tracked Notify objects.

Each Notify object tracks the set of un-notified Watchers via calls to complete_watcher(). Once the remaining set is empty or the timeout expires (cb, registered in init()) a notify completion is sent to the client.

Watch Lifecycle

A watch may be in one of 5 states:

1. Non existent.
2. On disk, but not registered with an object context.
3. Connected
4. Disconnected, callback registered with timer
5. Disconnected, callback in queue for scrub or is_degraded

Case 2 occurs between when an OSD goes active and the ObjectContext for an object with watchers is loaded into memory due to an access. During Case 2, no state is registered for the watch. Case 2 transitions to Case 4 in ReplicatedPG::populate_obc_watchers() during ReplicatedPG::find_object_context. Case 1 becomes case 3 via OSD::do_osd_op_effects due to a watch operation. Case 4,5 become case 3 in the same way. Case 3 becomes case 4 when the connection resets on a watcher's session.

Cases 4&5 can use some explanation. Normally, when a Watch enters Case 4, a callback is registered with the OSDService::watch_timer to be called at timeout expiration. At the time that the callback is called, however, the pg might be in a state where it cannot write to the object in order to remove the watch (i.e., during a scrub or while the object is degraded). In that case, we use Watch::get_delayed_cb() to generate another Context for use from the callbacks_for_degraded_object and Scrubber::callbacks lists. In either case, Watch::unregister_cb() does the right thing (SafeTimer::cancel_event() is harmless for contexts not registered with the timer).

Notify Lifecycle

The notify timeout is simpler: a timeout callback is registered when the notify is init()'d. If all watchers ack notifies before the timeout occurs, the timeout is canceled and the client is notified of the notify completion. Otherwise, the timeout fires, the Notify object pings each Watch via cancel_notify to remove itself, and sends the notify completion to the client early.

10.36.13 Writeback Throttle

Previously, the filestore had a problem when handling large numbers of small ios. We throttle dirty data implicitely via the journal, but a large number of inodes can be dirtied without filling the journal resulting in a very long sync time when the sync finally does happen. The flusher was not an adequate solution to this problem since it forced writeback of small writes too eagerly killing performance.

WBThrottle tracks unflushed io per hobject_t and ::fsyncs in lru order once the start_flusher threshold is exceeded for any of dirty bytes, dirty ios, or dirty inodes. While any of these exceed the hard_limit, we block on throttle() in _do_op.

See src/os/WBThrottle.h, src/osd/WBThrottle.cc

To track the open FDs through the writeback process, there is now an fdcache to cache open fds. lfn_open now returns a cached FDRef which implicitely closes the fd once all references have expired.

Filestore syncs have a sideeffect of flushing all outstanding objects in the wbthrottle.

lfn_unlink clears the cached FDRef and wbthrottle entries for the unlinked object when then last link is removed and asserts that all outstanding FDRefs for that object are dead.

10.37 MDS developer documentation

Contents

10.37.1 MDS internal data structures

CInode CInode contains the metadata of a file, there is one CInode for each file. The CInode stores information like who owns the file, how big the file is.

CDentry CDentry is the glue that holds inodes and files together by relating inode to file/directory names. A CDentry links to at most one CInode (it may not link to any CInode). A CInode may be linked by multiple CDentries.

CDir CDir only exists for directory inode, it's used to link CDentries under the directory. A CInode can have multiple CDir when the directory is fragmented.

These data structures are linked together as:

```
CInode
CDir
 |   \
 |      \
 |         \
CDentry   CDentry
CInode    CInode
CDir      CDir
 |         |  \
 |         |     \
 |         |        \
CDentry   CDentry  CDentry
CInode    CInode   CInode
```

As this doc is being written, size of CInode is about 1400 bytes, size of CDentry is about 400 bytes, size of CDir is about 700 bytes. These data structures are quite large. Please be careful if you want to add new fields to them.

10.38 RADOS Gateway developer documentation

Contents

10.38.1 Usage Design Overview

Testing

The current usage testing does the following:

Following these operations:

- Create a few buckets
- Remove buckets
- Create a bucket

- Put object
- Remove object

Test:

1. Verify that 'usage show' with delete_obj category isn't empty after no more than 45 seconds (wait to flush)
2. Check the following

- 'usage show'
  - does not error out
  - num of entries > 0
  - num of summary entries > 0
  - for every entry in categories check successful_ops > 0
  - check that correct uid in the user summary
- 'usage show' with specified uid (–uid=<uid>')
  - num of entries > 0
  - num of summary entries > 0
  - for every entry in categories check successful_ops > 0
  - check that correct uid in the user summary
- 'usage show' with specified uid and specified categories (create_bucket, put_obj, delete_obj, delete_bucket)
  - for each category: - does not error out - num of entries > 0 - user in user summary is correct user - length of categories entries under user summary is exactly 1 - name of category under user summary is correct name - successful ops for the category > 0
- 'usage trim' with specified uid - does not error - check following 'usage show' shows complete usage info cleared for user

Additional required testing:

- test multiple users

  Do the same as in (2), with multiple users being set up.

- test with multiple buckets (> 1000 * factor, e.g., 2000)

  Create multiple buckets, put objects in each. Account the number written data and verify that usage reports show the expected number (up to a certain delta).

- verify usage show with a date/time range

  Take timestamp of the beginning of the test, and the end of the test. Round timestamps to the nearest hour (downward from start of test, upward from the end of test). List data starting at end-time, make sure that no data is being shown. List data ending at start-time, make sure that no data is shown. List data beginning at start-time, make sure that correct data is displayed. List data ending end end-time, make sure that correct data is displayed. List data beginning in begin-time, ending in end-time, make sure that correct data is displayed.

- verify usage trim with a date/time range

  Take timestamp of the beginning of the test, and the end of the test. Round timestamps to the nearest hour (downward from start of test, upward from the end of test). Trim data starting at end-time, make sure that no data has been trimmed. Trim data ending at start-time, make sure that no data has been trimmed. Trim data beginning in begin-time, ending in end-time, make sure that all data has been trimmed.

10.38.2 Admin Operations

An admin API request will be done on a URI that starts with the configurable 'admin' resource entry point. Authorization for the admin API duplicates the S3 authorization mechanism. Some operations require that the user holds special administrative capabilities. The response entity type (XML or JSON) may be specified as the 'format' option in the request and defaults to JSON if not specified.

Get Object

Get an existing object. NOTE: Does not require owner to be non-suspended.

Syntax

```
GET /{admin}/bucket?object&format=json HTTP/1.1
Host {fqdn}
```

Request Parameters

`bucket`

Description The bucket containing the object to be retrieved.

Type String

Example `foo_bucket`

Required Yes

`object`

Description The object to be retrieved.

Type String

Example `foo.txt`

Required Yes

Response Entities

If successful, returns the desired object.

`object`

Description The desired object.

Type Object

Special Error Responses

`NoSuchObject`

Description Specified object does not exist.

Code 404 Not Found

Head Object

Verify the existence of an object. If the object exists, metadata headers for the object will be returned.

Syntax

```
HEAD /{admin}/bucket?object HTTP/1.1
Host {fqdn}
```

Request Parameters

`bucket`

Description The bucket containing the object to be retrieved.

Type String

Example `foo_bucket`

Required Yes

`object`

Description The object to be retrieved.

Type String

Example `foo.txt`

Required Yes

Response Entities

None.

Special Error Responses

`NoSuchObject`

Description Specified object does not exist.

Code 404 Not Found

Get Zone Info

Get cluster information.

Syntax

```
GET /{admin}/zone&format=json HTTP/1.1
Host {fqdn}
```

Response Entities

If successful, returns cluster pool configuration.

`zone`

Description Contains current cluster pool configuration.

Type Container

`domain_root`

Description root of all buckets.

Type String

Parent `cluster`

`control_pool`

Description

Type String

Parent `cluster`

`gc_pool`

Description Garbage collection pool.

Type String

Parent `cluster`

`log_pool`

Description Log pool.

Type String

Parent `cluster`

`intent_log_pool`

Description Intent log pool.

Type String

Parent `cluster`

`usage_log_pool`

Description Usage log pool.

Type String

Parent `cluster`

`user_keys_pool`

Description User key pool.

Type String

Parent `cluster`

`user_email_pool`

Description User email pool.

Type String

Parent `cluster`

`user_swift_pool`

Description Pool of swift users.

Type String

Parent `cluster`

Special Error Responses

None.

Example Response

```
HTTP/1.1 200
Content-Type: application/json

{
  "domain_root": ".rgw",
  "control_pool": ".rgw.control",
  "gc_pool": ".rgw.gc",
  "log_pool": ".log",
  "intent_log_pool": ".intent-log",
  "usage_log_pool": ".usage",
  "user_keys_pool": ".users",
  "user_email_pool": ".users.email",
  "user_swift_pool": ".users.swift",
  "user_uid_pool ": ".users.uid"
}
```

Add Placement Pool

Make a pool available for data placement.

Syntax

```
PUT /{admin}/pool?format=json HTTP/1.1
Host {fqdn}
```

Request Parameters

`pool`

Description The pool to be made available for data placement.

Type String

Example `foo_pool`

Required Yes

`create`

Description Creates the data pool if it does not exist.

Type Boolean

Example False [False]

Required No

Response Entities

TBD.

Special Error Responses

TBD.

Remove Placement Pool

Make a pool unavailable for data placement.

Syntax

```
DELETE /{admin}/pool?format=json HTTP/1.1
Host {fqdn}
```

Request Parameters

`pool`

Description The existing pool to be made available for data placement.

Type String

Example `foo_pool`

Required Yes

`destroy`

Description Destroys the pool after removing it from the active set.

Type Boolean

Example False [False]

Required No

Response Entities

TBD.

Special Error Responses

TBD.

List Available Data Placement Pools

List current pools available for data placement.

Syntax

```
GET /{admin}/pool?format=json HTTP/1.1
Host {fqdn}
```

Response Entities

If successful, returns a list of pools available for data placement.

`pools`

Description Contains currently available pools for data placement.

Type Container

List Expired Garbage Collection Items

List objects scheduled for garbage collection.

Syntax

```
GET /{admin}/garbage?format=json HTTP/1.1
Host {fqdn}
```

Request Parameters

None.

Response Entities

If expired garbage collection items exist, a list of such objects will be returned.

`garbage`

Description Expired garbage collection items.

Type Container

`object`

Description A container garbage collection object information.

Type Container

Parent `garbage`

`name`

Description The name of the object.

Type String

Parent `object`

`expired`

Description The date at which the object expired.

Type String

Parent `object`

Special Error Responses

TBD.

Manually Processes Garbage Collection Items

List objects scheduled for garbage collection.

Syntax

```
DELETE /{admin}/garbage?format=json HTTP/1.1
Host {fqdn}
```

Request Parameters

None.

Response Entities

If expired garbage collection items exist, a list of removed objects will be returned.

`garbage`

Description Expired garbage collection items.

Type Container

`object`

Description A container garbage collection object information.

Type Container

Parent `garbage`

`name`

Description The name of the object.

Type String

Parent `object`

`expired`

Description The date at which the object expired.

Type String

Parent `object`

Special Error Responses

TBD.

Show Log Objects

Show log objects

Syntax

```
GET /{admin}/log?format=json HTTP/1.1
Host {fqdn}
```

Request Parameters

`object`

Description The log object to return.

Type String:

Example `2012-10-11-09-4165.2-foo_bucket`

Required No

Response Entities

If no object is specified, returns the full list of log objects.

`log-objects`

Description A list of log objects.

Type Container

`object`

Description The name of the log object.

Type String

`log`

Description The contents of the log object.

Type Container

Special Error Responses

None.

Standard Error Responses

`AccessDenied`

Description Access denied.

Code 403 Forbidden

`InternalError`

Description Internal server error.

Code 500 Internal Server Error

`NoSuchUser`

Description User does not exist.

Code 404 Not Found

`NoSuchBucket`

Description Bucket does not exist.

Code 404 Not Found

`NoSuchKey`

Description No such access key.

Code 404 Not Found

10.38.3 Radios Gateway S3 API Compliance

Warning: This document is a draft, it might not be accurate

Naming code reference

Here comes a BNF definition on how to name a feature in the code for referencing purpose :

```
name ::= request_type "_" ( header | operation ) ( "_" header_option )?

request_type ::= "req" | "res"

header ::= string

operation ::= method resource

method ::= "GET" | "PUT" | "POST" | "DELETE" | "OPTIONS" | "HEAD"

resource ::= string

header_option ::= string
```

Common Request Headers

S3 Documentation reference : http://docs.aws.amazon.com/AmazonS3/latest/API/RESTCommonRequestHeaders.html

Header	Supported?	Code Links	Tests links
Authorization	Yes	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_s3.cc#L1 https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_s3.cc#L2	
Content-Length	Yes		
Content-Type	Yes		
Content-MD5	Yes	https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.cc#L1249 https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.cc#L1306	
Date	Yes	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_auth_s3.cc#L	
Expect	Yes	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest.cc#L1227 https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_s3.cc#L8 https://github.com/ceph/ceph/blob/76040d90f7eb9f9921a3b8dcd0f821ac2cd9c492/src/rgw/rgw_main.cc#L372	
Host	?		
x-amz-date	Yes	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_auth_s3.cc#L should take precedence over DATE as mentionned here -> http://docs.aws.amazon.com/AmazonS3/latest/API/RESTCommonRequestHeaders.html	
x-amz-security-token	No		

Common Response Headers

S3 Documentation reference : http://docs.aws.amazon.com/AmazonS3/latest/API/RESTCommonResponseHeaders.html

Header	Supported?	Code Links	Tests links
Content-Length	Yes		
Connection	?		
Date	?		
ETag	Yes	https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.cc#L1312 https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.cc#L1436 https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.cc#L2222 https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_s3.cc#L1 https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_s3.cc#L2 https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_s3.cc#L5 https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_s3.cc#L1 https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_s3.cc#L1 https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_s3.cc#L1	
Server	No		
x-amz-delete-marker	No		
x-amz-id-2	No		
x-amz-request-id	No		
x-amz-version-id	No		

Operations on the Service

S3 Documentation reference : http://docs.aws.amazon.com/AmazonS3/latest/API/RESTServiceOps.html

Type	Operation	Supported?	Code links	Tests links
GET	Service	Yes	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_s3.cc#L https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_s3.cc#L https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_s3.cc#L	

Operations on Buckets

S3 Documentation reference : http://docs.aws.amazon.com/AmazonS3/latest/API/RESTBucketOps.html

Type	Operation	Supported?	Code links	Tests links
DELETE	Bucket	Yes	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48	
DELETE	Bucket cors	?	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48	
DELETE	Bucket lifecycle	No		
DELETE	Bucket policy	?		
DELETE	Bucket tagging	?		
DELETE	Bucket website	No		
GET	Bucket	Yes	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48	
GET	Bucket acl	Yes	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48	
GET	Bucket cors	?	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48	
GET	Bucket lifecycle	No		
GET	Bucket location	No		
GET	Bucket policy	?	https://github.com/ceph/ceph/blob/e91042171939b6bf82a56a1015c5cae792d2	
GET	Bucket logging	?	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48	
GET	Bucket notification	No		
GET	Bucket tagging	No		
GET	Bucket Object versions	No		
GET	Bucket requestPayment	No		
GET	Bucket versionning	No		
GET	Bucket website	No		
GET	List Multipart uploads	Yes	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48	
HEAD	Bucket	Yes	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48	
PUT	Bucket	Yes	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48	
PUT	Bucket acl	Yes	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48	
PUT	Bucket cors	?	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48	
PUT	Bucket lifecycle	No		
PUT	Bucket policy	?		
PUT	Bucket logging	?		
PUT	Bucket notification	No		
PUT	Bucket tagging	?		
PUT	Bucket requestPayment	No		
PUT	Bucket versionning	No		
PUT	Bucket website	N0		

Operations on Objects

S3 Documentation reference : http://docs.aws.amazon.com/AmazonS3/latest/API/RESTObjectOps.html

Type	Operation	Supported?	Code links	Tests links
DELETE	Object	Yes	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c	
DELETE	Multiple objects	Yes	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c	
GET	Object	Yes	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c	
GET	Object acl	Yes		
GET	Object torrent	No		
HEAD	Object	Yes	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c	
OPTIONS	Object	Yes	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c	
POST	Object	Yes	https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/8a2eb18494005aa968b71f18121da8ebab48e950/src/rgw/rgw_rest_ https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c https://github.com/ceph/ceph/blob/b139a7cd34b4e203ab164ada7a8fa590b50d8b13/src/rgw/rgw_op.c	
POST	Object restore	?		
PUT	Object	Yes		
PUT	Object acl	Yes		
PUT	Object copy	Yes		

CHAPTER

ELEVEN

RELEASE NOTES

11.1 v9.0.0

This is the first development release for the Infernalis cycle, and the first Ceph release to sport a version number from the new numbering scheme. The "9" indicates this is the 9th release cycle–I (for Infernalis) is the 9th letter. The first "0" indicates this is a development release ("1" will mean release candidate and "2" will mean stable release), and the final "0" indicates this is the first such development release.

A few highlights include:

- a new 'ceph daemonperf' command to watch perfcounter stats in realtime
- reduced MDS memory usage
- many MDS snapshot fixes
- librbd can now store options in the image itself
- many fixes for RGW Swift API support
- OSD performance improvements
- many doc updates and misc bug fixes

11.1.1 Notable Changes

- aarch64: add optimized version of crc32c (Yazen Ghannam, Steve Capper)
- auth: reinit NSS after fork() (#11128 Yan, Zheng)
- build: disable LTTNG by default (#11333 Josh Durgin)
- build: fix ppc build (James Page)
- build: install-deps: support OpenSUSE (Loic Dachary)
- build: misc cmake fixes (Matt Benjamin)
- ceph-disk: follow ceph-osd hints when creating journal (#9580 Sage Weil)
- ceph-disk: handle re-using existing partition (#10987 Loic Dachary)
- ceph-disk: improve parted output parsing (#10983 Loic Dachary)
- ceph-disk: make suppression work for activate-all and activate-journal (Dan van der Ster)
- ceph-disk: misc fixes (Alfredo Deza)
- ceph-fuse, libcephfs: don't clear COMPLETE when trimming null (Yan, Zheng)

- ceph-fuse, libcephfs: hold exclusive caps on dirs we “own” (#11226 Greg Farnum)
- ceph-fuse: do not require successful remount when unmounting (#10982 Greg Farnum)
- ceph: new ‘ceph daemonperf’ command (John Spray, Mykola Golub)
- common: PriorityQueue tests (Kefu Chai)
- common: add descriptions to perfcounters (Kiseleva Alyona)
- common: fix LTTNG vs fork issue (Josh Durgin)
- crush: fix has_v4_buckets (#11364 Sage Weil)
- crushtool: fix order of operations, usage (Sage Weil)
- debian: minor package reorg (Ken Dreyer)
- doc: docuemnt object corpus generation (#11099 Alexis Normand)
- doc: fix gender neutrality (Alexandre Maragone)
- doc: fix install doc (#10957 Kefu Chai)
- doc: fix sphinx issues (Kefu Chai)
- doc: mds data structure docs (Yan, Zheng)
- doc: misc updates (Nilamdyuti Goswami, Vartika Rai, Florian Haas, Loic Dachary, Simon Guinot, Andy Allan, Alistair Israel, Ken Dreyer, Robin Rehu, Lee Revell, Florian Marsylle, Thomas Johnson, Bosse Klykken, Travis Rhoden, Ian Kelling)
- doc: swift tempurls (#10184 Abhishek Lekshmanan)
- doc: switch doxygen integration back to breathe (#6115 Kefu Chai)
- erasure-code: update ISA-L to 2.13 (Yuan Zhou)
- gmock: switch to submodule (Danny Al-Gaaf, Loic Dachary)
- hadoop: add terasort test (Noah Watkins)
- java: fix libcephfs bindings (Noah Watkins)
- libcephfs,ceph-fuse: fix request resend on cap reconnect (#10912 Yan, Zheng)
- librados: define C++ flags from C constants (Josh Durgin)
- librados: fix last_force_resent handling (#11026 Jianpeng Ma)
- librados: fix memory leak from C_TwoContexts (Xiong Yiliang)
- librados: fix striper when stripe_count = 1 and stripe_unit != object_size (#11120 Yan, Zheng)
- librados: op perf counters (John Spray)
- librados: pybind: fix write() method return code (Javier Guerra)
- libradosstriper: fix leak (Danny Al-Gaaf)
- librbd: add purge_on_error cache behavior (Jianpeng Ma)
- librbd: misc aio fixes (#5488 Jason Dillaman)
- librbd: misc rbd fixes (#11478 #11113 #11342 #11380 Jason Dillaman, Zhiqiang Wang)
- librbd: readahead fixes (Zhiqiang Wang)
- librbd: store metadata, including config options, in image (Haomai Wang)
- mds: add ‘damaged’ state to MDSMap (John Spray)

- mds: add nicknames for perfcounters (John Spray)
- mds: disable problematic rstat propagation into snap parents (Yan, Zheng)
- mds: fix mydir replica issue with shutdown (#10743 John Spray)
- mds: fix out-of-order messages (#11258 Yan, Zheng)
- mds: fix shutdown with strays (#10744 John Spray)
- mds: fix snapshot fixes (Yan, Zheng)
- mds: fix stray handling (John Spray)
- mds: flush immediately in do_open_truncate (#11011 John Spray)
- mds: improve dump methods (John Spray)
- mds: misc journal cleanups and fixes (#10368 John Spray)
- mds: new SessionMap storage using omap (#10649 John Spray)
- mds: reduce memory consumption (Yan, Zheng)
- mds: throttle purge stray operations (#10390 John Spray)
- mds: tolerate clock jumping backwards (#11053 Yan, Zheng)
- misc coverity fixes (Danny Al-Gaaf)
- mon: do not deactivate last mds (#10862 John Spray)
- mon: make osd get pool 'all' only return applicable fields (#10891 Michal Jarzabek)
- mon: warn on bogus cache tier config (Jianpeng Ma)
- msg/async: misc bug fixes and updates (Haomai Wang)
- msg/simple: fix connect_seq assert (Haomai Wang)
- msg/xio: misc fixes (#10735 Matt Benjamin, Kefu Chai, Danny Al-Gaaf, Raju Kurunkad, Vu Pham)
- msg: unit tests (Haomai Wang)
- objectcacher: misc bug fixes (Jianpeng Ma)
- os/filestore: enlarge getxattr buffer size (Jianpeng Ma)
- osd: EIO injection (David Zhang)
- osd: add misc perfcounters (Xinze Chi)
- osd: add simple sleep injection in recovery (Sage Weil)
- osd: allow SEEK_HOLE/SEEK_DATA for sparse read (Zhiqiang Wang)
- osd: avoid dup omap sets for in pg metadata (Sage Weil)
- osd: clean up some constness, privateness (Kefu Chai)
- osd: erasure-code: drop entries according to LRU (Andreas-Joachim Peters)
- osd: fix negative degraded stats during backfill (Guang Yang)
- osd: misc fixes (Ning Yao, Kefu Chai, Xinze Chi, Zhiqiang Wang, Jianpeng Ma)
- pybind: pep8 cleanups (Danny Al-Gaaf)
- qa: fix filelock_interrupt.py test (Yan, Zheng)
- qa: improve ceph-disk tests (Loic Dachary)

- qa: improve docker build layers (Loic Dachary)
- rados: translate erno to string in CLI (#10877 Kefu Chai)
- rbd: accept map options config option (Ilya Dryomov)
- rbd: cli: fix arg parsing with –io-pattern (Dmitry Yatsushkevich)
- rbd: fix error messages (#2862 Rajesh Nambiar)
- rbd: update rbd man page (Ilya Dryomov)
- rbd: update xfstests tests (Douglas Fuller)
- rgw: add X-Timestamp for Swift containers (#10938 Radoslaw Zarzynski)
- rgw: add missing headers to Swift container details (#10666 Ahmad Faheem, Dmytro Iurchenko)
- rgw: add stats to headers for account GET (#10684 Yuan Zhou)
- rgw: do not prefecth data for HEAD requests (Guang Yang)
- rgw: don't clobber bucket/object owner when setting ACLs (#10978 Yehuda Sadeh)
- rgw: don't use rgw_socket_path if frontend is configured (#11160 Yehuda Sadeh)
- rgw: enforce Content-Lenth for POST on Swift cont/obj (#10661 Radoslaw Zarzynski)
- rgw: fix handling empty metadata items on Swift container (#11088 Radoslaw Zarzynski)
- rgw: fix log rotation (Wuxingyi)
- rgw: generate Date header for civetweb (#10873 Radoslaw Zarzynski)
- rgw: make init script wait for radosgw to stop (#11140 Dmitry Yatsushkevich)
- rgw: make quota/gc threads configurable (#11047 Guang Yang)
- rgw: pass in civetweb configurables (#10907 Yehuda Sadeh)
- rgw: rectify 202 Accepted in PUT response (#11148 Radoslaw Zarzynski)
- rgw: remove meta file after deleting bucket (#11149 Orit Wasserman)
- rgw: swift: allow setting attributes with COPY (#10662 Ahmad Faheem, Dmytro Iurchenko)
- rgw: swift: fix metadata handling on copy (#10645 Radoslaw Zarzynski)
- rgw: swift: send Last-Modified header (#10650 Radoslaw Zarzynski)
- rgw: update keystone cache with token info (#11125 Yehuda Sadeh)
- rgw: update to latest civetweb, enable config for IPv6 (#10965 Yehuda Sadeh)
- rocksdb: update to latest (Xiaoxi Chen)
- rpm: loosen ceph-test dependencies (Ken Dreyer)

11.2 v0.94.1 Hammer

This bug fix release fixes a few critical issues with CRUSH. The most important addresses a bug in feature bit enforcement that may prevent pre-hammer clients from communicating with the cluster during an upgrade. This only manifests in some cases (for example, when the 'rack' type is in use in the CRUSH map, and possibly other cases), but for safety we strongly recommend that all users use 0.94.1 instead of 0.94 when upgrading.

There is also a fix in the new straw2 buckets when OSD weights are 0.

We recommend that all v0.94 users upgrade.

11.2.1 Notable changes

- crush: fix divide-by-0 in straw2 (#11357 Sage Weil)
- crush: fix has_v4_buckets (#11364 Sage Weil)
- osd: fix negative degraded objects during backfilling (#7737 Guang Yang)

For more detailed information, see `the complete changelog`.

11.3 v0.94 Hammer

This major release is expected to form the basis of the next long-term stable series. It is intended to supersede v0.80.x Firefly.

Highlights since Giant include:

- *RADOS Performance*: a range of improvements have been made in the OSD and client-side librados code that improve the throughput on flash backends and improve parallelism and scaling on fast machines.
- *Simplified RGW deployment*: the ceph-deploy tool now has a new 'ceph-deploy rgw create HOST' command that quickly deploys a instance of the S3/Swift gateway using the embedded Civetweb server. This is vastly simpler than the previous Apache-based deployment. There are a few rough edges (e.g., around SSL support) but we encourage users to try the new method.
- *RGW object versioning*: RGW now supports the S3 object versioning API, which preserves old version of objects instead of overwriting them.
- *RGW bucket sharding*: RGW can now shard the bucket index for large buckets across, improving performance for very large buckets.
- *RBD object maps*: RBD now has an object map function that tracks which parts of the image are allocating, improving performance for clones and for commands like export and delete.
- *RBD mandatory locking*: RBD has a new mandatory locking framework (still disabled by default) that adds additional safeguards to prevent multiple clients from using the same image at the same time.
- *RBD copy-on-read*: RBD now supports copy-on-read for image clones, improving performance for some workloads.
- *CephFS snapshot improvements*: Many many bugs have been fixed with CephFS snapshots. Although they are still disabled by default, stability has improved significantly.
- *CephFS Recovery tools*: We have built some journal recovery and diagnostic tools. Stability and performance of single-MDS systems is vastly improved in Giant, and more improvements have been made now in Hammer. Although we still recommend caution when storing important data in CephFS, we do encourage testing for non-critical workloads so that we can better guage the feature, usability, performance, and stability gaps.
- *CRUSH improvements*: We have added a new straw2 bucket algorithm that reduces the amount of data migration required when changes are made to the cluster.
- *Shingled erasure codes (SHEC)*: The OSDs now have experimental support for shingled erasure codes, which allow a small amount of additional storage to be traded for improved recovery performance.
- *RADOS cache tiering*: A series of changes have been made in the cache tiering code that improve performance and reduce latency.
- *RDMA support*: There is now experimental support the RDMA via the Accelio (libxio) library.

- *New administrator commands*: The 'ceph osd df' command shows pertinent details on OSD disk utilizations. The 'ceph pg ls ...' command makes it much simpler to query PG states while diagnosing cluster issues.

Other highlights since Firefly include:

- *CephFS*: we have fixed a raft of bugs in CephFS and built some basic journal recovery and diagnostic tools. Stability and performance of single-MDS systems is vastly improved in Giant. Although we do not yet recommend CephFS for production deployments, we do encourage testing for non-critical workloads so that we can better guage the feature, usability, performance, and stability gaps.
- *Local Recovery Codes*: the OSDs now support an erasure-coding scheme that stores some additional data blocks to reduce the IO required to recover from single OSD failures.
- *Degraded vs misplaced*: the Ceph health reports from 'ceph -s' and related commands now make a distinction between data that is degraded (there are fewer than the desired number of copies) and data that is misplaced (stored in the wrong location in the cluster). The distinction is important because the latter does not compromise data safety.
- *Tiering improvements*: we have made several improvements to the cache tiering implementation that improve performance. Most notably, objects are not promoted into the cache tier by a single read; they must be found to be sufficiently hot before that happens.
- *Monitor performance*: the monitors now perform writes to the local data store asynchronously, improving overall responsiveness.
- *Recovery tools*: the ceph-objectstore-tool is greatly expanded to allow manipulation of an individual OSDs data store for debugging and repair purposes. This is most heavily used by our QA infrastructure to exercise recovery code.

I would like to take this opportunity to call out the amazing growth in contributors to Ceph beyond the core development team from Inktank. Hammer features major new features and improvements from Intel, Fujitsu, UnitedStack, Yahoo, UbuntuKylin, CohortFS, Mellanox, CERN, Deutsche Telekom, Mirantis, and SanDisk.

11.3.1 Dedication

This release is dedicated in memoriam to Sandon Van Ness, aka Houkouonchi, who unexpectedly passed away a few weeks ago. Sandon was responsible for maintaining the large and complex Sepia lab that houses the Ceph project's build and test infrastructure. His efforts have made an important impact on our ability to reliably test Ceph with a relatively small group of people. He was a valued member of the team and we will miss him. H is also for Houkouonchi.

11.3.2 Upgrading

- If your existing cluster is running a version older than v0.80.x Firefly, please first upgrade to the latest Firefly release before moving on to Giant. We have not tested upgrades directly from Emperor, Dumpling, or older releases.

  We *have* tested:

  - Firefly to Hammer
  - Giant to Hammer
  - Dumpling to Firefly to Hammer

- Please upgrade daemons in the following order:

  1. Monitors
  2. OSDs

3. MDSs and/or radosgw

Note that the relative ordering of OSDs and monitors should not matter, but we primarily tested upgrading monitors first.

- The ceph-osd daemons will perform a disk-format upgrade improve the PG metadata layout and to repair a minor bug in the on-disk format. It may take a minute or two for this to complete, depending on how many objects are stored on the node; do not be alarmed if they do not marked "up" by the cluster immediately after starting.
- **If upgrading from v0.93, set** osd enable degraded writes = false

  on all osds prior to upgrading. The degraded writes feature has been reverted due to 11155.
- The LTTNG tracing in librbd and librados is disabled in the release packages until we find a way to avoid violating distro security policies when linking libust.

11.3.3 Upgrading from v0.80.x Giant

- librbd and librados include lttng tracepoints on distros with liblttng 2.4 or later (only Ubuntu Trusty for the ceph.com packages). When running a daemon that uses these libraries, i.e. an application that calls fork(2) or clone(2) without exec(3), you must set LD_PRELOAD=liblttng-ust-fork.so.0 to prevent a crash in the lttng atexit handler when the process exits. The only ceph tool that requires this is rbd-fuse.
- If rgw_socket_path is defined and rgw_frontends defines a socket_port and socket_host, we now allow the rgw_frontends settings to take precedence. This change should only affect users who have made non-standard changes to their radosgw configuration.
- If you are upgrading specifically from v0.92, you must stop all OSD daemons and flush their journals (`ceph-osd -i NNN --flush-journal`) before upgrading. There was a transaction encoding bug in v0.92 that broke compatibility. Upgrading from v0.93, v0.91, or anything earlier is safe.
- The experimental 'keyvaluestore-dev' OSD backend has been renamed 'keyvaluestore' (for simplicity) and marked as experimental. To enable this untested feature and acknowledge that you understand that it is untested and may destroy data, you need to add the following to your ceph.conf:

```
enable experimental unrecoverable data corrupting featuers = keyvaluestore
```

- The following librados C API function calls take a 'flags' argument whose value is now correctly interpreted:

  rados_write_op_operate() rados_aio_write_op_operate() rados_read_op_operate() rados_aio_read_op_operate()

  The flags were not correctly being translated from the librados constants to the internal values. Now they are. Any code that is passing flags to these methods should be audited to ensure that they are using the correct LIBRADOS_OP_FLAG_* constants.
- The 'rados' CLI 'copy' and 'cppool' commands now use the copy-from operation, which means the latest CLI cannot run these commands against pre-firefly OSDs.
- The librados watch/notify API now includes a watch_flush() operation to flush the async queue of notify operations. This should be called by any watch/notify user prior to rados_shutdown().
- The 'category' field for objects has been removed. This was originally added to track PG stat summations over different categories of objects for use by radosgw. It is no longer has any known users and is prone to abuse because it can lead to a pg_stat_t structure that is unbounded. The librados API calls that accept this field now ignore it, and the OSD no longers tracks the per-category summations.
- The output for 'rados df' has changed. The 'category' level has been eliminated, so there is now a single stat object per pool. The structure of the JSON output is different, and the plaintext output has one less column.

- The ‘rados create <objectname> [category]’ optional category argument is no longer supported or recognized.
- rados.py’s Rados class no longer has a __del__ method; it was causing problems on interpreter shutdown and use of threads. If your code has Rados objects with limited lifetimes and you’re concerned about locked resources, call Rados.shutdown() explicitly.
- There is a new version of the librados watch/notify API with vastly improved semantics. Any applications using this interface are encouraged to migrate to the new API. The old API calls are marked as deprecated and will eventually be removed.
- The librados rados_unwatch() call used to be safe to call on an invalid handle. The new version has undefined behavior when passed a bogus value (for example, when rados_watch() returns an error and handle is not defined).
- The structure of the formatted ‘pg stat’ command is changed for the portion that counts states by name to avoid using the ‘+’ character (which appears in state names) as part of the XML token (it is not legal).
- Previously, the formatted output of ‘ceph pg stat -f ...’ was a full pg dump that included all metadata about all PGs in the system. It is now a concise summary of high-level PG stats, just like the unformatted ‘ceph pg stat’ command.
- All JSON dumps of floating point values were incorrecting surrounding the value with quotes. These quotes have been removed. Any consumer of structured JSON output that was consuming the floating point values was previously having to interpret the quoted string and will most likely need to be fixed to take the unquoted number.
- New ability to list all objects from all namespaces can fail or return incomplete results when not all OSDs have been upgraded. Features rados –all ls, rados cppool, rados export, rados cache-flush-evict-all and rados cache-try-flush-evict-all can also fail or return incomplete results.
- Due to a change in the Linux kernel version 3.18 and the limits of the FUSE interface, ceph-fuse needs be mounted as root on at least some systems. See issues #9997, #10277, and #10542 for details.

11.3.4 Upgrading from v0.80x Firefly (additional notes)

- The client-side caching for librbd is now enabled by default (rbd cache = true). A safety option (rbd cache writethrough until flush = true) is also enabled so that writeback caching is not used until the library observes a ‘flush’ command, indicating that the librbd users is passing that operation through from the guest VM. This avoids potential data loss when used with older versions of qemu that do not support flush.

  leveldb_write_buffer_size = 8*1024*1024 = 33554432 // 8MB leveldb_cache_size = 512*1024*1204 = 536870912 // 512MB leveldb_block_size = 64*1024 = 65536 // 64KB leveldb_compression = false leveldb_log = “”

  OSDs will still maintain the following osd-specific defaults:

  leveldb_log = “”
- The ‘rados getxattr ...’ command used to add a gratuitous newline to the attr value; it now does not.
- The `*_kb perf` counters on the monitor have been removed. These are replaced with a new set of `*_bytes` counters (e.g., `cluster_osd_kb` is replaced by `cluster_osd_bytes`).
- The `rd_kb` and `wr_kb` fields in the JSON dumps for pool stats (accessed via the `ceph df detail -f json-pretty` and related commands) have been replaced with corresponding `*_bytes` fields. Similarly, the `total_space`, `total_used`, and `total_avail` fields are replaced with `total_bytes`, `total_used_bytes`, and `total_avail_bytes` fields.
- The `rados df --format=json` output `read_bytes` and `write_bytes` fields were incorrectly reporting ops; this is now fixed.

- The `rados df --format=json` output previously included `read_kb` and `write_kb` fields; these have been removed. Please use `read_bytes` and `write_bytes` instead (and divide by 1024 if appropriate).
- The experimental keyvaluestore-dev OSD backend had an on-disk format change that prevents existing OSD data from being upgraded. This affects developers and testers only.
- mon-specific and osd-specific leveldb options have been removed. From this point onward users should use the *leveldb_** generic options and add the options in the appropriate sections of their configuration files. Monitors will still maintain the following monitor-specific defaults:

  > leveldb_write_buffer_size = 8*1024*1024 = 33554432 // 8MB leveldb_cache_size = 512*1024*1204 = 536870912 // 512MB leveldb_block_size = 64*1024 = 65536 // 64KB leveldb_compression = false leveldb_log = ""

  OSDs will still maintain the following osd-specific defaults:

  > leveldb_log = ""

- CephFS support for the legacy anchor table has finally been removed. Users with file systems created before firefly should ensure that inodes with multiple hard links are modified *prior* to the upgrade to ensure that the backtraces are written properly. For example:

```
sudo find /mnt/cephfs -type f -links +1 -exec touch \{\} \;
```

- We disallow nonsensical 'tier cache-mode' transitions. From this point onward, 'writeback' can only transition to 'forward' and 'forward' can transition to 1) 'writeback' if there are dirty objects, or 2) any if there are no dirty objects.

11.3.5 Notable changes since v0.93

- build: a few cmake fixes (Matt Benjamin)
- build: fix build on RHEL/CentOS 5.9 (Rohan Mars)
- build: reorganize Makefile to allow modular builds (Boris Ranto)
- ceph-fuse: be more forgiving on remount (#10982 Greg Farnum)
- ceph: improve CLI parsing (#11093 David Zafman)
- common: fix cluster logging to default channel (#11177 Sage Weil)
- crush: fix parsing of straw2 buckets (#11015 Sage Weil)
- doc: update man pages (David Zafman)
- librados: fix leak in C_TwoContexts (Xiong Yiliang)
- librados: fix leak in watch/notify path (Sage Weil)
- librbd: fix and improve AIO cache invalidation (#10958 Jason Dillaman)
- librbd: fix memory leak (Jason Dillaman)
- librbd: fix ordering/queueing of resize operations (Jason Dillaman)
- librbd: validate image is r/w on resize/flatten (Jason Dillaman)
- librbd: various internal locking fixes (Jason Dillaman)
- lttng: tracing is disabled until we streamline dependencies (Josh Durgin)
- mon: add bootstrap-rgw profile (Sage Weil)
- mon: do not pollute mon dir with CSV files from CRUSH check (Loic Dachary)

- mon: fix clock drift time check interval (#10546 Joao Eduardo Luis)
- mon: fix units in store stats (Joao Eduardo Luis)
- mon: improve error handling on erasure code profile set (#10488, #11144 Loic Dachary)
- mon: set {read,write}_tier on 'osd tier add-cache ...' (Jianpeng Ma)
- ms: xio: fix misc bugs (Matt Benjamin, Vu Pham)
- osd: DBObjectMap: fix locking to prevent rare crash (#9891 Samuel Just)
- osd: fix and document last_epoch_started semantics (Samuel Just)
- osd: fix divergent entry handling on PG split (Samuel Just)
- osd: fix leak on shutdown (Kefu Chai)
- osd: fix recording of digest on scrub (Samuel Just)
- osd: fix whiteout handling (Sage Weil)
- rbd: allow v2 striping parameters for clones and imports (Jason Dillaman)
- rbd: fix formatted output of image features (Jason Dillaman)
- rbd: updat eman page (Ilya Dryomov)
- rgw: don't overwrite bucket/object owner when setting ACLs (#10978 Yehuda Sadeh)
- rgw: enable IPv6 for civetweb (#10965 Yehuda Sadeh)
- rgw: fix sysvinit script when rgw_socket_path is not defined (#11159 Yehuda Sadeh, Dan Mick)
- rgw: pass civetweb configurables through (#10907 Yehuda Sadeh)
- rgw: use new watch/notify API (Yehuda Sadeh, Sage Weil)
- osd: reverted degraded writes feature due to 11155

11.3.6 Notable changes since v0.87.x Giant

- add experimental features option (Sage Weil)
- arch: fix NEON feaeture detection (#10185 Loic Dachary)
- asyncmsgr: misc fixes (Haomai Wang)
- buffer: add 'shareable' construct (Matt Benjamin)
- buffer: add list::get_contiguous (Sage Weil)
- buffer: avoid rebuild if buffer already contiguous (Jianpeng Ma)
- build: CMake support (Ali Maredia, Casey Bodley, Adam Emerson, Marcus Watts, Matt Benjamin)
- build: a few cmake fixes (Matt Benjamin)
- build: aarch64 build fixes (Noah Watkins, Haomai Wang)
- build: adjust build deps for yasm, virtualenv (Jianpeng Ma)
- build: fix 'make check' races (#10384 Loic Dachary)
- build: fix build on RHEL/CentOS 5.9 (Rohan Mars)
- build: fix pkg names when libkeyutils is missing (Pankag Garg, Ken Dreyer)
- build: improve build dependency tooling (Loic Dachary)

- build: reorganize Makefile to allow modular builds (Boris Ranto)
- build: support for jemalloc (Shishir Gowda)
- ceph-disk: Scientific Linux support (Dan van der Ster)
- ceph-disk: allow journal partition re-use (#10146 Loic Dachary, Dav van der Ster)
- ceph-disk: call partx/partprobe consistency (#9721 Loic Dachary)
- ceph-disk: do not re-use partition if encryption is required (Loic Dachary)
- ceph-disk: fix dmcrypt key permissions (Loic Dachary)
- ceph-disk: fix umount race condition (#10096 Blaine Gardner)
- ceph-disk: improved systemd support (Owen Synge)
- ceph-disk: init=none option (Loic Dachary)
- ceph-disk: misc fixes (Christos Stavrakakis)
- ceph-disk: respect –statedir for keyring (Loic Dachary)
- ceph-disk: set guid if reusing journal partition (Dan van der Ster)
- ceph-disk: support LUKS for encrypted partitions (Andrew Bartlett, Loic Dachary)
- ceph-fuse, libcephfs: POSIX file lock support (Yan, Zheng)
- ceph-fuse, libcephfs: allow xattr caps in inject_release_failure (#9800 John Spray)
- ceph-fuse, libcephfs: fix I_COMPLETE_ORDERED checks (#9894 Yan, Zheng)
- ceph-fuse, libcephfs: fix cap flush overflow (Greg Farnum, Yan, Zheng)
- ceph-fuse, libcephfs: fix root inode xattrs (Yan, Zheng)
- ceph-fuse, libcephfs: preserve dir ordering (#9178 Yan, Zheng)
- ceph-fuse, libcephfs: trim inodes before reconnecting to MDS (Yan, Zheng)
- ceph-fuse,libcephfs: add support for O_NOFOLLOW and O_PATH (Greg Farnum)
- ceph-fuse,libcephfs: resend requests before completing cap reconnect (#10912 Yan, Zheng)
- ceph-fuse: be more forgiving on remount (#10982 Greg Farnum)
- ceph-fuse: fix dentry invalidation on 3.18+ kernels (#9997 Yan, Zheng)
- ceph-fuse: fix kernel cache trimming (#10277 Yan, Zheng)
- ceph-fuse: select kernel cache invalidation mechanism based on kernel version (Greg Farnum)
- ceph-monstore-tool: fix shutdown (#10093 Loic Dachary)
- ceph-monstore-tool: fix/improve CLI (Joao Eduardo Luis)
- ceph-objectstore-tool: fix import (#10090 David Zafman)
- ceph-objectstore-tool: improved import (David Zafman)
- ceph-objectstore-tool: many improvements and tests (David Zafman)
- ceph-objectstore-tool: many many improvements (David Zafman)
- ceph-objectstore-tool: misc improvements, fixes (#9870 #9871 David Zafman)
- ceph.spec: package rbd-replay-prep (Ken Dreyer)
- ceph: add 'ceph osd df [tree]' command (#10452 Mykola Golub)

- ceph: do not parse injectargs twice (Loic Dachary)
- ceph: fix 'ceph tell ...' command validation (#10439 Joao Eduardo Luis)
- ceph: improve 'ceph osd tree' output (Mykola Golub)
- ceph: improve CLI parsing (#11093 David Zafman)
- ceph: make 'ceph -s' output more readable (Sage Weil)
- ceph: make 'ceph -s' show PG state counts in sorted order (Sage Weil)
- ceph: make 'ceph tell mon.* version' work (Mykola Golub)
- ceph: new 'ceph tell mds.$name_or_rank_or_gid' (John Spray)
- ceph: show primary-affinity in 'ceph osd tree' (Mykola Golub)
- ceph: test robustness (Joao Eduardo Luis)
- ceph_objectstore_tool: behave with sharded flag (#9661 David Zafman)
- cephfs-journal-tool: add recover_dentries function (#9883 John Spray)
- cephfs-journal-tool: fix journal import (#10025 John Spray)
- cephfs-journal-tool: skip up to expire_pos (#9977 John Spray)
- cleanup rados.h definitions with macros (Ilya Dryomov)
- common: add 'perf reset ...' admin command (Jianpeng Ma)
- common: add TableFormatter (Andreas Peters)
- common: add newline to flushed json output (Sage Weil)
- common: check syncfs() return code (Jianpeng Ma)
- common: do not unlock rwlock on destruction (Federico Simoncelli)
- common: filtering for 'perf dump' (John Spray)
- common: fix Formatter factory breakage (#10547 Loic Dachary)
- common: fix block device discard check (#10296 Sage Weil)
- common: make json-pretty output prettier (Sage Weil)
- common: remove broken CEPH_LOCKDEP optoin (Kefu Chai)
- common: shared_cache unit tests (Cheng Cheng)
- common: support new gperftools header locations (Key Dreyer)
- config: add $cctid meta variable (Adam Crume)
- crush: fix buffer overrun for poorly formed rules (#9492 Johnu George)
- crush: fix detach_bucket (#10095 Sage Weil)
- crush: fix parsing of straw2 buckets (#11015 Sage Weil)
- crush: fix several bugs in adjust_item_weight (Rongze Zhu)
- crush: fix tree bucket behavior (Rongze Zhu)
- crush: improve constness (Loic Dachary)
- crush: new and improved straw2 bucket type (Sage Weil, Christina Anderson, Xiaoxi Chen)
- crush: straw bucket weight calculation fixes (#9998 Sage Weil)

- crush: update tries stats for indep rules (#10349 Loic Dachary)
- crush: use larger choose_tries value for erasure code rulesets (#10353 Loic Dachary)
- crushtool: add –location <id> command (Sage Weil, Loic Dachary)
- debian,rpm: move RBD udev rules to ceph-common (#10864 Ken Dreyer)
- debian: split python-ceph into python-{rbd,rados,cephfs} (Boris Ranto)
- default to libnss instead of crypto++ (Federico Gimenez)
- doc: CephFS disaster recovery guidance (John Spray)
- doc: CephFS for early adopters (John Spray)
- doc: add build-doc guidlines for Fedora and CentOS/RHEL (Nilamdyuti Goswami)
- doc: add dumpling to firefly upgrade section (#7679 John Wilkins)
- doc: ceph osd reweight vs crush weight (Laurent Guerby)
- doc: do not suggest dangerous XFS nobarrier option (Dan van der Ster)
- doc: document erasure coded pool operations (#9970 Loic Dachary)
- doc: document the LRC per-layer plugin configuration (Yuan Zhou)
- doc: enable rbd cache on openstack deployments (Sebastien Han)
- doc: erasure code doc updates (Loic Dachary)
- doc: file system osd config settings (Kevin Dalley)
- doc: fix OpenStack Glance docs (#10478 Sebastien Han)
- doc: improved installation nots on CentOS/RHEL installs (John Wilkins)
- doc: key/value store config reference (John Wilkins)
- doc: misc cleanups (Adam Spiers, Sebastien Han, Nilamdyuti Goswami, Ken Dreyer, John Wilkins)
- doc: misc improvements (Nilamdyuti Goswami, John Wilkins, Chris Holcombe)
- doc: misc updates (#9793 #9922 #10204 #10203 Travis Rhoden, Hazem, Ayari, Florian Coste, Andy Allan, Frank Yu, Baptiste Veuillez-Mainard, Yuan Zhou, Armando Segnini, Robert Jansen, Tyler Brekke, Viktor Suprun)
- doc: misc updates (Alfredo Deza, VRan Liu)
- doc: misc updates (Nilamdyuti Goswami, John Wilkins)
- doc: new man pages (Nilamdyuti Goswami)
- doc: preflight doc fixes (John Wilkins)
- doc: replace cloudfiles with swiftclient Python Swift example (Tim Freund)
- doc: update PG count guide (Gerben Meijer, Laurent Guerby, Loic Dachary)
- doc: update man pages (David Zafman)
- doc: update openstack docs for Juno (Sebastien Han)
- doc: update release descriptions (Ken Dreyer)
- doc: update sepia hardware inventory (Sandon Van Ness)
- erasure-code: add mSHEC erasure code support (Takeshi Miyamae)
- erasure-code: improved docs (#10340 Loic Dachary)

- erasure-code: set max_size to 20 (#10363 Loic Dachary)
- fix cluster logging from non-mon daemons (Sage Weil)
- init-ceph: check for systemd-run before using it (Boris Ranto)
- install-deps.sh: do not require sudo when root (Loic Dachary)
- keyvaluestore: misc fixes (Haomai Wang)
- keyvaluestore: performance improvements (Haomai Wang)
- libcephfs,ceph-fuse: add 'status' asok (John Spray)
- libcephfs,ceph-fuse: fix getting zero-length xattr (#10552 Yan, Zheng)
- libcephfs: fix dirfrag trimming (#10387 Yan, Zheng)
- libcephfs: fix mount timeout (#10041 Yan, Zheng)
- libcephfs: fix test (#10415 Yan, Zheng)
- libcephfs: fix use-afer-free on umount (#10412 Yan, Zheng)
- libcephfs: include ceph and git version in client metadata (Sage Weil)
- librados, osd: new watch/notify implementation (Sage Weil)
- librados: add blacklist_add convenience method (Jason Dillaman)
- librados: add rados_pool_get_base_tier() call (Adam Crume)
- librados: add watch_flush() operation (Sage Weil, Haomai Wang)
- librados: avoid memcpy on getxattr, read (Jianpeng Ma)
- librados: cap buffer length (Loic Dachary)
- librados: create ioctx by pool id (Jason Dillaman)
- librados: do notify completion in fast-dispatch (Sage Weil)
- librados: drop 'category' feature (Sage Weil)
- librados: expose rados_{read|write}_op_assert_version in C API (Kim Vandry)
- librados: fix infinite loop with skipped map epochs (#9986 Ding Dinghua)
- librados: fix iterator operator= bugs (#10082 David Zafman, Yehuda Sadeh)
- librados: fix leak in C_TwoContexts (Xiong Yiliang)
- librados: fix leak in watch/notify path (Sage Weil)
- librados: fix null deref when pool DNE (#9944 Sage Weil)
- librados: fix objecter races (#9617 Josh Durgin)
- librados: fix pool deletion handling (#10372 Sage Weil)
- librados: fix pool name caching (#10458 Radoslaw Zarzynski)
- librados: fix resource leak, misc bugs (#10425 Radoslaw Zarzynski)
- librados: fix some watch/notify locking (Jason Dillaman, Josh Durgin)
- librados: fix timer race from recent refactor (Sage Weil)
- librados: new fadvise API (Ma Jianpeng)
- librados: only export public API symbols (Jason Dillaman)

- librados: remove shadowed variable (Kefu Chain)
- librados: translate op flags from C APIs (Matthew Richards)
- libradosstriper: fix remove() (Dongmao Zhang)
- libradosstriper: fix shutdown hang (Dongmao Zhang)
- libradosstriper: fix stat strtoll (Dongmao Zhang)
- libradosstriper: fix trunc method (#10129 Sebastien Ponce)
- libradosstriper: fix write_full when ENOENT (#10758 Sebastien Ponce)
- libradosstriper: misc fixes (Sebastien Ponce)
- librbd: CRC protection for RBD image map (Jason Dillaman)
- librbd: add missing python docstrings (Jason Dillaman)
- librbd: add per-image object map for improved performance (Jason Dillaman)
- librbd: add readahead (Adam Crume)
- librbd: add support for an "object map" indicating which objects exist (Jason Dillaman)
- librbd: adjust internal locking (Josh Durgin, Jason Dillaman)
- librbd: better handling of watch errors (Jason Dillaman)
- librbd: complete pending ops before closing image (#10299 Josh Durgin)
- librbd: coordinate maint operations through lock owner (Jason Dillaman)
- librbd: copy-on-read (Min Chen, Li Wang, Yunchuan Wen, Cheng Cheng, Jason Dillaman)
- librbd: differentiate between R/O vs R/W features (Jason Dillaman)
- librbd: don't close a closed parent in failure path (#10030 Jason Dillaman)
- librbd: enforce write ordering with a snapshot (Jason Dillaman)
- librbd: exclusive image locking (Jason Dillaman)
- librbd: fadvise API (Ma Jianpeng)
- librbd: fadvise-style hints; add misc hints for certain operations (Jianpeng Ma)
- librbd: fix and improve AIO cache invalidation (#10958 Jason Dillaman)
- librbd: fix cache tiers in list_children and snap_unprotect (Adam Crume)
- librbd: fix coverity false-positives (Jason Dillaman)
- librbd: fix diff test (#10002 Josh Durgin)
- librbd: fix list_children from invalid pool ioctxs (#10123 Jason Dillaman)
- librbd: fix locking for readahead (#10045 Jason Dillaman)
- librbd: fix memory leak (Jason Dillaman)
- librbd: fix ordering/queueing of resize operations (Jason Dillaman)
- librbd: fix performance regression in ObjectCacher (#9513 Adam Crume)
- librbd: fix snap create races (Jason Dillaman)
- librbd: fix write vs import race (#10590 Jason Dillaman)
- librbd: flush AIO operations asynchronously (#10714 Jason Dillaman)

- librbd: gracefully handle deleted/renamed pools (#10270 Jason Dillaman)
- librbd: lttng tracepoints (Adam Crume)
- librbd: make async versions of long-running maint operations (Jason Dillaman)
- librbd: misc fixes (Xinxin Shu, Jason Dillaman)
- librbd: mock tests (Jason Dillaman)
- librbd: only export public API symbols (Jason Dillaman)
- librbd: optionally blacklist clients before breaking locks (#10761 Jason Dillaman)
- librbd: prevent copyup during shrink (Jason Dillaman)
- librbd: refactor unit tests to use fixtures (Jason Dillaman)
- librbd: validate image is r/w on resize/flatten (Jason Dillaman)
- librbd: various internal locking fixes (Jason Dillaman)
- many coverity fixes (Danny Al-Gaaf)
- many many coverity cleanups (Danny Al-Gaaf)
- mds: 'flush journal' admin command (John Spray)
- mds: ENOSPC and OSDMap epoch barriers (#7317 John Spray)
- mds: a whole bunch of initial scrub infrastructure (Greg Farnum)
- mds: add cephfs-table-tool (John Spray)
- mds: asok command for fetching subtree map (John Spray)
- mds: avoid sending traceless replies in most cases (Yan, Zheng)
- mds: constify MDSCacheObjects (John Spray)
- mds: dirfrag buf fix (Yan, Zheng)
- mds: disallow most commands on inactive MDS's (Greg Farnum)
- mds: drop dentries, leases on deleted directories (#10164 Yan, Zheng)
- mds: export dir asok command (John Spray)
- mds: fix MDLog IO callback deadlock (John Spray)
- mds: fix compat_version for MClientSession (#9945 John Spray)
- mds: fix deadlock during journal probe vs purge (#10229 Yan, Zheng)
- mds: fix race trimming log segments (Yan, Zheng)
- mds: fix reply snapbl (Yan, Zheng)
- mds: fix sessionmap lifecycle bugs (Yan, Zheng)
- mds: fix stray/purge perfcounters (#10388 John Spray)
- mds: handle heartbeat_reset during shutdown (#10382 John Spray)
- mds: handle zero-size xattr (#10335 Yan, Zheng)
- mds: initialize root inode xattr version (Yan, Zheng)
- mds: introduce auth caps (John Spray)
- mds: many many snapshot-related fixes (Yan, Zheng)

- mds: misc bugs (Greg Farnum, John Spray, Yan, Zheng, Henry Change)
- mds: refactor, improve Session storage (John Spray)
- mds: store backtrace for stray dir (Yan, Zheng)
- mds: subtree quota support (Yunchuan Wen)
- mds: verify backtrace when fetching dirfrag (#9557 Yan, Zheng)
- memstore: free space tracking (John Spray)
- misc cleanup (Danny Al-Gaaf, David Anderson)
- misc coverity fixes (Danny Al-Gaaf)
- misc coverity fixes (Danny Al-Gaaf)
- misc: various valgrind fixes and cleanups (Danny Al-Gaaf)
- mon: 'osd crush reweight-all' command (Sage Weil)
- mon: add 'ceph osd rename-bucket ...' command (Loic Dachary)
- mon: add bootstrap-rgw profile (Sage Weil)
- mon: add max pgs per osd warning (Sage Weil)
- mon: add noforward flag for some mon commands (Mykola Golub)
- mon: allow adding tiers to fs pools (#10135 John Spray)
- mon: allow full flag to be manually cleared (#9323 Sage Weil)
- mon: clean up auth list output (Loic Dachary)
- mon: delay failure injection (Joao Eduardo Luis)
- mon: disallow empty pool names (#10555 Wido den Hollander)
- mon: do not deactivate last mds (#10862 John Spray)
- mon: do not pollute mon dir with CSV files from CRUSH check (Loic Dachary)
- mon: drop old ceph_mon_store_converter (Sage Weil)
- mon: fix 'ceph pg dump_stuck degraded' (Xinxin Shu)
- mon: fix 'mds fail' for standby MDSs (John Spray)
- mon: fix 'osd crush link' id resolution (John Spray)
- mon: fix 'profile osd' use of config-key function on mon (#10844 Joao Eduardo Luis)
- mon: fix *_ratio* units and types (Sage Weil)
- mon: fix JSON dumps to dump floats as flots and not strings (Sage Weil)
- mon: fix MDS health status from peons (#10151 John Spray)
- mon: fix caching for min_last_epoch_clean (#9987 Sage Weil)
- mon: fix clock drift time check interval (#10546 Joao Eduardo Luis)
- mon: fix compatset initalization during mkfs (Joao Eduardo Luis)
- mon: fix error output for add_data_pool (#9852 Joao Eduardo Luis)
- mon: fix feature tracking during elections (Joao Eduardo Luis)
- mon: fix formatter 'pg stat' command output (Sage Weil)

- mon: fix mds gid/rank/state parsing (John Spray)
- mon: fix misc error paths (Joao Eduardo Luis)
- mon: fix paxos off-by-one corner case (#9301 Sage Weil)
- mon: fix paxos timeouts (#10220 Joao Eduardo Luis)
- mon: fix stashed monmap encoding (#5203 Xie Rui)
- mon: fix units in store stats (Joao Eduardo Luis)
- mon: get canonical OSDMap from leader (#10422 Sage Weil)
- mon: ignore failure reports from before up_from (#10762 Dan van der Ster, Sage Weil)
- mon: implement 'fs reset' command (John Spray)
- mon: improve error handling on erasure code profile set (#10488, #11144 Loic Dachary)
- mon: improved corrupt CRUSH map detection (Joao Eduardo Luis)
- mon: include entity name in audit log for forwarded requests (#9913 Joao Eduardo Luis)
- mon: include pg_temp count in osdmap summary (Sage Weil)
- mon: log health summary to cluster log (#9440 Joao Eduardo Luis)
- mon: make 'mds fail' idempotent (John Spray)
- mon: make pg dump {sum,pgs,pgs_brief} work for format=plain (#5963 #6759 Mykola Golub)
- mon: new 'ceph pool ls [detail]' command (Sage Weil)
- mon: new pool safety flags nodelete, nopgchange, nosizechange (#9792 Mykola Golub)
- mon: new, friendly 'ceph pg ls ...' command (Xinxin Shu)
- mon: paxos: allow reads while proposing (#9321 #9322 Joao Eduardo Luis)
- mon: prevent MDS transition from STOPPING (#10791 Greg Farnum)
- mon: propose all pending work in one transaction (Sage Weil)
- mon: remove pg_temps for nonexistent pools (Joao Eduardo Luis)
- mon: require mon_allow_pool_delete option to remove pools (Sage Weil)
- mon: respect down flag when promoting standbys (John Spray)
- mon: set globalid prealloc to larger value (Sage Weil)
- mon: set {read,write}_tier on 'osd tier add-cache ...' (Jianpeng Ma)
- mon: skip zeroed osd stats in get_rule_avail (#10257 Joao Eduardo Luis)
- mon: validate min_size range (Jianpeng Ma)
- mon: wait for writeable before cross-proposing (#9794 Joao Eduardo Luis)
- mount.ceph: fix suprious error message (#10351 Yan, Zheng)
- ms: xio: fix misc bugs (Matt Benjamin, Vu Pham)
- msgr: async: bind threads to CPU cores, improved poll (Haomai Wang)
- msgr: async: many fixes, unit tests (Haomai Wang)
- msgr: async: several fixes (Haomai Wang)
- msgr: asyncmessenger: add kqueue support (#9926 Haomai Wang)

- msgr: avoid useless new/delete (Haomai Wang)
- msgr: fix RESETSESSION bug (#10080 Greg Farnum)
- msgr: fix crc configuration (Mykola Golub)
- msgr: fix delay injection bug (#9910 Sage Weil, Greg Farnum)
- msgr: misc unit tests (Haomai Wang)
- msgr: new AsymcMessenger alternative implementation (Haomai Wang)
- msgr: prefetch data when doing recv (Yehuda Sadeh)
- msgr: simple: fix rare deadlock (Greg Farnum)
- msgr: simple: retry binding to port on failure (#10029 Wido den Hollander)
- msgr: xio: XioMessenger RDMA support (Casey Bodley, Vu Pham, Matt Benjamin)
- objectstore: deprecate collection attrs (Sage Weil)
- osd, librados: fadvise-style librados hints (Jianpeng Ma)
- osd, librados: fix xattr_cmp_u64 (Dongmao Zhang)
- osd, librados: revamp PG listing API to handle namespaces (#9031 #9262 #9438 David Zafman)
- osd, mds: 'ops' as shorthand for 'dump_ops_in_flight' on asok (Sage Weil)
- osd, mon: add checksums to all OSDMaps (Sage Weil)
- osd, mon: send intiial pg create time from mon to osd (#9887 David Zafman)
- osd,mon: add 'norebalance' flag (Kefu Chai)
- osd,mon: specify OSD features explicitly in MOSDBoot (#10911 Sage Weil)
- osd: DBObjectMap: fix locking to prevent rare crash (#9891 Samuel Just)
- osd: EIO on whole-object reads when checksum is wrong (Sage Weil)
- osd: add erasure code corpus (Loic Dachary)
- osd: add fadvise flags to ObjectStore API (Jianpeng Ma)
- osd: add get_latest_osdmap asok command (#9483 #9484 Mykola Golub)
- osd: add misc tests (Loic Dachary, Danny Al-Gaaf)
- osd: add option to prioritize heartbeat network traffic (Jian Wen)
- osd: add support for the SHEC erasure-code algorithm (Takeshi Miyamae, Loic Dachary)
- osd: allow deletion of objects with watcher (#2339 Sage Weil)
- osd: allow recovery while below min_size (Samuel Just)
- osd: allow recovery with fewer than min_size OSDs (Samuel Just)
- osd: allow sparse read for Push/Pull (Haomai Wang)
- osd: allow whiteout deletion in cache pool (Sage Weil)
- osd: allow writes to degraded objects (Samuel Just)
- osd: allow writes to degraded objects (Samuel Just)
- osd: avoid publishing unchanged PG stats (Sage Weil)
- osd: batch pg log trim (Xinze Chi)

- osd: cache pool: ignore min flush age when cache is full (Xinze Chi)
- osd: cache recent ObjectContexts (Dong Yuan)
- osd: cache reverse_nibbles hash value (Dong Yuan)
- osd: clean up internal ObjectStore interface (Sage Weil)
- osd: cleanup boost optionals (William Kennington)
- osd: clear cache on interval change (Samuel Just)
- osd: do no proxy reads unless target OSDs are new (#10788 Sage Weil)
- osd: do not abort deep scrub on missing hinfo (#10018 Loic Dachary)
- osd: do not update digest on inconsistent object (#10524 Samuel Just)
- osd: don’t record digests for snapdirs (#10536 Samuel Just)
- osd: drop upgrade support for pre-dumpling (Sage Weil)
- osd: enable and use posix_fadvise (Sage Weil)
- osd: erasure coding: allow bench.sh to test ISA backend (Yuan Zhou)
- osd: erasure-code: encoding regression tests, corpus (#9420 Loic Dachary)
- osd: erasure-code: enforce chunk size alignment (#10211 Loic Dachary)
- osd: erasure-code: jerasure support for NEON (Loic Dachary)
- osd: erasure-code: relax cauchy w restrictions (#10325 David Zhang, Loic Dachary)
- osd: erasure-code: update gf-complete to latest upstream (Loic Dachary)
- osd: expose non-journal backends via ceph-osd CLI (Hoamai Wang)
- osd: filejournal: don’t cache journal when not using direct IO (Jianpeng Ma)
- osd: fix JSON output for stray OSDs (Loic Dachary)
- osd: fix OSDCap parser on old (el6) boost::spirit (#10757 Kefu Chai)
- osd: fix OSDCap parsing on el6 (#10757 Kefu Chai)
- osd: fix ObjectStore::Transaction encoding version (#10734 Samuel Just)
- osd: fix WBTHrottle perf counters (Haomai Wang)
- osd: fix and document last_epoch_started semantics (Samuel Just)
- osd: fix auth object selection during repair (#10524 Samuel Just)
- osd: fix backfill bug (#10150 Samuel Just)
- osd: fix bug in pending digest updates (#10840 Samuel Just)
- osd: fix cancel_proxy_read_ops (Sage Weil)
- osd: fix cleanup of interrupted pg deletion (#10617 Sage Weil)
- osd: fix divergent entry handling on PG split (Samuel Just)
- osd: fix ghobject_t formatted output to include shard (#10063 Loic Dachary)
- osd: fix ioprio option (Mykola Golub)
- osd: fix ioprio options (Loic Dachary)
- osd: fix journal shutdown race (Sage Weil)

- osd: fix journal wrapping bug (#10883 David Zafman)
- osd: fix leak in SnapTrimWQ (#10421 Kefu Chai)
- osd: fix leak on shutdown (Kefu Chai)
- osd: fix memstore free space calculation (Xiaoxi Chen)
- osd: fix mixed-version peering issues (Samuel Just)
- osd: fix object age eviction (Zhiqiang Wang)
- osd: fix object atime calculation (Xinze Chi)
- osd: fix object digest update bug (#10840 Samuel Just)
- osd: fix occasional peering stalls (#10431 Sage Weil)
- osd: fix ordering issue with new transaction encoding (#10534 Dong Yuan)
- osd: fix osd peer check on scrub messages (#9555 Sage Weil)
- osd: fix past_interval display bug (#9752 Loic Dachary)
- osd: fix past_interval generation (#10427 #10430 David Zafman)
- osd: fix pgls filter ops (#9439 David Zafman)
- osd: fix recording of digest on scrub (Samuel Just)
- osd: fix scrub delay bug (#10693 Samuel Just)
- osd: fix scrub vs try-flush bug (#8011 Samuel Just)
- osd: fix short read handling on push (#8121 David Zafman)
- osd: fix stderr with -f or -d (Dan Mick)
- osd: fix transaction accounting (Jianpeng Ma)
- osd: fix watch reconnect race (#10441 Sage Weil)
- osd: fix watch timeout cache state update (#10784 David Zafman)
- osd: fix whiteout handling (Sage Weil)
- osd: flush snapshots from cache tier immediately (Sage Weil)
- osd: force promotion of watch/notify ops (Zhiqiang Wang)
- osd: handle no-op write with snapshot (#10262 Sage Weil)
- osd: improve idempotency detection across cache promotion/demotion (#8935 Sage Weil, Samuel Just)
- osd: include activating peers in blocked_by (#10477 Sage Weil)
- osd: jerasure and gf-complete updates from upstream (#10216 Loic Dachary)
- osd: journal: check fsync/fdatasync result (Jianpeng Ma)
- osd: journal: fix alignment checks, avoid useless memmove (Jianpeng Ma)
- osd: journal: fix hang on shutdown (#10474 David Zafman)
- osd: journal: fix header.committed_up_to (Xinze Chi)
- osd: journal: fix journal zeroing when direct IO is enabled (Xie Rui)
- osd: journal: initialize throttle (Ning Yao)
- osd: journal: misc bug fixes (#6003 David Zafman, Samuel Just)

- osd: journal: update committed_thru after replay (#6756 Samuel Just)
- osd: keyvaluestore: cleanup dead code (Ning Yao)
- osd: keyvaluestore: fix getattr semantics (Haomai Wang)
- osd: keyvaluestore: fix key ordering (#10119 Haomai Wang)
- osd: keyvaluestore_dev: optimization (Chendi Xue)
- osd: limit in-flight read requests (Jason Dillaman)
- osd: log when scrub or repair starts (Loic Dachary)
- osd: make misdirected op checks robust for EC pools (#9835 Sage Weil)
- osd: memstore: fix size limit (Xiaoxi Chen)
- osd: misc FIEMAP fixes (Ma Jianpeng)
- osd: misc cleanup (Xinze Chi, Yongyue Sun)
- osd: misc optimizations (Xinxin Shu, Zhiqiang Wang, Xinze Chi)
- osd: misc scrub fixes (#10017 Loic Dachary)
- osd: new 'activating' state between peering and active (Sage Weil)
- osd: new optimized encoding for ObjectStore::Transaction (Dong Yuan)
- osd: optimize Finisher (Xinze Chi)
- osd: optimize WBThrottle map with unordered_map (Ning Yao)
- osd: optimize filter_snapc (Ning Yao)
- osd: preserve reqids for idempotency checks for promote/demote (Sage Weil, Zhiqiang Wang, Samuel Just)
- osd: proxy read support (Zhiqiang Wang)
- osd: proxy reads during cache promote (Zhiqiang Wang)
- osd: remove dead locking code (Xinxin Shu)
- osd: remove legacy classic scrub code (Sage Weil)
- osd: remove unused fields in MOSDSubOp (Xiaoxi Chen)
- osd: removed some dead code (Xinze Chi)
- osd: replace MOSDSubOp messages with simpler, optimized MOSDRepOp (Xiaoxi Chen)
- osd: restrict scrub to certain times of day (Xinze Chi)
- osd: rocksdb: fix shutdown (Hoamai Wang)
- osd: store PG metadata in per-collection objects for better concurrency (Sage Weil)
- osd: store whole-object checksums on scrub, write_full (Sage Weil)
- osd: support for discard for journal trim (Jianpeng Ma)
- osd: use FIEMAP_FLAGS_SYNC instead of fsync (Jianpeng Ma)
- osd: verify kernel is new enough before using XFS extsize ioctl, enable by default (#9956 Sage Weil)
- pybind: fix memory leak in librados bindings (Billy Olsen)
- pyrados: add object lock support (#6114 Mehdi Abaakouk)
- pyrados: fix misnamed wait_* routings (#10104 Dan Mick)

- pyrados: misc cleanups (Kefu Chai)
- qa: add large auth ticket tests (Ilya Dryomov)
- qa: fix mds tests (#10539 John Spray)
- qa: fix osd create dup tests (#10083 Loic Dachary)
- qa: ignore duplicates in rados ls (Josh Durgin)
- qa: improve hadoop tests (Noah Watkins)
- qa: many 'make check' improvements (Loic Dachary)
- qa: misc tests (Loic Dachary, Yan, Zheng)
- qa: parallelize make check (Loic Dachary)
- qa: reorg fs quota tests (Greg Farnum)
- qa: tolerate nearly-full disk for make check (Loic Dachary)
- rados: fix put of /dev/null (Loic Dachary)
- rados: fix usage (Jianpeng Ma)
- rados: parse command-line arguments more strictly (#8983 Adam Crume)
- rados: use copy-from operation for copy, cppool (Sage Weil)
- radosgw-admin: add replicalog update command (Yehuda Sadeh)
- rbd-fuse: clean up on shutdown (Josh Durgin)
- rbd-fuse: fix memory leak (Adam Crume)
- rbd-replay-many (Adam Crume)
- rbd-replay: –anonymize flag to rbd-replay-prep (Adam Crume)
- rbd: add 'merge-diff' function (MingXin Liu, Yunchuan Wen, Li Wang)
- rbd: allow v2 striping parameters for clones and imports (Jason Dillaman)
- rbd: fix 'rbd diff' for non-existent objects (Adam Crume)
- rbd: fix buffer handling on image import (#10590 Jason Dillaman)
- rbd: fix error when striping with format 1 (Sebastien Han)
- rbd: fix export for image sizes over 2GB (Vicente Cheng)
- rbd: fix formatted output of image features (Jason Dillaman)
- rbd: leave exclusive lockin goff by default (Jason Dillaman)
- rbd: updat eman page (Ilya Dryomov)
- rbd: update init-rbdmap to fix dup mount point (Karel Striegel)
- rbd: use IO hints for import, export, and bench operations (#10462 Jason Dillaman)
- rbd: use rolling average for rbd bench-write throughput (Jason Dillaman)
- rbd_recover_tool: RBD image recovery tool (Min Chen)
- rgw: S3-style object versioning support (Yehuda Sadeh)
- rgw: add location header when object is in another region (VRan Liu)
- rgw: change multipart upload id magic (#10271 Yehuda Sadeh)

- rgw: check keystone auth for S3 POST requests (#10062 Abhishek Lekshmanan)
- rgw: check timestamp on s3 keystone auth (#10062 Abhishek Lekshmanan)
- rgw: conditional PUT on ETag (#8562 Ray Lv)
- rgw: create subuser if needed when creating user (#10103 Yehuda Sadeh)
- rgw: decode http query params correction (#10271 Yehuda Sadeh)
- rgw: don't overwrite bucket/object owner when setting ACLs (#10978 Yehuda Sadeh)
- rgw: enable IPv6 for civetweb (#10965 Yehuda Sadeh)
- rgw: extend replica log API (purge-all) (Yehuda Sadeh)
- rgw: fail S3 POST if keystone not configured (#10688 Valery Tschopp, Yehuda Sadeh)
- rgw: fix If-Modified-Since (VRan Liu)
- rgw: fix XML header on get ACL request (#10106 Yehuda Sadeh)
- rgw: fix bucket removal with data purge (Yehuda Sadeh)
- rgw: fix content length check (#10701 Axel Dunkel, Yehuda Sadeh)
- rgw: fix content-length update (#9576 Yehuda Sadeh)
- rgw: fix disabling of max_size quota (#9907 Dong Lei)
- rgw: fix error codes (#10334 #10329 Yehuda Sadeh)
- rgw: fix incorrect len when len is 0 (#9877 Yehuda Sadeh)
- rgw: fix object copy content type (#9478 Yehuda Sadeh)
- rgw: fix partial GET in swift (#10553 Yehuda Sadeh)
- rgw: fix replica log indexing (#8251 Yehuda Sadeh)
- rgw: fix shutdown (#10472 Yehuda Sadeh)
- rgw: fix swift metadata header name (Dmytro Iurchenko)
- rgw: fix sysvinit script when rgw_socket_path is not defined (#11159 Yehuda Sadeh, Dan Mick)
- rgw: fix user stags in get-user-info API (#9359 Ray Lv)
- rgw: include XML ns on get ACL request (#10106 Yehuda Sadeh)
- rgw: index swift keys appropriately (#10471 Yehuda Sadeh)
- rgw: make sysvinit script set ulimit -n properly (Sage Weil)
- rgw: misc fixes (#10307 Yehuda Sadeh)
- rgw: only track cleanup for objects we write (#10311 Yehuda Sadeh)
- rgw: pass civetweb configurables through (#10907 Yehuda Sadeh)
- rgw: prevent illegal bucket policy that doesn't match placement rule (Yehuda Sadeh)
- rgw: remove multipart entries from bucket index on abort (#10719 Yehuda Sadeh)
- rgw: remove swift user manifest (DLO) hash calculation (#9973 Yehuda Sadeh)
- rgw: respond with 204 to POST on containers (#10667 Yuan Zhou)
- rgw: return timestamp on GET/HEAD (#8911 Yehuda Sadeh)
- rgw: reuse fcgx connection struct (#10194 Yehuda Sadeh)

- rgw: run radosgw as apache with systemd (#10125 Loic Dachary)
- rgw: send explicit HTTP status string (Yehuda Sadeh)
- rgw: set ETag on object copy (#9479 Yehuda Sadeh)
- rgw: set length for keystone token validation request (#7796 Yehuda Sadeh, Mark Kirkwood)
- rgw: support X-Storage-Policy header for Swift storage policy compat (Yehuda Sadeh)
- rgw: support multiple host names (#7467 Yehuda Sadeh)
- rgw: swift: dump container's custom metadata (#10665 Ahmad Faheem, Dmytro Iurchenko)
- rgw: swift: support Accept header for response format (#10746 Dmytro Iurchenko)
- rgw: swift: support for X-Remove-Container-Meta-{key} (#10475 Dmytro Iurchenko)
- rgw: tweak error codes (#10329 #10334 Yehuda Sadeh)
- rgw: update bucket index on attr changes, for multi-site sync (#5595 Yehuda Sadeh)
- rgw: use rn for http headers (#9254 Yehuda Sadeh)
- rgw: use gc for multipart abort (#10445 Aaron Bassett, Yehuda Sadeh)
- rgw: use new watch/notify API (Yehuda Sadeh, Sage Weil)
- rpm: misc fixes (Key Dreyer)
- rpm: move rgw logrotate to radosgw subpackage (Ken Dreyer)
- systemd: better systemd unit files (Owen Synge)
- sysvinit: fix race in 'stop' (#10389 Loic Dachary)
- test: fix bufferlist tests (Jianpeng Ma)
- tests: ability to run unit tests under docker (Loic Dachary)
- tests: centos-6 dockerfile (#10755 Loic Dachary)
- tests: improve docker-based tests (Loic Dachary)
- tests: unit tests for shared_cache (Dong Yuan)
- udev: fix rules for CentOS7/RHEL7 (Loic Dachary)
- use clock_gettime instead of gettimeofday (Jianpeng Ma)
- vstart.sh: set up environment for s3-tests (Luis Pabon)
- vstart.sh: work with cmake (Yehuda Sadeh)

11.4 v0.93

This is the first release candidate for Hammer, and includes all of the features that will be present in the final release. We welcome and encourage any and all testing in non-production clusters to identify any problems with functionality, stability, or performance before the final Hammer release.

We suggest some caution in one area: librbd. There is a lot of new functionality around object maps and locking that is disabled by default but may still affect stability for existing images. We are continuing to shake out those bugs so that the final Hammer release (probably v0.94) will be rock solid.

Major features since Giant include:

- cephfs: journal scavenger repair tool (John Spray)

- crush: new and improved straw2 bucket type (Sage Weil, Christina Anderson, Xiaoxi Chen)
- doc: improved guidance for CephFS early adopters (John Spray)
- librbd: add per-image object map for improved performance (Jason Dillaman)
- librbd: copy-on-read (Min Chen, Li Wang, Yunchuan Wen, Cheng Cheng)
- librados: fadvise-style IO hints (Jianpeng Ma)
- mds: many many snapshot-related fixes (Yan, Zheng)
- mon: new 'ceph osd df' command (Mykola Golub)
- mon: new 'ceph pg ls ...' command (Xinxin Shu)
- osd: improved performance for high-performance backends
- osd: improved recovery behavior (Samuel Just)
- osd: improved cache tier behavior with reads (Zhiqiang Wang)
- rgw: S3-compatible bucket versioning support (Yehuda Sadeh)
- rgw: large bucket index sharding (Guang Yang, Yehuda Sadeh)
- RDMA "xio" messenger support (Matt Benjamin, Vu Pham)

11.4.1 Upgrading

- If you are upgrading from v0.92, you must stop all OSD daemons and flush their journals (`ceph-osd -i NNN --flush-journal`) before upgrading. There was a transaction encoding bug in v0.92 that broke compatibility. Upgrading from v0.91 or anything earlier is safe.
- No special restrictions when upgrading from firefly or giant.

11.4.2 Notable Changes

- build: CMake support (Ali Maredia, Casey Bodley, Adam Emerson, Marcus Watts, Matt Benjamin)
- ceph-disk: do not re-use partition if encryption is required (Loic Dachary)
- ceph-disk: support LUKS for encrypted partitions (Andrew Bartlett, Loic Dachary)
- ceph-fuse,libcephfs: add support for O_NOFOLLOW and O_PATH (Greg Farnum)
- ceph-fuse,libcephfs: resend requests before completing cap reconnect (#10912 Yan, Zheng)
- ceph-fuse: select kernel cache invalidation mechanism based on kernel version (Greg Farnum)
- ceph-objectstore-tool: improved import (David Zafman)
- ceph-objectstore-tool: misc improvements, fixes (#9870 #9871 David Zafman)
- ceph: add 'ceph osd df [tree]' command (#10452 Mykola Golub)
- ceph: fix 'ceph tell ...' command validation (#10439 Joao Eduardo Luis)
- ceph: improve 'ceph osd tree' output (Mykola Golub)
- cephfs-journal-tool: add recover_dentries function (#9883 John Spray)
- common: add newline to flushed json output (Sage Weil)
- common: filtering for 'perf dump' (John Spray)

- common: fix Formatter factory breakage (#10547 Loic Dachary)
- common: make json-pretty output prettier (Sage Weil)
- crush: new and improved straw2 bucket type (Sage Weil, Christina Anderson, Xiaoxi Chen)
- crush: update tries stats for indep rules (#10349 Loic Dachary)
- crush: use larger choose_tries value for erasure code rulesets (#10353 Loic Dachary)
- debian,rpm: move RBD udev rules to ceph-common (#10864 Ken Dreyer)
- debian: split python-ceph into python-{rbd,rados,cephfs} (Boris Ranto)
- doc: CephFS disaster recovery guidance (John Spray)
- doc: CephFS for early adopters (John Spray)
- doc: fix OpenStack Glance docs (#10478 Sebastien Han)
- doc: misc updates (#9793 #9922 #10204 #10203 Travis Rhoden, Hazem, Ayari, Florian Coste, Andy Allan, Frank Yu, Baptiste Veuillez-Mainard, Yuan Zhou, Armando Segnini, Robert Jansen, Tyler Brekke, Viktor Suprun)
- doc: replace cloudfiles with swiftclient Python Swift example (Tim Freund)
- erasure-code: add mSHEC erasure code support (Takeshi Miyamae)
- erasure-code: improved docs (#10340 Loic Dachary)
- erasure-code: set max_size to 20 (#10363 Loic Dachary)
- libcephfs,ceph-fuse: fix getting zero-length xattr (#10552 Yan, Zheng)
- librados: add blacklist_add convenience method (Jason Dillaman)
- librados: expose rados_{read|write}_op_assert_version in C API (Kim Vandry)
- librados: fix pool name caching (#10458 Radoslaw Zarzynski)
- librados: fix resource leak, misc bugs (#10425 Radoslaw Zarzynski)
- librados: fix some watch/notify locking (Jason Dillaman, Josh Durgin)
- libradosstriper: fix write_full when ENOENT (#10758 Sebastien Ponce)
- librbd: CRC protection for RBD image map (Jason Dillaman)
- librbd: add per-image object map for improved performance (Jason Dillaman)
- librbd: add support for an “object map” indicating which objects exist (Jason Dillaman)
- librbd: adjust internal locking (Josh Durgin, Jason Dillaman)
- librbd: better handling of watch errors (Jason Dillaman)
- librbd: coordinate maint operations through lock owner (Jason Dillaman)
- librbd: copy-on-read (Min Chen, Li Wang, Yunchuan Wen, Cheng Cheng, Jason Dillaman)
- librbd: enforce write ordering with a snapshot (Jason Dillaman)
- librbd: fadvise-style hints; add misc hints for certain operations (Jianpeng Ma)
- librbd: fix coverity false-positives (Jason Dillaman)
- librbd: fix snap create races (Jason Dillaman)
- librbd: flush AIO operations asynchronously (#10714 Jason Dillaman)
- librbd: make async versions of long-running maint operations (Jason Dillaman)

- librbd: mock tests (Jason Dillaman)
- librbd: optionally blacklist clients before breaking locks (#10761 Jason Dillaman)
- librbd: prevent copyup during shrink (Jason Dillaman)
- mds: add cephfs-table-tool (John Spray)
- mds: avoid sending traceless replies in most cases (Yan, Zheng)
- mds: export dir asok command (John Spray)
- mds: fix stray/purge perfcounters (#10388 John Spray)
- mds: handle heartbeat_reset during shutdown (#10382 John Spray)
- mds: many many snapshot-related fixes (Yan, Zheng)
- mds: refactor, improve Session storage (John Spray)
- misc coverity fixes (Danny Al-Gaaf)
- mon: add noforward flag for some mon commands (Mykola Golub)
- mon: disallow empty pool names (#10555 Wido den Hollander)
- mon: do not deactivate last mds (#10862 John Spray)
- mon: drop old ceph_mon_store_converter (Sage Weil)
- mon: fix 'ceph pg dump_stuck degraded' (Xinxin Shu)
- mon: fix 'profile osd' use of config-key function on mon (#10844 Joao Eduardo Luis)
- mon: fix compatset initalization during mkfs (Joao Eduardo Luis)
- mon: fix feature tracking during elections (Joao Eduardo Luis)
- mon: fix mds gid/rank/state parsing (John Spray)
- mon: ignore failure reports from before up_from (#10762 Dan van der Ster, Sage Weil)
- mon: improved corrupt CRUSH map detection (Joao Eduardo Luis)
- mon: include pg_temp count in osdmap summary (Sage Weil)
- mon: log health summary to cluster log (#9440 Joao Eduardo Luis)
- mon: make 'mds fail' idempotent (John Spray)
- mon: make pg dump {sum,pgs,pgs_brief} work for format=plain (#5963 #6759 Mykola Golub)
- mon: new pool safety flags nodelete, nopgchange, nosizechange (#9792 Mykola Golub)
- mon: new, friendly 'ceph pg ls ...' command (Xinxin Shu)
- mon: prevent MDS transition from STOPPING (#10791 Greg Farnum)
- mon: propose all pending work in one transaction (Sage Weil)
- mon: remove pg_temps for nonexistent pools (Joao Eduardo Luis)
- mon: require mon_allow_pool_delete option to remove pools (Sage Weil)
- mon: set globalid prealloc to larger value (Sage Weil)
- mon: skip zeroed osd stats in get_rule_avail (#10257 Joao Eduardo Luis)
- mon: validate min_size range (Jianpeng Ma)
- msgr: async: bind threads to CPU cores, improved poll (Haomai Wang)

- msgr: fix crc configuration (Mykola Golub)
- msgr: misc unit tests (Haomai Wang)
- msgr: xio: XioMessenger RDMA support (Casey Bodley, Vu Pham, Matt Benjamin)
- osd, librados: fadvise-style librados hints (Jianpeng Ma)
- osd, librados: fix xattr_cmp_u64 (Dongmao Zhang)
- osd,mon: add 'norebalance' flag (Kefu Chai)
- osd,mon: specify OSD features explicitly in MOSDBoot (#10911 Sage Weil)
- osd: add option to prioritize heartbeat network traffic (Jian Wen)
- osd: add support for the SHEC erasure-code algorithm (Takeshi Miyamae, Loic Dachary)
- osd: allow recovery while below min_size (Samuel Just)
- osd: allow recovery with fewer than min_size OSDs (Samuel Just)
- osd: allow writes to degraded objects (Samuel Just)
- osd: allow writes to degraded objects (Samuel Just)
- osd: avoid publishing unchanged PG stats (Sage Weil)
- osd: cache recent ObjectContexts (Dong Yuan)
- osd: clear cache on interval change (Samuel Just)
- osd: do no proxy reads unless target OSDs are new (#10788 Sage Weil)
- osd: do not update digest on inconsistent object (#10524 Samuel Just)
- osd: don't record digests for snapdirs (#10536 Samuel Just)
- osd: fix OSDCap parser on old (el6) boost::spirit (#10757 Kefu Chai)
- osd: fix OSDCap parsing on el6 (#10757 Kefu Chai)
- osd: fix ObjectStore::Transaction encoding version (#10734 Samuel Just)
- osd: fix auth object selection during repair (#10524 Samuel Just)
- osd: fix bug in pending digest updates (#10840 Samuel Just)
- osd: fix cancel_proxy_read_ops (Sage Weil)
- osd: fix cleanup of interrupted pg deletion (#10617 Sage Weil)
- osd: fix journal wrapping bug (#10883 David Zafman)
- osd: fix leak in SnapTrimWQ (#10421 Kefu Chai)
- osd: fix memstore free space calculation (Xiaoxi Chen)
- osd: fix mixed-version peering issues (Samuel Just)
- osd: fix object digest update bug (#10840 Samuel Just)
- osd: fix ordering issue with new transaction encoding (#10534 Dong Yuan)
- osd: fix past_interval generation (#10427 #10430 David Zafman)
- osd: fix short read handling on push (#8121 David Zafman)
- osd: fix watch timeout cache state update (#10784 David Zafman)
- osd: force promotion of watch/notify ops (Zhiqiang Wang)

- osd: improve idempotency detection across cache promotion/demotion (#8935 Sage Weil, Samuel Just)
- osd: include activating peers in blocked_by (#10477 Sage Weil)
- osd: jerasure and gf-complete updates from upstream (#10216 Loic Dachary)
- osd: journal: check fsync/fdatasync result (Jianpeng Ma)
- osd: journal: fix hang on shutdown (#10474 David Zafman)
- osd: journal: fix header.committed_up_to (Xinze Chi)
- osd: journal: initialize throttle (Ning Yao)
- osd: journal: misc bug fixes (#6003 David Zafman, Samuel Just)
- osd: misc cleanup (Xinze Chi, Yongyue Sun)
- osd: new 'activating' state between peering and active (Sage Weil)
- osd: preserve reqids for idempotency checks for promote/demote (Sage Weil, Zhiqiang Wang, Samuel Just)
- osd: remove dead locking code (Xinxin Shu)
- osd: restrict scrub to certain times of day (Xinze Chi)
- osd: rocksdb: fix shutdown (Hoamai Wang)
- pybind: fix memory leak in librados bindings (Billy Olsen)
- qa: fix mds tests (#10539 John Spray)
- qa: ignore duplicates in rados ls (Josh Durgin)
- qa: improve hadoop tests (Noah Watkins)
- qa: reorg fs quota tests (Greg Farnum)
- rados: fix usage (Jianpeng Ma)
- radosgw-admin: add replicalog update command (Yehuda Sadeh)
- rbd-fuse: clean up on shutdown (Josh Durgin)
- rbd: add 'merge-diff' function (MingXin Liu, Yunchuan Wen, Li Wang)
- rbd: fix buffer handling on image import (#10590 Jason Dillaman)
- rbd: leave exclusive lockin goff by default (Jason Dillaman)
- rbd: update init-rbdmap to fix dup mount point (Karel Striegel)
- rbd: use IO hints for import, export, and bench operations (#10462 Jason Dillaman)
- rbd_recover_tool: RBD image recovery tool (Min Chen)
- rgw: S3-style object versioning support (Yehuda Sadeh)
- rgw: check keystone auth for S3 POST requests (#10062 Abhishek Lekshmanan)
- rgw: extend replica log API (purge-all) (Yehuda Sadeh)
- rgw: fail S3 POST if keystone not configured (#10688 Valery Tschopp, Yehuda Sadeh)
- rgw: fix XML header on get ACL request (#10106 Yehuda Sadeh)
- rgw: fix bucket removal with data purge (Yehuda Sadeh)
- rgw: fix replica log indexing (#8251 Yehuda Sadeh)
- rgw: fix swift metadata header name (Dmytro Iurchenko)

- rgw: remove multipart entries from bucket index on abort (#10719 Yehuda Sadeh)
- rgw: respond with 204 to POST on containers (#10667 Yuan Zhou)
- rgw: reuse fcgx connection struct (#10194 Yehuda Sadeh)
- rgw: support multiple host names (#7467 Yehuda Sadeh)
- rgw: swift: dump container's custom metadata (#10665 Ahmad Faheem, Dmytro Iurchenko)
- rgw: swift: support Accept header for response format (#10746 Dmytro Iurchenko)
- rgw: swift: support for X-Remove-Container-Meta-{key} (#10475 Dmytro Iurchenko)
- rpm: move rgw logrotate to radosgw subpackage (Ken Dreyer)
- tests: centos-6 dockerfile (#10755 Loic Dachary)
- tests: unit tests for shared_cache (Dong Yuan)
- vstart.sh: work with cmake (Yehuda Sadeh)

11.5 v0.92

This is the second-to-last chunk of new stuff before Hammer. Big items include additional checksums on OSD objects, proxied reads in the cache tier, image locking in RBD, optimized OSD Transaction and replication messages, and a big pile of RGW and MDS bug fixes.

11.5.1 Upgrading

- The experimental 'keyvaluestore-dev' OSD backend has been renamed 'keyvaluestore' (for simplicity) and marked as experimental. To enable this untested feature and acknowledge that you understand that it is untested and may destroy data, you need to add the following to your ceph.conf:

```
enable experimental unrecoverable data corrupting featuers = keyvaluestore
```

- The following librados C API function calls take a 'flags' argument whose value is now correctly interpreted:

  rados_write_op_operate() rados_aio_write_op_operate() rados_read_op_operate() rados_aio_read_op_operate()

  The flags were not correctly being translated from the librados constants to the internal values. Now they are. Any code that is passing flags to these methods should be audited to ensure that they are using the correct LIBRADOS_OP_FLAG_* constants.
- The 'rados' CLI 'copy' and 'cppool' commands now use the copy-from operation, which means the latest CLI cannot run these commands against pre-firefly OSDs.
- The librados watch/notify API now includes a watch_flush() operation to flush the async queue of notify operations. This should be called by any watch/notify user prior to rados_shutdown().

11.5.2 Notable Changes

- add experimental features option (Sage Weil)
- build: fix 'make check' races (#10384 Loic Dachary)
- build: fix pkg names when libkeyutils is missing (Pankag Garg, Ken Dreyer)
- ceph: make 'ceph -s' show PG state counts in sorted order (Sage Weil)

- ceph: make ‘ceph tell mon.* version’ work (Mykola Golub)
- ceph-monstore-tool: fix/improve CLI (Joao Eduardo Luis)
- ceph: show primary-affinity in ‘ceph osd tree’ (Mykola Golub)
- common: add TableFormatter (Andreas Peters)
- common: check syncfs() return code (Jianpeng Ma)
- doc: do not suggest dangerous XFS nobarrier option (Dan van der Ster)
- doc: misc updates (Nilamdyuti Goswami, John Wilkins)
- install-deps.sh: do not require sudo when root (Loic Dachary)
- libcephfs: fix dirfrag trimming (#10387 Yan, Zheng)
- libcephfs: fix mount timeout (#10041 Yan, Zheng)
- libcephfs: fix test (#10415 Yan, Zheng)
- libcephfs: fix use-afer-free on umount (#10412 Yan, Zheng)
- libcephfs: include ceph and git version in client metadata (Sage Weil)
- librados: add watch_flush() operation (Sage Weil, Haomai Wang)
- librados: avoid memcpy on getxattr, read (Jianpeng Ma)
- librados: create ioctx by pool id (Jason Dillaman)
- librados: do notify completion in fast-dispatch (Sage Weil)
- librados: remove shadowed variable (Kefu Chain)
- librados: translate op flags from C APIs (Matthew Richards)
- librbd: differentiate between R/O vs R/W features (Jason Dillaman)
- librbd: exclusive image locking (Jason Dillaman)
- librbd: fix write vs import race (#10590 Jason Dillaman)
- librbd: gracefully handle deleted/renamed pools (#10270 Jason Dillaman)
- mds: asok command for fetching subtree map (John Spray)
- mds: constify MDSCacheObjects (John Spray)
- misc: various valgrind fixes and cleanups (Danny Al-Gaaf)
- mon: fix ‘mds fail’ for standby MDSs (John Spray)
- mon: fix stashed monmap encoding (#5203 Xie Rui)
- mon: implement ‘fs reset’ command (John Spray)
- mon: respect down flag when promoting standbys (John Spray)
- mount.ceph: fix suprious error message (#10351 Yan, Zheng)
- msgr: async: many fixes, unit tests (Haomai Wang)
- msgr: simple: retry binding to port on failure (#10029 Wido den Hollander)
- osd: add fadvise flags to ObjectStore API (Jianpeng Ma)
- osd: add get_latest_osdmap asok command (#9483 #9484 Mykola Golub)
- osd: EIO on whole-object reads when checksum is wrong (Sage Weil)

- osd: filejournal: don't cache journal when not using direct IO (Jianpeng Ma)
- osd: fix ioprio option (Mykola Golub)
- osd: fix scrub delay bug (#10693 Samuel Just)
- osd: fix watch reconnect race (#10441 Sage Weil)
- osd: handle no-op write with snapshot (#10262 Sage Weil)
- osd: journal: fix journal zeroing when direct IO is enabled (Xie Rui)
- osd: keyvaluestore: cleanup dead code (Ning Yao)
- osd, mds: 'ops' as shorthand for 'dump_ops_in_flight' on asok (Sage Weil)
- osd: memstore: fix size limit (Xiaoxi Chen)
- osd: misc scrub fixes (#10017 Loic Dachary)
- osd: new optimized encoding for ObjectStore::Transaction (Dong Yuan)
- osd: optimize filter_snapc (Ning Yao)
- osd: optimize WBThrottle map with unordered_map (Ning Yao)
- osd: proxy reads during cache promote (Zhiqiang Wang)
- osd: proxy read support (Zhiqiang Wang)
- osd: remove legacy classic scrub code (Sage Weil)
- osd: remove unused fields in MOSDSubOp (Xiaoxi Chen)
- osd: replace MOSDSubOp messages with simpler, optimized MOSDRepOp (Xiaoxi Chen)
- osd: store whole-object checksums on scrub, write_full (Sage Weil)
- osd: verify kernel is new enough before using XFS extsize ioctl, enable by default (#9956 Sage Weil)
- rados: use copy-from operation for copy, cppool (Sage Weil)
- rgw: change multipart upload id magic (#10271 Yehuda Sadeh)
- rgw: decode http query params correction (#10271 Yehuda Sadeh)
- rgw: fix content length check (#10701 Axel Dunkel, Yehuda Sadeh)
- rgw: fix partial GET in swift (#10553 Yehuda Sadeh)
- rgw: fix shutdown (#10472 Yehuda Sadeh)
- rgw: include XML ns on get ACL request (#10106 Yehuda Sadeh)
- rgw: misc fixes (#10307 Yehuda Sadeh)
- rgw: only track cleanup for objects we write (#10311 Yehuda Sadeh)
- rgw: tweak error codes (#10329 #10334 Yehuda Sadeh)
- rgw: use gc for multipart abort (#10445 Aaron Bassett, Yehuda Sadeh)
- sysvinit: fix race in 'stop' (#10389 Loic Dachary)
- test: fix bufferlist tests (Jianpeng Ma)
- tests: improve docker-based tests (Loic Dachary)

11.6 v0.91

We are quickly approaching the Hammer feature freeze but have a few more dev releases to go before we get there. The headline items are subtree-based quota support in CephFS (ceph-fuse/libcephfs client support only for now), a rewrite of the watch/notify librados API used by RBD and RGW, OSDMap checksums to ensure that maps are always consistent inside the cluster, new API calls in librados and librbd for IO hinting modeled after posix_fadvise, and improved storage of per-PG state.

We expect two more releases before the Hammer feature freeze (v0.93).

11.6.1 Upgrading

- The ‘category’ field for objects has been removed. This was originally added to track PG stat summations over different categories of objects for use by radosgw. It is no longer has any known users and is prone to abuse because it can lead to a pg_stat_t structure that is unbounded. The librados API calls that accept this field now ignore it, and the OSD no longers tracks the per-category summations.
- The output for ‘rados df’ has changed. The ‘category’ level has been eliminated, so there is now a single stat object per pool. The structure of the JSON output is different, and the plaintext output has one less column.
- The ‘rados create <objectname> [category]’ optional category argument is no longer supported or recognized.
- rados.py’s Rados class no longer has a __del__ method; it was causing problems on interpreter shutdown and use of threads. If your code has Rados objects with limited lifetimes and you’re concerned about locked resources, call Rados.shutdown() explicitly.
- There is a new version of the librados watch/notify API with vastly improved semantics. Any applications using this interface are encouraged to migrate to the new API. The old API calls are marked as deprecated and will eventually be removed.
- The librados rados_unwatch() call used to be safe to call on an invalid handle. The new version has undefined behavior when passed a bogus value (for example, when rados_watch() returns an error and handle is not defined).
- The structure of the formatted ‘pg stat’ command is changed for the portion that counts states by name to avoid using the ‘+’ character (which appears in state names) as part of the XML token (it is not legal).

11.6.2 Notable Changes

- asyncmsgr: misc fixes (Haomai Wang)
- buffer: add ‘shareable’ construct (Matt Benjamin)
- build: aarch64 build fixes (Noah Watkins, Haomai Wang)
- build: support for jemalloc (Shishir Gowda)
- ceph-disk: allow journal partition re-use (#10146 Loic Dachary, Dav van der Ster)
- ceph-disk: misc fixes (Christos Stavrakakis)
- ceph-fuse: fix kernel cache trimming (#10277 Yan, Zheng)
- ceph-objectstore-tool: many many improvements (David Zafman)
- common: support new gperftools header locations (Key Dreyer)
- crush: straw bucket weight calculation fixes (#9998 Sage Weil)
- doc: misc improvements (Nilamdyuti Goswami, John Wilkins, Chris Holcombe)

- libcephfs,ceph-fuse: add 'status' asok (John Spray)
- librados, osd: new watch/notify implementation (Sage Weil)
- librados: drop 'category' feature (Sage Weil)
- librados: fix pool deletion handling (#10372 Sage Weil)
- librados: new fadvise API (Ma Jianpeng)
- libradosstriper: fix remove() (Dongmao Zhang)
- librbd: complete pending ops before closing image (#10299 Josh Durgin)
- librbd: fadvise API (Ma Jianpeng)
- mds: ENOSPC and OSDMap epoch barriers (#7317 John Spray)
- mds: dirfrag buf fix (Yan, Zheng)
- mds: disallow most commands on inactive MDS's (Greg Farnum)
- mds: drop dentries, leases on deleted directories (#10164 Yan, Zheng)
- mds: handle zero-size xattr (#10335 Yan, Zheng)
- mds: subtree quota support (Yunchuan Wen)
- memstore: free space tracking (John Spray)
- misc cleanup (Danny Al-Gaaf, David Anderson)
- mon: 'osd crush reweight-all' command (Sage Weil)
- mon: allow full flag to be manually cleared (#9323 Sage Weil)
- mon: delay failure injection (Joao Eduardo Luis)
- mon: fix paxos timeouts (#10220 Joao Eduardo Luis)
- mon: get canonical OSDMap from leader (#10422 Sage Weil)
- msgr: fix RESETSESSION bug (#10080 Greg Farnum)
- objectstore: deprecate collection attrs (Sage Weil)
- osd, mon: add checksums to all OSDMaps (Sage Weil)
- osd: allow deletion of objects with watcher (#2339 Sage Weil)
- osd: allow sparse read for Push/Pull (Haomai Wang)
- osd: cache reverse_nibbles hash value (Dong Yuan)
- osd: drop upgrade support for pre-dumpling (Sage Weil)
- osd: enable and use posix_fadvise (Sage Weil)
- osd: erasure-code: enforce chunk size alignment (#10211 Loic Dachary)
- osd: erasure-code: jerasure support for NEON (Loic Dachary)
- osd: erasure-code: relax cauchy w restrictions (#10325 David Zhang, Loic Dachary)
- osd: erasure-code: update gf-complete to latest upstream (Loic Dachary)
- osd: fix WBTHrottle perf counters (Haomai Wang)
- osd: fix backfill bug (#10150 Samuel Just)
- osd: fix occasional peering stalls (#10431 Sage Weil)

- osd: fix scrub vs try-flush bug (#8011 Samuel Just)
- osd: fix stderr with -f or -d (Dan Mick)
- osd: misc FIEMAP fixes (Ma Jianpeng)
- osd: optimize Finisher (Xinze Chi)
- osd: store PG metadata in per-collection objects for better concurrency (Sage Weil)
- pyrados: add object lock support (#6114 Mehdi Abaakouk)
- pyrados: fix misnamed wait_* routings (#10104 Dan Mick)
- pyrados: misc cleanups (Kefu Chai)
- qa: add large auth ticket tests (Ilya Dryomov)
- qa: many 'make check' improvements (Loic Dachary)
- qa: misc tests (Loic Dachary, Yan, Zheng)
- rgw: conditional PUT on ETag (#8562 Ray Lv)
- rgw: fix error codes (#10334 #10329 Yehuda Sadeh)
- rgw: index swift keys appropriately (#10471 Yehuda Sadeh)
- rgw: prevent illegal bucket policy that doesn't match placement rule (Yehuda Sadeh)
- rgw: run radosgw as apache with systemd (#10125 Loic Dachary)
- rgw: support X-Storage-Policy header for Swift storage policy compat (Yehuda Sadeh)
- rgw: use rn for http headers (#9254 Yehuda Sadeh)
- rpm: misc fixes (Key Dreyer)

11.7 v0.90

This is the last development release before Christmas. There are some API cleanups for librados and librbd, and lots of bug fixes across the board for the OSD, MDS, RGW, and CRUSH. The OSD also gets support for discard (potentially helpful on SSDs, although it is off by default), and there are several improvements to ceph-disk.

The next two development releases will be getting a slew of new functionality for hammer. Stay tuned!

11.7.1 Upgrading

- Previously, the formatted output of 'ceph pg stat -f ...' was a full pg dump that included all metadata about all PGs in the system. It is now a concise summary of high-level PG stats, just like the unformatted 'ceph pg stat' command.
- All JSON dumps of floating point values were incorrecting surrounding the value with quotes. These quotes have been removed. Any consumer of structured JSON output that was consuming the floating point values was previously having to interpret the quoted string and will most likely need to be fixed to take the unquoted number.

11.7.2 Notable Changes

- arch: fix NEON feaeture detection (#10185 Loic Dachary)
- build: adjust build deps for yasm, virtualenv (Jianpeng Ma)
- build: improve build dependency tooling (Loic Dachary)
- ceph-disk: call partx/partprobe consistency (#9721 Loic Dachary)
- ceph-disk: fix dmcrypt key permissions (Loic Dachary)
- ceph-disk: fix umount race condition (#10096 Blaine Gardner)
- ceph-disk: init=none option (Loic Dachary)
- ceph-monstore-tool: fix shutdown (#10093 Loic Dachary)
- ceph-objectstore-tool: fix import (#10090 David Zafman)
- ceph-objectstore-tool: many improvements and tests (David Zafman)
- ceph.spec: package rbd-replay-prep (Ken Dreyer)
- common: add 'perf reset ...' admin command (Jianpeng Ma)
- common: do not unlock rwlock on destruction (Federico Simoncelli)
- common: fix block device discard check (#10296 Sage Weil)
- common: remove broken CEPH_LOCKDEP optoin (Kefu Chai)
- crush: fix tree bucket behavior (Rongze Zhu)
- doc: add build-doc guidlines for Fedora and CentOS/RHEL (Nilamdyuti Goswami)
- doc: enable rbd cache on openstack deployments (Sebastien Han)
- doc: improved installation nots on CentOS/RHEL installs (John Wilkins)
- doc: misc cleanups (Adam Spiers, Sebastien Han, Nilamdyuti Goswami, Ken Dreyer, John Wilkins)
- doc: new man pages (Nilamdyuti Goswami)
- doc: update release descriptions (Ken Dreyer)
- doc: update sepia hardware inventory (Sandon Van Ness)
- librados: only export public API symbols (Jason Dillaman)
- libradosstriper: fix stat strtoll (Dongmao Zhang)
- libradosstriper: fix trunc method (#10129 Sebastien Ponce)
- librbd: fix list_children from invalid pool ioctxs (#10123 Jason Dillaman)
- librbd: only export public API symbols (Jason Dillaman)
- many coverity fixes (Danny Al-Gaaf)
- mds: 'flush journal' admin command (John Spray)
- mds: fix MDLog IO callback deadlock (John Spray)
- mds: fix deadlock during journal probe vs purge (#10229 Yan, Zheng)
- mds: fix race trimming log segments (Yan, Zheng)
- mds: store backtrace for stray dir (Yan, Zheng)
- mds: verify backtrace when fetching dirfrag (#9557 Yan, Zheng)

- mon: add max pgs per osd warning (Sage Weil)
- mon: fix *_ratio* units and types (Sage Weil)
- mon: fix JSON dumps to dump floats as flots and not strings (Sage Weil)
- mon: fix formatter 'pg stat' command output (Sage Weil)
- msgr: async: several fixes (Haomai Wang)
- msgr: simple: fix rare deadlock (Greg Farnum)
- osd: batch pg log trim (Xinze Chi)
- osd: clean up internal ObjectStore interface (Sage Weil)
- osd: do not abort deep scrub on missing hinfo (#10018 Loic Dachary)
- osd: fix ghobject_t formatted output to include shard (#10063 Loic Dachary)
- osd: fix osd peer check on scrub messages (#9555 Sage Weil)
- osd: fix pgls filter ops (#9439 David Zafman)
- osd: flush snapshots from cache tier immediately (Sage Weil)
- osd: keyvaluestore: fix getattr semantics (Haomai Wang)
- osd: keyvaluestore: fix key ordering (#10119 Haomai Wang)
- osd: limit in-flight read requests (Jason Dillaman)
- osd: log when scrub or repair starts (Loic Dachary)
- osd: support for discard for journal trim (Jianpeng Ma)
- qa: fix osd create dup tests (#10083 Loic Dachary)
- rgw: add location header when object is in another region (VRan Liu)
- rgw: check timestamp on s3 keystone auth (#10062 Abhishek Lekshmanan)
- rgw: make sysvinit script set ulimit -n properly (Sage Weil)
- systemd: better systemd unit files (Owen Synge)
- tests: ability to run unit tests under docker (Loic Dachary)

11.8 v0.89

This is the second development release since Giant. The big items include the first batch of scrub patchs from Greg for CephFS, a rework in the librados object listing API to properly handle namespaces, and a pile of bug fixes for RGW. There are also several smaller issues fixed up in the performance area with buffer alignment and memory copies, osd cache tiering agent, and various CephFS fixes.

11.8.1 Upgrading

- New ability to list all objects from all namespaces can fail or return incomplete results when not all OSDs have been upgraded. Features rados –all ls, rados cppool, rados export, rados cache-flush-evict-all and rados cache-try-flush-evict-all can also fail or return incomplete results.

11.8.2 Notable Changes

- buffer: add list::get_contiguous (Sage Weil)
- buffer: avoid rebuild if buffer already contiguous (Jianpeng Ma)
- ceph-disk: improved systemd support (Owen Synge)
- ceph-disk: set guid if reusing journal partition (Dan van der Ster)
- ceph-fuse, libcephfs: allow xattr caps in inject_release_failure (#9800 John Spray)
- ceph-fuse, libcephfs: fix I_COMPLETE_ORDERED checks (#9894 Yan, Zheng)
- ceph-fuse: fix dentry invalidation on 3.18+ kernels (#9997 Yan, Zheng)
- crush: fix detach_bucket (#10095 Sage Weil)
- crush: fix several bugs in adjust_item_weight (Rongze Zhu)
- doc: add dumpling to firefly upgrade section (#7679 John Wilkins)
- doc: document erasure coded pool operations (#9970 Loic Dachary)
- doc: file system osd config settings (Kevin Dalley)
- doc: key/value store config reference (John Wilkins)
- doc: update openstack docs for Juno (Sebastien Han)
- fix cluster logging from non-mon daemons (Sage Weil)
- init-ceph: check for systemd-run before using it (Boris Ranto)
- librados: fix infinite loop with skipped map epochs (#9986 Ding Dinghua)
- librados: fix iterator operator= bugs (#10082 David Zafman, Yehuda Sadeh)
- librados: fix null deref when pool DNE (#9944 Sage Weil)
- librados: fix timer race from recent refactor (Sage Weil)
- libradosstriper: fix shutdown hang (Dongmao Zhang)
- librbd: don't close a closed parent in failure path (#10030 Jason Dillaman)
- librbd: fix diff test (#10002 Josh Durgin)
- librbd: fix locking for readahead (#10045 Jason Dillaman)
- librbd: refactor unit tests to use fixtures (Jason Dillaman)
- many many coverity cleanups (Danny Al-Gaaf)
- mds: a whole bunch of initial scrub infrastructure (Greg Farnum)
- mds: fix compat_version for MClientSession (#9945 John Spray)
- mds: fix reply snapbl (Yan, Zheng)
- mon: allow adding tiers to fs pools (#10135 John Spray)
- mon: fix MDS health status from peons (#10151 John Spray)
- mon: fix caching for min_last_epoch_clean (#9987 Sage Weil)
- mon: fix error output for add_data_pool (#9852 Joao Eduardo Luis)
- mon: include entity name in audit log for forwarded requests (#9913 Joao Eduardo Luis)
- mon: paxos: allow reads while proposing (#9321 #9322 Joao Eduardo Luis)

- msgr: asyncmessenger: add kqueue support (#9926 Haomai Wang)
- osd, librados: revamp PG listing API to handle namespaces (#9031 #9262 #9438 David Zafman)
- osd, mon: send intiial pg create time from mon to osd (#9887 David Zafman)
- osd: allow whiteout deletion in cache pool (Sage Weil)
- osd: cache pool: ignore min flush age when cache is full (Xinze Chi)
- osd: erasure coding: allow bench.sh to test ISA backend (Yuan Zhou)
- osd: erasure-code: encoding regression tests, corpus (#9420 Loic Dachary)
- osd: fix journal shutdown race (Sage Weil)
- osd: fix object age eviction (Zhiqiang Wang)
- osd: fix object atime calculation (Xinze Chi)
- osd: fix past_interval display bug (#9752 Loic Dachary)
- osd: journal: fix alignment checks, avoid useless memmove (Jianpeng Ma)
- osd: journal: update committed_thru after replay (#6756 Samuel Just)
- osd: keyvaluestore_dev: optimization (Chendi Xue)
- osd: make misdirected op checks robust for EC pools (#9835 Sage Weil)
- osd: removed some dead code (Xinze Chi)
- qa: parallelize make check (Loic Dachary)
- qa: tolerate nearly-full disk for make check (Loic Dachary)
- rgw: create subuser if needed when creating user (#10103 Yehuda Sadeh)
- rgw: fix If-Modified-Since (VRan Liu)
- rgw: fix content-length update (#9576 Yehuda Sadeh)
- rgw: fix disabling of max_size quota (#9907 Dong Lei)
- rgw: fix incorrect len when len is 0 (#9877 Yehuda Sadeh)
- rgw: fix object copy content type (#9478 Yehuda Sadeh)
- rgw: fix user stags in get-user-info API (#9359 Ray Lv)
- rgw: remove swift user manifest (DLO) hash calculation (#9973 Yehuda Sadeh)
- rgw: return timestamp on GET/HEAD (#8911 Yehuda Sadeh)
- rgw: set ETag on object copy (#9479 Yehuda Sadeh)
- rgw: update bucket index on attr changes, for multi-site sync (#5595 Yehuda Sadeh)

11.9 v0.88

This is the first development release after Giant. The two main features merged this round are the new AsyncMessenger (an alternative implementation of the network layer) from Haomai Wang at UnitedStack, and support for POSIX file locks in ceph-fuse and libcephfs from Yan, Zheng. There is also a big pile of smaller items that re merged while we were stabilizing Giant, including a range of smaller performance and bug fixes and some new tracepoints for LTTNG.

11.9.1 Notable Changes

- ceph-disk: Scientific Linux support (Dan van der Ster)
- ceph-disk: respect –statedir for keyring (Loic Dachary)
- ceph-fuse, libcephfs: POSIX file lock support (Yan, Zheng)
- ceph-fuse, libcephfs: fix cap flush overflow (Greg Farnum, Yan, Zheng)
- ceph-fuse, libcephfs: fix root inode xattrs (Yan, Zheng)
- ceph-fuse, libcephfs: preserve dir ordering (#9178 Yan, Zheng)
- ceph-fuse, libcephfs: trim inodes before reconnecting to MDS (Yan, Zheng)
- ceph: do not parse injectargs twice (Loic Dachary)
- ceph: make 'ceph -s' output more readable (Sage Weil)
- ceph: new 'ceph tell mds.$name_or_rank_or_gid' (John Spray)
- ceph: test robustness (Joao Eduardo Luis)
- ceph_objectstore_tool: behave with sharded flag (#9661 David Zafman)
- cephfs-journal-tool: fix journal import (#10025 John Spray)
- cephfs-journal-tool: skip up to expire_pos (#9977 John Spray)
- cleanup rados.h definitions with macros (Ilya Dryomov)
- common: shared_cache unit tests (Cheng Cheng)
- config: add $cctid meta variable (Adam Crume)
- crush: fix buffer overrun for poorly formed rules (#9492 Johnu George)
- crush: improve constness (Loic Dachary)
- crushtool: add –location <id> command (Sage Weil, Loic Dachary)
- default to libnss instead of crypto++ (Federico Gimenez)
- doc: ceph osd reweight vs crush weight (Laurent Guerby)
- doc: document the LRC per-layer plugin configuration (Yuan Zhou)
- doc: erasure code doc updates (Loic Dachary)
- doc: misc updates (Alfredo Deza, VRan Liu)
- doc: preflight doc fixes (John Wilkins)
- doc: update PG count guide (Gerben Meijer, Laurent Guerby, Loic Dachary)
- keyvaluestore: misc fixes (Haomai Wang)
- keyvaluestore: performance improvements (Haomai Wang)
- librados: add rados_pool_get_base_tier() call (Adam Crume)
- librados: cap buffer length (Loic Dachary)
- librados: fix objecter races (#9617 Josh Durgin)
- libradosstriper: misc fixes (Sebastien Ponce)
- librbd: add missing python docstrings (Jason Dillaman)
- librbd: add readahead (Adam Crume)

- librbd: fix cache tiers in list_children and snap_unprotect (Adam Crume)
- librbd: fix performance regression in ObjectCacher (#9513 Adam Crume)
- librbd: lttng tracepoints (Adam Crume)
- librbd: misc fixes (Xinxin Shu, Jason Dillaman)
- mds: fix sessionmap lifecycle bugs (Yan, Zheng)
- mds: initialize root inode xattr version (Yan, Zheng)
- mds: introduce auth caps (John Spray)
- mds: misc bugs (Greg Farnum, John Spray, Yan, Zheng, Henry Change)
- misc coverity fixes (Danny Al-Gaaf)
- mon: add 'ceph osd rename-bucket ...' command (Loic Dachary)
- mon: clean up auth list output (Loic Dachary)
- mon: fix 'osd crush link' id resolution (John Spray)
- mon: fix misc error paths (Joao Eduardo Luis)
- mon: fix paxos off-by-one corner case (#9301 Sage Weil)
- mon: new 'ceph pool ls [detail]' command (Sage Weil)
- mon: wait for writeable before cross-proposing (#9794 Joao Eduardo Luis)
- msgr: avoid useless new/delete (Haomai Wang)
- msgr: fix delay injection bug (#9910 Sage Weil, Greg Farnum)
- msgr: new AsymcMessenger alternative implementation (Haomai Wang)
- msgr: prefetch data when doing recv (Yehuda Sadeh)
- osd: add erasure code corpus (Loic Dachary)
- osd: add misc tests (Loic Dachary, Danny Al-Gaaf)
- osd: cleanup boost optionals (William Kennington)
- osd: expose non-journal backends via ceph-osd CLI (Hoamai Wang)
- osd: fix JSON output for stray OSDs (Loic Dachary)
- osd: fix ioprio options (Loic Dachary)
- osd: fix transaction accounting (Jianpeng Ma)
- osd: misc optimizations (Xinxin Shu, Zhiqiang Wang, Xinze Chi)
- osd: use FIEMAP_FLAGS_SYNC instead of fsync (Jianpeng Ma)
- rados: fix put of /dev/null (Loic Dachary)
- rados: parse command-line arguments more strictly (#8983 Adam Crume)
- rbd-fuse: fix memory leak (Adam Crume)
- rbd-replay-many (Adam Crume)
- rbd-replay: –anonymize flag to rbd-replay-prep (Adam Crume)
- rbd: fix 'rbd diff' for non-existent objects (Adam Crume)
- rbd: fix error when striping with format 1 (Sebastien Han)

- rbd: fix export for image sizes over 2GB (Vicente Cheng)
- rbd: use rolling average for rbd bench-write throughput (Jason Dillaman)
- rgw: send explicit HTTP status string (Yehuda Sadeh)
- rgw: set length for keystone token validation request (#7796 Yehuda Sadeh, Mark Kirkwood)
- udev: fix rules for CentOS7/RHEL7 (Loic Dachary)
- use clock_gettime instead of gettimeofday (Jianpeng Ma)
- vstart.sh: set up environment for s3-tests (Luis Pabon)

11.10 v0.87.2 Giant

This is the second (and possibly final) point release for Giant.

We recommend all v0.87.x Giant users upgrade to this release.

11.10.1 Notable Changes

- ceph-objectstore-tool: only output unsupported features when incompatible (#11176 David Zafman)
- common: do not implicitly unlock rwlock on destruction (Federico Simoncelli)
- common: make wait timeout on empty queue configurable (#10818 Samuel Just)
- crush: pick ruleset id that matches and rule id (Xiaoxi Chen)
- crush: set_choose_tries = 100 for new erasure code rulesets (#10353 Loic Dachary)
- librados: check initialized atomic safely (#9617 Josh Durgin)
- librados: fix failed tick_event assert (#11183 Zhiqiang Wang)
- librados: fix looping on skipped maps (#9986 Ding Dinghua)
- librados: fix op submit with timeout (#10340 Samuel Just)
- librados: pybind: fix memory leak (#10723 Billy Olsen)
- librados: pybind: keep reference to callbacks (#10775 Josh Durgin)
- librados: translate operation flags from C APIs (Matthew Richards)
- libradosstriper: fix write_full on ENOENT (#10758 Sebastien Ponce)
- libradosstriper: use strtoll instead of strtol (Dongmao Zhang)
- mds: fix assertion caused by system time moving backwards (#11053 Yan, Zheng)
- mon: allow injection of random delays on writes (Joao Eduardo Luis)
- mon: do not trust small osd epoch cache values (#10787 Sage Weil)
- mon: fail non-blocking flush if object is being scrubbed (#8011 Samuel Just)
- mon: fix division by zero in stats dump (Joao Eduardo Luis)
- mon: fix get_rule_avail when no osds (#10257 Joao Eduardo Luis)
- mon: fix timeout rounds period (#10546 Joao Eduardo Luis)
- mon: ignore osd failures before up_from (#10762 Dan van der Ster, Sage Weil)

- mon: paxos: reset accept timeout before writing to store (#10220 Joao Eduardo Luis)
- mon: return if fs exists on 'fs new' (Joao Eduardo Luis)
- mon: use EntityName when expanding profiles (#10844 Joao Eduardo Luis)
- mon: verify cross-service proposal preconditions (#10643 Joao Eduardo Luis)
- mon: wait for osdmon to be writeable when requesting proposal (#9794 Joao Eduardo Luis)
- mount.ceph: avoid spurious error message about /etc/mtab (#10351 Yan, Zheng)
- msg/simple: allow RESETSESSION when we forget an endpoint (#10080 Greg Farnum)
- msg/simple: discard delay queue before incoming queue (#9910 Sage Weil)
- osd: clear_primary_state when leaving Primary (#10059 Samuel Just)
- osd: do not ignore deleted pgs on startup (#10617 Sage Weil)
- osd: fix FileJournal wrap to get header out first (#10883 David Zafman)
- osd: fix PG leak in SnapTrimWQ (#10421 Kefu Chai)
- osd: fix journalq population in do_read_entry (#6003 Samuel Just)
- osd: fix operator== for op_queue_age_hit and fs_perf_stat (#10259 Samuel Just)
- osd: fix rare assert after split (#10430 David Zafman)
- osd: get pgid ancestor from last_map when building past intervals (#10430 David Zafman)
- osd: include rollback_info_trimmed_to in {read,write}_log (#10157 Samuel Just)
- osd: lock header_lock in DBObjectMap::sync (#9891 Samuel Just)
- osd: requeue blocked op before flush it was blocked on (#10512 Sage Weil)
- osd: tolerate missing object between list and attr get on backfill (#10150 Samuel Just)
- osd: use correct atime for eviction decision (Xinze Chi)
- rgw: flush XML header on get ACL request (#10106 Yehuda Sadeh)
- rgw: index swift keys appropriately (#10471 Hemant Bruman, Yehuda Sadeh)
- rgw: send cancel for bucket index pending ops (#10770 Baijiaruo, Yehuda Sadeh)
- rgw: swift: support X_Remove_Container-Meta-{key} (#01475 Dmytro Iurchenko)

For more detailed information, see `the complete changelog`.

11.11 v0.87.1 Giant

This is the first (and possibly final) point release for Giant. Our focus on stability fixes will be directed towards Hammer and Firefly.

We recommend that all v0.87 Giant users upgrade to this release.

11.11.1 Upgrading

- Due to a change in the Linux kernel version 3.18 and the limits of the FUSE interface, ceph-fuse needs be mounted as root on at least some systems. See issues #9997, #10277, and #10542 for details.

11.11.2 Notable Changes

- build: disable stack-execute bit on assembler objects (#10114 Dan Mick)
- build: support boost 1.57.0 (#10688 Ken Dreyer)
- ceph-disk: fix dmcrypt file permissions (#9785 Loic Dachary)
- ceph-disk: run partprobe after zap, behave with partx or partprobe (#9665 #9721 Loic Dachary)
- cephfs-journal-tool: fix import for aged journals (#9977 John Spray)
- cephfs-journal-tool: fix journal import (#10025 John Spray)
- ceph-fuse: use remount to trim kernel dcache (#10277 Yan, Zheng)
- common: add cctid meta variable (#6228 Adam Crume)
- common: fix dump of shard for ghobject_t (#10063 Loic Dachary)
- crush: fix bucket weight underflow (#9998 Pawel Sadowski)
- erasure-code: enforce chunk size alignment (#10211 Loic Dachary)
- erasure-code: regression test suite (#9420 Loic Dachary)
- erasure-code: relax caucy w restrictions (#10325 Loic Dachary)
- libcephfs,ceph-fuse: allow xattr caps on inject_release_failure (#9800 John Spray)
- libcephfs,ceph-fuse: fix cap flush tid comparison (#9869 Greg Farnum)
- libcephfs,ceph-fuse: new flag to indicated sorted dcache (#9178 Yan, Zheng)
- libcephfs,ceph-fuse: prune cache before reconnecting to MDS (Yan, Zheng)
- librados: limit number of in-flight read requests (#9854 Jason Dillaman)
- libradospy: fix thread shutdown (#8797 Dan Mick)
- libradosstriper: fix locking issue in truncate (#10129 Sebastien Ponce)
- librbd: complete pending ops before closing mage (#10299 Jason Dillaman)
- librbd: fix error path on image open failure (#10030 Jason Dillaman)
- librbd: gracefully handle deleted/renamed pools (#10270 Jason Dillaman)
- librbd: handle errors when creating ioctx while listing children (#10123 Jason Dillaman)
- mds: fix compat version in MClientSession (#9945 John Spray)
- mds: fix journaler write error handling (#10011 John Spray)
- mds: fix locking for file size recovery (#10229 Yan, Zheng)
- mds: handle heartbeat_reset during shutdown (#10382 John Spray)
- mds: store backtrace for straydir (Yan, Zheng)
- mon: allow tiers for FS pools (#10135 John Spray)
- mon: fix caching of last_epoch_clean, osdmap trimming (#9987 Sage Weil)
- mon: fix 'fs ls' on peons (#10288 John Spray)
- mon: fix MDS health status from peons (#10151 John Spray)
- mon: fix paxos off-by-one (#9301 Sage Weil)
- msgr: simple: do not block on takeover while holding global lock (#9921 Greg Farnum)

- osd: deep scrub must not abort if hinfo is missing (#10018 Loic Dachary)
- osd: fix misdirected op detection (#9835 Sage Weil)
- osd: fix past_interval display for acting (#9752 Loic Dachary)
- osd: fix PG peering backoff when behind on osdmaps (#10431 Sage Weil)
- osd: handle no-op write with snapshot case (#10262 Ssage Weil)
- osd: use fast-dispatch (Sage Weil, Greg Farnum)
- rados: fix write to /dev/null (Loic Dachary)
- radosgw-admin: create subuser when needed (#10103 Yehuda Sadeh)
- rbd: avoid invalidating aio_write buffer during image import (#10590 Jason Dillaman)
- rbd: fix export with images > 2GB (Vicente Cheng)
- rgw: change multipart upload id magic (#10271 Georgios Dimitrakakis, Yehuda Sadeh)
- rgw: check keystone auth for S3 POST (#10062 Abhishek Lekshmanan)
- rgw: check timestamp for S3 keystone auth (#10062 Abhishek Lekshmanan)
- rgw: fix partial GET with swift (#10553 Yehuda Sadeh)
- rgw: fix quota disable (#9907 Dong Lei)
- rgw: fix rare corruption of object metadata on put (#9576 Yehuda Sadeh)
- rgw: fix S3 object copy content-type (#9478 Yehuda Sadeh)
- rgw: headers end with rn (#9254 Benedikt Fraunhofer, Yehuda Sadeh)
- rgw: remove swift user manifest DLO hash calculation (#9973 Yehuda Sadeh)
- rgw: return correct len when len is 0 (#9877 Yehuda Sadeh)
- rgw: return X-Timestamp field (#8911 Yehuda Sadeh)
- rgw: run radosgw as apache with systemd (#10125)
- rgw: sent ETag on S3 object copy (#9479 Yehuda Sadeh)
- rgw: sent HTTP status reason explicitly in fastcgi (Yehuda Sadeh)
- rgw: set length for keystone token validation (#7796 Mark Kirkwood, Yehuda Sadeh)
- rgw: set ulimit -n on sysvinit before starting daemon (#9587 Sage Weil)
- rgw: update bucket index on set_attrs (#5595 Yehuda Sadeh)
- rgw: update swift subuser permission masks when authenticating (#9918 Yehuda Sadeh)
- rgw: URL decode HTTP query params correction (#10271 Georgios Dimitrakakis, Yehuda Sadeh)
- rgw: use cached attrs while reading object attrs (#10307 Yehuda Sadeh)
- rgw: use strict_strtoll for content length (#10701 Axel Dunkel, Yehuda Sadeh)

For more detailed information, see `the complete changelog`.

11.12 v0.87 Giant

This release will form the basis for the stable release Giant, v0.87.x. Highlights for Giant include:

- *RADOS Performance*: a range of improvements have been made in the OSD and client-side librados code that improve the throughput on flash backends and improve parallelism and scaling on fast machines.
- *CephFS*: we have fixed a raft of bugs in CephFS and built some basic journal recovery and diagnostic tools. Stability and performance of single-MDS systems is vastly improved in Giant. Although we do not yet recommend CephFS for production deployments, we do encourage testing for non-critical workloads so that we can better guage the feature, usability, performance, and stability gaps.
- *Local Recovery Codes*: the OSDs now support an erasure-coding scheme that stores some additional data blocks to reduce the IO required to recover from single OSD failures.
- *Degraded vs misplaced*: the Ceph health reports from 'ceph -s' and related commands now make a distinction between data that is degraded (there are fewer than the desired number of copies) and data that is misplaced (stored in the wrong location in the cluster). The distinction is important because the latter does not compromise data safety.
- *Tiering improvements*: we have made several improvements to the cache tiering implementation that improve performance. Most notably, objects are not promoted into the cache tier by a single read; they must be found to be sufficiently hot before that happens.
- *Monitor performance*: the monitors now perform writes to the local data store asynchronously, improving overall responsiveness.
- *Recovery tools*: the ceph_objectstore_tool is greatly expanded to allow manipulation of an individual OSDs data store for debugging and repair purposes. This is most heavily used by our QA infrastructure to exercise recovery code.

11.12.1 Upgrade Sequencing

- If your existing cluster is running a version older than v0.80.x Firefly, please first upgrade to the latest Firefly release before moving on to Giant. We have not tested upgrades directly from Emperor, Dumpling, or older releases.

  We *have* tested:

  - Firefly to Giant
  - Dumpling to Firefly to Giant

- Please upgrade daemons in the following order:

  1. Monitors
  2. OSDs
  3. MDSs and/or radosgw

  Note that the relative ordering of OSDs and monitors should not matter, but we primarily tested upgrading monitors first.

11.12.2 Upgrading from v0.80x Firefly

- The client-side caching for librbd is now enabled by default (rbd cache = true). A safety option (rbd cache writethrough until flush = true) is also enabled so that writeback caching is not used until the library observes a 'flush' command, indicating that the librbd users is passing that operation through from the guest VM. This avoids potential data loss when used with older versions of qemu that do not support flush.

> leveldb_write_buffer_size = 8*1024*1024 = 33554432 // 8MB leveldb_cache_size = 512*1024*1204 = 536870912 // 512MB leveldb_block_size = 64*1024 = 65536 // 64KB leveldb_compression = false leveldb_log = ""

OSDs will still maintain the following osd-specific defaults:

> leveldb_log = ""

- The 'rados getxattr ...' command used to add a gratuitous newline to the attr value; it now does not.
- The `*_kb perf` counters on the monitor have been removed. These are replaced with a new set of `*_bytes` counters (e.g., `cluster_osd_kb` is replaced by `cluster_osd_bytes`).
- The `rd_kb` and `wr_kb` fields in the JSON dumps for pool stats (accessed via the `ceph df detail -f json-pretty` and related commands) have been replaced with corresponding `*_bytes` fields. Similarly, the `total_space`, `total_used`, and `total_avail` fields are replaced with `total_bytes`, `total_used_bytes`, and `total_avail_bytes` fields.
- The `rados df --format=json` output `read_bytes` and `write_bytes` fields were incorrectly reporting ops; this is now fixed.
- The `rados df --format=json` output previously included `read_kb` and `write_kb` fields; these have been removed. Please use `read_bytes` and `write_bytes` instead (and divide by 1024 if appropriate).
- The experimental keyvaluestore-dev OSD backend had an on-disk format change that prevents existing OSD data from being upgraded. This affects developers and testers only.
- mon-specific and osd-specific leveldb options have been removed. From this point onward users should use the *leveldb_** generic options and add the options in the appropriate sections of their configuration files. Monitors will still maintain the following monitor-specific defaults:

  > leveldb_write_buffer_size = 8*1024*1024 = 33554432 // 8MB leveldb_cache_size = 512*1024*1204 = 536870912 // 512MB leveldb_block_size = 64*1024 = 65536 // 64KB leveldb_compression = false leveldb_log = ""

  OSDs will still maintain the following osd-specific defaults:

  > leveldb_log = ""

- CephFS support for the legacy anchor table has finally been removed. Users with file systems created before firefly should ensure that inodes with multiple hard links are modified *prior* to the upgrade to ensure that the backtraces are written properly. For example:

```
sudo find /mnt/cephfs -type f -links +1 -exec touch \{\} \;
```

- We disallow nonsensical 'tier cache-mode' transitions. From this point onward, 'writeback' can only transition to 'forward' and 'forward' can transition to 1) 'writeback' if there are dirty objects, or 2) any if there are no dirty objects.

11.12.3 Notable Changes since v0.86

- ceph-disk: use new udev rules for centos7/rhel7 (#9747 Loic Dachary)
- libcephfs-java: fix fstat mode (Noah Watkins)
- librados: fix deadlock when listing PG contents (Guang Yang)
- librados: misc fixes to the new threading model (#9582 #9706 #9845 #9873 Sage Weil)
- mds: fix inotable initialization (Henry C Chang)

- mds: gracefully handle unknown lock type in flock requests (Yan, Zheng)
- mon: add read-only, read-write, and role-definer rols (Joao Eduardo Luis)
- mon: fix mon cap checks (Joao Eduardo Luis)
- mon: misc fixes for new paxos async writes (#9635 Sage Weil)
- mon: set scrub timestamps on PG creation (#9496 Joao Eduardo Luis)
- osd: erasure code: fix buffer alignment (Janne Grunau, Loic Dachary)
- osd: fix alloc hint induced crashes on mixed clusters (#9419 David Zafman)
- osd: fix backfill reservation release on rejection (#9626, Samuel Just)
- osd: fix ioprio option parsing (#9676 #9677 Loic Dachary)
- osd: fix memory leak during snap trimming (#9113 Samuel Just)
- osd: misc peering and recovery fixes (#9614 #9696 #9731 #9718 #9821 #9875 Samuel Just, Guang Yang)

11.12.4 Notable Changes since v0.80.x Firefly

- bash completion improvements (Wido den Hollander)
- brag: fixes, improvements (Loic Dachary)
- buffer: improve rebuild_page_aligned (Ma Jianpeng)
- build: fix build on alpha (Michael Cree, Dmitry Smirnov)
- build: fix CentOS 5 (Gerben Meijer)
- build: fix yasm check for x32 (Daniel Schepler, Sage Weil)
- ceph-brag: add tox tests (Alfredo Deza)
- ceph-conf: flush log on exit (Sage Weil)
- ceph.conf: update sample (Sebastien Han)
- ceph-dencoder: refactor build a bit to limit dependencies (Sage Weil, Dan Mick)
- ceph-disk: add Scientific Linux support (Dan van der Ster)
- ceph-disk: do not inadvertantly create directories (Owne Synge)
- ceph-disk: fix dmcrypt support (Sage Weil)
- ceph-disk: fix dmcrypt support (Stephen Taylor)
- ceph-disk: handle corrupt volumes (Stuart Longlang)
- ceph-disk: linter cleanup, logging improvements (Alfredo Deza)
- ceph-disk: partprobe as needed (Eric Eastman)
- ceph-disk: show information about dmcrypt in 'ceph-disk list' output (Sage Weil)
- ceph-disk: use partition type UUIDs and blkid (Sage Weil)
- ceph: fix for non-default cluster names (#8944, Dan Mick)
- ceph-fuse, libcephfs: asok hooks for handling session resets, timeouts (Yan, Zheng)
- ceph-fuse, libcephfs: fix crash in trim_caps (John Spray)
- ceph-fuse, libcephfs: improve cap trimming (John Spray)

- ceph-fuse, libcephfs: improve traceless reply handling (Sage Weil)
- ceph-fuse, libcephfs: virtual xattrs for rstat (Yan, Zheng)
- ceph_objectstore_tool: vastly improved and extended tool for working offline with OSD data stores (David Zafman)
- ceph.spec: many fixes (Erik Logtenberg, Boris Ranto, Dan Mick, Sandon Van Ness)
- ceph.spec: split out ceph-common package, other fixes (Sandon Van Ness)
- ceph_test_librbd_fsx: fix RNG, make deterministic (Ilya Dryomov)
- cephtool: fix help (Yilong Zhao)
- cephtool: refactor and improve CLI tests (Joao Eduardo Luis)
- cephtool: test cleanup (Joao Eduardo Luis)
- clang build fixes (John Spray, Danny Al-Gaaf)
- client: improved MDS session dumps (John Spray)
- common: add config diff admin socket command (Joao Eduardo Luis)
- common: add rwlock assertion checks (Yehuda Sadeh)
- common: fix dup log messages (#9080, Sage Weil)
- common: perfcounters now use atomics and go faster (Sage Weil)
- config: support G, M, K, etc. suffixes (Joao Eduardo Luis)
- coverity cleanups (Danny Al-Gaaf)
- crush: clean up CrushWrapper interface (Xioaxi Chen)
- crush: include new tunables in dump (Sage Weil)
- crush: make ruleset ids unique (Xiaoxi Chen, Loic Dachary)
- crush: only require rule features if the rule is used (#8963, Sage Weil)
- crushtool: send output to stdout, not stderr (Wido den Hollander)
- doc: cache tiering (John Wilkins)
- doc: CRUSH updates (John Wilkins)
- doc: document new upstream wireshark dissector (Kevin Cox)
- doc: improve manual install docs (Francois Lafont)
- doc: keystone integration docs (John Wilkins)
- doc: librados example fixes (Kevin Dalley)
- doc: many doc updates (John Wilkins)
- doc: many install doc updates (John Wilkins)
- doc: misc updates (John Wilkins, Loic Dachary, David Moreau Simard, Wido den Hollander. Volker Voigt, Alfredo Deza, Stephen Jahl, Dan van der Ster)
- doc: osd primary affinity (John Wilkins)
- doc: pool quotas (John Wilkins)
- doc: pre-flight doc improvements (Kevin Dalley)
- doc: switch to an unencumbered font (Ross Turk)

- doc: updated simple configuration guides (John Wilkins)
- doc: update erasure docs (Loic Dachary, Venky Shankar)
- doc: update openstack docs (Josh Durgin)
- filestore: disable use of XFS hint (buggy on old kernels) (Samuel Just)
- filestore: fix xattr spillout (Greg Farnum, Haomai Wang)
- fix hppa arch build (Dmitry Smirnov)
- fix i386 builds (Sage Weil)
- fix struct vs class inconsistencies (Thorsten Behrens)
- global: write pid file even when running in foreground (Alexandre Oliva)
- hadoop: improve tests (Huamin Chen, Greg Farnum, John Spray)
- hadoop: update hadoop tests for Hadoop 2.0 (Haumin Chen)
- init-ceph: continue starting other daemons on crush or mount failure (#8343, Sage Weil)
- journaler: fix locking (Zheng, Yan)
- keyvaluestore: fix hint crash (#8381, Haomai Wang)
- keyvaluestore: header cache (Haomai Wang)
- libcephfs-java: build against older JNI headers (Greg Farnum)
- libcephfs-java: fix gcj-jdk build (Dmitry Smirnov)
- librados: fix crash on read op timeout (#9362 Matthias Kiefer, Sage Weil)
- librados: fix lock leaks in error paths (#9022, Paval Rallabhandi)
- librados: fix pool existence check (#8835, Pavan Rallabhandi)
- librados: fix rados_pool_list bounds checks (Sage Weil)
- librados: fix shutdown race (#9130 Sage Weil)
- librados: fix watch/notify test (#7934 David Zafman)
- librados: fix watch reregistration on acting set change (#9220 Samuel Just)
- librados: give Objecter fine-grained locks (Yehuda Sadeh, Sage Weil, John Spray)
- librados: lttng tracepoitns (Adam Crume)
- librados, osd: return ETIMEDOUT on failed notify (Sage Weil)
- librados: pybind: fix reads when 0 is present (#9547 Mohammad Salehe)
- librados_striper: striping library for librados (Sebastien Ponce)
- librbd, ceph-fuse: reduce cache flush overhead (Haomai Wang)
- librbd: check error code on cache invalidate (Josh Durgin)
- librbd: enable caching by default (Sage Weil)
- librbd: enforce cache size on read requests (Jason Dillaman)
- librbd: fix crash using clone of flattened image (#8845, Josh Durgin)
- librbd: fix error path when opening image (#8912, Josh Durgin)
- librbd: handle blacklisting during shutdown (#9105 John Spray)

- librbd: lttng tracepoints (Adam Crume)
- librbd: new libkrbd library for kernel map/unmap/showmapped (Ilya Dryomov)
- librbd: store and retrieve snapshot metadata based on id (Josh Durgin)
- libs3: update to latest (Danny Al-Gaaf)
- log: fix derr level (Joao Eduardo Luis)
- logrotate: fix osd log rotation on ubuntu (Sage Weil)
- lttng: tracing infrastructure (Noah Watkins, Adam Crume)
- mailmap: many updates (Loic Dachary)
- mailmap: updates (Loic Dachary, Abhishek Lekshmanan, M Ranga Swami Reddy)
- Makefile: fix out of source builds (Stefan Eilemann)
- many many coverity fixes, cleanups (Danny Al-Gaaf)
- mds: adapt to new Objecter locking, give types to all Contexts (John Spray)
- mds: add file system name, enabled flag (John Spray)
- mds: add internal health checks (John Spray)
- mds: add min/max UID for snapshot creation/deletion (#9029, Wido den Hollander)
- mds: avoid tight mon reconnect loop (#9428 Sage Weil)
- mds: boot refactor, cleanup (John Spray)
- mds: cephfs-journal-tool (John Spray)
- mds: fix crash killing sessions (#9173 John Spray)
- mds: fix ctime updates (#9514 Greg Farnum)
- mds: fix journal conversion with standby-replay (John Spray)
- mds: fix replay locking (Yan, Zheng)
- mds: fix standby-replay cache trimming (#8648 Zheng, Yan)
- mds: fix xattr bug triggered by ACLs (Yan, Zheng)
- mds: give perfcounters meaningful names (Sage Weil)
- mds: improve health reporting to monitor (John Spray)
- mds: improve Journaler on-disk format (John Spray)
- mds: improve journal locking (Zheng, Yan)
- mds, libcephfs: use client timestamp for mtime/ctime (Sage Weil)
- mds: make max file recoveries tunable (Sage Weil)
- mds: misc encoding improvements (John Spray)
- mds: misc fixes for multi-mds (Yan, Zheng)
- mds: multi-mds fixes (Yan, Zheng)
- mds: OPTracker integration, dump_ops_in_flight (Greg Farnum)
- mds: prioritize file recovery when appropriate (Sage Weil)
- mds: refactor beacon, improve reliability (John Spray)

- mds: remove legacy anchor table (Yan, Zheng)
- mds: remove legacy discover ino (Yan, Zheng)
- mds: restart on EBLACKLISTED (John Spray)
- mds: separate inode recovery queue (John Spray)
- mds: session ls, evict commands (John Spray)
- mds: submit log events in async thread (Yan, Zheng)
- mds: track RECALL progress, report failure (#9284 John Spray)
- mds: update segment references during journal write (John Spray, Greg Farnum)
- mds: use client-provided timestamp for user-visible file metadata (Yan, Zheng)
- mds: use meaningful names for clients (John Spray)
- mds: validate journal header on load and save (John Spray)
- mds: warn clients which aren't revoking caps (Zheng, Yan, John Spray)
- misc build errors/warnings for Fedora 20 (Boris Ranto)
- misc build fixes for OS X (John Spray)
- misc cleanup (Christophe Courtaut)
- misc integer size cleanups (Kevin Cox)
- misc memory leaks, cleanups, fixes (Danny Al-Gaaf, Sahid Ferdjaoui)
- misc suse fixes (Danny Al-Gaaf)
- misc word size fixes (Kevin Cox)
- mon: add audit log for all admin commands (Joao Eduardo Luis)
- mon: add cluster fingerprint (Sage Weil)
- mon: add get-quota commands (Joao Eduardo Luis)
- mon: add 'osd blocked-by' command to easily see which OSDs are blocking peering progress (Sage Weil)
- mon: add 'osd reweight-by-pg' command (Sage Weil, Guang Yang)
- mon: add perfcounters for paxos operations (Sage Weil)
- mon: avoid creating unnecessary rule on pool create (#9304 Loic Dachary)
- monclient: fix hang (Sage Weil)
- mon: create default EC profile if needed (Loic Dachary)
- mon: do not create file system by default (John Spray)
- mon: do not spam log (Aanchal Agrawal, Sage Weil)
- mon: drop mon- and osd- specific leveldb options (Joao Eduardo Luis)
- mon: ec pool profile fixes (Loic Dachary)
- mon: fix bug when no auth keys are present (#8851, Joao Eduardo Luis)
- mon: fix 'ceph df' output for available space (Xiaoxi Chen)
- mon: fix compat version for MForward (Joao Eduardo Luis)
- mon: fix crash on loopback messages and paxos timeouts (#9062, Sage Weil)

- mon: fix default replication pool ruleset choice (#8373, John Spray)
- mon: fix divide by zero when pg_num is adjusted before OSDs are added (#9101, Sage Weil)
- mon: fix double-free of old MOSDBoot (Sage Weil)
- mon: fix health down messages (Sage Weil)
- mon: fix occasional memory leak after session reset (#9176, Sage Weil)
- mon: fix op write latency perfcounter (#9217 Xinxin Shu)
- mon: fix 'osd perf' reported latency (#9269 Samuel Just)
- mon: fix quorum feature check (#8738, Greg Farnum)
- mon: fix ruleset/ruleid bugs (#9044, Loic Dachary)
- mon: fix set cache_target_full_ratio (#8440, Geoffrey Hartz)
- mon: fix store check on startup (Joao Eduardo Luis)
- mon: include per-pool 'max avail' in df output (Sage Weil)
- mon: make paxos transaction commits asynchronous (Sage Weil)
- mon: make usage dumps in terms of bytes, not kB (Sage Weil)
- mon: 'osd crush reweight-subtree ...' (Sage Weil)
- mon, osd: relax client EC support requirements (Sage Weil)
- mon: preload erasure plugins (#9153 Loic Dachary)
- mon: prevent cache pools from being used directly by CephFS (#9435 John Spray)
- mon: prevent EC pools from being used with cephfs (Joao Eduardo Luis)
- mon: prevent implicit destruction of OSDs with 'osd setmaxosd ...' (#8865, Anand Bhat)
- mon: prevent nonsensical cache-mode transitions (Joao Eduardo Luis)
- mon: restore original weight when auto-marked out OSDs restart (Sage Weil)
- mon: restrict some pool properties to tiered pools (Joao Eduardo Luis)
- mon: some instrumentation (Sage Weil)
- mon: use msg header tid for MMonGetVersionReply (Ilya Dryomov)
- mon: use user-provided ruleset for replicated pool (Xiaoxi Chen)
- mon: verify all quorum members are contiguous at end of Paxos round (#9053, Sage Weil)
- mon: verify available disk space on startup (#9502 Joao Eduardo Luis)
- mon: verify erasure plugin version on load (Loic Dachary)
- msgr: avoid big lock when sending (most) messages (Greg Farnum)
- msgr: fix logged address (Yongyue Sun)
- msgr: misc locking fixes for fast dispatch (#8891, Sage Weil)
- msgr: refactor to cleanly separate SimpleMessenger implemenetation, move toward Connection-based calls (Matt Benjamin, Sage Wei)
- objecter: flag operations that are redirected by caching (Sage Weil)
- objectstore: clean up KeyValueDB interface for key/value backends (Sage Weil)

- osd: account for hit_set_archive bytes (Sage Weil)
- osd: add ability to prehash filestore directories (Guang Yang)
- osd: add 'dump_reservations' admin socket command (Sage Weil)
- osd: add feature bit for erasure plugins (Loic Dachary)
- osd: add header cache for KeyValueStore (Haomai Wang)
- osd: add ISA erasure plugin table cache (Andreas-Joachim Peters)
- osd: add local_mtime for use by cache agent (Zhiqiang Wang)
- osd: add local recovery code (LRC) erasure plugin (Loic Dachary)
- osd: add prototype KineticStore based on Seagate Kinetic (Josh Durgin)
- osd: add READFORWARD caching mode (Luis Pabon)
- osd: add superblock for KeyValueStore backend (Haomai Wang)
- osd: add support for Intel ISA-L erasure code library (Andreas-Joachim Peters)
- osd: allow map cache size to be adjusted at runtime (Sage Weil)
- osd: avoid refcounting overhead by passing a few things by ref (Somnath Roy)
- osd: avoid sharing PG info that is not durable (Samuel Just)
- osd: bound osdmap epoch skew between PGs (Sage Weil)
- osd: cache tier flushing fixes for snapped objects (Samuel Just)
- osd: cap hit_set size (#9339 Samuel Just)
- osd: clean up shard_id_t, shard_t (Loic Dachary)
- osd: clear FDCache on unlink (#8914 Loic Dachary)
- osd: clear slow request latency info on osd up/down (Sage Weil)
- osd: do not evict blocked objects (#9285 Zhiqiang Wang)
- osd: do not skip promote for write-ordered reads (#9064, Samuel Just)
- osd: fix agent early finish looping (David Zafman)
- osd: fix ambigous encoding order for blacklisted clients (#9211, Sage Weil)
- osd: fix bogus assert during OSD shutdown (Sage Weil)
- osd: fix bug with long object names and rename (#8701, Sage Weil)
- osd: fix cache flush corner case for snapshotted objects (#9054, Samuel Just)
- osd: fix cache full -> not full requeueing (#8931, Sage Weil)
- osd: fix clone deletion case (#8334, Sam Just)
- osd: fix clone vs cache_evict bug (#8629 Sage Weil)
- osd: fix connection reconnect race (Greg Farnum)
- osd: fix crash from duplicate backfill reservation (#8863 Sage Weil)
- osd: fix dead peer connection checks (#9295 Greg Farnum, Sage Weil)
- osd: fix discard of old/obsolete subop replies (#9259, Samuel Just)
- osd: fix discard of peer messages from previous intervals (Greg Farnum)

- osd: fix dump of open fds on EMFILE (Sage Weil)
- osd: fix dumps (Joao Eduardo Luis)
- osd: fix erasure-code lib initialization (Loic Dachary)
- osd: fix extent normalization (Adam Crume)
- osd: fix filestore removal corner case (#8332, Sam Just)
- osd: fix flush vs OpContext (Samuel Just)
- osd: fix gating of messages from old OSD instances (Greg Farnum)
- osd: fix hang waiting for osdmap (#8338, Greg Farnum)
- osd: fix interval check corner case during peering (#8104, Sam Just)
- osd: fix ISA erasure alignment (Loic Dachary, Andreas-Joachim Peters)
- osd: fix journal dump (Ma Jianpeng)
- osd: fix journal-less operation (Sage Weil)
- osd: fix keyvaluestore scrub (#8589 Haomai Wang)
- osd: fix keyvaluestore upgrade (Haomai Wang)
- osd: fix loopback msgr issue (Ma Jianpeng)
- osd: fix LSB release parsing (Danny Al-Gaaf)
- osd: fix MarkMeDown and other shutdown races (Sage Weil)
- osd: fix memstore bugs with collection_move_rename, lock ordering (Sage Weil)
- osd: fix min_read_recency_for_promote default on upgrade (Zhiqiang Wang)
- osd: fix mon feature bit requirements bug and resulting log spam (Sage Weil)
- osd: fix mount/remount sync race (#9144 Sage Weil)
- osd: fix PG object listing/ordering bug (Guang Yang)
- osd: fix PG stat errors with tiering (#9082, Sage Weil)
- osd: fix purged_snap initialization on backfill (Sage Weil, Samuel Just, Dan van der Ster, Florian Haas)
- osd: fix race condition on object deletion (#9480 Somnath Roy)
- osd: fix recovery chunk size usage during EC recovery (Ma Jianpeng)
- osd: fix recovery reservation deadlock for EC pools (Samuel Just)
- osd: fix removal of old xattrs when overwriting chained xattrs (Ma Jianpeng)
- osd: fix requesting queueing on PG split (Samuel Just)
- osd: fix scrub vs cache bugs (Samuel Just)
- osd: fix snap object writeback from cache tier (#9054 Samuel Just)
- osd: fix trim of hitsets (Sage Weil)
- osd: force new xattrs into leveldb if fs returns E2BIG (#7779, Sage Weil)
- osd: implement alignment on chunk sizes (Loic Dachary)
- osd: improved backfill priorities (Sage Weil)
- osd: improve journal shutdown (Ma Jianpeng, Mark Kirkwood)

- osd: improve locking for KeyValueStore (Haomai Wang)
- osd: improve locking in OpTracker (Pavan Rallabhandi, Somnath Roy)
- osd: improve prioritization of recovery of degraded over misplaced objects (Sage Weil)
- osd: improve tiering agent arithmetic (Zhiqiang Wang, Sage Weil, Samuel Just)
- osd: include backend information in metadata reported to mon (Sage Weil)
- osd: locking, sharding, caching improvements in FileStore's FDCache (Somnath Roy, Greg Farnum)
- osd: lttng tracepoints for filestore (Noah Watkins)
- osd: make blacklist encoding deterministic (#9211 Sage Weil)
- osd: make tiering behave if hit_sets aren't enabled (Sage Weil)
- osd: many important bug fixes (Samuel Just)
- osd: many many core fixes (Samuel Just)
- osd: many many important fixes (#8231 #8315 #9113 #9179 #9293 #9294 #9326 #9453 #9481 #9482 #9497 #9574 Samuel Just)
- osd: mark pools with incomplete clones (Sage Weil)
- osd: misc erasure code plugin fixes (Loic Dachary)
- osd: misc locking fixes for fast dispatch (Samuel Just, Ma Jianpeng)
- osd, mon: add rocksdb support (Xinxin Shu, Sage Weil)
- osd, mon: config sanity checks on start (Sage Weil, Joao Eduardo Luis)
- osd, mon: distinguish between "misplaced" and "degraded" objects in cluster health and PG state reporting (Sage Weil)
- osd, msgr: fast-dispatch of OSD ops (Greg Farnum, Samuel Just)
- osd, objecter: resend ops on last_force_op_resend barrier; fix cache overlay op ordering (Sage Weil)
- osd: preload erasure plugins (#9153 Loic Dachary)
- osd: prevent old rados clients from using tiered pools (#8714, Sage Weil)
- osd: reduce OpTracker overhead (Somnath Roy)
- osd: refactor some ErasureCode functionality into command parent class (Loic Dachary)
- osd: remove obsolete classic scrub code (David Zafman)
- osd: scrub PGs with invalid stats (Sage Weil)
- osd: set configurable hard limits on object and xattr names (Sage Weil, Haomai Wang)
- osd: set rollback_info_completed on create (#8625, Samuel Just)
- osd: sharded threadpool to improve parallelism (Somnath Roy)
- osd: shard OpTracker to improve performance (Somnath Roy)
- osd: simple io prioritization for scrub (Sage Weil)
- osd: simple scrub throttling (Sage Weil)
- osd: simple snap trimmer throttle (Sage Weil)
- osd: tests for bench command (Loic Dachary)
- osd: trim old EC objects quickly; verify on scrub (Samuel Just)

- osd: use FIEMAP to inform copy_range (Haomai Wang)
- osd: use local time for tiering decisions (Zhiqiang Wang)
- osd: use xfs hint less frequently (Ilya Dryomov)
- osd: verify erasure plugin version on load (Loic Dachary)
- osd: work around GCC 4.8 bug in journal code (Matt Benjamin)
- pybind/rados: fix small timeouts (John Spray)
- qa: xfstests updates (Ilya Dryomov)
- rados: allow setxattr value to be read from stdin (Sage Weil)
- rados bench: fix arg order (Kevin Dalley)
- rados: drop gratuitous n from getxattr command (Sage Weil)
- rados: fix bench write arithmetic (Jiangheng)
- rados: fix {read,write}_ops values for df output (Sage Weil)
- rbd: add rbdmap pre- and post post- hooks, fix misc bugs (Dmitry Smirnov)
- rbd-fuse: allow exposing single image (Stephen Taylor)
- rbd-fuse: fix unlink (Josh Durgin)
- rbd: improve option default behavior (Josh Durgin)
- rbd: parallelize rbd import, export (Jason Dillaman)
- rbd: rbd-replay utility to replay captured rbd workload traces (Adam Crume)
- rbd: use write-back (not write-through) when caching is enabled (Jason Dillaman)
- removed mkcephfs (deprecated since dumpling)
- rest-api: fix help (Ailing Zhang)
- rgw: add civetweb as default frontent on port 7490 (#9013 Yehuda Sadeh)
- rgw: add –min-rewrite-stripe-size for object restriper (Yehuda Sadeh)
- rgw: add powerdns hook for dynamic DNS for global clusters (Wido den Hollander)
- rgw: add S3 bucket get location operation (Abhishek Lekshmanan)
- rgw: allow : in S3 access key (Roman Haritonov)
- rgw: automatically align writes to EC pool (#8442, Yehuda Sadeh)
- rgw: bucket link uses instance id (Yehuda Sadeh)
- rgw: cache bucket info (Yehuda Sadeh)
- rgw: cache decoded user info (Yehuda Sadeh)
- rgw: check entity permission for put_metadata (#8428, Yehuda Sadeh)
- rgw: copy object data is target bucket is in a different pool (#9039, Yehuda Sadeh)
- rgw: do not try to authenticate CORS preflight requests (#8718, Robert Hubbard, Yehuda Sadeh)
- rgw: fix admin create user op (#8583 Ray Lv)
- rgw: fix civetweb URL decoding (#8621, Yehuda Sadeh)
- rgw: fix crash on swift CORS preflight request (#8586, Yehuda Sadeh)

- rgw: fix log filename suffix (#9353 Alexandre Marangone)
- rgw: fix memory leak following chunk read error (Yehuda Sadeh)
- rgw: fix memory leaks (Andrey Kuznetsov)
- rgw: fix multipart object attr regression (#8452, Yehuda Sadeh)
- rgw: fix multipart upload (#8846, Silvain Munaut, Yehuda Sadeh)
- rgw: fix radosgw-admin 'show log' command (#8553, Yehuda Sadeh)
- rgw: fix removal of objects during object creation (Patrycja Szablowska, Yehuda Sadeh)
- rgw: fix striping for copied objects (#9089, Yehuda Sadeh)
- rgw: fix test for identify whether an object has a tail (#9226, Yehuda Sadeh)
- rgw: fix URL decoding (#8702, Brian Rak)
- rgw: fix URL escaping (Yehuda Sadeh)
- rgw: fix usage (Abhishek Lekshmanan)
- rgw: fix user manifest (Yehuda Sadeh)
- rgw: fix when stripe size is not a multiple of chunk size (#8937, Yehuda Sadeh)
- rgw: handle empty extra pool name (Yehuda Sadeh)
- rgw: improve civetweb logging (Yehuda Sadeh)
- rgw: improve delimited listing of bucket, misc fixes (Yehuda Sadeh)
- rgw: improve -h (Abhishek Lekshmanan)
- rgw: many fixes for civetweb (Yehuda Sadeh)
- rgw: misc civetweb fixes (Yehuda Sadeh)
- rgw: misc civetweb frontend fixes (Yehuda Sadeh)
- rgw: object and bucket rewrite functions to allow restriping old objects (Yehuda Sadeh)
- rgw: powerdns backend for global namespaces (Wido den Hollander)
- rgw: prevent multiobject PUT race (Yehuda Sadeh)
- rgw: send user manifest header (Yehuda Sadeh)
- rgw: subuser creation fixes (#8587 Yehuda Sadeh)
- rgw: use systemd-run from sysvinit script (JuanJose Galvez)
- rpm: do not restart daemons on upgrade (Alfredo Deza)
- rpm: misc packaging fixes for rhel7 (Sandon Van Ness)
- rpm: split ceph-common from ceph (Sandon Van Ness)
- systemd: initial systemd config files (Federico Simoncelli)
- systemd: wrap started daemons in new systemd environment (Sage Weil, Dan Mick)
- sysvinit: add support for non-default cluster names (Alfredo Deza)
- sysvinit: less sensitive to failures (Sage Weil)
- test_librbd_fsx: test krbd as well as librbd (Ilya Dryomov)
- unit test improvements (Loic Dachary)

- upstart: increase max open files limit (Sage Weil)
- vstart.sh: fix/improve rgw support (Luis Pabon, Abhishek Lekshmanan)

11.13 v0.86

This is a release candidate for Giant, which will hopefully be out in another week or two. We did a feature freeze about a month ago and since then have been doing only stabilization and bug fixing (and a handful on low-risk enhancements). A fair bit of new functionality went into the final sprint, but it's baked for quite a while now and we're feeling pretty good about it.

Major items include:

- librados locking refactor to improve scaling and client performance
- local recovery code (LRC) erasure code plugin to trade some additional storage overhead for improved recovery performance
- LTTNG tracing framework, with initial tracepoints in librados, librbd, and the OSD FileStore backend
- separate monitor audit log for all administrative commands
- asynchronos monitor transaction commits to reduce the impact on monitor read requests while processing updates
- low-level tool for working with individual OSD data stores for debugging, recovery, and testing
- many MDS improvements (bug fixes, health reporting)

There are still a handful of known bugs in this release, but nothing severe enough to prevent a release. By and large we are pretty pleased with the stability and expect the final Giant release to be quite reliable.

Please try this out on your non-production clusters for a preview

11.13.1 Notable Changes

- buffer: improve rebuild_page_aligned (Ma Jianpeng)
- build: fix CentOS 5 (Gerben Meijer)
- build: fix build on alpha (Michael Cree, Dmitry Smirnov)
- build: fix yasm check for x32 (Daniel Schepler, Sage Weil)
- ceph-disk: add Scientific Linux support (Dan van der Ster)
- ceph-fuse, libcephfs: fix crash in trim_caps (John Spray)
- ceph-fuse, libcephfs: improve cap trimming (John Spray)
- ceph-fuse, libcephfs: virtual xattrs for rstat (Yan, Zheng)
- ceph.conf: update sample (Sebastien Han)
- ceph.spec: many fixes (Erik Logtenberg, Boris Ranto, Dan Mick, Sandon Van Ness)
- ceph_objectstore_tool: vastly improved and extended tool for working offline with OSD data stores (David Zafman)
- common: add config diff admin socket command (Joao Eduardo Luis)
- common: add rwlock assertion checks (Yehuda Sadeh)

- crush: clean up CrushWrapper interface (Xioaxi Chen)
- crush: make ruleset ids unique (Xiaoxi Chen, Loic Dachary)
- doc: improve manual install docs (Francois Lafont)
- doc: misc updates (John Wilkins, Loic Dachary, David Moreau Simard, Wido den Hollander. Volker Voigt, Alfredo Deza, Stephen Jahl, Dan van der Ster)
- global: write pid file even when running in foreground (Alexandre Oliva)
- hadoop: improve tests (Huamin Chen, Greg Farnum, John Spray)
- journaler: fix locking (Zheng, Yan)
- librados, osd: return ETIMEDOUT on failed notify (Sage Weil)
- librados: fix crash on read op timeout (#9362 Matthias Kiefer, Sage Weil)
- librados: fix shutdown race (#9130 Sage Weil)
- librados: fix watch reregistration on acting set change (#9220 Samuel Just)
- librados: fix watch/notify test (#7934 David Zafman)
- librados: give Objecter fine-grained locks (Yehuda Sadeh, Sage Weil, John Spray)
- librados: lttng tracepoitns (Adam Crume)
- librados: pybind: fix reads when 0 is present (#9547 Mohammad Salehe)
- librbd: enforce cache size on read requests (Jason Dillaman)
- librbd: handle blacklisting during shutdown (#9105 John Spray)
- librbd: lttng tracepoints (Adam Crume)
- lttng: tracing infrastructure (Noah Watkins, Adam Crume)
- mailmap: updates (Loic Dachary, Abhishek Lekshmanan, M Ranga Swami Reddy)
- many many coverity fixes, cleanups (Danny Al-Gaaf)
- mds: adapt to new Objecter locking, give types to all Contexts (John Spray)
- mds: add internal health checks (John Spray)
- mds: avoid tight mon reconnect loop (#9428 Sage Weil)
- mds: fix crash killing sessions (#9173 John Spray)
- mds: fix ctime updates (#9514 Greg Farnum)
- mds: fix replay locking (Yan, Zheng)
- mds: fix standby-replay cache trimming (#8648 Zheng, Yan)
- mds: give perfcounters meaningful names (Sage Weil)
- mds: improve health reporting to monitor (John Spray)
- mds: improve journal locking (Zheng, Yan)
- mds: make max file recoveries tunable (Sage Weil)
- mds: prioritize file recovery when appropriate (Sage Weil)
- mds: refactor beacon, improve reliability (John Spray)
- mds: restart on EBLACKLISTED (John Spray)

- mds: track RECALL progress, report failure (#9284 John Spray)
- mds: update segment references during journal write (John Spray, Greg Farnum)
- mds: use meaningful names for clients (John Spray)
- mds: warn clients which aren’t revoking caps (Zheng, Yan, John Spray)
- mon: add ‘osd reweight-by-pg’ command (Sage Weil, Guang Yang)
- mon: add audit log for all admin commands (Joao Eduardo Luis)
- mon: add cluster fingerprint (Sage Weil)
- mon: avoid creating unnecessary rule on pool create (#9304 Loic Dachary)
- mon: do not spam log (Aanchal Agrawal, Sage Weil)
- mon: fix ‘osd perf’ reported latency (#9269 Samuel Just)
- mon: fix double-free of old MOSDBoot (Sage Weil)
- mon: fix op write latency perfcounter (#9217 Xinxin Shu)
- mon: fix store check on startup (Joao Eduardo Luis)
- mon: make paxos transaction commits asynchronous (Sage Weil)
- mon: preload erasure plugins (#9153 Loic Dachary)
- mon: prevent cache pools from being used directly by CephFS (#9435 John Spray)
- mon: use user-provided ruleset for replicated pool (Xiaoxi Chen)
- mon: verify available disk space on startup (#9502 Joao Eduardo Luis)
- mon: verify erasure plugin version on load (Loic Dachary)
- msgr: fix logged address (Yongyue Sun)
- osd: account for hit_set_archive bytes (Sage Weil)
- osd: add ISA erasure plugin table cache (Andreas-Joachim Peters)
- osd: add ability to prehash filestore directories (Guang Yang)
- osd: add feature bit for erasure plugins (Loic Dachary)
- osd: add local recovery code (LRC) erasure plugin (Loic Dachary)
- osd: cap hit_set size (#9339 Samuel Just)
- osd: clear FDCache on unlink (#8914 Loic Dachary)
- osd: do not evict blocked objects (#9285 Zhiqiang Wang)
- osd: fix ISA erasure alignment (Loic Dachary, Andreas-Joachim Peters)
- osd: fix clone vs cache_evict bug (#8629 Sage Weil)
- osd: fix crash from duplicate backfill reservation (#8863 Sage Weil)
- osd: fix dead peer connection checks (#9295 Greg Farnum, Sage Weil)
- osd: fix keyvaluestore scrub (#8589 Haomai Wang)
- osd: fix keyvaluestore upgrade (Haomai Wang)
- osd: fix min_read_recency_for_promote default on upgrade (Zhiqiang Wang)
- osd: fix mount/remount sync race (#9144 Sage Weil)

- osd: fix purged_snap initialization on backfill (Sage Weil, Samuel Just, Dan van der Ster, Florian Haas)
- osd: fix race condition on object deletion (#9480 Somnath Roy)
- osd: fix snap object writeback from cache tier (#9054 Samuel Just)
- osd: improve journal shutdown (Ma Jianpeng, Mark Kirkwood)
- osd: improve locking in OpTracker (Pavan Rallabhandi, Somnath Roy)
- osd: improve tiering agent arithmetic (Zhiqiang Wang, Sage Weil, Samuel Just)
- osd: lttng tracepoints for filestore (Noah Watkins)
- osd: make blacklist encoding deterministic (#9211 Sage Weil)
- osd: many many important fixes (#8231 #8315 #9113 #9179 #9293 #9294 #9326 #9453 #9481 #9482 #9497 #9574 Samuel Just)
- osd: misc erasure code plugin fixes (Loic Dachary)
- osd: preload erasure plugins (#9153 Loic Dachary)
- osd: shard OpTracker to improve performance (Somnath Roy)
- osd: use local time for tiering decisions (Zhiqiang Wang)
- osd: verify erasure plugin version on load (Loic Dachary)
- rados: fix bench write arithmetic (Jiangheng)
- rbd: parallelize rbd import, export (Jason Dillaman)
- rbd: rbd-replay utility to replay captured rbd workload traces (Adam Crume)
- rbd: use write-back (not write-through) when caching is enabled (Jason Dillaman)
- rgw: add S3 bucket get location operation (Abhishek Lekshmanan)
- rgw: add civetweb as default frontent on port 7490 (#9013 Yehuda Sadeh)
- rgw: allow : in S3 access key (Roman Haritonov)
- rgw: fix admin create user op (#8583 Ray Lv)
- rgw: fix log filename suffix (#9353 Alexandre Marangone)
- rgw: fix usage (Abhishek Lekshmanan)
- rgw: many fixes for civetweb (Yehuda Sadeh)
- rgw: subuser creation fixes (#8587 Yehuda Sadeh)
- rgw: use systemd-run from sysvinit script (JuanJose Galvez)
- unit test improvements (Loic Dachary)
- vstart.sh: fix/improve rgw support (Luis Pabon, Abhishek Lekshmanan)

11.14 v0.85

This is the second-to-last development release before Giant that contains new functionality. The big items to land during this cycle are the messenger refactoring from Matt Benjmain that lays some groundwork for RDMA support, a performance improvement series from SanDisk that improves performance on SSDs, lots of improvements to our new standalone civetweb-based RGW frontend, and a new 'osd blocked-by' mon command that allows admins to easily identify which OSDs are blocking peering progress. The other big change is that the OSDs and Monitors now distinguish between "misplaced" and "degraded" objects: the latter means there are fewer copies than we'd like, while the former simply means the are not stored in the locations where we want them to be.

Also of note is a change to librbd that enables client-side caching by default. This is coupled with another option that makes the cache write-through until a "flush" operations is observed: this implies that the librbd user (usually a VM guest OS) supports barriers and flush and that it is safe for the cache to switch into writeback mode without compromising data safety or integrity. It has long been recommended practice that these options be enabled (e.g., in OpenStack environments) but until now it has not been the default.

We have frozen the tree for the looming Giant release, and the next development release will be a release candidate with a final batch of new functionality.

11.14.1 Upgrading

- The client-side caching for librbd is now enabled by default (rbd cache = true). A safety option (rbd cache writethrough until flush = true) is also enabled so that writeback caching is not used until the library observes a 'flush' command, indicating that the librbd users is passing that operation through from the guest VM. This avoids potential data loss when used with older versions of qemu that do not support flush.

  > leveldb_write_buffer_size = 32*1024*1024 = 33554432 // 32MB leveldb_cache_size = 512*1024*1204 = 536870912 // 512MB leveldb_block_size = 64*1024 = 65536 // 64KB leveldb_compression = false leveldb_log = ""

  OSDs will still maintain the following osd-specific defaults:

  > leveldb_log = ""

- The 'rados getxattr ...' command used to add a gratuitous newline to the attr value; it now does not.

11.14.2 Notable Changes

- ceph-disk: do not inadvertantly create directories (Owne Synge)
- ceph-disk: fix dmcrypt support (Sage Weil)
- ceph-disk: linter cleanup, logging improvements (Alfredo Deza)
- ceph-disk: show information about dmcrypt in 'ceph-disk list' output (Sage Weil)
- ceph-disk: use partition type UUIDs and blkid (Sage Weil)
- ceph: fix for non-default cluster names (#8944, Dan Mick)
- doc: document new upstream wireshark dissector (Kevin Cox)
- doc: many install doc updates (John Wilkins)
- librados: fix lock leaks in error paths (#9022, Paval Rallabhandi)
- librados: fix pool existence check (#8835, Pavan Rallabhandi)
- librbd: enable caching by default (Sage Weil)
- librbd: fix crash using clone of flattened image (#8845, Josh Durgin)
- librbd: store and retrieve snapshot metadata based on id (Josh Durgin)
- mailmap: many updates (Loic Dachary)
- mds: add min/max UID for snapshot creation/deletion (#9029, Wido den Hollander)
- misc build errors/warnings for Fedora 20 (Boris Ranto)

- mon: add 'osd blocked-by' command to easily see which OSDs are blocking peering progress (Sage Weil)
- mon: add perfcounters for paxos operations (Sage Weil)
- mon: create default EC profile if needed (Loic Dachary)
- mon: fix crash on loopback messages and paxos timeouts (#9062, Sage Weil)
- mon: fix divide by zero when pg_num is adjusted before OSDs are added (#9101, Sage Weil)
- mon: fix occasional memory leak after session reset (#9176, Sage Weil)
- mon: fix ruleset/ruleid bugs (#9044, Loic Dachary)
- mon: make usage dumps in terms of bytes, not kB (Sage Weil)
- mon: prevent implicit destruction of OSDs with 'osd setmaxosd ...' (#8865, Anand Bhat)
- mon: verify all quorum members are contiguous at end of Paxos round (#9053, Sage Weil)
- msgr: refactor to cleanly separate SimpleMessenger implemenetation, move toward Connection-based calls (Matt Benjamin, Sage Wei)
- objectstore: clean up KeyValueDB interface for key/value backends (Sage Weil)
- osd: add local_mtime for use by cache agent (Zhiqiang Wang)
- osd: add superblock for KeyValueStore backend (Haomai Wang)
- osd: add support for Intel ISA-L erasure code library (Andreas-Joachim Peters)
- osd: do not skip promote for write-ordered reads (#9064, Samuel Just)
- osd: fix ambigous encoding order for blacklisted clients (#9211, Sage Weil)
- osd: fix cache flush corner case for snapshotted objects (#9054, Samuel Just)
- osd: fix discard of old/obsolete subop replies (#9259, Samuel Just)
- osd: fix discard of peer messages from previous intervals (Greg Farnum)
- osd: fix dump of open fds on EMFILE (Sage Weil)
- osd: fix journal dump (Ma Jianpeng)
- osd: fix mon feature bit requirements bug and resulting log spam (Sage Weil)
- osd: fix recovery chunk size usage during EC recovery (Ma Jianpeng)
- osd: fix recovery reservation deadlock for EC pools (Samuel Just)
- osd: fix removal of old xattrs when overwriting chained xattrs (Ma Jianpeng)
- osd: fix requesting queueing on PG split (Samuel Just)
- osd: force new xattrs into leveldb if fs returns E2BIG (#7779, Sage Weil)
- osd: implement alignment on chunk sizes (Loic Dachary)
- osd: improve prioritization of recovery of degraded over misplaced objects (Sage Weil)
- osd: locking, sharding, caching improvements in FileStore's FDCache (Somnath Roy, Greg Farnum)
- osd: many important bug fixes (Samuel Just)
- osd, mon: add rocksdb support (Xinxin Shu, Sage Weil)
- osd, mon: distinguish between "misplaced" and "degraded" objects in cluster health and PG state reporting (Sage Weil)
- osd: refactor some ErasureCode functionality into command parent class (Loic Dachary)

- osd: set rollback_info_completed on create (#8625, Samuel Just)
- rados: allow setxattr value to be read from stdin (Sage Weil)
- rados: drop gratuitous n from getxattr command (Sage Weil)
- rgw: add –min-rewrite-stripe-size for object restriper (Yehuda Sadeh)
- rgw: add powerdns hook for dynamic DNS for global clusters (Wido den Hollander)
- rgw: copy object data is target bucket is in a different pool (#9039, Yehuda Sadeh)
- rgw: do not try to authenticate CORS preflight requests (#8718, Robert Hubbard, Yehuda Sadeh)
- rgw: fix civetweb URL decoding (#8621, Yehuda Sadeh)
- rgw: fix removal of objects during object creation (Patrycja Szablowska, Yehuda Sadeh)
- rgw: fix striping for copied objects (#9089, Yehuda Sadeh)
- rgw: fix test for identify whether an object has a tail (#9226, Yehuda Sadeh)
- rgw: fix when stripe size is not a multiple of chunk size (#8937, Yehuda Sadeh)
- rgw: improve civetweb logging (Yehuda Sadeh)
- rgw: misc civetweb frontend fixes (Yehuda Sadeh)
- sysvinit: add support for non-default cluster names (Alfredo Deza)

11.15 v0.84

The next Ceph development release is here! This release contains several meaty items, including some MDS improvements for journaling, the ability to remove the CephFS file system (and name it), several mon cleanups with tiered pools, several OSD performance branches, a new "read forward" RADOS caching mode, a prototype Kinetic OSD backend, and various radosgw improvements (especially with the new standalone civetweb frontend). And there are a zillion OSD bug fixes. Things are looking pretty good for the Giant release that is coming up in the next month.

11.15.1 Upgrading

- The `*_kb perf` counters on the monitor have been removed. These are replaced with a new set of `*_bytes` counters (e.g., `cluster_osd_kb` is replaced by `cluster_osd_bytes`).
- The `rd_kb` and `wr_kb` fields in the JSON dumps for pool stats (accessed via the `ceph df detail -f json-pretty` and related commands) have been replaced with corresponding `*_bytes` fields. Similarly, the `total_space`, `total_used`, and `total_avail` fields are replaced with `total_bytes`, `total_used_bytes`, and `total_avail_bytes` fields.
- The `rados df --format=json` output `read_bytes` and `write_bytes` fields were incorrectly reporting ops; this is now fixed.
- The `rados df --format=json` output previously included `read_kb` and `write_kb` fields; these have been removed. Please use `read_bytes` and `write_bytes` instead (and divide by 1024 if appropriate).

11.15.2 Notable Changes

- ceph-conf: flush log on exit (Sage Weil)
- ceph-dencoder: refactor build a bit to limit dependencies (Sage Weil, Dan Mick)

- ceph.spec: split out ceph-common package, other fixes (Sandon Van Ness)
- ceph_test_librbd_fsx: fix RNG, make deterministic (Ilya Dryomov)
- cephtool: refactor and improve CLI tests (Joao Eduardo Luis)
- client: improved MDS session dumps (John Spray)
- common: fix dup log messages (#9080, Sage Weil)
- crush: include new tunables in dump (Sage Weil)
- crush: only require rule features if the rule is used (#8963, Sage Weil)
- crushtool: send output to stdout, not stderr (Wido den Hollander)
- fix i386 builds (Sage Weil)
- fix struct vs class inconsistencies (Thorsten Behrens)
- hadoop: update hadoop tests for Hadoop 2.0 (Haumin Chen)
- librbd, ceph-fuse: reduce cache flush overhead (Haomai Wang)
- librbd: fix error path when opening image (#8912, Josh Durgin)
- mds: add file system name, enabled flag (John Spray)
- mds: boot refactor, cleanup (John Spray)
- mds: fix journal conversion with standby-replay (John Spray)
- mds: separate inode recovery queue (John Spray)
- mds: session ls, evict commands (John Spray)
- mds: submit log events in async thread (Yan, Zheng)
- mds: use client-provided timestamp for user-visible file metadata (Yan, Zheng)
- mds: validate journal header on load and save (John Spray)
- misc build fixes for OS X (John Spray)
- misc integer size cleanups (Kevin Cox)
- mon: add get-quota commands (Joao Eduardo Luis)
- mon: do not create file system by default (John Spray)
- mon: fix 'ceph df' output for available space (Xiaoxi Chen)
- mon: fix bug when no auth keys are present (#8851, Joao Eduardo Luis)
- mon: fix compat version for MForward (Joao Eduardo Luis)
- mon: restrict some pool properties to tiered pools (Joao Eduardo Luis)
- msgr: misc locking fixes for fast dispatch (#8891, Sage Weil)
- osd: add 'dump_reservations' admin socket command (Sage Weil)
- osd: add READFORWARD caching mode (Luis Pabon)
- osd: add header cache for KeyValueStore (Haomai Wang)
- osd: add prototype KineticStore based on Seagate Kinetic (Josh Durgin)
- osd: allow map cache size to be adjusted at runtime (Sage Weil)
- osd: avoid refcounting overhead by passing a few things by ref (Somnath Roy)

- osd: avoid sharing PG info that is not durable (Samuel Just)
- osd: clear slow request latency info on osd up/down (Sage Weil)
- osd: fix PG object listing/ordering bug (Guang Yang)
- osd: fix PG stat errors with tiering (#9082, Sage Weil)
- osd: fix bug with long object names and rename (#8701, Sage Weil)
- osd: fix cache full -> not full requeueing (#8931, Sage Weil)
- osd: fix gating of messages from old OSD instances (Greg Farnum)
- osd: fix memstore bugs with collection_move_rename, lock ordering (Sage Weil)
- osd: improve locking for KeyValueStore (Haomai Wang)
- osd: make tiering behave if hit_sets aren't enabled (Sage Weil)
- osd: mark pools with incomplete clones (Sage Weil)
- osd: misc locking fixes for fast dispatch (Samuel Just, Ma Jianpeng)
- osd: prevent old rados clients from using tiered pools (#8714, Sage Weil)
- osd: reduce OpTracker overhead (Somnath Roy)
- osd: set configurable hard limits on object and xattr names (Sage Weil, Haomai Wang)
- osd: trim old EC objects quickly; verify on scrub (Samuel Just)
- osd: work around GCC 4.8 bug in journal code (Matt Benjamin)
- rados bench: fix arg order (Kevin Dalley)
- rados: fix {read,write}_ops values for df output (Sage Weil)
- rbd: add rbdmap pre- and post post- hooks, fix misc bugs (Dmitry Smirnov)
- rbd: improve option default behavior (Josh Durgin)
- rgw: automatically align writes to EC pool (#8442, Yehuda Sadeh)
- rgw: fix crash on swift CORS preflight request (#8586, Yehuda Sadeh)
- rgw: fix memory leaks (Andrey Kuznetsov)
- rgw: fix multipart upload (#8846, Silvain Munaut, Yehuda Sadeh)
- rgw: improve -h (Abhishek Lekshmanan)
- rgw: improve delimited listing of bucket, misc fixes (Yehuda Sadeh)
- rgw: misc civetweb fixes (Yehuda Sadeh)
- rgw: powerdns backend for global namespaces (Wido den Hollander)
- systemd: initial systemd config files (Federico Simoncelli)

11.16 v0.83

Another Ceph development release! This has been a longer cycle, so there has been quite a bit of bug fixing and stabilization in this round. There is also a bunch of packaging fixes for RPM distros (RHEL/CentOS, Fedora, and SUSE) and for systemd. We've also added a new librados-striper library from Sebastien Ponce that provides a generic striping API for applications to code to.

11.16.1 Upgrading

- The experimental keyvaluestore-dev OSD backend had an on-disk format change that prevents existing OSD data from being upgraded. This affects developers and testers only.
- mon-specific and osd-specific leveldb options have been removed. From this point onward users should use the *leveldb_** generic options and add the options in the appropriate sections of their configuration files. Monitors will still maintain the following monitor-specific defaults:

  leveldb_write_buffer_size = 32*1024*1024 = 33554432 // 32MB leveldb_cache_size = 512*1024*1204 = 536870912 // 512MB leveldb_block_size = 64*1024 = 65536 // 64KB leveldb_compression = false leveldb_log = ""

  OSDs will still maintain the following osd-specific defaults:

  leveldb_log = ""

11.16.2 Notable Changes

- ceph-disk: fix dmcrypt support (Stephen Taylor)
- cephtool: fix help (Yilong Zhao)
- cephtool: test cleanup (Joao Eduardo Luis)
- doc: librados example fixes (Kevin Dalley)
- doc: many doc updates (John Wilkins)
- doc: update erasure docs (Loic Dachary, Venky Shankar)
- filestore: disable use of XFS hint (buggy on old kernels) (Samuel Just)
- filestore: fix xattr spillout (Greg Farnum, Haomai Wang)
- keyvaluestore: header cache (Haomai Wang)
- librados_striper: striping library for librados (Sebastien Ponce)
- libs3: update to latest (Danny Al-Gaaf)
- log: fix derr level (Joao Eduardo Luis)
- logrotate: fix osd log rotation on ubuntu (Sage Weil)
- mds: fix xattr bug triggered by ACLs (Yan, Zheng)
- misc memory leaks, cleanups, fixes (Danny Al-Gaaf, Sahid Ferdjaoui)
- misc suse fixes (Danny Al-Gaaf)
- misc word size fixes (Kevin Cox)
- mon: drop mon- and osd- specific leveldb options (Joao Eduardo Luis)
- mon: ec pool profile fixes (Loic Dachary)
- mon: fix health down messages (Sage Weil)
- mon: fix quorum feature check (#8738, Greg Farnum)
- mon: 'osd crush reweight-subtree ...' (Sage Weil)
- mon, osd: relax client EC support requirements (Sage Weil)
- mon: some instrumentation (Sage Weil)

- objecter: flag operations that are redirected by caching (Sage Weil)
- osd: clean up shard_id_t, shard_t (Loic Dachary)
- osd: fix connection reconnect race (Greg Farnum)
- osd: fix dumps (Joao Eduardo Luis)
- osd: fix erasure-code lib initialization (Loic Dachary)
- osd: fix extent normalization (Adam Crume)
- osd: fix loopback msgr issue (Ma Jianpeng)
- osd: fix LSB release parsing (Danny Al-Gaaf)
- osd: improved backfill priorities (Sage Weil)
- osd: many many core fixes (Samuel Just)
- osd, mon: config sanity checks on start (Sage Weil, Joao Eduardo Luis)
- osd: sharded threadpool to improve parallelism (Somnath Roy)
- osd: simple io prioritization for scrub (Sage Weil)
- osd: simple scrub throttling (Sage Weil)
- osd: tests for bench command (Loic Dachary)
- osd: use xfs hint less frequently (Ilya Dryomov)
- pybind/rados: fix small timeouts (John Spray)
- qa: xfstests updates (Ilya Dryomov)
- rgw: cache bucket info (Yehuda Sadeh)
- rgw: cache decoded user info (Yehuda Sadeh)
- rgw: fix multipart object attr regression (#8452, Yehuda Sadeh)
- rgw: fix radosgw-admin 'show log' command (#8553, Yehuda Sadeh)
- rgw: fix URL decoding (#8702, Brian Rak)
- rgw: handle empty extra pool name (Yehuda Sadeh)
- rpm: do not restart daemons on upgrade (Alfredo Deza)
- rpm: misc packaging fixes for rhel7 (Sandon Van Ness)
- rpm: split ceph-common from ceph (Sandon Van Ness)
- systemd: wrap started daemons in new systemd environment (Sage Weil, Dan Mick)
- sysvinit: less sensitive to failures (Sage Weil)
- upstart: increase max open files limit (Sage Weil)

11.17 v0.82

This is the second post-firefly development release. It includes a range of bug fixes and some usability improvements. There are some MDS debugging and diagnostic tools, an improved 'ceph df', and some OSD backend refactoring and cleanup.

11.17.1 Notable Changes

- ceph-brag: add tox tests (Alfredo Deza)
- common: perfcounters now use atomics and go faster (Sage Weil)
- doc: CRUSH updates (John Wilkins)
- doc: osd primary affinity (John Wilkins)
- doc: pool quotas (John Wilkins)
- doc: pre-flight doc improvements (Kevin Dalley)
- doc: switch to an unencumbered font (Ross Turk)
- doc: update openstack docs (Josh Durgin)
- fix hppa arch build (Dmitry Smirnov)
- init-ceph: continue starting other daemons on crush or mount failure (#8343, Sage Weil)
- keyvaluestore: fix hint crash (#8381, Haomai Wang)
- libcephfs-java: build against older JNI headers (Greg Farnum)
- librados: fix rados_pool_list bounds checks (Sage Weil)
- mds: cephfs-journal-tool (John Spray)
- mds: improve Journaler on-disk format (John Spray)
- mds, libcephfs: use client timestamp for mtime/ctime (Sage Weil)
- mds: misc encoding improvements (John Spray)
- mds: misc fixes for multi-mds (Yan, Zheng)
- mds: OPTracker integration, dump_ops_in_flight (Greg Farnum)
- misc cleanup (Christophe Courtaut)
- mon: fix default replication pool ruleset choice (#8373, John Spray)
- mon: fix set cache_target_full_ratio (#8440, Geoffrey Hartz)
- mon: include per-pool 'max avail' in df output (Sage Weil)
- mon: prevent EC pools from being used with cephfs (Joao Eduardo Luis)
- mon: restore original weight when auto-marked out OSDs restart (Sage Weil)
- mon: use msg header tid for MMonGetVersionReply (Ilya Dryomov)
- osd: fix bogus assert during OSD shutdown (Sage Weil)
- osd: fix clone deletion case (#8334, Sam Just)
- osd: fix filestore removal corner case (#8332, Sam Just)
- osd: fix hang waiting for osdmap (#8338, Greg Farnum)
- osd: fix interval check corner case during peering (#8104, Sam Just)
- osd: fix journal-less operation (Sage Weil)
- osd: include backend information in metadata reported to mon (Sage Weil)
- rest-api: fix help (Ailing Zhang)
- rgw: check entity permission for put_metadata (#8428, Yehuda Sadeh)

11.18 v0.81

This is the first development release since Firefly. It includes a lot of work that we delayed merging while stabilizing things. Lots of new functionality, as well as several fixes that are baking a bit before getting backported.

11.18.1 Upgrading

- CephFS support for the legacy anchor table has finally been removed. Users with file systems created before firefly should ensure that inodes with multiple hard links are modified *prior* to the upgrade to ensure that the backtraces are written properly. For example:

```
sudo find /mnt/cephfs -type f -links +1 -exec touch \{\} \;
```

- Disallow nonsensical 'tier cache-mode' transitions. From this point onward, 'writeback' can only transition to 'forward' and 'forward' can transition to 1) 'writeback' if there are dirty objects, or 2) any if there are no dirty objects.

11.18.2 Notable Changes

- bash completion improvements (Wido den Hollander)
- brag: fixes, improvements (Loic Dachary)
- ceph-disk: handle corrupt volumes (Stuart Longlang)
- ceph-disk: partprobe as needed (Eric Eastman)
- ceph-fuse, libcephfs: asok hooks for handling session resets, timeouts (Yan, Zheng)
- ceph-fuse, libcephfs: improve traceless reply handling (Sage Weil)
- clang build fixes (John Spray, Danny Al-Gaaf)
- config: support G, M, K, etc. suffixes (Joao Eduardo Luis)
- coverity cleanups (Danny Al-Gaaf)
- doc: cache tiering (John Wilkins)
- doc: keystone integration docs (John Wilkins)
- doc: updated simple configuration guides (John Wilkins)
- libcephfs-java: fix gcj-jdk build (Dmitry Smirnov)
- librbd: check error code on cache invalidate (Josh Durgin)
- librbd: new libkrbd library for kernel map/unmap/showmapped (Ilya Dryomov)
- Makefile: fix out of source builds (Stefan Eilemann)
- mds: multi-mds fixes (Yan, Zheng)
- mds: remove legacy anchor table (Yan, Zheng)
- mds: remove legacy discover ino (Yan, Zheng)
- monclient: fix hang (Sage Weil)
- mon: prevent nonsensical cache-mode transitions (Joao Eduardo Luis)
- msgr: avoid big lock when sending (most) messages (Greg Farnum)

- osd: bound osdmap epoch skew between PGs (Sage Weil)
- osd: cache tier flushing fixes for snapped objects (Samuel Just)
- osd: fix agent early finish looping (David Zafman)
- osd: fix flush vs OpContext (Samuel Just)
- osd: fix MarkMeDown and other shutdown races (Sage Weil)
- osd: fix scrub vs cache bugs (Samuel Just)
- osd: fix trim of hitsets (Sage Weil)
- osd, msgr: fast-dispatch of OSD ops (Greg Farnum, Samuel Just)
- osd, objecter: resend ops on last_force_op_resend barrier; fix cache overlay op ordering (Sage Weil)
- osd: remove obsolete classic scrub code (David Zafman)
- osd: scrub PGs with invalid stats (Sage Weil)
- osd: simple snap trimmer throttle (Sage Weil)
- osd: use FIEMAP to inform copy_range (Haomai Wang)
- rbd-fuse: allow exposing single image (Stephen Taylor)
- rbd-fuse: fix unlink (Josh Durgin)
- removed mkcephfs (deprecated since dumpling)
- rgw: bucket link uses instance id (Yehuda Sadeh)
- rgw: fix memory leak following chunk read error (Yehuda Sadeh)
- rgw: fix URL escaping (Yehuda Sadeh)
- rgw: fix user manifest (Yehuda Sadeh)
- rgw: object and bucket rewrite functions to allow restriping old objects (Yehuda Sadeh)
- rgw: prevent multiobject PUT race (Yehuda Sadeh)
- rgw: send user manifest header (Yehuda Sadeh)
- test_librbd_fsx: test krbd as well as librbd (Ilya Dryomov)

11.19 v0.80.9 Firefly

This is a bugfix release for firefly. It fixes a performance regression in librbd, an important CRUSH misbehavior (see below), and several RGW bugs. We have also backported support for flock/fcntl locks to ceph-fuse and libcephfs.

We recommend that all Firefly users upgrade.

For more detailed information, see `the complete changelog`.

11.19.1 Adjusting CRUSH maps

- This point release fixes several issues with CRUSH that trigger excessive data migration when adjusting OSD weights. These are most obvious when a very small weight change (e.g., a change from 0 to .01) triggers a large amount of movement, but the same set of bugs can also lead to excessive (though less noticeable) movement in other cases.

However, because the bug may already have affected your cluster, fixing it may trigger movement *back* to the more correct location. For this reason, you must manually opt-in to the fixed behavior.

In order to set the new tunable to correct the behavior:

```
ceph osd crush set-tunable straw_calc_version 1
```

Note that this change will have no immediate effect. However, from this point forward, any ‘straw’ bucket in your CRUSH map that is adjusted will get non-buggy internal weights, and that transition may trigger some rebalancing.

You can estimate how much rebalancing will eventually be necessary on your cluster with:

```
  ceph osd getcrushmap -o /tmp/cm
  crushtool -i /tmp/cm --num-rep 3 --test --show-mappings > /tmp/a 2>&1
  crushtool -i /tmp/cm --set-straw-calc-version 1 -o /tmp/cm2
  crushtool -i /tmp/cm2 --reweight -o /tmp/cm2
  crushtool -i /tmp/cm2 --num-rep 3 --test --show-mappings > /tmp/b 2>&1
  wc -l /tmp/a                          # num total mappings
  diff -u /tmp/a /tmp/b | grep -c ^+    # num changed mappings

Divide the total number of lines in /tmp/a with the number of lines
changed.  We've found that most clusters are under 10%.

You can force all of this rebalancing to happen at once with::

  ceph osd crush reweight-all

Otherwise, it will happen at some unknown point in the future when
CRUSH weights are next adjusted.
```

11.19.2 Notable Changes

- ceph-fuse: flock, fcntl lock support (Yan, Zheng, Greg Farnum)
- crush: fix straw bucket weight calculation, add straw_calc_version tunable (#10095 Sage Weil)
- crush: fix tree bucket (Rongzu Zhu)
- crush: fix underflow of tree weights (Loic Dachary, Sage Weil)
- crushtool: add –reweight (Sage Weil)
- librbd: complete pending operations before losing image (#10299 Jason Dillaman)
- librbd: fix read caching performance regression (#9854 Jason Dillaman)
- librbd: gracefully handle deleted/renamed pools (#10270 Jason Dillaman)
- mon: fix dump of chooseleaf_vary_r tunable (Sage Weil)
- osd: fix PG ref leak in snaptrimmer on peering (#10421 Kefu Chai)
- osd: handle no-op write with snapshot (#10262 Sage Weil)
- radosgw-admin: create subuser when creating user (#10103 Yehuda Sadeh)
- rgw: change multipart uplaod id magic (#10271 Georgio Dimitrakakis, Yehuda Sadeh)
- rgw: don’t overwrite bucket/object owner when setting ACLs (#10978 Yehuda Sadeh)
- rgw: enable IPv6 for embedded civetweb (#10965 Yehuda Sadeh)
- rgw: fix partial swift GET (#10553 Yehuda Sadeh)

- rgw: fix quota disable (#9907 Dong Lei)
- rgw: index swift keys appropriately (#10471 Hemant Burman, Yehuda Sadeh)
- rgw: make setattrs update bucket index (#5595 Yehuda Sadeh)
- rgw: pass civetweb configurables (#10907 Yehuda Sadeh)
- rgw: remove swift user manifest (DLO) hash calculation (#9973 Yehuda Sadeh)
- rgw: return correct len for 0-len objects (#9877 Yehuda Sadeh)
- rgw: S3 object copy content-type fix (#9478 Yehuda Sadeh)
- rgw: send ETag on S3 object copy (#9479 Yehuda Sadeh)
- rgw: send HTTP status reason explicitly in fastcgi (Yehuda Sadeh)
- rgw: set ulimit -n from sysvinit (el6) init script (#9587 Sage Weil)
- rgw: update swift subuser permission masks when authenticating (#9918 Yehuda Sadeh)
- rgw: URL decode query params correctly (#10271 Georgio Dimitrakakis, Yehuda Sadeh)
- rgw: use attrs when reading object attrs (#10307 Yehuda Sadeh)
- rgw: use rn for http headers (#9254 Benedikt Fraunhofer, Yehuda Sadeh)

11.20 v0.80.8 Firefly

This is a long-awaited bugfix release for firefly. It has several imporant (but relatively rare) OSD peering fixes, performance issues when snapshots are trimmed, several RGW fixes, a paxos corner case fix, and some packaging updates.

We recommend that all users for v0.80.x firefly upgrade when it is convenient to do so.

For more detailed information, see `the complete changelog`.

11.20.1 Notable Changes

- build: remove stack-execute bit from assembled code sections (#10114 Dan Mick)
- ceph-disk: fix dmcrypt key permissions (#9785 Loic Dachary)
- ceph-disk: fix keyring location (#9653 Loic Dachary)
- ceph-disk: make partition checks more robust (#9721 #9665 Loic Dachary)
- ceph: cleanly shut down librados context on shutdown (#8797 Dan Mick)
- common: add $cctid config metavariable (#6228 Adam Crume)
- crush: align rule and ruleset ids (#9675 Xiaoxi Chen)
- crush: fix negative weight bug during create_or_move_item (#9998 Pawel Sadowski)
- crush: fix potential buffer overflow in erasure rules (#9492 Johnu George)
- debian: fix python-ceph -> ceph file movement (Sage Weil)
- libcephfs,ceph-fuse: fix flush tid wraparound bug (#9869 Greg Farnum, Yan, Zheng)
- libcephfs: close fd befure umount (#10415 Yan, Zheng)
- librados: fix crash from C API when read timeout is enabled (#9582 Sage Weil)

- librados: handle reply race with pool deletion (#10372 Sage Weil)
- librbd: cap memory utilization for read requests (Jason Dillaman)
- librbd: do not close a closed parent image on failure (#10030 Jason Dillaman)
- librbd: fix diff tests (#10002 Josh Durgin)
- librbd: protect list_children from invalid pools (#10123 Jason Dillaman)
- make check improvemens (Loic Dachary)
- mds: fix ctime updates (#9514 Greg Farnum)
- mds: fix journal import tool (#10025 John Spray)
- mds: fix rare NULL deref in cap flush handler (Greg Farnum)
- mds: handle unknown lock messages (Yan, Zheng)
- mds: store backtrace for straydir (Yan, Zheng)
- mon: abort startup if disk is full (#9502 Joao Eduardo Luis)
- mon: add paxos instrumentation (Sage Weil)
- mon: fix double-free in rare OSD startup path (Sage Weil)
- mon: fix osdmap trimming (#9987 Sage Weil)
- mon: fix paxos corner cases (#9301 #9053 Sage Weil)
- osd: cancel callback on blacklisted watchers (#8315 Samuel Just)
- osd: cleanly abort set-alloc-hint operations during upgrade (#9419 David Zafman)
- osd: clear rollback PG metadata on PG deletion (#9293 Samuel Just)
- osd: do not abort deep scrub if hinfo is missing (#10018 Loic Dachary)
- osd: erasure-code regression tests (Loic Dachary)
- osd: fix distro metadata reporting for SUSE (#8654 Danny Al-Gaaf)
- osd: fix full OSD checks during backfill (#9574 Samuel Just)
- osd: fix ioprio parsing (#9677 Loic Dachary)
- osd: fix journal direct-io shutdown (#9073 Mark Kirkwood, Ma Jianpeng, Somnath Roy)
- osd: fix journal dump (Ma Jianpeng)
- osd: fix occasional stall during peering or activation (Sage Weil)
- osd: fix past_interval display bug (#9752 Loic Dachary)
- osd: fix rare crash triggered by admin socket dump_ops_in_filght (#9916 Dong Lei)
- osd: fix snap trimming performance issues (#9487 #9113 Samuel Just, Sage Weil, Dan van der Ster, Florian Haas)
- osd: fix snapdir handling on cache eviction (#8629 Sage Weil)
- osd: handle map gaps in map advance code (Sage Weil)
- osd: handle undefined CRUSH results in interval check (#9718 Samuel Just)
- osd: include shard in JSON dump of ghobject (#10063 Loic Dachary)
- osd: make backfill reservation denial handling more robust (#9626 Samuel Just)

- osd: make misdirected op checks handle EC + primary affinity (#9835 Samuel Just, Sage Weil)
- osd: mount XFS with inode64 by default (Sage Weil)
- osd: other misc bugs (#9821 #9875 Samuel Just)
- rgw: add .log to default log path (#9353 Alexandre Marangone)
- rgw: clean up fcgi request context (#10194 Yehuda Sadeh)
- rgw: convet header underscores to dashes (#9206 Yehuda Sadeh)
- rgw: copy object data if copy target is in different pool (#9039 Yehuda Sadeh)
- rgw: don't try to authenticate CORS peflight request (#8718 Robert Hubbard, Yehuda Sadeh)
- rgw: fix civetweb URL decoding (#8621 Yehuda Sadeh)
- rgw: fix hash calculation during PUT (Yehuda Sadeh)
- rgw: fix misc bugs (#9089 #9201 Yehuda Sadeh)
- rgw: fix object tail test (#9226 Sylvain Munaut, Yehuda Sadeh)
- rgw: make sysvinit script run rgw under systemd context as needed (#10125 Loic Dachary)
- rgw: separate civetweb log from rgw log (Yehuda Sadeh)
- rgw: set length for keystone token validations (#7796 Mark Kirkwood, Yehuda Sadeh)
- rgw: subuser creation fixes (#8587 Yehuda Sadeh)
- rpm: misc packaging improvements (Sandon Van Ness, Dan Mick, Erik Logthenberg, Boris Ranto)
- rpm: use standard udev rules for CentOS7/RHEL7 (#9747 Loic Dachary)

11.21 v0.80.7 Firefly

This release fixes a few critical issues with v0.80.6, particularly with clusters running mixed versions.

We recommend that all v0.80.x Firefly users upgrade to this release.

For more detailed information, see `the complete changelog`.

11.21.1 Notable Changes

- osd: fix invalid memory reference in log trimming (#9731 Samuel Just)
- osd: fix use-after-free in cache tiering code (#7588 Sage Weil)
- osd: remove bad backfill assertion for mixed-version clusters (#9696 Samuel Just)

11.22 v0.80.6 Firefly

This is a major bugfix release for firefly, fixing a range of issues in the OSD and monitor, particularly with cache tiering. There are also important fixes in librados, with the watch/notify mechanism used by librbd, and in radosgw.

A few pieces of new functionality of been backported, including improved 'ceph df' output (view amount of writeable space per pool), support for non-default cluster names when using sysvinit or systemd, and improved (and fixed) support for dmcrypt.

We recommend that all v0.80.x Firefly users upgrade to this release.

For more detailed information, see `the complete changelog`.

11.22.1 Notable Changes

- build: fix atomic64_t on i386 (#8969 Sage Weil)
- build: fix build on alpha (Michael Cree, Dmitry Smirnov)
- build: fix build on hppa (Dmitry Smirnov)
- build: fix yasm detection on x32 arch (Sage Weil)
- ceph-disk: fix 'list' function with dmcrypt (Sage Weil)
- ceph-disk: fix dmcrypt support (Alfredo Deza)
- ceph: allow non-default cluster to be specified (#8944)
- common: fix dup log messages to mon (#9080 Sage Weil)
- global: write pid file when -f is used (systemd, upstart) (Alexandre Oliva)
- librados: fix crash when read timeout is enabled (#9362 Matthias Kiefer, Sage Weil)
- librados: fix lock leaks in error paths (#9022 Pavan Rallabhandi)
- librados: fix watch resend on PG acting set change (#9220 Samuel Just)
- librados: python: fix aio_read handling with 0 (Mohammad Salehe)
- librbd: add interface to invalidate cached data (Josh Durgin)
- librbd: fix crash when using clone of flattened image (#8845 Josh Durgin)
- librbd: fix error path cleanup on open (#8912 Josh Durgin)
- librbd: fix null pointer check (Danny Al-Gaaf)
- librbd: limit dirty object count (Haomai Wang)
- mds: fix rstats for root and mdsdir (Yan, Zheng)
- mon: add 'get' command for new cache tier pool properties (Joao Eduardo Luis)
- mon: add 'osd pool get-quota' (#8523 Joao Eduardo Luis)
- mon: add cluster fingerprint (Sage Weil)
- mon: disallow nonsensical cache-mode transitions (#8155 Joao Eduardo Luis)
- mon: fix cache tier rounding error on i386 (Sage Weil)
- mon: fix occasional memory leak (#9176 Sage Weil)
- mon: fix reported latency for 'osd perf' (#9269 Samuel Just)
- mon: include 'max avail' in 'ceph df' output (Sage Weil, Xioaxi Chen)
- mon: persistently mark pools where scrub may find incomplete clones (#8882 Sage Weil)
- mon: preload erasure plugins (Loic Dachary)
- mon: prevent cache-specific settings on non-tier pools (#8696 Joao Eduardo Luis)
- mon: reduce log spam (Aanchal Agrawal, Sage Weil)
- mon: warn when cache pools have no hit_sets enabled (Sage Weil)

- msgr: fix trivial memory leak (Sage Weil)
- osd: automatically scrub PGs with invalid stats (#8147 Sage Weil)
- osd: avoid sharing PG metadata that is not durable (Samuel Just)
- osd: cap hit_set size (#9339 Samuel Just)
- osd: create default erasure profile if needed (#8601 Loic Dachary)
- osd: dump tid as JSON int (not string) where appropriate (Joao Eduardo Luis)
- osd: encode blacklist in deterministic order (#9211 Sage Weil)
- osd: fix behavior when cache tier has no hit_sets enabled (#8982 Sage Weil)
- osd: fix cache tier flushing of snapshots (#9054 Samuel Just)
- osd: fix cache tier op ordering when going from full to non-full (#8931 Sage Weil)
- osd: fix crash on dup recovery reservation (#8863 Sage Weil)
- osd: fix division by zero when pg_num adjusted with no OSDs (#9052 Sage Weil)
- osd: fix hint crash in experimental keyvaluestore_dev backend (Hoamai Wang)
- osd: fix leak in copyfrom cancellation (#8894 Samuel Just)
- osd: fix locking for copyfrom finish (#8889 Sage Weil)
- osd: fix long filename handling in backend (#8701 Sage Weil)
- osd: fix min_size check with backfill (#9497 Samuel Just)
- osd: fix mount/remount sync race (#9144 Sage Weil)
- osd: fix object listing + erasure code bug (Guang Yang)
- osd: fix race on reconnect to failed OSD (#8944 Greg Farnum)
- osd: fix recovery reservation deadlock (Samuel Just)
- osd: fix tiering agent arithmetic for negative values (#9082 Karan Singh)
- osd: improve shutdown order (#9218 Sage Weil)
- osd: improve subop discard logic (#9259 Samuel Just)
- osd: introduce optional sleep, io priority for scrub and snap trim (Sage Weil)
- osd: make scrub check for and remove stale erasure-coded objects (Samuel Just)
- osd: misc fixes (#9481 #9482 #9179 Sameul Just)
- osd: mix keyvaluestore_dev improvements (Haomai Wang)
- osd: only require CRUSH features for rules that are used (#8963 Sage Weil)
- osd: preload erasure plugins on startup (Loic Dachary)
- osd: prevent PGs from falling behind when consuming OSDMaps (#7576 Sage Weil)
- osd: prevent old clients from using tiered pools (#8714 Sage Weil)
- osd: set min_size on erasure pools to data chunk count (Sage Weil)
- osd: trim old erasure-coded objects more aggressively (Samuel Just)
- rados: enforce erasure code alignment (Lluis Pamies-Juarez)
- rgw: align object stripes with erasure pool alignment (#8442 Yehuda Sadeh)

- rgw: don't send error body on HEAD for civetweb (#8539 Yehuda Sadeh)
- rgw: fix crash in CORS preflight request (Yehuda Sadeh)
- rgw: fix decoding of + in URL (#8702 Brian Rak)
- rgw: fix object removal on object create (#8972 Patrycja Szabowska, Yehuda Sadeh)
- systemd: use systemd-run when starting radosgw (JuanJose Galvez)
- sysvinit: support non-default cluster name (Alfredo Deza)

11.23 v0.80.5 Firefly

This release fixes a few important bugs in the radosgw and fixes several packaging and environment issues, including OSD log rotation, systemd environments, and daemon restarts on upgrade.

We recommend that all v0.80.x Firefly users upgrade, particularly if they are using upstart, systemd, or radosgw.

11.23.1 Notable Changes

- ceph-dencoder: do not needlessly link to librgw, librados, etc. (Sage Weil)
- do not needlessly link binaries to leveldb (Sage Weil)
- mon: fix mon crash when no auth keys are present (#8851, Joao Eduardo Luis)
- osd: fix cleanup (and avoid occasional crash) during shutdown (#7981, Sage Weil)
- osd: fix log rotation under upstart (Sage Weil)
- rgw: fix multipart upload when object has irregular size (#8846, Yehuda Sadeh, Sylvain Munaut)
- rgw: improve bucket listing S3 compatibility (#8858, Yehuda Sadeh)
- rgw: improve delimited bucket listing (Yehuda Sadeh)
- rpm: do not restart daemons on upgrade (#8849, Alfredo Deza)

For more detailed information, see `the complete changelog`.

11.24 v0.80.4 Firefly

This Firefly point release fixes an potential data corruption problem when ceph-osd daemons run on top of XFS and service Firefly librbd clients. A recently added allocation hint that RBD utilizes triggers an XFS bug on some kernels (Linux 3.2, and likely others) that leads to data corruption and deep-scrub errors (and inconsistent PGs). This release avoids the situation by disabling the allocation hint until we can validate which kernels are affected and/or are known to be safe to use the hint on.

We recommend that all v0.80.x Firefly users urgently upgrade, especially if they are using RBD.

11.24.1 Notable Changes

- osd: disable XFS extsize hint by default (#8830, Samuel Just)
- rgw: fix extra data pool default name (Yehuda Sadeh)

For more detailed information, see `the complete changelog`.

11.25 v0.80.3 Firefly

This is the third Firefly point release. It includes a single fix for a radosgw regression that was discovered in v0.80.2 right after it was released.

We recommand that all v0.80.x Firefly users upgrade.

11.25.1 Notable Changes

- radosgw: fix regression in manifest decoding (#8804, Sage Weil)

For more detailed information, see `the complete changelog`.

11.26 v0.80.2 Firefly

This is the second Firefly point release. It contains a range of important fixes, including several bugs in the OSD cache tiering, some compatibility checks that affect upgrade situations, several radosgw bugs, and an irritating and unnecessary feature bit check that prevents older clients from communicating with a cluster with any erasure coded pools.

One someone large change in this point release is that the ceph RPM package is separated into a ceph and ceph-common package, similar to Debian. The ceph-common package contains just the client libraries without any of the server-side daemons.

We recommend that all v0.80.x Firefly users skip this release and use v0.80.3.

11.26.1 Notable Changes

- ceph-disk: better debug logging (Alfredo Deza)
- ceph-disk: fix preparation of OSDs with dmcrypt (#6700, Stephen F Taylor)
- ceph-disk: partprobe on prepare to fix dm-crypt (#6966, Eric Eastman)
- do not require ERASURE_CODE feature from clients (#8556, Sage Weil)
- libcephfs-java: build with older JNI headers (Greg Farnum)
- libcephfs-java: fix build with gcj-jdk (Dmitry Smirnov)
- librados: fix osd op tid for redirected ops (#7588, Samuel Just)
- librados: fix rados_pool_list buffer bounds checks (#8447, Sage Weil)
- librados: resend ops when pool overlay changes (#8305, Sage Weil)
- librbd, ceph-fuse: reduce CPU overhead for clean object check in cache (Haomai Wang)
- mon: allow deletion of cephfs pools (John Spray)
- mon: fix default pool ruleset choice (#8373, John Spray)
- mon: fix health summary for mon low disk warning (Sage Weil)
- mon: fix 'osd pool set <pool> cache_target_full_ratio' (Geoffrey Hartz)
- mon: fix quorum feature check (Greg Farnum)
- mon: fix request forwarding in mixed firefly+dumpling clusters 9#8727, Joao Eduardo Luis)

- mon: fix rule vs ruleset check in 'osd pool set ... crush_ruleset' command (John Spray)
- mon: make osd 'down' count accurate (Sage Weil)
- mon: set 'next commit' in primary-affinity reply (Ilya Dryomov)
- mon: verify CRUSH features are supported by all mons (#8738, Greg Farnum)
- msgr: fix sequence negotiation during connection reset (Guang Yang)
- osd: block scrub on blocked objects (#8011, Samuel Just)
- osd: call XFS hint ioctl less often (#8241, Ilya Dryomov)
- osd: copy xattr spill out marker on clone (Haomai Wang)
- osd: fix flush of snapped objects (#8334, Samuel Just)
- osd: fix hashindex restart of merge operation (#8332, Samuel Just)
- osd: fix osdmap subscription bug causing startup hang (Greg Farnum)
- osd: fix potential null deref (#8328, Sage Weil)
- osd: fix shutdown race (#8319, Sage Weil)
- osd: handle 'none' in CRUSH results properly during peering (#8507, Samuel Just)
- osd: set no spill out marker on new objects (Greg Farnum)
- osd: skip op ordering debug checks on tiered pools (#8380, Sage Weil)
- rados: enforce 'put' alignment (Lluis Pamies-Juarez)
- rest-api: fix for 'rx' commands (Ailing Zhang)
- rgw: calc user manifest etag and fix check (#8169, #8436, Yehuda Sadeh)
- rgw: fetch attrs on multipart completion (#8452, Yehuda Sadeh, Sylvain Munaut)
- rgw: fix buffer overflow for long instance ids (#8608, Yehuda Sadeh)
- rgw: fix entity permission check on metadata put (#8428, Yehuda Sadeh)
- rgw: fix multipart retry race (#8269, Yehuda Sadeh)
- rpm: split ceph into ceph and ceph-common RPMs (Sandon Van Ness, Dan Mick)
- sysvinit: continue startin daemons after failure doing mount (#8554, Sage Weil)

For more detailed information, see `the complete changelog`.

11.27 v0.80.1 Firefly

This first Firefly point release fixes a few bugs, the most visible being a problem that prevents scrub from completing in some cases.

11.27.1 Notable Changes

- osd: revert incomplete scrub fix (Samuel Just)
- rgw: fix stripe calculation for manifest objects (Yehuda Sadeh)
- rgw: improve handling, memory usage for abort reads (Yehuda Sadeh)

- rgw: send Swift user manifest HTTP header (Yehuda Sadeh)
- libcephfs, ceph-fuse: expose MDS session state via admin socket (Yan, Zheng)
- osd: add simple throttle for snap trimming (Sage Weil)
- monclient: fix possible hang from ill-timed monitor connection failure (Sage Weil)
- osd: fix trimming of past HitSets (Sage Weil)
- osd: fix whiteouts for non-writeback cache modes (Sage Weil)
- osd: prevent divide by zero in tiering agent (David Zafman)
- osd: prevent busy loop when tiering agent can do no work (David Zafman)

For more detailed information, see `the complete changelog`.

11.28 v0.80 Firefly

This release will form the basis for our long-term supported release Firefly, v0.80.x. The big new features are support for erasure coding and cache tiering, although a broad range of other features, fixes, and improvements have been made across the code base. Highlights include:

- *Erasure coding*: support for a broad range of erasure codes for lower storage overhead and better data durability.
- *Cache tiering*: support for creating 'cache pools' that store hot, recently accessed objects with automatic demotion of colder data to a base tier. Typically the cache pool is backed by faster storage devices like SSDs.
- *Primary affinity*: Ceph now has the ability to skew selection of OSDs as the "primary" copy, which allows the read workload to be cheaply skewed away from parts of the cluster without migrating any data.
- *Key/value OSD backend* (experimental): An alternative storage backend for Ceph OSD processes that puts all data in a key/value database like leveldb. This provides better performance for workloads dominated by key/value operations (like radosgw bucket indices).
- *Standalone radosgw* (experimental): The radosgw process can now run in a standalone mode without an apache (or similar) web server or fastcgi. This simplifies deployment and can improve performance.

We expect to maintain a series of stable releases based on v0.80 Firefly for as much as a year. In the meantime, development of Ceph continues with the next release, Giant, which will feature work on the CephFS distributed file system, more alternative storage backends (like RocksDB and f2fs), RDMA support, support for pyramid erasure codes, and additional functionality in the block device (RBD) like copy-on-read and multisite mirroring.

11.28.1 Upgrade Sequencing

- If your existing cluster is running a version older than v0.67 Dumpling, please first upgrade to the latest Dumpling release before upgrading to v0.80 Firefly. Please refer to the *Dumpling upgrade* documentation.
- We recommand adding the following to the [mon] section of your ceph.conf prior to upgrade:

```
mon warn on legacy crush tunables = false
```

  This will prevent health warnings due to the use of legacy CRUSH placement. Although it is possible to rebalance existing data across your cluster (see the upgrade notes below), we do not normally recommend it for production environments as a large amount of data will move and there is a significant performance impact from the rebalancing.

- Upgrade daemons in the following order:

1. Monitors
2. OSDs
3. MDSs and/or radosgw

If the ceph-mds daemon is restarted first, it will wait until all OSDs have been upgraded before finishing its startup sequence. If the ceph-mon daemons are not restarted prior to the ceph-osd daemons, they will not correctly register their new capabilities with the cluster and new features may not be usable until they are restarted a second time.

- Upgrade radosgw daemons together. There is a subtle change in behavior for multipart uploads that prevents a multipart request that was initiated with a new radosgw from being completed by an old radosgw.

11.28.2 Upgrading from v0.79

- OSDMap's json-formatted dump changed for keys 'full' and 'nearfull'. What was previously being outputted as 'true' or 'false' strings are now being outputted 'true' and 'false' booleans according to json syntax.
- HEALTH_WARN on 'mon osd down out interval == 0'. Having this option set to zero on the leader acts much like having the 'noout' flag set. This warning will only be reported if the monitor getting the 'health' or 'status' request has this option set to zero.
- Monitor 'auth' commands now require the mon 'x' capability. This matches dumpling v0.67.x and earlier, but differs from emperor v0.72.x.
- A librados WATCH operation on a non-existent object now returns ENOENT; previously it did not.
- Librados interface change: As there are no partial writes, the rados_write() and rados_append() operations now return 0 on success like rados_write_full() always has. This includes the C++ interface equivalents and AIO return values for the aio variants.
- The radosgw init script (sysvinit) how requires that the 'host = ...' line in ceph.conf, if present, match the short hostname (the output of 'hostname -s'), not the fully qualified hostname or the (occasionally non-short) output of 'hostname'. Failure to adjust this when upgrading from emperor or dumpling may prevent the radosgw daemon from starting.

11.28.3 Upgrading from v0.72 Emperor

- See notes above.
- The 'ceph -s' or 'ceph status' command's 'num_in_osds' field in the JSON and XML output has been changed from a string to an int.
- The recently added 'ceph mds set allow_new_snaps' command's syntax has changed slightly; it is now 'ceph mds set allow_new_snaps true'. The 'unset' command has been removed; instead, set the value to 'false'.
- The syntax for allowing snapshots is now 'mds set allow_new_snaps <true|false>' instead of 'mds <set,unset> allow_new_snaps'.
- 'rbd ls' on a pool which never held rbd images now exits with code 0. It outputs nothing in plain format, or an empty list in non-plain format. This is consistent with the behavior for a pool which used to hold images, but contains none. Scripts relying on this behavior should be updated.
- The MDS requires a new OSD operation TMAP2OMAP, added in this release. When upgrading, be sure to upgrade and restart the ceph-osd daemons before the ceph-mds daemon. The MDS will refuse to start if any up OSDs do not support the new feature.
- The 'ceph mds set_max_mds N' command is now deprecated in favor of 'ceph mds set max_mds N'.

- The ‘osd pool create ...’ syntax has changed for erasure pools.
- The default CRUSH rules and layouts are now using the ‘bobtail’ tunables and defaults. Upgaded clusters using the old values will now present with a health WARN state. This can be disabled by adding ‘mon warn on legacy crush tunables = false’ to ceph.conf and restarting the monitors. Alternatively, you can switch to the new tunables with ‘ceph osd crush tunables firefly,’ but keep in mind that this will involve moving a *significant* portion of the data already stored in the cluster and in a large cluster may take several days to complete. We do not recommend adjusting tunables on a production cluster.
- We now default to the ‘bobtail’ CRUSH tunable values that are first supported by Ceph clients in bobtail (v0.56) and Linux kernel version v3.9. If you plan to access a newly created Ceph cluster with an older kernel client, you should use ‘ceph osd crush tunables legacy’ to switch back to the legacy behavior. Note that making that change will likely result in some data movement in the system, so adjust the setting before populating the new cluster with data.
- We now set the HASHPSPOOL flag on newly created pools (and new clusters) by default. Support for this flag first appeared in v0.64; v0.67 Dumpling is the first major release that supports it. It is first supported by the Linux kernel version v3.9. If you plan to access a newly created Ceph cluster with an older kernel or clients (e.g, librados, librbd) from a pre-dumpling Ceph release, you should add ‘osd pool default flag hashpspool = false’ to the ‘[global]’ section of your ‘ceph.conf’ prior to creating your monitors (e.g., after ‘ceph-deploy new’ but before ‘ceph-deploy mon create ...’).
- The configuration option ‘osd pool default crush rule’ is deprecated and replaced with ‘osd pool default crush replicated ruleset’. ‘osd pool default crush rule’ takes precedence for backward compatibility and a deprecation warning is displayed when it is used.
- As part of fix for #6796, ‘ceph osd pool set <pool> <var> <arg>’ now receives <arg> as an integer instead of a string. This affects how ‘hashpspool’ flag is set/unset: instead of ‘true’ or ‘false’, it now must be ‘0’ or ‘1’.
- The behavior of the CRUSH ‘indep’ choose mode has been changed. No ceph cluster should have been using this behavior unless someone has manually extracted a crush map, modified a CRUSH rule to replace ‘firstn’ with ‘indep’, recompiled, and reinjected the new map into the cluster. If the ‘indep’ mode is currently in use on a cluster, the rule should be modified to use ‘firstn’ instead, and the administrator should wait until any data movement completes before upgrading.
- The ‘osd dump’ command now dumps pool snaps as an array instead of an object.

11.28.4 Upgrading from v0.67 Dumpling

- See notes above.
- ceph-fuse and radosgw now use the same default values for the admin socket and log file paths that the other daemons (ceph-osd, ceph-mon, etc.) do. If you run these daemons as non-root, you may need to adjust your ceph.conf to disable these options or to adjust the permissions on /var/run/ceph and /var/log/ceph.
- The MDS now disallows snapshots by default as they are not considered stable. The command ‘ceph mds set allow_snaps’ will enable them.
- For clusters that were created before v0.44 (pre-argonaut, Spring 2012) and store radosgw data, the auto-upgrade from TMAP to OMAP objects has been disabled. Before upgrading, make sure that any buckets created on pre-argonaut releases have been modified (e.g., by PUTing and then DELETEing an object from each bucket). Any cluster created with argonaut (v0.48) or a later release or not using radosgw never relied on the automatic conversion and is not affected by this change.
- Any direct users of the ‘tmap’ portion of the librados API should be aware that the automatic tmap -> omap conversion functionality has been removed.
- Most output that used K or KB (e.g., for kilobyte) now uses a lower-case k to match the official SI convention. Any scripts that parse output and check for an upper-case K will need to be modified.

- librados::Rados::pool_create_async() and librados::Rados::pool_delete_async() don't drop a reference to the completion object on error, caller needs to take care of that. This has never really worked correctly and we were leaking an object
- 'ceph osd crush set <id> <weight> <loc..>' no longer adds the osd to the specified location, as that's a job for 'ceph osd crush add'. It will however continue to work just the same as long as the osd already exists in the crush map.
- The OSD now enforces that class write methods cannot both mutate an object and return data. The rbd.assign_bid method, the lone offender, has been removed. This breaks compatibility with pre-bobtail librbd clients by preventing them from creating new images.
- librados now returns on commit instead of ack for synchronous calls. This is a bit safer in the case where both OSDs and the client crash, and is probably how it should have been acting from the beginning. Users are unlikely to notice but it could result in lower performance in some circumstances. Those who care should switch to using the async interfaces, which let you specify safety semantics precisely.
- The C++ librados AioComplete::get_version() method was incorrectly returning an int (usually 32-bits). To avoid breaking library compatibility, a get_version64() method is added that returns the full-width value. The old method is deprecated and will be removed in a future release. Users of the C++ librados API that make use of the get_version() method should modify their code to avoid getting a value that is truncated from 64 to to 32 bits.

11.28.5 Notable changes since v0.79

- ceph-fuse, libcephfs: fix several caching bugs (Yan, Zheng)
- ceph-fuse: trim inodes in response to mds memory pressure (Yan, Zheng)
- librados: fix inconsistencies in API error values (David Zafman)
- librados: fix watch operations with cache pools (Sage Weil)
- librados: new snap rollback operation (David Zafman)
- mds: fix respawn (John Spray)
- mds: misc bugs (Yan, Zheng)
- mds: misc multi-mds fixes (Yan, Zheng)
- mds: use shared_ptr for requests (Greg Farnum)
- mon: fix peer feature checks (Sage Weil)
- mon: require 'x' mon caps for auth operations (Joao Luis)
- mon: shutdown when removed from mon cluster (Joao Luis)
- msgr: fix locking bug in authentication (Josh Durgin)
- osd: fix bug in journal replay/restart (Sage Weil)
- osd: many many many bug fixes with cache tiering (Samuel Just)
- osd: track omap and hit_set objects in pg stats (Samuel Just)
- osd: warn if agent cannot enable due to invalid (post-split) stats (Sage Weil)
- rados bench: track metadata for multiple runs separately (Guang Yang)
- rgw: fixed subuser modify (Yehuda Sadeh)
- rpm: fix redhat-lsb dependency (Sage Weil, Alfredo Deza)

11.28.6 Notable changes since v0.72 Emperor

- buffer: some zero-copy groundwork (Josh Durgin)
- build: misc improvements (Ken Dreyer)
- ceph-conf: stop creating bogus log files (Josh Durgin, Sage Weil)
- ceph-crush-location: new hook for setting CRUSH location of osd daemons on start)
- ceph-disk: avoid fd0 (Loic Dachary)
- ceph-disk: generalize path names, add tests (Loic Dachary)
- ceph-disk: misc improvements for puppet (Loic Dachary)
- ceph-disk: several bug fixes (Loic Dachary)
- ceph-fuse: fix race for sync reads (Sage Weil)
- ceph-fuse, libcephfs: fix several caching bugs (Yan, Zheng)
- ceph-fuse: trim inodes in response to mds memory pressure (Yan, Zheng)
- ceph-kvstore-tool: expanded command set and capabilities (Joao Eduardo Luis)
- ceph.spec: fix build dependency (Loic Dachary)
- common: bloom filter improvements (Sage Weil)
- common: check preexisting admin socket for active daemon before removing (Loic Dachary)
- common: fix aligned buffer allocation (Loic Dachary)
- common: fix authentication on big-endian architectures (Dan Mick)
- common: fix config variable substitution (Loic Dachary)
- common: portability changes to support libc++ (Noah Watkins)
- common: switch to unordered_map from hash_map (Noah Watkins)
- config: recursive metavariable expansion (Loic Dachary)
- crush: default to bobtail tunables (Sage Weil)
- crush: fix off-by-one error in recent refactor (Sage Weil)
- crush: many additional tests (Loic Dachary)
- crush: misc fixes, cleanups (Loic Dachary)
- crush: new rule steps to adjust retry attempts (Sage Weil)
- crush, osd: s/rep/replicated/ for less confusion (Loic Dachary)
- crush: refactor descend_once behavior; support set_choose*_tries for replicated rules (Sage Weil)
- crush: usability and test improvements (Loic Dachary)
- debian: change directory ownership between ceph and ceph-common (Sage Weil)
- debian: integrate misc fixes from downstream packaging (James Page)
- doc: big update to install docs (John Wilkins)
- doc: many many install doc improvements (John Wilkins)
- doc: many many updates (John Wilkins)
- doc: misc fixes (David Moreau Simard, Kun Huang)

- erasure-code: improve buffer alignment (Loic Dachary)
- erasure-code: rewrite region-xor using vector operations (Andreas Peters)
- init: fix startup ordering/timeout problem with OSDs (Dmitry Smirnov)
- libcephfs: fix resource leak (Zheng Yan)
- librados: add C API coverage for atomic write operations (Christian Marie)
- librados: fix inconsistencies in API error values (David Zafman)
- librados: fix throttle leak (and eventual deadlock) (Josh Durgin)
- librados: fix watch operations with cache pools (Sage Weil)
- librados: new snap rollback operation (David Zafman)
- librados, osd: new TMAP2OMAP operation (Yan, Zheng)
- librados: read directly into user buffer (Rutger ter Borg)
- librbd: fix use-after-free aio completion bug #5426 (Josh Durgin)
- librbd: localize/distribute parent reads (Sage Weil)
- librbd: skip zeroes/holes when copying sparse images (Josh Durgin)
- mailmap: affiliation updates (Loic Dachary)
- mailmap updates (Loic Dachary)
- many portability improvements (Noah Watkins)
- many unit test improvements (Loic Dachary)
- mds: always store backtrace in default pool (Yan, Zheng)
- mds: cope with MDS failure during creation (John Spray)
- mds: fix cap migration behavior (Yan, Zheng)
- mds: fix client session flushing (Yan, Zheng)
- mds: fix crash from client sleep/resume (Zheng Yan)
- mds: fix many many multi-mds bugs (Yan, Zheng)
- mds: fix readdir end check (Zheng Yan)
- mds: fix Resetter locking (Alexandre Oliva)
- mds: fix respawn (John Spray)
- mds: inline data support (Li Wang, Yunchuan Wen)
- mds: misc bugs (Yan, Zheng)
- mds: misc fixes for directory fragments (Zheng Yan)
- mds: misc fixes for larger directories (Zheng Yan)
- mds: misc fixes for multiple MDSs (Zheng Yan)
- mds: misc multi-mds fixes (Yan, Zheng)
- mds: remove .ceph directory (John Spray)
- mds: store directories in omap instead of tmap (Yan, Zheng)
- mds: update old-format backtraces opportunistically (Zheng Yan)

- mds: use shared_ptr for requests (Greg Farnum)
- misc cleanups from coverity (Xing Lin)
- misc coverity fixes, cleanups (Danny Al-Gaaf)
- misc coverity fixes (Xing Lin, Li Wang, Danny Al-Gaaf)
- misc portability fixes (Noah Watkins, Alan Somers)
- misc portability fixes (Noah Watkins, Christophe Courtaut, Alan Somers, huanjun)
- misc portability work (Noah Watkins)
- mon: add erasure profiles and improve erasure pool creation (Loic Dachary)
- mon: add 'mon getmap EPOCH' (Joao Eduardo Luis)
- mon: allow adjustment of cephfs max file size via 'ceph mds set max_file_size' (Sage Weil)
- mon: allow debug quorum_{enter,exit} commands via admin socket
- mon: 'ceph osd pg-temp ...' and primary-temp commands (Ilya Dryomov)
- mon: change mds allow_new_snaps syntax to be more consistent (Sage Weil)
- mon: clean up initial crush rule creation (Loic Dachary)
- mon: collect misc metadata about osd (os, kernel, etc.), new 'osd metadata' command (Sage Weil)
- mon: do not create erasure rules by default (Sage Weil)
- mon: do not generate spurious MDSMaps in certain cases (Sage Weil)
- mon: do not use keyring if auth = none (Loic Dachary)
- mon: fix peer feature checks (Sage Weil)
- mon: fix pg_temp leaks (Joao Eduardo Luis)
- mon: fix pool count in 'ceph -s' output (Sage Weil)
- mon: handle more whitespace (newline, tab) in mon capabilities (Sage Weil)
- mon: improve (replicate or erasure) pool creation UX (Loic Dachary)
- mon: infrastructure to handle mixed-version mon cluster and cli/rest API (Greg Farnum)
- mon: MForward tests (Loic Dachary)
- mon: mkfs now idempotent (Loic Dachary)
- mon: only seed new osdmaps to current OSDs (Sage Weil)
- mon, osd: create erasure style crush rules (Loic Dachary, Sage Weil)
- mon: 'osd crush show-tunables' (Sage Weil)
- mon: 'osd dump' dumps pool snaps as array, not object (Dan Mick)
- mon, osd: new 'erasure' pool type (still not fully supported)
- mon: persist quorum features to disk (Greg Farnum)
- mon: prevent extreme changes in pool pg_num (Greg Farnum)
- mon: require 'x' mon caps for auth operations (Joao Luis)
- mon: shutdown when removed from mon cluster (Joao Luis)
- mon: take 'osd pool set ...' value as an int, not string (Joao Eduardo Luis)

- mon: track osd features in OSDMap (Joao Luis, David Zafman)
- mon: trim MDSMaps (Joao Eduardo Luis)
- mon: warn if crush has non-optimal tunables (Sage Weil)
- mount.ceph: add -n for autofs support (Steve Stock)
- msgr: fix locking bug in authentication (Josh Durgin)
- msgr: fix messenger restart race (Xihui He)
- msgr: improve connection error detection between clients and monitors (Greg Farnum, Sage Weil)
- osd: add/fix CPU feature detection for jerasure (Loic Dachary)
- osd: add HitSet tracking for read ops (Sage Weil, Greg Farnum)
- osd: avoid touching leveldb for some xattrs (Haomai Wang, Sage Weil)
- osd: backfill to multiple targets (David Zafman)
- osd: backfill to osds not in acting set (David Zafman)
- osd: cache pool support for snapshots (Sage Weil)
- osd: client IO path changes for EC (Samuel Just)
- osd: default to 3x replication
- osd: do not include backfill targets in acting set (David Zafman)
- osd: enable new hashpspool layout by default (Sage Weil)
- osd: erasure plugin benchmarking tool (Loic Dachary)
- osd: fix and cleanup misc backfill issues (David Zafman)
- osd: fix bug in journal replay/restart (Sage Weil)
- osd: fix copy-get omap bug (Sage Weil)
- osd: fix linux kernel version detection (Ilya Dryomov)
- osd: fix memstore segv (Haomai Wang)
- osd: fix object_info_t encoding bug from emperor (Sam Just)
- osd: fix omap_clear operation to not zap xattrs (Sam Just, Yan, Zheng)
- osd: fix several bugs with tier infrastructure
- osd: fix throttle thread (Haomai Wang)
- osd: fix XFS detection (Greg Farnum, Sushma Gurram)
- osd: generalize scrubbing infrastructure to allow EC (David Zafman)
- osd: handle more whitespace (newline, tab) in osd capabilities (Sage Weil)
- osd: ignore num_objects_dirty on scrub for old pools (Sage Weil)
- osd: improved scrub checks on clones (Sage Weil, Sam Just)
- osd: improve locking in fd lookup cache (Samuel Just, Greg Farnum)
- osd: include more info in pg query result (Sage Weil)
- osd, librados: fix full cluster handling (Josh Durgin)
- osd: many erasure fixes (Sam Just)

- osd: many many many bug fixes with cache tiering (Samuel Just)
- osd: move to jerasure2 library (Loic Dachary)
- osd: new 'chassis' type in default crush hierarchy (Sage Weil)
- osd: new keyvaluestore-dev backend based on leveldb (Haomai Wang)
- osd: new OSDMap encoding (Greg Farnum)
- osd: new tests for erasure pools (David Zafman)
- osd: preliminary cache pool support (no snaps) (Greg Farnum, Sage Weil)
- osd: reduce scrub lock contention (Guang Yang)
- osd: requery unfound on stray notify (#6909) (Samuel Just)
- osd: some PGBackend infrastructure (Samuel Just)
- osd: support for new 'memstore' (memory-backed) backend (Sage Weil)
- osd: track erasure compatibility (David Zafman)
- osd: track omap and hit_set objects in pg stats (Samuel Just)
- osd: warn if agent cannot enable due to invalid (post-split) stats (Sage Weil)
- rados: add 'crush location', smart replica selection/balancing (Sage Weil)
- rados bench: track metadata for multiple runs separately (Guang Yang)
- rados: some performance optimizations (Yehuda Sadeh)
- rados tool: fix listomapvals (Josh Durgin)
- rbd: add 'rbdmap' init script for mapping rbd images on book (Adam Twardowski)
- rbd: add rbdmap support for upstart (Laurent Barbe)
- rbd: expose kernel rbd client options via 'rbd map' (Ilya Dryomov)
- rbd: fix bench-write command (Hoamai Wang)
- rbd: make 'rbd list' return empty list and success on empty pool (Josh Durgin)
- rbd: prevent deletion of images with watchers (Ilya Dryomov)
- rbd: support for 4096 mapped devices, up from ~250 (Ilya Dryomov)
- rest-api: do not fail when no OSDs yet exist (Dan Mick)
- rgw: add 'status' command to sysvinit script (David Moreau Simard)
- rgw: allow multiple frontends (Yehuda Sadeh)
- rgw: allow use of an erasure data pool (Yehuda Sadeh)
- rgw: convert bucket info to new format on demand (Yehuda Sadeh)
- rgw: fixed subuser modify (Yehuda Sadeh)
- rgw: fix error setting empty owner on ACLs (Yehuda Sadeh)
- rgw: fix fastcgi deadlock (do not return data from librados callback) (Yehuda Sadeh)
- rgw: fix many-part multipart uploads (Yehuda Sadeh)
- rgw: fix misc CORS bugs (Robin H. Johnson)
- rgw: fix object placement read op (Yehuda Sadeh)

- rgw: fix reading bucket policy (#6940)
- rgw: fix read_user_buckets 'max' behavior (Yehuda Sadeh)
- rgw: fix several CORS bugs (Robin H. Johnson)
- rgw: fix use-after-free when releasing completion handle (Yehuda Sadeh)
- rgw: improve swift temp URL support (Yehuda Sadeh)
- rgw: make multi-object delete idempotent (Yehuda Sadeh)
- rgw: optionally defer to bucket ACLs instead of object ACLs (Liam Monahan)
- rgw: prototype mongoose frontend (Yehuda Sadeh)
- rgw: several doc fixes (Alexandre Marangone)
- rgw: support for password (instead of admin token) for keystone authentication (Christophe Courtaut)
- rgw: switch from mongoose to civetweb (Yehuda Sadeh)
- rgw: user quotas (Yehuda Sadeh)
- rpm: fix redhat-lsb dependency (Sage Weil, Alfredo Deza)
- specfile: fix RPM build on RHEL6 (Ken Dreyer, Derek Yarnell)
- specfile: ship libdir/ceph (Key Dreyer)
- sysvinit, upstart: prevent both init systems from starting the same daemons (Josh Durgin)

11.28.7 Notable changes since v0.67 Dumpling

- build cleanly under clang (Christophe Courtaut)
- build: Makefile refactor (Roald J. van Loon)
- build: fix [/usr]/sbin locations (Alan Somers)
- ceph-disk: fix journal preallocation
- ceph-fuse, radosgw: enable admin socket and logging by default
- ceph-fuse: fix problem with readahead vs truncate race (Yan, Zheng)
- ceph-fuse: trim deleted inodes from cache (Yan, Zheng)
- ceph-fuse: use newer fuse api (Jianpeng Ma)
- ceph-kvstore-tool: new tool for working with leveldb (copy, crc) (Joao Luis)
- ceph-post-file: new command to easily share logs or other files with ceph devs
- ceph: improve parsing of CEPH_ARGS (Benoit Knecht)
- ceph: make -h behave when monitors are down
- ceph: parse CEPH_ARGS env variable
- common: bloom_filter improvements, cleanups
- common: cache crc32c values where possible
- common: correct SI is kB not KB (Dan Mick)
- common: fix looping on BSD (Alan Somers)
- common: migrate SharedPtrRegistry to use boost::shared_ptr<> (Loic Dachary)

- common: misc portability fixes (Noah Watkins)
- crc32c: fix optimized crc32c code (it now detects arch support properly)
- crc32c: improved intel-optimized crc32c support (~8x faster on my laptop!)
- crush: fix name caching
- doc: erasure coding design notes (Loic Dachary)
- hadoop: removed old version of shim to avoid confusing users (Noah Watkins)
- librados, mon: ability to query/ping out-of-quorum monitor status (Joao Luis)
- librados: fix async aio completion wakeup
- librados: fix installed header #includes (Dan Mick)
- librados: get_version64() method for C++ API
- librados: hello_world example (Greg Farnum)
- librados: sync calls now return on commit (instead of ack) (Greg Farnum)
- librbd python bindings: fix parent image name limit (Josh Durgin)
- librbd, ceph-fuse: avoid some sources of ceph-fuse, rbd cache stalls
- mds: avoid leaking objects when deleting truncated files (Yan, Zheng)
- mds: fix F_GETLK (Yan, Zheng)
- mds: fix LOOKUPSNAP bug
- mds: fix heap profiler commands (Joao Luis)
- mds: fix locking deadlock (David Disseldorp)
- mds: fix many bugs with stray (unlinked) inodes (Yan, Zheng)
- mds: fix many directory fragmentation bugs (Yan, Zheng)
- mds: fix mds rejoin with legacy parent backpointer xattrs (Alexandre Oliva)
- mds: fix rare restart/failure race during fs creation
- mds: fix standby-replay when we fall behind (Yan, Zheng)
- mds: fix stray directory purging (Yan, Zheng)
- mds: notify clients about deleted files (so they can release from their cache) (Yan, Zheng)
- mds: several bug fixes with clustered mds (Yan, Zheng)
- mon, osd: improve osdmap trimming logic (Samuel Just)
- mon, osd: initial CLI for configuring tiering
- mon: a few 'ceph mon add' races fixed (command is now idempotent) (Joao Luis)
- mon: allow (un)setting HASHPSPOOL flag on existing pools (Joao Luis)
- mon: allow cap strings with . to be unquoted
- mon: allow logging level of cluster log (/var/log/ceph/ceph.log) to be adjusted
- mon: avoid rewriting full osdmaps on restart (Joao Luis)
- mon: continue to discover peer addr info during election phase
- mon: disallow CephFS snapshots until 'ceph mds set allow_new_snaps' (Greg Farnum)

- mon: do not expose uncommitted state from 'osd crush {add,set} ...' (Joao Luis)
- mon: fix 'ceph osd crush reweight ...' (Joao Luis)
- mon: fix 'osd crush move ...' command for buckets (Joao Luis)
- mon: fix byte counts (off by factor of 4) (Dan Mick, Joao Luis)
- mon: fix paxos corner case
- mon: kv properties for pools to support EC (Loic Dachary)
- mon: make 'osd pool rename' idempotent (Joao Luis)
- mon: modify 'auth add' semantics to make a bit more sense (Joao Luis)
- mon: new 'osd perf' command to dump recent performance information (Samuel Just)
- mon: new and improved 'ceph -s' or 'ceph status' command (more info, easier to read)
- mon: some auth check cleanups (Joao Luis)
- mon: track per-pool stats (Joao Luis)
- mon: warn about pools with bad pg_num
- mon: warn when mon data stores grow very large (Joao Luis)
- monc: fix small memory leak
- new wireshark patches pulled into the tree (Kevin Jones)
- objecter, librados: redirect requests based on cache tier config
- objecter: fix possible hang when cluster is unpaused (Josh Durgin)
- osd, librados: add new COPY_FROM rados operation
- osd, librados: add new COPY_GET rados operations (used by COPY_FROM)
- osd: 'osd recover clone overlap limit' option to limit cloning during recovery (Samuel Just)
- osd: COPY_GET on-wire encoding improvements (Greg Farnum)
- osd: add 'osd heartbeat min healthy ratio' configurable (was hard-coded at 33%)
- osd: add option to disable pg log debug code (which burns CPU)
- osd: allow cap strings with . to be unquoted
- osd: automatically detect proper xattr limits (David Zafman)
- osd: avoid extra copy in erasure coding reference implementation (Loic Dachary)
- osd: basic cache pool redirects (Greg Farnum)
- osd: basic whiteout, dirty flag support (not yet used)
- osd: bloom_filter encodability, fixes, cleanups (Loic Dachary, Sage Weil)
- osd: clean up and generalize copy-from code (Greg Farnum)
- osd: cls_hello OSD class example
- osd: erasure coding doc updates (Loic Dachary)
- osd: erasure coding plugin infrastructure, tests (Loic Dachary)
- osd: experiemental support for ZFS (zfsonlinux.org) (Yan, Zheng)
- osd: fix RWORDER flags

- osd: fix exponential backoff of slow request warnings (Loic Dachary)
- osd: fix handling of racing read vs write (Samuel Just)
- osd: fix version value returned by various operations (Greg Farnum)
- osd: generalized temp object infrastructure
- osd: ghobject_t infrastructure for EC (David Zafman)
- osd: improvements for compatset support and storage (David Zafman)
- osd: infrastructure to copy objects from other OSDs
- osd: instrument peering states (David Zafman)
- osd: misc copy-from improvements
- osd: opportunistic crc checking on stored data (off by default)
- osd: properly enforce RD/WR flags for rados classes
- osd: reduce blocking on backing fs (Samuel Just)
- osd: refactor recovery using PGBackend (Samuel Just)
- osd: remove old magical tmap->omap conversion
- osd: remove old pg log on upgrade (Samuel Just)
- osd: revert xattr size limit (fixes large rgw uploads)
- osd: use fdatasync(2) instead of fsync(2) to improve performance (Sam Just)
- pybind: fix blacklisting nonce (Loic Dachary)
- radosgw-agent: multi-region replication/DR
- rgw: complete in-progress requests before shutting down
- rgw: default log level is now more reasonable (Yehuda Sadeh)
- rgw: fix S3 auth with response-* query string params (Sylvain Munaut, Yehuda Sadeh)
- rgw: fix a few minor memory leaks (Yehuda Sadeh)
- rgw: fix acl group check (Yehuda Sadeh)
- rgw: fix inefficient use of std::list::size() (Yehuda Sadeh)
- rgw: fix major CPU utilization bug with internal caching (Yehuda Sadeh, Mark Nelson)
- rgw: fix ordering of write operations (preventing data loss on crash) (Yehuda Sadeh)
- rgw: fix ordering of writes for mulitpart upload (Yehuda Sadeh)
- rgw: fix various CORS bugs (Yehuda Sadeh)
- rgw: fix/improve swift COPY support (Yehuda Sadeh)
- rgw: improve help output (Christophe Courtaut)
- rgw: misc fixes to support DR (Josh Durgin, Yehuda Sadeh)
- rgw: per-bucket quota (Yehuda Sadeh)
- rgw: validate S3 tokens against keystone (Roald J. van Loon)
- rgw: wildcard support for keystone roles (Christophe Courtaut)
- rpm: fix junit dependencies (Alan Grosskurth)

- sysvinit radosgw: fix status return code (Danny Al-Gaaf)
- sysvinit rbdmap: fix error 'service rbdmap stop' (Laurent Barbe)
- sysvinit: add condrestart command (Dan van der Ster)
- sysvinit: fix shutdown order (mons last) (Alfredo Deza)

11.29 v0.79

This release is intended to serve as a release candidate for firefly, which will hopefully be v0.80. No changes are being made to the code base at this point except those that fix bugs. Please test this release if you intend to make use of the new erasure-coded pools or cache tiers in firefly.

This release fixes a range of bugs found in v0.78 and streamlines the user experience when creating erasure-coded pools. There is also a raft of fixes for the MDS (multi-mds, directory fragmentation, and large directories). The main notable new piece of functionality is a small change to allow radosgw to use an erasure-coded pool for object data.

11.29.1 Upgrading

- Erasure pools created with v0.78 will no longer function with v0.79. You will need to delete the old pool and create a new one.
- A bug was fixed in the authentication handshake with big-endian architectures that prevent authentication between big- and little-endian machines in the same cluster. If you have a cluster that consists entirely of big-endian machines, you will need to upgrade all daemons and clients and restart.
- The 'ceph.file.layout' and 'ceph.dir.layout' extended attributes are no longer included in the listxattr(2) results to prevent problems with 'cp -a' and similar tools.
- Monitor 'auth' read-only commands now expect the user to have 'rx' caps. This is the same behavior that was present in dumpling, but in emperor and more recent development releases the 'r' cap was sufficient. The affected commands are:

```
ceph auth export
ceph auth get
ceph auth get-key
ceph auth print-key
ceph auth list
```

11.29.2 Notable Changes

- ceph-conf: stop creating bogus log files (Josh Durgin, Sage Weil)
- common: fix authentication on big-endian architectures (Dan Mick)
- debian: change directory ownership between ceph and ceph-common (Sage Weil)
- init: fix startup ordering/timeout problem with OSDs (Dmitry Smirnov)
- librbd: skip zeroes/holes when copying sparse images (Josh Durgin)
- mds: cope with MDS failure during creation (John Spray)
- mds: fix crash from client sleep/resume (Zheng Yan)
- mds: misc fixes for directory fragments (Zheng Yan)

- mds: misc fixes for larger directories (Zheng Yan)
- mds: misc fixes for multiple MDSs (Zheng Yan)
- mds: remove .ceph directory (John Spray)
- misc coverity fixes, cleanups (Danny Al-Gaaf)
- mon: add erasure profiles and improve erasure pool creation (Loic Dachary)
- mon: 'ceph osd pg-temp ...' and primary-temp commands (Ilya Dryomov)
- mon: fix pool count in 'ceph -s' output (Sage Weil)
- msgr: improve connection error detection between clients and monitors (Greg Farnum, Sage Weil)
- osd: add/fix CPU feature detection for jerasure (Loic Dachary)
- osd: improved scrub checks on clones (Sage Weil, Sam Just)
- osd: many erasure fixes (Sam Just)
- osd: move to jerasure2 library (Loic Dachary)
- osd: new tests for erasure pools (David Zafman)
- osd: reduce scrub lock contention (Guang Yang)
- rgw: allow use of an erasure data pool (Yehuda Sadeh)

11.30 v0.78

This development release includes two key features: erasure coding and cache tiering. A huge amount of code was merged for this release and several additional weeks were spent stabilizing the code base, and it is now in a state where it is ready to be tested by a broader user base.

This is *not* the firefly release. Firefly will be delayed for at least another sprint so that we can get some operational experience with the new code and do some additional testing before committing to long term support.

Note: Please note that while it is possible to create and test erasure coded pools in this release, the pools will not be usable when you upgrade to v0.79 as the OSDMap encoding will subtlely change. Please do not populate your test pools with important data that can't be reloaded.

11.30.1 Upgrading

- Upgrade daemons in the following order:
  1. Monitors
  2. OSDs
  3. MDSs and/or radosgw

  If the ceph-mds daemon is restarted first, it will wait until all OSDs have been upgraded before finishing its startup sequence. If the ceph-mon daemons are not restarted prior to the ceph-osd daemons, they will not correctly register their new capabilities with the cluster and new features may not be usable until they are restarted a second time.
- Upgrade radosgw daemons together. There is a subtle change in behavior for multipart uploads that prevents a multipart request that was initiated with a new radosgw from being completed by an old radosgw.

- CephFS recently added support for a new 'backtrace' attribute on file data objects that is used for lookup by inode number (i.e., NFS reexport and hard links), and will later be used by fsck repair. This replaces the existing anchor table mechanism that is used for hard link resolution. In order to completely phase that out, any inode that has an outdated backtrace attribute will get updated when the inode itself is modified. This will result in some extra workload after a legacy CephFS file system is upgraded.
- The per-op return code in librados' ObjectWriteOperation interface is now filled in.
- The librados cmpxattr operation now handles xattrs containing null bytes as data rather than null-terminated strings.
- Compound operations in librados that create and then delete the same object are now explicitly disallowed (they fail with -EINVAL).
- The default leveldb cache size for the ceph-osd daemon has been increased from 4 MB to 128 MB. This will increase the memory footprint of that process but tends to increase performance of omap (key/value) objects (used for CephFS and the radosgw). If memory in your deployment is tight, you can preserve the old behavio by adding:

```
leveldb write buffer size = 0
leveldb cache size = 0
```

  to your ceph.conf to get back the (leveldb) defaults.

11.30.2 Notable Changes

- ceph-brag: new client and server tools (Sebastien Han, Babu Shanmugam)
- ceph-disk: use partx on RHEL or CentOS instead of partprobe (Alfredo Deza)
- ceph: fix combination of 'tell' and interactive mode (Joao Eduardo Luis)
- ceph-fuse: fix bugs with inline data and multiple MDSs (Zheng Yan)
- client: fix getcwd() to use new LOOKUPPARENT operation (Zheng Yan)
- common: fall back to json-pretty for admin socket (Loic Dachary)
- common: fix 'config dump' debug prefix (Danny Al-Gaaf)
- common: misc coverity fixes (Danny Al-Gaaf)
- common: throtller, shared_cache performance improvements, TrackedOp (Greg Farnum, Samuel Just)
- crush: fix JSON schema for dump (John Spray)
- crush: misc cleanups, tests (Loic Dachary)
- crush: new vary_r tunable (Sage Weil)
- crush: prevent invalid buckets of type 0 (Sage Weil)
- keyvaluestore: add perfcounters, misc bug fixes (Haomai Wang)
- keyvaluestore: portability improvements (Noah Watkins)
- libcephfs: API changes to better support NFS reexport via Ganesha (Matt Benjamin, Adam Emerson, Andrey Kuznetsov, Casey Bodley, David Zafman)
- librados: API documentation improvements (John Wilkins, Josh Durgin)
- librados: fix object enumeration bugs; allow iterator assignment (Josh Durgin)
- librados: streamline tests (Josh Durgin)

- librados: support for atomic read and omap operations for C API (Josh Durgin)
- librados: support for osd and mon command timeouts (Josh Durgin)
- librbd: pass allocation hints to OSD (Ilya Dryomov)
- logrotate: fix bug that prevented rotation for some daemons (Loic Dachary)
- mds: avoid duplicated discovers during recovery (Zheng Yan)
- mds: fix file lock owner checks (Zheng Yan)
- mds: fix LOOKUPPARENT, new LOOKUPNAME ops for reliable NFS reexport (Zheng Yan)
- mds: fix xattr handling on setxattr (Zheng Yan)
- mds: fix xattrs in getattr replies (Sage Weil)
- mds: force backtrace updates for old inodes on update (Zheng Yan)
- mds: several multi-mds and dirfrag bug fixes (Zheng Yan)
- mon: encode erasure stripe width in pool metadata (Loic Dachary)
- mon: erasure code crush rule creation (Loic Dachary)
- mon: erasure code plugin support (Loic Dachary)
- mon: fix bugs in initial post-mkfs quorum creation (Sage Weil)
- mon: fix error output to terminal during startup (Joao Eduardo Luis)
- mon: fix legacy CRUSH tunables warning (Sage Weil)
- mon: fix osd_epochs lower bound tracking for map trimming (Sage Weil)
- mon: fix OSDMap encoding features (Sage Weil, Aaron Ten Clay)
- mon: fix 'pg dump' JSON output (John Spray)
- mon: include dirty stats in 'ceph df detail' (Sage Weil)
- mon: list quorum member names in quorum order (Sage Weil)
- mon: prevent addition of non-empty cache tier (Sage Weil)
- mon: prevent deletion of CephFS pools (John Spray)
- mon: warn when cache tier approaches 'full' (Sage Weil)
- osd: allocation hint, with XFS support (Ilya Dryomov)
- osd: erasure coded pool support (Samuel Just)
- osd: fix bug causing slow/stalled recovery (#7706) (Samuel Just)
- osd: fix bugs in log merging (Samuel Just)
- osd: fix/clarify end-of-object handling on read (Loic Dachary)
- osd: fix impolite mon session backoff, reconnect behavior (Greg Farnum)
- osd: fix SnapContext cache id bug (Samuel Just)
- osd: increase default leveldb cache size and write buffer (Sage Weil, Dmitry Smirnov)
- osd: limit size of 'osd bench ...' arguments (Joao Eduardo Luis)
- osdmaptool: new –test-map-pgs mode (Sage Weil, Ilya Dryomov)
- osd, mon: add primary-affinity to adjust selection of primaries (Sage Weil)

- osd: new ‘status’ admin socket command (Sage Weil)
- osd: simple tiering agent (Sage Weil)
- osd: store checksums for erasure coded object stripes (Samuel Just)
- osd: tests for objectstore backends (Haomai Wang)
- osd: various refactoring and bug fixes (Samuel Just, David Zafman)
- rados: add ‘set-alloc-hint’ command (Ilya Dryomov)
- rbd-fuse: fix enumerate_images overflow, memory leak (Ilya Dryomov)
- rbdmap: fix upstart script (Stephan Renatus)
- rgw: avoid logging system events to usage log (Yehuda Sadeh)
- rgw: fix Swift range reponse (Yehuda Sadeh)
- rgw: improve scalability for manifest objects (Yehuda Sadeh)
- rgw: misc fixes for multipart objects, policies (Yehuda Sadeh)
- rgw: support non-standard MultipartUpload command (Yehuda Sadeh)

11.31 v0.77

This is the final development release before the Firefly feature freeze. The main items in this release include some additional refactoring work in the OSD IO path (include some locking improvements), per-user quotas for the radosgw, a switch to civetweb from mongoose for the prototype radosgw standalone mode, and a prototype leveldb-based backend for the OSD. The C librados API also got support for atomic write operations (read side transactions will appear in v0.78).

11.31.1 Upgrading

- The ‘ceph -s’ or ‘ceph status’ command’s ‘num_in_osds’ field in the JSON and XML output has been changed from a string to an int.
- The recently added ‘ceph mds set allow_new_snaps’ command’s syntax has changed slightly; it is now ‘ceph mds set allow_new_snaps true’. The ‘unset’ command has been removed; instead, set the value to ‘false’.
- The syntax for allowing snapshots is now ‘mds set allow_new_snaps <true|false>’ instead of ‘mds <set,unset> allow_new_snaps’.

11.31.2 Notable Changes

- osd: client IO path changes for EC (Samuel Just)
- common: portability changes to support libc++ (Noah Watkins)
- common: switch to unordered_map from hash_map (Noah Watkins)
- rgw: switch from mongoose to civetweb (Yehuda Sadeh)
- osd: improve locking in fd lookup cache (Samuel Just, Greg Farnum)
- doc: many many updates (John Wilkins)
- rgw: user quotas (Yehuda Sadeh)

- mon: persist quorum features to disk (Greg Farnum)
- mon: MForward tests (Loic Dachary)
- mds: inline data support (Li Wang, Yunchuan Wen)
- rgw: fix many-part multipart uploads (Yehuda Sadeh)
- osd: new keyvaluestore-dev backend based on leveldb (Haomai Wang)
- rbd: prevent deletion of images with watchers (Ilya Dryomov)
- osd: avoid touching leveldb for some xattrs (Haomai Wang, Sage Weil)
- mailmap: affiliation updates (Loic Dachary)
- osd: new OSDMap encoding (Greg Farnum)
- osd: generalize scrubbing infrastructure to allow EC (David Zafman)
- rgw: several doc fixes (Alexandre Marangone)
- librados: add C API coverage for atomic write operations (Christian Marie)
- rgw: improve swift temp URL support (Yehuda Sadeh)
- rest-api: do not fail when no OSDs yet exist (Dan Mick)
- common: check preexisting admin socket for active daemon before removing (Loic Dachary)
- osd: handle more whitespace (newline, tab) in osd capabilities (Sage Weil)
- mon: handle more whitespace (newline, tab) in mon capabilities (Sage Weil)
- rgw: make multi-object delete idempotent (Yehuda Sadeh)
- crush: fix off-by-one error in recent refactor (Sage Weil)
- rgw: fix read_user_buckets 'max' behavior (Yehuda Sadeh)
- mon: change mds allow_new_snaps syntax to be more consistent (Sage Weil)

11.32 v0.76

This release includes another batch of updates for firefly functionality. Most notably, the cache pool infrastructure now support snapshots, the OSD backfill functionality has been generalized to include multiple targets (necessary for the coming erasure pools), and there were performance improvements to the erasure code plugin on capable processors. The MDS now properly utilizes (and seamlessly migrates to) the OSD key/value interface (aka omap) for storing directory objects. There continue to be many other fixes and improvements for usability and code portability across the tree.

11.32.1 Upgrading

- 'rbd ls' on a pool which never held rbd images now exits with code 0. It outputs nothing in plain format, or an empty list in non-plain format. This is consistent with the behavior for a pool which used to hold images, but contains none. Scripts relying on this behavior should be updated.
- The MDS requires a new OSD operation TMAP2OMAP, added in this release. When upgrading, be sure to upgrade and restart the ceph-osd daemons before the ceph-mds daemon. The MDS will refuse to start if any up OSDs do not support the new feature.
- The 'ceph mds set_max_mds N' command is now deprecated in favor of 'ceph mds set max_mds N'.

11.32.2 Notable Changes

- build: misc improvements (Ken Dreyer)
- ceph-disk: generalize path names, add tests (Loic Dachary)
- ceph-disk: misc improvements for puppet (Loic Dachary)
- ceph-disk: several bug fixes (Loic Dachary)
- ceph-fuse: fix race for sync reads (Sage Weil)
- config: recursive metavariable expansion (Loic Dachary)
- crush: usability and test improvements (Loic Dachary)
- doc: misc fixes (David Moreau Simard, Kun Huang)
- erasure-code: improve buffer alignment (Loic Dachary)
- erasure-code: rewrite region-xor using vector operations (Andreas Peters)
- librados, osd: new TMAP2OMAP operation (Yan, Zheng)
- mailmap updates (Loic Dachary)
- many portability improvements (Noah Watkins)
- many unit test improvements (Loic Dachary)
- mds: always store backtrace in default pool (Yan, Zheng)
- mds: store directories in omap instead of tmap (Yan, Zheng)
- mon: allow adjustment of cephfs max file size via 'ceph mds set max_file_size' (Sage Weil)
- mon: do not create erasure rules by default (Sage Weil)
- mon: do not generate spurious MDSMaps in certain cases (Sage Weil)
- mon: do not use keyring if auth = none (Loic Dachary)
- mon: fix pg_temp leaks (Joao Eduardo Luis)
- osd: backfill to multiple targets (David Zafman)
- osd: cache pool support for snapshots (Sage Weil)
- osd: fix and cleanup misc backfill issues (David Zafman)
- osd: fix omap_clear operation to not zap xattrs (Sam Just, Yan, Zheng)
- osd: ignore num_objects_dirty on scrub for old pools (Sage Weil)
- osd: include more info in pg query result (Sage Weil)
- osd: track erasure compatibility (David Zafman)
- rbd: make 'rbd list' return empty list and success on empty pool (Josh Durgin)
- rgw: fix object placement read op (Yehuda Sadeh)
- rgw: fix several CORS bugs (Robin H. Johnson)
- specfile: fix RPM build on RHEL6 (Ken Dreyer, Derek Yarnell)
- specfile: ship libdir/ceph (Key Dreyer)

11.33 v0.75

This is a big release, with lots of infrastructure going in for firefly. The big items include a prototype standalone frontend for radosgw (which does not require apache or fastcgi), tracking for read activity on the osds (to inform tiering decisions), preliminary cache pool support (no snapshots yet), and lots of bug fixes and other work across the tree to get ready for the next batch of erasure coding patches.

For comparison, here are the diff stats for the last few versions:

```
v0.75 291 files changed, 82713 insertions(+), 33495 deletions(-)
v0.74 192 files changed, 17980 insertions(+), 1062 deletions(-)
v0.73 148 files changed, 4464 insertions(+), 2129 deletions(-)
```

11.33.1 Upgrading

- The 'osd pool create ...' syntax has changed for erasure pools.
- The default CRUSH rules and layouts are now using the latest and greatest tunables and defaults. Clusters using the old values will now present with a health WARN state. This can be disabled by adding 'mon warn on legacy crush tunables = false' to ceph.conf.

11.33.2 Notable Changes

- common: bloom filter improvements (Sage Weil)
- common: fix config variable substitution (Loic Dachary)
- crush, osd: s/rep/replicated/ for less confusion (Loic Dachary)
- crush: refactor descend_once behavior; support set_choose*_tries for replicated rules (Sage Weil)
- librados: fix throttle leak (and eventual deadlock) (Josh Durgin)
- librados: read directly into user buffer (Rutger ter Borg)
- librbd: fix use-after-free aio completion bug #5426 (Josh Durgin)
- librbd: localize/distribute parent reads (Sage Weil)
- mds: fix Resetter locking (Alexandre Oliva)
- mds: fix cap migration behavior (Yan, Zheng)
- mds: fix client session flushing (Yan, Zheng)
- mds: fix many many multi-mds bugs (Yan, Zheng)
- misc portability work (Noah Watkins)
- mon, osd: create erasure style crush rules (Loic Dachary, Sage Weil)
- mon: 'osd crush show-tunables' (Sage Weil)
- mon: clean up initial crush rule creation (Loic Dachary)
- mon: improve (replicate or erasure) pool creation UX (Loic Dachary)
- mon: infrastructure to handle mixed-version mon cluster and cli/rest API (Greg Farnum)
- mon: mkfs now idempotent (Loic Dachary)
- mon: only seed new osdmaps to current OSDs (Sage Weil)

- mon: track osd features in OSDMap (Joao Luis, David Zafman)
- mon: warn if crush has non-optimal tunables (Sage Weil)
- mount.ceph: add -n for autofs support (Steve Stock)
- msgr: fix messenger restart race (Xihui He)
- osd, librados: fix full cluster handling (Josh Durgin)
- osd: add HitSet tracking for read ops (Sage Weil, Greg Farnum)
- osd: backfill to osds not in acting set (David Zafman)
- osd: enable new hashpspool layout by default (Sage Weil)
- osd: erasure plugin benchmarking tool (Loic Dachary)
- osd: fix XFS detection (Greg Farnum, Sushma Gurram)
- osd: fix copy-get omap bug (Sage Weil)
- osd: fix linux kernel version detection (Ilya Dryomov)
- osd: fix memstore segv (Haomai Wang)
- osd: fix several bugs with tier infrastructure
- osd: fix throttle thread (Haomai Wang)
- osd: preliminary cache pool support (no snaps) (Greg Farnum, Sage Weil)
- rados tool: fix listomapvals (Josh Durgin)
- rados: add 'crush location', smart replica selection/balancing (Sage Weil)
- rados: some performance optimizations (Yehuda Sadeh)
- rbd: add rbdmap support for upstart (Laurent Barbe)
- rbd: expose kernel rbd client options via 'rbd map' (Ilya Dryomov)
- rbd: fix bench-write command (Hoamai Wang)
- rbd: support for 4096 mapped devices, up from ~250 (Ilya Dryomov)
- rgw: allow multiple frontends (Yehuda Sadeh)
- rgw: convert bucket info to new format on demand (Yehuda Sadeh)
- rgw: fix misc CORS bugs (Robin H. Johnson)
- rgw: prototype mongoose frontend (Yehuda Sadeh)

11.34 v0.74

This release includes a few substantial pieces for Firefly, including a long-overdue switch to 3x replication by default and a switch to the "new" CRUSH tunables by default (supported since bobtail). There is also a fix for a long-standing radosgw bug (stalled GET) that has already been backported to emperor and dumpling.

11.34.1 Upgrading

- We now default to the 'bobtail' CRUSH tunable values that are first supported by Ceph clients in bobtail (v0.56) and Linux kernel version v3.9. If you plan to access a newly created Ceph cluster with an older kernel client, you should use 'ceph osd crush tunables legacy' to switch back to the legacy behavior. Note that making that change will likely result in some data movement in the system, so adjust the setting before populating the new cluster with data.
- We now set the HASHPSPOOL flag on newly created pools (and new clusters) by default. Support for this flag first appeared in v0.64; v0.67 Dumpling is the first major release that supports it. It is first supported by the Linux kernel version v3.9. If you plan to access a newly created Ceph cluster with an older kernel or clients (e.g, librados, librbd) from a pre-dumpling Ceph release, you should add 'osd pool default flag hashpspool = false' to the '[global]' section of your 'ceph.conf' prior to creating your monitors (e.g., after 'ceph-deploy new' but before 'ceph-deploy mon create ...').
- The configuration option 'osd pool default crush rule' is deprecated and replaced with 'osd pool default crush replicated ruleset'. 'osd pool default crush rule' takes precedence for backward compatibility and a deprecation warning is displayed when it is used.

11.34.2 Notable Changes

- buffer: some zero-copy groundwork (Josh Durgin)
- ceph-disk: avoid fd0 (Loic Dachary)
- crush: default to bobtail tunables (Sage Weil)
- crush: many additional tests (Loic Dachary)
- crush: misc fixes, cleanups (Loic Dachary)
- crush: new rule steps to adjust retry attempts (Sage Weil)
- debian: integrate misc fixes from downstream packaging (James Page)
- doc: big update to install docs (John Wilkins)
- libcephfs: fix resource leak (Zheng Yan)
- misc coverity fixes (Xing Lin, Li Wang, Danny Al-Gaaf)
- misc portability fixes (Noah Watkins, Alan Somers)
- mon, osd: new 'erasure' pool type (still not fully supported)
- mon: add 'mon getmap EPOCH' (Joao Eduardo Luis)
- mon: collect misc metadata about osd (os, kernel, etc.), new 'osd metadata' command (Sage Weil)
- osd: default to 3x replication
- osd: do not include backfill targets in acting set (David Zafman)
- osd: new 'chassis' type in default crush hierarchy (Sage Weil)
- osd: requery unfound on stray notify (#6909) (Samuel Just)
- osd: some PGBackend infrastructure (Samuel Just)
- osd: support for new 'memstore' (memory-backed) backend (Sage Weil)
- rgw: fix fastcgi deadlock (do not return data from librados callback) (Yehuda Sadeh)
- rgw: fix reading bucket policy (#6940)

- rgw: fix use-after-free when releasing completion handle (Yehuda Sadeh)

11.35 v0.73

This release, the first development release after emperor, includes many bug fixes and a few additional pieces of functionality. The first batch of larger changes will be landing in the next version, v0.74.

11.35.1 Upgrading

- As part of fix for #6796, ‘ceph osd pool set <pool> <var> <arg>’ now receives <arg> as an integer instead of a string. This affects how ‘hashpspool’ flag is set/unset: instead of ‘true’ or ‘false’, it now must be ‘0’ or ‘1’.
- The behavior of the CRUSH ‘indep’ choose mode has been changed. No ceph cluster should have been using this behavior unless someone has manually extracted a crush map, modified a CRUSH rule to replace ‘firstn’ with ‘indep’, recompiled, and reinjected the new map into the cluster. If the ‘indep’ mode is currently in use on a cluster, the rule should be modified to use ‘firstn’ instead, and the administrator should wait until any data movement completes before upgrading.
- The ‘osd dump’ command now dumps pool snaps as an array instead of an object.
- The radosgw init script (sysvinit) how requires that the ‘host = ...’ line in ceph.conf, if present, match the short hostname (the output of ‘hostname -s’), not the fully qualified hostname or the (occasionally non-short) output of ‘hostname’. Failure to adjust this when upgrading from emperor or dumpling may prevent the radosgw daemon from starting.

11.35.2 Notable Changes

- ceph-crush-location: new hook for setting CRUSH location of osd daemons on start
- ceph-kvstore-tool: expanded command set and capabilities (Joao Eduardo Luis)
- ceph.spec: fix build dependency (Loic Dachary)
- common: fix aligned buffer allocation (Loic Dachary)
- doc: many many install doc improvements (John Wilkins)
- mds: fix readdir end check (Zheng Yan)
- mds: update old-format backtraces opportunistically (Zheng Yan)
- misc cleanups from coverity (Xing Lin)
- misc portability fixes (Noah Watkins, Christophe Courtaut, Alan Somers, huanjun)
- mon: ‘osd dump’ dumps pool snaps as array, not object (Dan Mick)
- mon: allow debug quorum_{enter,exit} commands via admin socket
- mon: prevent extreme changes in pool pg_num (Greg Farnum)
- mon: take ‘osd pool set ...’ value as an int, not string (Joao Eduardo Luis)
- mon: trim MDSMaps (Joao Eduardo Luis)
- osd: fix object_info_t encoding bug from emperor (Sam Just)
- rbd: add ‘rbdmap’ init script for mapping rbd images on book (Adam Twardowski)
- rgw: add ‘status’ command to sysvinit script (David Moreau Simard)

- rgw: fix error setting empty owner on ACLs (Yehuda Sadeh)
- rgw: optionally defer to bucket ACLs instead of object ACLs (Liam Monahan)
- rgw: support for password (instead of admin token) for keystone authentication (Christophe Courtaut)
- sysvinit, upstart: prevent both init systems from starting the same daemons (Josh Durgin)

11.36 v0.72.3 Emperor (pending release)

11.36.1 Upgrading

- Monitor 'auth' read-only commands now expect the user to have 'rx' caps. This is the same behavior that was present in dumpling, but in emperor and more recent development releases the 'r' cap was sufficient. Note that this backported security fix will break mon keys that are using the following commands but do not have the 'x' bit in the mon capability:

```
ceph auth export
ceph auth get
ceph auth get-key
ceph auth print-key
ceph auth list
```

11.37 v0.72.2 Emperor

This is the second bugfix release for the v0.72.x Emperor series. We have fixed a hang in radosgw, and fixed (again) a problem with monitor CLI compatiblity with mixed version monitors. (In the future this will no longer be a problem.)

11.37.1 Upgrading

- The JSON schema for the 'osd pool set ...' command changed slightly. Please avoid issuing this particular command via the CLI while there is a mix of v0.72.1 and v0.72.2 monitor daemons running.
- As part of fix for #6796, 'ceph osd pool set <pool> <var> <arg>' now receives <arg> as an integer instead of a string. This affects how 'hashpspool' flag is set/unset: instead of 'true' or 'false', it now must be '0' or '1'.

11.37.2 Changes

- mon: 'osd pool set ...' syntax change
- osd: added test for missing on-disk HEAD object
- osd: fix osd bench block size argument
- rgw: fix hang on large object GET
- rgw: fix rare use-after-free
- rgw: various DR bug fixes
- rgw: do not return error on empty owner when setting ACL
- sysvinit, upstart: prevent starting daemons using both init systems

For more detailed information, see `the complete changelog`.

11.38 v0.72.1 Emperor

11.38.1 Important Note

When you are upgrading from Dumpling to Emperor, do not run any of the "ceph osd pool set" commands while your monitors are running separate versions. Doing so could result in inadvertently changing cluster configuration settings that exhaust compute resources in your OSDs.

11.38.2 Changes

- osd: fix upgrade bug #6761
- ceph_filestore_tool: introduced tool to repair errors caused by #6761

This release addresses issue #6761. Upgrading to Emperor can cause reads to begin returning ENFILE (too many open files). v0.72.1 fixes that upgrade issue and adds a tool ceph_filestore_tool to repair osd stores affected by this bug.

To repair a cluster affected by this bug:

1. Upgrade all osd machines to v0.72.1
2. Install the ceph-test package on each osd machine to get ceph_filestore_tool
3. Stop all osd processes
4. To see all lost objects, run the following on each osd with the osd stopped and the osd data directory mounted:

```
ceph_filestore_tool --list-lost-objects=true --filestore-path=<path-to-osd-filestore> --journal-
```

5. To fix all lost objects, run the following on each osd with the osd stopped and the osd data directory mounted:

```
ceph_filestore_tool --fix-lost-objects=true --list-lost-objects=true --filestore-path=<path-to-o
```

6. Once lost objects have been repaired on each osd, you can restart the cluster.

Note, the ceph_filestore_tool performs a scan of all objects on the osd and may take some time.

11.39 v0.72 Emperor

This is the fifth major release of Ceph, the fourth since adopting a 3-month development cycle. This release brings several new features, including multi-datacenter replication for the radosgw, improved usability, and lands a lot of incremental performance and internal refactoring work to support upcoming features in Firefly.

11.39.1 Important Note

When you are upgrading from Dumpling to Emperor, do not run any of the "ceph osd pool set" commands while your monitors are running separate versions. Doing so could result in inadvertently changing cluster configuration settings that exhaust compute resources in your OSDs.

11.39.2 Highlights

- common: improved crc32c performance
- librados: new example client and class code

- mds: many bug fixes and stability improvements
- mon: health warnings when pool pg_num values are not reasonable
- mon: per-pool performance stats
- osd, librados: new object copy primitives
- osd: improved interaction with backend file system to reduce latency
- osd: much internal refactoring to support ongoing erasure coding and tiering support
- rgw: bucket quotas
- rgw: improved CORS support
- rgw: performance improvements
- rgw: validate S3 tokens against Keystone

Coincident with core Ceph, the Emperor release also brings:

- radosgw-agent: support for multi-datacenter replication for disaster recovery
- tgt: improved support for iSCSI via upstream tgt

Packages for both are available on ceph.com.

11.39.3 Upgrade sequencing

There are no specific upgrade restrictions on the order or sequence of upgrading from 0.67.x Dumpling. However, you cannot run any of the "ceph osd pool set" commands while your monitors are running separate versions. Doing so could result in inadvertently changing cluster configuration settings and exhausting compute resources in your OSDs.

It is also possible to do a rolling upgrade from 0.61.x Cuttlefish, but there are ordering restrictions. (This is the same set of restrictions for Cuttlefish to Dumpling.)

1. Upgrade ceph-common on all nodes that will use the command line 'ceph' utility.
2. Upgrade all monitors (upgrade ceph package, restart ceph-mon daemons). This can happen one daemon or host at a time. Note that because cuttlefish and dumpling monitors can't talk to each other, all monitors should be upgraded in relatively short succession to minimize the risk that an a untimely failure will reduce availability.
3. Upgrade all osds (upgrade ceph package, restart ceph-osd daemons). This can happen one daemon or host at a time.
4. Upgrade radosgw (upgrade radosgw package, restart radosgw daemons).

11.39.4 Upgrading from v0.71

- ceph-fuse and radosgw now use the same default values for the admin socket and log file paths that the other daemons (ceph-osd, ceph-mon, etc.) do. If you run these daemons as non-root, you may need to adjust your ceph.conf to disable these options or to adjust the permissions on /var/run/ceph and /var/log/ceph.

11.39.5 Upgrading from v0.67 Dumpling

- ceph-fuse and radosgw now use the same default values for the admin socket and log file paths that the other daemons (ceph-osd, ceph-mon, etc.) do. If you run these daemons as non-root, you may need to adjust your ceph.conf to disable these options or to adjust the permissions on /var/run/ceph and /var/log/ceph.

- The MDS now disallows snapshots by default as they are not considered stable. The command 'ceph mds set allow_snaps' will enable them.
- For clusters that were created before v0.44 (pre-argonaut, Spring 2012) and store radosgw data, the auto-upgrade from TMAP to OMAP objects has been disabled. Before upgrading, make sure that any buckets created on pre-argonaut releases have been modified (e.g., by PUTing and then DELETEing an object from each bucket). Any cluster created with argonaut (v0.48) or a later release or not using radosgw never relied on the automatic conversion and is not affected by this change.
- Any direct users of the 'tmap' portion of the librados API should be aware that the automatic tmap -> omap conversion functionality has been removed.
- Most output that used K or KB (e.g., for kilobyte) now uses a lower-case k to match the official SI convention. Any scripts that parse output and check for an upper-case K will need to be modified.
- librados::Rados::pool_create_async() and librados::Rados::pool_delete_async() don't drop a reference to the completion object on error, caller needs to take care of that. This has never really worked correctly and we were leaking an object
- 'ceph osd crush set <id> <weight> <loc..>' no longer adds the osd to the specified location, as that's a job for 'ceph osd crush add'. It will however continue to work just the same as long as the osd already exists in the crush map.
- The OSD now enforces that class write methods cannot both mutate an object and return data. The rbd.assign_bid method, the lone offender, has been removed. This breaks compatibility with pre-bobtail librbd clients by preventing them from creating new images.
- librados now returns on commit instead of ack for synchronous calls. This is a bit safer in the case where both OSDs and the client crash, and is probably how it should have been acting from the beginning. Users are unlikely to notice but it could result in lower performance in some circumstances. Those who care should switch to using the async interfaces, which let you specify safety semantics precisely.
- The C++ librados AioComplete::get_version() method was incorrectly returning an int (usually 32-bits). To avoid breaking library compatibility, a get_version64() method is added that returns the full-width value. The old method is deprecated and will be removed in a future release. Users of the C++ librados API that make use of the get_version() method should modify their code to avoid getting a value that is truncated from 64 to to 32 bits.

11.39.6 Notable Changes since v0.71

- build: fix [/usr]/sbin locations (Alan Somers)
- ceph-fuse, radosgw: enable admin socket and logging by default
- ceph: make -h behave when monitors are down
- common: cache crc32c values where possible
- common: fix looping on BSD (Alan Somers)
- librados, mon: ability to query/ping out-of-quorum monitor status (Joao Luis)
- librbd python bindings: fix parent image name limit (Josh Durgin)
- mds: avoid leaking objects when deleting truncated files (Yan, Zheng)
- mds: fix F_GETLK (Yan, Zheng)
- mds: fix many bugs with stray (unlinked) inodes (Yan, Zheng)
- mds: fix many directory fragmentation bugs (Yan, Zheng)
- mon: allow (un)setting HASHPSPOOL flag on existing pools (Joao Luis)

- mon: make ‘osd pool rename’ idempotent (Joao Luis)
- osd: COPY_GET on-wire encoding improvements (Greg Farnum)
- osd: bloom_filter encodability, fixes, cleanups (Loic Dachary, Sage Weil)
- osd: fix handling of racing read vs write (Samuel Just)
- osd: reduce blocking on backing fs (Samuel Just)
- radosgw-agent: multi-region replication/DR
- rgw: fix/improve swift COPY support (Yehuda Sadeh)
- rgw: misc fixes to support DR (Josh Durgin, Yehuda Sadeh)
- rgw: per-bucket quota (Yehuda Sadeh)
- rpm: fix junit dependencies (Alan Grosskurth)

11.39.7 Notable Changes since v0.67 Dumpling

- build cleanly under clang (Christophe Courtaut)
- build: Makefile refactor (Roald J. van Loon)
- build: fix [/usr]/sbin locations (Alan Somers)
- ceph-disk: fix journal preallocation
- ceph-fuse, radosgw: enable admin socket and logging by default
- ceph-fuse: fix problem with readahead vs truncate race (Yan, Zheng)
- ceph-fuse: trim deleted inodes from cache (Yan, Zheng)
- ceph-fuse: use newer fuse api (Jianpeng Ma)
- ceph-kvstore-tool: new tool for working with leveldb (copy, crc) (Joao Luis)
- ceph-post-file: new command to easily share logs or other files with ceph devs
- ceph: improve parsing of CEPH_ARGS (Benoit Knecht)
- ceph: make -h behave when monitors are down
- ceph: parse CEPH_ARGS env variable
- common: bloom_filter improvements, cleanups
- common: cache crc32c values where possible
- common: correct SI is kB not KB (Dan Mick)
- common: fix looping on BSD (Alan Somers)
- common: migrate SharedPtrRegistry to use boost::shared_ptr<> (Loic Dachary)
- common: misc portability fixes (Noah Watkins)
- crc32c: fix optimized crc32c code (it now detects arch support properly)
- crc32c: improved intel-optimized crc32c support (~8x faster on my laptop!)
- crush: fix name caching
- doc: erasure coding design notes (Loic Dachary)
- hadoop: removed old version of shim to avoid confusing users (Noah Watkins)

- librados, mon: ability to query/ping out-of-quorum monitor status (Joao Luis)
- librados: fix async aio completion wakeup
- librados: fix installed header #includes (Dan Mick)
- librados: get_version64() method for C++ API
- librados: hello_world example (Greg Farnum)
- librados: sync calls now return on commit (instead of ack) (Greg Farnum)
- librbd python bindings: fix parent image name limit (Josh Durgin)
- librbd, ceph-fuse: avoid some sources of ceph-fuse, rbd cache stalls
- mds: avoid leaking objects when deleting truncated files (Yan, Zheng)
- mds: fix F_GETLK (Yan, Zheng)
- mds: fix LOOKUPSNAP bug
- mds: fix heap profiler commands (Joao Luis)
- mds: fix locking deadlock (David Disseldorp)
- mds: fix many bugs with stray (unlinked) inodes (Yan, Zheng)
- mds: fix many directory fragmentation bugs (Yan, Zheng)
- mds: fix mds rejoin with legacy parent backpointer xattrs (Alexandre Oliva)
- mds: fix rare restart/failure race during fs creation
- mds: fix standby-replay when we fall behind (Yan, Zheng)
- mds: fix stray directory purging (Yan, Zheng)
- mds: notify clients about deleted files (so they can release from their cache) (Yan, Zheng)
- mds: several bug fixes with clustered mds (Yan, Zheng)
- mon, osd: improve osdmap trimming logic (Samuel Just)
- mon, osd: initial CLI for configuring tiering
- mon: a few 'ceph mon add' races fixed (command is now idempotent) (Joao Luis)
- mon: allow (un)setting HASHPSPOOL flag on existing pools (Joao Luis)
- mon: allow cap strings with . to be unquoted
- mon: allow logging level of cluster log (/var/log/ceph/ceph.log) to be adjusted
- mon: avoid rewriting full osdmaps on restart (Joao Luis)
- mon: continue to discover peer addr info during election phase
- mon: disallow CephFS snapshots until 'ceph mds set allow_new_snaps' (Greg Farnum)
- mon: do not expose uncommitted state from 'osd crush {add,set} ...' (Joao Luis)
- mon: fix 'ceph osd crush reweight ...' (Joao Luis)
- mon: fix 'osd crush move ...' command for buckets (Joao Luis)
- mon: fix byte counts (off by factor of 4) (Dan Mick, Joao Luis)
- mon: fix paxos corner case
- mon: kv properties for pools to support EC (Loic Dachary)

- mon: make ‘osd pool rename’ idempotent (Joao Luis)
- mon: modify ‘auth add’ semantics to make a bit more sense (Joao Luis)
- mon: new ‘osd perf’ command to dump recent performance information (Samuel Just)
- mon: new and improved ‘ceph -s’ or ‘ceph status’ command (more info, easier to read)
- mon: some auth check cleanups (Joao Luis)
- mon: track per-pool stats (Joao Luis)
- mon: warn about pools with bad pg_num
- mon: warn when mon data stores grow very large (Joao Luis)
- monc: fix small memory leak
- new wireshark patches pulled into the tree (Kevin Jones)
- objecter, librados: redirect requests based on cache tier config
- objecter: fix possible hang when cluster is unpaused (Josh Durgin)
- osd, librados: add new COPY_FROM rados operation
- osd, librados: add new COPY_GET rados operations (used by COPY_FROM)
- osd: ‘osd recover clone overlap limit’ option to limit cloning during recovery (Samuel Just)
- osd: COPY_GET on-wire encoding improvements (Greg Farnum)
- osd: add ‘osd heartbeat min healthy ratio’ configurable (was hard-coded at 33%)
- osd: add option to disable pg log debug code (which burns CPU)
- osd: allow cap strings with . to be unquoted
- osd: automatically detect proper xattr limits (David Zafman)
- osd: avoid extra copy in erasure coding reference implementation (Loic Dachary)
- osd: basic cache pool redirects (Greg Farnum)
- osd: basic whiteout, dirty flag support (not yet used)
- osd: bloom_filter encodability, fixes, cleanups (Loic Dachary, Sage Weil)
- osd: clean up and generalize copy-from code (Greg Farnum)
- osd: cls_hello OSD class example
- osd: erasure coding doc updates (Loic Dachary)
- osd: erasure coding plugin infrastructure, tests (Loic Dachary)
- osd: experiemental support for ZFS (zfsonlinux.org) (Yan, Zheng)
- osd: fix RWORDER flags
- osd: fix exponential backoff of slow request warnings (Loic Dachary)
- osd: fix handling of racing read vs write (Samuel Just)
- osd: fix version value returned by various operations (Greg Farnum)
- osd: generalized temp object infrastructure
- osd: ghobject_t infrastructure for EC (David Zafman)
- osd: improvements for compatset support and storage (David Zafman)

- osd: infrastructure to copy objects from other OSDs
- osd: instrument peering states (David Zafman)
- osd: misc copy-from improvements
- osd: opportunistic crc checking on stored data (off by default)
- osd: properly enforce RD/WR flags for rados classes
- osd: reduce blocking on backing fs (Samuel Just)
- osd: refactor recovery using PGBackend (Samuel Just)
- osd: remove old magical tmap->omap conversion
- osd: remove old pg log on upgrade (Samuel Just)
- osd: revert xattr size limit (fixes large rgw uploads)
- osd: use fdatasync(2) instead of fsync(2) to improve performance (Sam Just)
- pybind: fix blacklisting nonce (Loic Dachary)
- radosgw-agent: multi-region replication/DR
- rgw: complete in-progress requests before shutting down
- rgw: default log level is now more reasonable (Yehuda Sadeh)
- rgw: fix S3 auth with response-* query string params (Sylvain Munaut, Yehuda Sadeh)
- rgw: fix a few minor memory leaks (Yehuda Sadeh)
- rgw: fix acl group check (Yehuda Sadeh)
- rgw: fix inefficient use of std::list::size() (Yehuda Sadeh)
- rgw: fix major CPU utilization bug with internal caching (Yehuda Sadeh, Mark Nelson)
- rgw: fix ordering of write operations (preventing data loss on crash) (Yehuda Sadeh)
- rgw: fix ordering of writes for mulitpart upload (Yehuda Sadeh)
- rgw: fix various CORS bugs (Yehuda Sadeh)
- rgw: fix/improve swift COPY support (Yehuda Sadeh)
- rgw: improve help output (Christophe Courtaut)
- rgw: misc fixes to support DR (Josh Durgin, Yehuda Sadeh)
- rgw: per-bucket quota (Yehuda Sadeh)
- rgw: validate S3 tokens against keystone (Roald J. van Loon)
- rgw: wildcard support for keystone roles (Christophe Courtaut)
- rpm: fix junit dependencies (Alan Grosskurth)
- sysvinit radosgw: fix status return code (Danny Al-Gaaf)
- sysvinit rbdmap: fix error 'service rbdmap stop' (Laurent Barbe)
- sysvinit: add condrestart command (Dan van der Ster)
- sysvinit: fix shutdown order (mons last) (Alfredo Deza)

11.40 v0.71

This development release includes a significant amount of new code and refactoring, as well as a lot of preliminary functionality that will be needed for erasure coding and tiering support. There are also several significant patch sets improving this with the MDS.

11.40.1 Upgrading

- The MDS now disallows snapshots by default as they are not considered stable. The command 'ceph mds set allow_snaps' will enable them.
- For clusters that were created before v0.44 (pre-argonaut, Spring 2012) and store radosgw data, the auto-upgrade from TMAP to OMAP objects has been disabled. Before upgrading, make sure that any buckets created on pre-argonaut releases have been modified (e.g., by PUTing and then DELETEing an object from each bucket). Any cluster created with argonaut (v0.48) or a later release or not using radosgw never relied on the automatic conversion and is not affected by this change.
- Any direct users of the 'tmap' portion of the librados API should be aware that the automatic tmap -> omap conversion functionality has been removed.
- Most output that used K or KB (e.g., for kilobyte) now uses a lower-case k to match the official SI convention. Any scripts that parse output and check for an upper-case K will need to be modified.

11.40.2 Notable Changes

- build: Makefile refactor (Roald J. van Loon)
- ceph-disk: fix journal preallocation
- ceph-fuse: trim deleted inodes from cache (Yan, Zheng)
- ceph-fuse: use newer fuse api (Jianpeng Ma)
- ceph-kvstore-tool: new tool for working with leveldb (copy, crc) (Joao Luis)
- common: bloom_filter improvements, cleanups
- common: correct SI is kB not KB (Dan Mick)
- common: misc portability fixes (Noah Watkins)
- hadoop: removed old version of shim to avoid confusing users (Noah Watkins)
- librados: fix installed header #includes (Dan Mick)
- librbd, ceph-fuse: avoid some sources of ceph-fuse, rbd cache stalls
- mds: fix LOOKUPSNAP bug
- mds: fix standby-replay when we fall behind (Yan, Zheng)
- mds: fix stray directory purging (Yan, Zheng)
- mon: disallow CephFS snapshots until 'ceph mds set allow_new_snaps' (Greg Farnum)
- mon, osd: improve osdmap trimming logic (Samuel Just)
- mon: kv properties for pools to support EC (Loic Dachary)
- mon: some auth check cleanups (Joao Luis)
- mon: track per-pool stats (Joao Luis)

- mon: warn about pools with bad pg_num
- osd: automatically detect proper xattr limits (David Zafman)
- osd: avoid extra copy in erasure coding reference implementation (Loic Dachary)
- osd: basic cache pool redirects (Greg Farnum)
- osd: basic whiteout, dirty flag support (not yet used)
- osd: clean up and generalize copy-from code (Greg Farnum)
- osd: erasure coding doc updates (Loic Dachary)
- osd: erasure coding plugin infrastructure, tests (Loic Dachary)
- osd: fix RWORDER flags
- osd: fix exponential backoff of slow request warnings (Loic Dachary)
- osd: generalized temp object infrastructure
- osd: ghobject_t infrastructure for EC (David Zafman)
- osd: improvements for compatset support and storage (David Zafman)
- osd: misc copy-from improvements
- osd: opportunistic crc checking on stored data (off by default)
- osd: refactor recovery using PGBackend (Samuel Just)
- osd: remove old magical tmap->omap conversion
- pybind: fix blacklisting nonce (Loic Dachary)
- rgw: default log level is now more reasonable (Yehuda Sadeh)
- rgw: fix acl group check (Yehuda Sadeh)
- sysvinit: fix shutdown order (mons last) (Alfredo Deza)

11.41 v0.70

11.41.1 Upgrading

- librados::Rados::pool_create_async() and librados::Rados::pool_delete_async() don't drop a reference to the completion object on error, caller needs to take care of that. This has never really worked correctly and we were leaking an object
- 'ceph osd crush set <id> <weight> <loc..>' no longer adds the osd to the specified location, as that's a job for 'ceph osd crush add'. It will however continue to work just the same as long as the osd already exists in the crush map.

11.41.2 Notable Changes

- mon: a few 'ceph mon add' races fixed (command is now idempotent) (Joao Luis)
- crush: fix name caching
- rgw: fix a few minor memory leaks (Yehuda Sadeh)
- ceph: improve parsing of CEPH_ARGS (Benoit Knecht)

- mon: avoid rewriting full osdmaps on restart (Joao Luis)
- crc32c: fix optimized crc32c code (it now detects arch support properly)
- mon: fix 'ceph osd crush reweight ...' (Joao Luis)
- osd: revert xattr size limit (fixes large rgw uploads)
- mds: fix heap profiler commands (Joao Luis)
- rgw: fix inefficient use of std::list::size() (Yehuda Sadeh)

11.42 v0.69

11.42.1 Upgrading

- The sysvinit /etc/init.d/ceph script will, by default, update the CRUSH location of an OSD when it starts. Previously, if the monitors were not available, this command would hang indefinitely. Now, that step will time out after 10 seconds and the ceph-osd daemon will not be started.
- Users of the librados C++ API should replace users of get_version() with get_version64() as the old method only returns a 32-bit value for a 64-bit field. The existing 32-bit get_version() method is now deprecated.
- The OSDs are now more picky that request payload match their declared size. A write operation across N bytes that includes M bytes of data will now be rejected. No known clients do this, but the because the server-side behavior has changed it is possible that an application misusing the interface may now get errors.
- The OSD now enforces that class write methods cannot both mutate an object and return data. The rbd.assign_bid method, the lone offender, has been removed. This breaks compatibility with pre-bobtail librbd clients by preventing them from creating new images.
- librados now returns on commit instead of ack for synchronous calls. This is a bit safer in the case where both OSDs and the client crash, and is probably how it should have been acting from the beginning. Users are unlikely to notice but it could result in lower performance in some circumstances. Those who care should switch to using the async interfaces, which let you specify safety semantics precisely.
- The C++ librados AioComplete::get_version() method was incorrectly returning an int (usually 32-bits). To avoid breaking library compatibility, a get_version64() method is added that returns the full-width value. The old method is deprecated and will be removed in a future release. Users of the C++ librados API that make use of the get_version() method should modify their code to avoid getting a value that is truncated from 64 to to 32 bits.

11.42.2 Notable Changes

- build cleanly under clang (Christophe Courtaut)
- common: migrate SharedPtrRegistry to use boost::shared_ptr<> (Loic Dachary)
- doc: erasure coding design notes (Loic Dachary)
- improved intel-optimized crc32c support (~8x faster on my laptop!)
- librados: get_version64() method for C++ API
- mds: fix locking deadlock (David Disseldorp)
- mon, osd: initial CLI for configuring tiering
- mon: allow cap strings with . to be unquoted

- mon: continue to discover peer addr info during election phase
- mon: fix 'osd crush move ...' command for buckets (Joao Luis)
- mon: warn when mon data stores grow very large (Joao Luis)
- objecter, librados: redirect requests based on cache tier config
- osd, librados: add new COPY_FROM rados operation
- osd, librados: add new COPY_GET rados operations (used by COPY_FROM)
- osd: add 'osd heartbeat min healthy ratio' configurable (was hard-coded at 33%)
- osd: add option to disable pg log debug code (which burns CPU)
- osd: allow cap strings with . to be unquoted
- osd: fix version value returned by various operations (Greg Farnum)
- osd: infrastructure to copy objects from other OSDs
- osd: use fdatasync(2) instead of fsync(2) to improve performance (Sam Just)
- rgw: fix major CPU utilization bug with internal caching (Yehuda Sadeh, Mark Nelson)
- rgw: fix ordering of write operations (preventing data loss on crash) (Yehuda Sadeh)
- rgw: fix ordering of writes for mulitpart upload (Yehuda Sadeh)
- rgw: fix various CORS bugs (Yehuda Sadeh)
- rgw: improve help output (Christophe Courtaut)
- rgw: validate S3 tokens against keystone (Roald J. van Loon)
- rgw: wildcard support for keystone roles (Christophe Courtaut)
- sysvinit radosgw: fix status return code (Danny Al-Gaaf)
- sysvinit rbdmap: fix error 'service rbdmap stop' (Laurent Barbe)

11.43 v0.68

11.43.1 Upgrading

- 'ceph osd crush set <id> <weight> <loc..>' no longer adds the osd to the specified location, as that's a job for 'ceph osd crush add'. It will however continue to work just the same as long as the osd already exists in the crush map.
- The OSD now enforces that class write methods cannot both mutate an object and return data. The rbd.assign_bid method, the lone offender, has been removed. This breaks compatibility with pre-bobtail librbd clients by preventing them from creating new images.
- librados now returns on commit instead of ack for synchronous calls. This is a bit safer in the case where both OSDs and the client crash, and is probably how it should have been acting from the beginning. Users are unlikely to notice but it could result in lower performance in some circumstances. Those who care should switch to using the async interfaces, which let you specify safety semantics precisely.
- The C++ librados AioComplete::get_version() method was incorrectly returning an int (usually 32-bits). To avoid breaking library compatibility, a get_version64() method is added that returns the full-width value. The old method is deprecated and will be removed in a future release. Users of the C++ librados API that make use

of the get_version() method should modify their code to avoid getting a value that is truncated from 64 to to 32 bits.

11.43.2 Notable Changes

- ceph-fuse: fix problem with readahead vs truncate race (Yan, Zheng)
- ceph-post-file: new command to easily share logs or other files with ceph devs
- ceph: parse CEPH_ARGS env variable
- librados: fix async aio completion wakeup
- librados: hello_world example (Greg Farnum)
- librados: sync calls now return on commit (instead of ack) (Greg Farnum)
- mds: fix mds rejoin with legacy parent backpointer xattrs (Alexandre Oliva)
- mds: fix rare restart/failure race during fs creation
- mds: notify clients about deleted files (so they can release from their cache) (Yan, Zheng)
- mds: several bug fixes with clustered mds (Yan, Zheng)
- mon: allow logging level of cluster log (/var/log/ceph/ceph.log) to be adjusted
- mon: do not expose uncommitted state from 'osd crush {add,set} ...' (Joao Luis)
- mon: fix byte counts (off by factor of 4) (Dan Mick, Joao Luis)
- mon: fix paxos corner case
- mon: modify 'auth add' semantics to make a bit more sense (Joao Luis)
- mon: new 'osd perf' command to dump recent performance information (Samuel Just)
- mon: new and improved 'ceph -s' or 'ceph status' command (more info, easier to read)
- monc: fix small memory leak
- new wireshark patches pulled into the tree (Kevin Jones)
- objecter: fix possible hang when cluster is unpaused (Josh Durgin)
- osd: 'osd recover clone overlap limit' option to limit cloning during recovery (Samuel Just)
- osd: cls_hello OSD class example
- osd: experiemental support for ZFS (zfsonlinux.org) (Yan, Zheng)
- osd: instrument peering states (David Zafman)
- osd: properly enforce RD/WR flags for rados classes
- osd: remove old pg log on upgrade (Samuel Just)
- rgw: complete in-progress requests before shutting down
- rgw: fix S3 auth with response-* query string params (Sylvain Munaut, Yehuda Sadeh)
- sysvinit: add condrestart command (Dan van der Ster)

11.44 v0.67.12 “Dumpling” (draft)

This stable update for Dumpling fixes a few longstanding issues with backfill in the OSD that can lead to stalled IOs. There is also a fix for memory utilization for reads in librbd when caching is enabled, and then several other small fixes across the rest of the system.

Dumpling users who have encountered IO stalls during backfill and who do not expect to upgrade to Firefly soon should upgrade. Everyone else should upgrade to Firefly already. This is likely to be the last stable release for the 0.67.x Dumpling series.

11.44.1 Notable Changes

- buffer: fix buffer rebuild alignment corner case (#6614 #6003 Loic Dachary, Samuel Just)
- ceph-disk: reprobe partitions after zap (#9665 #9721 Loic Dachary)
- ceph-disk: use partx instead of partprobe when appropriate (Loic Dachary)
- common: add $cctid meta variable (#6228 Adam Crume)
- crush: fix get_full_location_ordered (Sage Weil)
- crush: pick ruleset id that matches rule_id (#9675 Xiaoxi Chen)
- libcephfs: fix tid wrap bug (#9869 Greg Farnum)
- libcephfs: get osd location on -1 should return EINVAL (Sage Weil)
- librados: fix race condition with C API and op timeouts (#9582 Sage Weil)
- librbd: constrain max number of in-flight read requests (#9854 Jason Dillaman)
- librbd: enforce cache size on read requests (Jason Dillaman)
- librbd: fix invalid close in image open failure path (#10030 Jason Dillaman)
- librbd: fix read hang on sparse files (Jason Dillaman)
- librbd: gracefully handle deleted/renamed pools (#10270 #10122 Jason Dillaman)
- librbd: protect list_children from invalid child pool ioctxs (#10123 Jason Dillaman)
- mds: fix ctime updates from clients without dirty caps (#9514 Greg Farnum)
- mds: fix rare NULL dereference in cap update path (Greg Farnum)
- mds: fix assertion caused by system clock backwards (#11053 Yan, Zheng)
- mds: store backtrace on straydir (Yan, Zheng)
- osd: fix journal committed_thru update after replay (#6756 Samuel Just)
- osd: fix memory leak, busy loop on snap trim (#9113 Samuel Just)
- osd: fix misc peering, recovery bugs (#10168 Samuel Just)
- osd: fix purged_snap field on backfill start (#9487 Sage Weil, Samuel Just)
- osd: handle no-op write with snapshot corner case (#10262 Sage Weil, Loic Dachary)
- osd: respect RWORDERED rados flag (Sage Weil)
- osd: several backfill fixes and refactors (Samuel Just, David Zafman)
- rgw: send http status reason explicitly in fastcgi (Yehuda Sadeh)

11.45 v0.67.11 “Dumpling”

This stable update for Dumpling fixes several important bugs that affect a small set of users.

We recommend that all Dumpling users upgrade at their convenience. If none of these issues are affecting your deployment there is no urgency.

11.45.1 Notable Changes

- common: fix sending dup cluster log items (#9080 Sage Weil)
- doc: several doc updates (Alfredo Deza)
- libcephfs-java: fix build against older JNI headesr (Greg Farnum)
- librados: fix crash in op timeout path (#9362 Matthias Kiefer, Sage Weil)
- librbd: fix crash using clone of flattened image (#8845 Josh Durgin)
- librbd: fix error path cleanup when failing to open image (#8912 Josh Durgin)
- mon: fix crash when adjusting pg_num before any OSDs are added (#9052 Sage Weil)
- mon: reduce log noise from paxos (Aanchal Agrawal, Sage Weil)
- osd: allow scrub and snap trim thread pool IO priority to be adjusted (Sage Weil)
- osd: fix mount/remount sync race (#9144 Sage Weil)

11.46 v0.67.10 “Dumpling”

This stable update release for Dumpling includes primarily fixes for RGW, including several issues with bucket listings and a potential data corruption problem when multiple multi-part uploads race. There is also some throttling capability added in the OSD for scrub that can mitigate the performance impact on production clusters.

We recommend that all Dumpling users upgrade at their convenience.

11.46.1 Notable Changes

- ceph-disk: partprobe befoere settle, fixing dm-crypt (#6966, Eric Eastman)
- librbd: add invalidate cache interface (Josh Durgin)
- librbd: close image if remove_child fails (Ilya Dryomov)
- librbd: fix potential null pointer dereference (Danny Al-Gaaf)
- librbd: improve writeback checks, performance (Haomai Wang)
- librbd: skip zeroes when copying image (#6257, Josh Durgin)
- mon: fix rule(set) check on ‘ceph pool set ... crush_ruleset ...’ (#8599, John Spray)
- mon: shut down if mon is removed from cluster (#6789, Joao Eduardo Luis)
- osd: fix filestore perf reports to mon (Sage Weil)
- osd: force any new or updated xattr into leveldb if E2BIG from XFS (#7779, Sage Weil)
- osd: lock snapdir object during write to fix race with backfill (Samuel Just)

- osd: option sleep during scrub (Sage Weil)
- osd: set io priority on scrub and snap trim threads (Sage Weil)
- osd: 'status' admin socket command (Sage Weil)
- rbd: tolerate missing NULL terminator on block_name_prefix (#7577, Dan Mick)
- rgw: calculate user manifest (#8169, Yehuda Sadeh)
- rgw: fix abort on chunk read error, avoid using extra memory (#8289, Yehuda Sadeh)
- rgw: fix buffer overflow on bucket instance id (#8608, Yehuda Sadeh)
- rgw: fix crash in swift CORS preflight request (#8586, Yehuda Sadeh)
- rgw: fix implicit removal of old objects on object creation (#8972, Patrycja Szablowska, Yehuda Sadeh)
- rgw: fix MaxKeys in bucket listing (Yehuda Sadeh)
- rgw: fix race with multiple updates to a single multipart object (#8269, Yehuda Sadeh)
- rgw: improve bucket listing with delimiter (Yehuda Sadeh)
- rgw: include NextMarker in bucket listing (#8858, Yehuda Sadeh)
- rgw: return error early on non-existent bucket (#7064, Yehuda Sadeh)
- rgw: set truncation flag correctly in bucket listing (Yehuda Sadeh)
- sysvinit: continue starting daemons after pre-mount error (#8554, Sage Weil)

For more detailed information, see `the complete changelog`.

11.47 v0.67.9 "Dumpling"

This Dumpling point release fixes several minor bugs. The most prevalent in the field is one that occasionally prevents OSDs from starting on recently created clusters.

We recommend that all Dumpling users upgrade at their convenience.

11.47.1 Notable Changes

- ceph-fuse, libcephfs: client admin socket command to kick and inspect MDS sessions (#8021, Zheng Yan)
- monclient: fix failure detection during mon handshake (#8278, Sage Weil)
- mon: set tid on no-op PGStatsAck messages (#8280, Sage Weil)
- msgr: fix a rare bug with connection negotiation between OSDs (Guang Yang)
- osd: allow snap trim throttling with simple delay (#6278, Sage Weil)
- osd: check for splitting when processing recover/backfill reservations (#6565, Samuel Just)
- osd: fix backfill position tracking (#8162, Samuel Just)
- osd: fix bug in backfill stats (Samuel Just)
- osd: fix bug preventing OSD startup for infant clusters (#8162, Greg Farnum)
- osd: fix rare PG resurrection race causing an incomplete PG (#7740, Samuel Just)
- osd: only complete replicas count toward min_size (#7805, Samuel Just)

- rgw: allow setting ACLs with empty owner (#6892, Yehuda Sadeh)
- rgw: send user manifest header field (#8170, Yehuda Sadeh)

For more detailed information, see `the complete changelog`.

11.48 v0.67.8 "Dumpling"

This Dumpling point release fixes several non-critical issues since v0.67.7. The most notable bug fixes are an auth fix in librbd (observed as an occasional crash from KVM), an improvement in the network failure detection with the monitor, and several hard to hit OSD crashes or hangs.

We recommend that all users upgrade at their convenience.

11.48.1 Upgrading

- The 'rbd ls' function now returns success and returns an empty when a pool does not store any rbd images. Previously it would return an ENOENT error.
- Ceph will now issue a health warning if the 'mon osd down out interval' config option is set to zero. This warning can be disabled by adding 'mon warn on osd down out interval zero = false' to ceph.conf.

11.48.2 Notable Changes

- all: improve keepalive detection of failed monitor connections (#7888, Sage Weil)
- ceph-fuse, libcephfs: pin inodes during readahead, fixing rare crash (#7867, Sage Weil)
- librbd: make cache writeback a bit less aggressive (Sage Weil)
- librbd: make symlink for qemu to detect librbd in RPM (#7293, Josh Durgin)
- mon: allow 'hashpspool' pool flag to be set and unset (Loic Dachary)
- mon: commit paxos state only after entire quorum acks, fixing rare race where prior round state is readable (#7736, Sage Weil)
- mon: make elections and timeouts a bit more robust (#7212, Sage Weil)
- mon: prevent extreme pool split operations (Greg Farnum)
- mon: wait for quorum for get_version requests to close rare pool creation race (#7997, Sage Weil)
- mon: warn on 'mon osd down out interval = 0' (#7784, Joao Luis)
- msgr: fix byte-order for auth challenge, fixing auth errors on big-endian clients (#7977, Dan Mick)
- msgr: fix occasional crash in authentication code (usually triggered by librbd) (#6840, Josh Durgin)
- msgr: fix rebind() race (#6992, Xihui He)
- osd: avoid timeouts during slow PG deletion (#6528, Samuel Just)
- osd: fix bug in pool listing during recovery (#6633, Samuel Just)
- osd: fix queue limits, fixing recovery stalls (#7706, Samuel Just)
- osd: fix rare peering crashes (#6722, #6910, Samuel Just)
- osd: fix rare recovery hang (#6681, Samuel Just)
- osd: improve error handling on journal errors (#7738, Sage Weil)

- osd: reduce load on the monitor from OSDMap subscriptions (Greg Farnum)
- osd: rery GetLog on peer osd startup, fixing some rare peering stalls (#6909, Samuel Just)
- osd: reset journal state on remount to fix occasional crash on OSD startup (#8019, Sage Weil)
- osd: share maps with peers more aggressively (Greg Farnum)
- rbd: make it harder to delete an rbd image that is currently in use (#7076, Ilya Drymov)
- rgw: deny writes to secondary zone by non-system users (#6678, Yehuda Sadeh)
- rgw: do'nt log system requests in usage log (#6889, Yehuda Sadeh)
- rgw: fix bucket recreation (#6951, Yehuda Sadeh)
- rgw: fix Swift range response (#7099, Julien Calvet, Yehuda Sadeh)
- rgw: fix URL escaping (#8202, Yehuda Sadeh)
- rgw: fix whitespace trimming in http headers (#7543, Yehuda Sadeh)
- rgw: make multi-object deletion idempotent (#7346, Yehuda Sadeh)

For more detailed information, see `the complete changelog`.

11.49 v0.67.7 "Dumpling"

This Dumpling point release fixes a few critical issues in v0.67.6.

All v0.67.6 users are urgently encouraged to upgrade. We also recommend that all v0.67.5 (or older) users upgrade.

11.49.1 Upgrading

- Once you have upgraded a radosgw instance or OSD to v0.67.7, you should not downgrade to a previous version.

11.49.2 Notable Changes

- ceph-disk: additional unit tests
- librbd: revert caching behavior change in v0.67.6
- osd: fix problem reading xattrs due to incomplete backport in v0.67.6
- radosgw-admin: fix reading object policy

For more detailed information, see `the complete changelog`.

11.50 v0.67.6 "Dumpling"

This Dumpling point release contains a number of important fixed for the OSD, monitor, and radosgw. Most significantly, a change that forces large object attributes to spill over into leveldb has been backported that can prevent objects and the cluster from being damaged by large attributes (which can be induced via the radosgw). There is also a set of fixes that improves data safety and RADOS semantics when the cluster becomes full and then non-full.

We recommend that all 0.67.x Dumpling users skip this release and upgrade to v0.67.7.

11.50.1 Upgrading

- The OSD has long contained a feature that allows large xattrs to spill over into the leveldb backing store in situations where not all local file systems are able to store them reliably. This option is now enabled unconditionally in order to avoid rare cases where storing large xattrs renders the object unreadable. This is known to be triggered by very large multipart objects, but could be caused by other workloads as well. Although there is some small risk that performance for certain workloads will degrade, it is more important that data be retrievable. Note that newer versions of Ceph (e.g., firefly) do some additional work to avoid the potential performance regression in this case, but that is current considered too complex for backport to the Dumpling stable series.
- It is very dangerous to downgrade from v0.67.6 to a prior version of Dumpling. If the old version does not have 'filestore xattr use omap = true' it may not be able to read all xattrs for an object and can cause undefined behavior.

11.50.2 Notable changes

- ceph-disk: misc bug fixes, particularly on RHEL (Loic Dachary, Alfredo Deza, various)
- ceph-fuse, libcephfs: fix crash from read over certain sparseness patterns (Sage Weil)
- ceph-fuse, libcephfs: fix integer overflow for sync reads racing with appends (Sage Weil)
- ceph.spec: fix udev rule when building RPM under RHEL (Derek Yarnell)
- common: fix crash from bad format from admin socket (Loic Dachary)
- librados: add optional timeouts (Josh Durgin)
- librados: do not leak budget when resending localized or redirected ops (Josh Durgin)
- librados, osd: fix and improve full cluster handling (Josh Durgin)
- librbd: fix use-after-free when updating perfcounters during image close (Josh Durgin)
- librbd: remove limit on objects in cache (Josh Durgin)
- mon: avoid on-disk full OSDMap corruption from pg_temp removal (Sage Weil)
- mon: avoid stray pg_temp entries from pool deletion race (Joao Eduardo Luis)
- mon: do not generate spurious MDSMaps from laggy daemons (Joao Eduardo Luis)
- mon: fix error code from 'osd rm|down|out|in ...' commands (Loic Dachary)
- mon: include all health items in summary output (John Spray)
- osd: fix occasional race/crash during startup (Sage Weil)
- osd: ignore stray OSDMap messages during init (Sage Weil)
- osd: unconditionally let xattrs overflow into leveldb (David Zafman)
- rados: fix a few error checks for the CLI (Josh Durgin)
- rgw: convert legacy bucket info objects on demand (Yehuda Sadeh)
- rgw: fix bug causing system users to lose privileges (Yehuda Sadeh)
- rgw: fix CORS bugs related to headers and case sensitivity (Robin H. Johnson)
- rgw: fix multipart object listing (Yehuda Sadeh)
- rgw: fix racing object creations (Yehuda Sadeh)
- rgw: fix racing object put and delete (Yehuda Sadeh)

- rgw: fix S3 auth when using response-* query string params (Sylvain Munaut)
- rgw: use correct secret key for POST authentication (Robin H. Johnson)

For more detailed information, see `the complete changelog`.

11.51 v0.67.5 "Dumpling"

This release includes a few critical bug fixes for the radosgw, including a fix for hanging operations on large objects. There are also several bug fixes for radosgw multi-site replications, and a few backported features. Also, notably, the 'osd perf' command (which dumps recent performance information about active OSDs) has been backported.

We recommend that all 0.67.x Dumpling users upgrade.

11.51.1 Notable changes

- ceph-fuse: fix crash in caching code
- mds: fix looping in populate_mydir()
- mds: fix standby-replay race
- mon: accept 'osd pool set ...' as string
- mon: backport: 'osd perf' command to dump recent OSD performance stats
- osd: add feature compat check for upcoming object sharding
- osd: fix osd bench block size argument
- rbd.py: increase parent name size limit
- rgw: backport: allow wildcard in supported keystone roles
- rgw: backport: improve swift COPY behavior
- rgw: backport: log and open admin socket by default
- rgw: backport: validate S3 tokens against keystone
- rgw: fix bucket removal
- rgw: fix client error code for chunked PUT failure
- rgw: fix hang on large object GET
- rgw: fix rare use-after-free
- rgw: various DR bug fixes
- sysvinit, upstart: prevent starting daemons using both init systems

For more detailed information, see `the complete changelog`.

11.52 v0.67.4 "Dumpling"

This point release fixes an important performance issue with radosgw, keystone authentication token caching, and CORS. All users (especially those of rgw) are encouraged to upgrade.

11.52.1 Notable changes

- crush: fix invalidation of cached names
- crushtool: do not crash on non-unique bucket ids
- mds: be more careful when decoding LogEvents
- mds: fix heap check debugging commands
- mon: avoid rebuilding old full osdmaps
- mon: fix 'ceph crush move ...'
- mon: fix 'ceph osd crush reweight ...'
- mon: fix writeout of full osdmaps during trim
- mon: limit size of transactions
- mon: prevent both unmanaged and pool snaps
- osd: disable xattr size limit (prevents upload of large rgw objects)
- osd: fix recovery op throttling
- osd: fix throttling of log messages for very slow requests
- rgw: drain pending requests before completing write
- rgw: fix CORS
- rgw: fix inefficient list::size() usage
- rgw: fix keystone token expiration
- rgw: fix minor memory leaks
- rgw: fix null termination of buffer

For more detailed information, see `the complete changelog`.

11.53 v0.67.3 "Dumpling"

This point release fixes a few important performance regressions with the OSD (both with CPU and disk utilization), as well as several other important but less common problems. We recommend that all production users upgrade.

11.53.1 Notable Changes

- ceph-disk: partprobe after creation journal partition
- ceph-disk: specify fs type when mounting
- ceph-post-file: new utility to help share logs and other files with ceph developers
- libcephfs: fix truncate vs readahead race (crash)
- mds: fix flock/fcntl lock deadlock
- mds: fix rejoin loop when encountering pre-dumpling backpointers
- mon: allow name and addr discovery during election stage
- mon: always refresh after Paxos store_state (fixes recovery corner case)

- mon: fix off-by-4x bug with osd byte counts
- osd: add and disable 'pg log keys debug' by default
- osd: add option to disable throttling
- osd: avoid leveldb iterators for pg log append and trim
- osd: fix readdir_r invocations
- osd: use fdatasync instead of sync
- radosgw: fix sysvinit script return status
- rbd: relicense as LGPL2
- rgw: flush pending data on multipart upload
- rgw: recheck object name during S3 POST
- rgw: reorder init/startup
- rpm: fix debuginfo package build

For more detailed information, see `the complete changelog`.

11.54 v0.67.2 "Dumpling"

This is an imporant point release for Dumpling. Most notably, it fixes a problem when upgrading directly from v0.56.x Bobtail to v0.67.x Dumpling (without stopping at v0.61.x Cuttlefish along the way). It also fixes a problem with the CLI parsing of the CEPH_ARGS environment variable, high CPU utilization by the ceph-osd daemons, and cleans up the radosgw shutdown sequence.

11.54.1 Notable Changes

- objecter: resend linger requests when cluster goes from full to non-full
- ceph: parse CEPH_ARGS environment variable
- librados: fix small memory leak
- osd: remove old log objects on upgrade (fixes bobtail -> dumpling jump)
- osd: disable PGLog::check() via config option (fixes CPU burn)
- rgw: drain requests on shutdown
- rgw: misc memory leaks on shutdown

For more detailed information, see `the complete changelog`.

11.55 v0.67.1 "Dumpling"

This is a minor point release for Dumpling that fixes problems with OpenStack and librbd hangs when caching is disabled.

11.55.1 Notable changes

- librados, librbd: fix constructor for python bindings with certain usages (in particular, that used by OpenStack)
- librados, librbd: fix aio_flush wakeup when cache is disabled
- librados: fix locking for aio completion refcounting
- fixes 'ceph –admin-daemon ...' command error code on error
- fixes 'ceph daemon ... config set ...' command for boolean config options.

For more detailed information, see `the complete changelog`.

11.56 v0.67 "Dumpling"

This is the fourth major release of Ceph, code-named "Dumpling." The headline features for this release include:

- Multi-site support for radosgw. This includes the ability to set up separate "regions" in the same or different Ceph clusters that share a single S3/Swift bucket/container namespace.
- RESTful API endpoint for Ceph cluster administration. ceph-rest-api, a wrapper around ceph_rest_api.py, can be used to start up a test single-threaded HTTP server that provides access to cluster information and administration in very similar ways to the ceph commandline tool. ceph_rest_api.py can be used as a WSGI application for deployment in a more-capable web server. See ceph-rest-api.8 for more.
- Object namespaces in librados.

11.56.1 Upgrade Sequencing

It is possible to do a rolling upgrade from Cuttlefish to Dumpling.

1. Upgrade ceph-common on all nodes that will use the command line 'ceph' utility.
2. Upgrade all monitors (upgrade ceph package, restart ceph-mon daemons). This can happen one daemon or host at a time. Note that because cuttlefish and dumpling monitors can't talk to each other, all monitors should be upgraded in relatively short succession to minimize the risk that an a untimely failure will reduce availability.
3. Upgrade all osds (upgrade ceph package, restart ceph-osd daemons). This can happen one daemon or host at a time.
4. Upgrade radosgw (upgrade radosgw package, restart radosgw daemons).

11.56.2 Upgrading from v0.66

- There is monitor internal protocol change, which means that v0.67 ceph-mon daemons cannot talk to v0.66 or older daemons. We recommend upgrading all monitors at once (or in relatively quick succession) to minimize the possibility of downtime.
- The output of 'ceph status –format=json' or 'ceph -s –format=json' has changed to return status information in a more structured and usable format.
- The 'ceph pg dump_stuck [threshold]' command used to require a –threshold or -t prefix to the threshold argument, but now does not.
- Many more ceph commands now output formatted information; select with '–format=<format>', where <format> can be 'json', 'json-pretty', 'xml', or 'xml-pretty'.

- The ‘ceph pg <pgid> ...’ commands (like ‘ceph pg <pgid> query’) are deprecated in favor of ‘ceph tell <pgid> ...’. This makes the distinction between ‘ceph pg <command> <pgid>’ and ‘ceph pg <pgid> <command>’ less awkward by making it clearer that the ‘tell’ commands are talking to the OSD serving the placement group, not the monitor.
- The ‘ceph –admin-daemon <path> <command ...>’ used to accept the command and arguments as either a single string or as separate arguments. It will now only accept the command spread across multiple arguments. This means that any script which does something like:

```
ceph --admin-daemon /var/run/ceph/ceph-osd.0.asok 'config set debug_ms 1'
```

  needs to remove the quotes. Also, note that the above can now be shortened to:

```
ceph daemon osd.0 config set debug_ms 1
```

- The radosgw caps were inconsistently documented to be either ‘mon = allow r’ or ‘mon = allow rw’. The ‘mon = allow rw’ is required for radosgw to create its own pools. All documentation has been updated accordingly.
- The radosgw copy object operation may return extra progress info during the operation. At this point it will only happen when doing cross zone copy operations. The S3 response will now return extra <Progress> field under the <CopyResult> container. The Swift response will now send the progress as a json array.
- In v0.66 and v0.65 the HASHPSPOOL pool flag was enabled by default for new pools, but has been disabled again until Linux kernel client support reaches more distributions and users.
- ceph-osd now requires a max file descriptor limit (e.g., `ulimit -n ...`) of at least filestore_wbthrottle_(xfs|btrfs)_inodes_hard_limit (5000 by default) in order to accomodate the new write back throttle system. On Ubuntu, upstart now sets the fd limit to 32k. On other platforms, the sysvinit script will set it to 32k by default (still overrideable via max_open_files). If this field has been customized in ceph.conf it should likely be adjusted upwards.

11.56.3 Upgrading from v0.61 “Cuttlefish”

In addition to the above notes about upgrading from v0.66:

- There has been a huge revamp of the ‘ceph’ command-line interface implementation. The `ceph-common` client library needs to be upgrade before `ceph-mon` is restarted in order to avoid problems using the CLI (the old `ceph` client utility cannot talk to the new `ceph-mon`).
- The CLI is now very careful about sending the ‘status’ one-liner output to stderr and command output to stdout. Scripts relying on output should take care.
- The ‘ceph osd tell ...’ and ‘ceph mon tell ...’ commands are no longer supported. Any callers should use:

```
ceph tell osd.<id or *> ...
ceph tell mon.<id or name or *> ...
```

  The ‘ceph mds tell ...’ command is still there, but will soon also transition to ‘ceph tell mds.<id or name or *> ...’

- The ‘ceph osd crush add ...’ command used to take one of two forms:

```
ceph osd crush add 123 osd.123 <weight> <location ...>
ceph osd crush add osd.123 <weight> <location ...>
```

  This is because the id and crush name are redundant. Now only the simple form is supported, where the osd name/id can either be a bare id (integer) or name (osd.<id>):

```
ceph osd crush add osd.123 <weight> <location ...>
ceph osd crush add 123 <weight> <location ...>
```

- There is now a maximum RADOS object size, configurable via 'osd max object size', defaulting to 100 GB. Note that this has no effect on RBD, CephFS, or radosgw, which all stripe over objects. If you are using librados and storing objects larger than that, you will need to adjust 'osd max object size', and should consider using smaller objects instead.
- The 'osd min down {reporters|reports}' config options have been renamed to 'mon osd min down {reporters|reports}', and the documentation has been updated to reflect that these options apply to the monitors (who process failure reports) and not OSDs. If you have adjusted these settings, please update your `ceph.conf` accordingly.

11.56.4 Notable changes since v0.66

- mon: sync improvements (performance and robustness)
- mon: many bug fixes (paxos and services)
- mon: fixed bugs in recovery and io rate reporting (negative/large values)
- mon: collect metadata on osd performance
- mon: generate health warnings from slow or stuck requests
- mon: expanded –format=<json|xml|...> support for monitor commands
- mon: scrub function for verifying data integrity
- mon, osd: fix old osdmap trimming logic
- mon: enable leveldb caching by default
- mon: more efficient storage of PG metadata
- ceph-rest-api: RESTful endpoint for administer cluster (mirrors CLI)
- rgw: multi-region support
- rgw: infrastructure to support georeplication of bucket and user metadata
- rgw: infrastructure to support georeplication of bucket data
- rgw: COPY object support between regions
- rbd: /etc/ceph/rbdmap file for mapping rbd images on startup
- osd: many bug fixes
- osd: limit number of incremental osdmaps sent to peers (could cause osds to be wrongly marked down)
- osd: more efficient small object recovery
- osd, librados: support for object namespaces
- osd: automatically enable xattrs on leveldb as necessary
- mds: fix bug in LOOKUPINO (used by nfs reexport)
- mds: fix O_TRUNC locking
- msgr: fixed race condition in inter-osd network communication
- msgr: fixed various memory leaks related to network sessions
- ceph-disk: fixes for unusual device names, partition detection
- hypertable: fixes for hypertable CephBroker bindings
- use SSE4.2 crc32c instruction if present

11.56.5 Notable changes since v0.61 “Cuttlefish”

- add ‘config get’ admin socket command
- ceph-conf: –show-config-value now reflects daemon defaults
- ceph-disk: add ‘[un]suppress-active DEV’ command
- ceph-disk: avoid mounting over an existing osd in /var/lib/ceph/osd/*
- ceph-disk: fixes for unusual device names, partition detection
- ceph-disk: improved handling of odd device names
- ceph-disk: many fixes for RHEL/CentOS, Fedora, wheezy
- ceph-disk: simpler, more robust locking
- ceph-fuse, libcephfs: fix a few caps revocation bugs
- ceph-fuse, libcephfs: fix read zeroing at EOF
- ceph-fuse, libcephfs: fix request refcounting bug (hang on shutdown)
- ceph-fuse, libcephfs: fix truncatation bug on >4MB files (Yan, Zheng)
- ceph-fuse, libcephfs: fix for cap release/hang
- ceph-fuse: add ioctl support
- ceph-fuse: fixed long-standing O_NOATIME vs O_LAZY bug
- ceph-rest-api: RESTful endpoint for administer cluster (mirrors CLI)
- ceph, librados: fix resending of commands on mon reconnect
- daemons: create /var/run/ceph as needed
- debian wheezy: fix udev rules
- debian, specfile: packaging cleanups
- debian: fix upstart behavior with upgrades
- debian: rgw: stop daemon on uninstall
- debian: stop daemons on uninstall; fix dependencies
- hypertable: fixes for hypertable CephBroker bindings
- librados python binding cleanups
- librados python: fix xattrs > 4KB (Josh Durgin)
- librados: configurable max object size (default 100 GB)
- librados: new calls to administer the cluster
- librbd: ability to read from local replicas
- librbd: locking tests (Josh Durgin)
- librbd: make default options/features for newly created images (e.g., via qemu-img) configurable
- librbd: parallelize delete, rollback, flatten, copy, resize
- many many fixes from static code analysis (Danny Al-Gaaf)
- mds: fix O_TRUNC locking
- mds: fix bug in LOOKUPINO (used by nfs reexport)

- mds: fix rare hang after client restart
- mds: fix several bugs (Yan, Zheng)
- mds: many backpointer improvements (Yan, Zheng)
- mds: many fixes for mds clustering
- mds: misc stability fixes (Yan, Zheng, Greg Farnum)
- mds: new robust open-by-ino support (Yan, Zheng)
- mds: support robust lookup by ino number (good for NFS) (Yan, Zheng)
- mon, ceph: huge revamp of CLI and internal admin API. (Dan Mick)
- mon, osd: fix old osdmap trimming logic
- mon, osd: many memory leaks fixed
- mon: better trim/compaction behavior
- mon: collect metadata on osd performance
- mon: enable leveldb caching by default
- mon: expanded –format=<json|xml|...> support for monitor commands
- mon: fix election timeout
- mon: fix leveldb compression, trimming
- mon: fix start fork behavior
- mon: fix units in 'ceph df' output
- mon: fix validation of mds ids from CLI commands
- mon: fixed bugs in recovery and io rate reporting (negative/large values)
- mon: generate health warnings from slow or stuck requests
- mon: many bug fixes (paxos and services, sync)
- mon: many stability fixes (Joao Luis)
- mon: more efficient storage of PG metadata
- mon: new –extract-monmap to aid disaster recovery
- mon: new capability syntax
- mon: scrub function for verifying data integrity
- mon: simplify PaxosService vs Paxos interaction, fix readable/writeable checks
- mon: sync improvements (performance and robustness)
- mon: tuning, performance improvements
- msgr: fix various memory leaks
- msgr: fixed race condition in inter-osd network communication
- msgr: fixed various memory leaks related to network sessions
- osd, librados: support for object namespaces
- osd, mon: optionally dump leveldb transactions to a log
- osd: automatically enable xattrs on leveldb as necessary

- osd: avoid osd flapping from asymmetric network failure
- osd: break blacklisted client watches (David Zafman)
- osd: close narrow journal race
- osd: do not use fadvise(DONTNEED) on XFS (data corruption on power cycle)
- osd: fix for an op ordering bug
- osd: fix handling for split after upgrade from bobtail
- osd: fix incorrect mark-down of osds
- osd: fix internal heartbeart timeouts when scrubbing very large objects
- osd: fix memory/network inefficiency during deep scrub
- osd: fixed problem with front-side heartbeats and mixed clusters (David Zafman)
- osd: limit number of incremental osdmaps sent to peers (could cause osds to be wrongly marked down)
- osd: many bug fixes
- osd: monitor both front and back interfaces
- osd: more efficient small object recovery
- osd: new writeback throttling (for less bursty write performance) (Sam Just)
- osd: pg log (re)writes are now vastly more efficient (faster peering) (Sam Just)
- osd: ping/heartbeat on public and private interfaces
- osd: prioritize recovery for degraded PGs
- osd: re-use partially deleted PG contents when present (Sam Just)
- osd: recovery and peering performance improvements
- osd: resurrect partially deleted PGs
- osd: verify both front and back network are working before rejoining cluster
- rados: clonedata command for cli
- radosgw-admin: create keys for new users by default
- rbd: /etc/ceph/rbdmap file for mapping rbd images on startup
- rgw: COPY object support between regions
- rgw: fix CORS bugs
- rgw: fix locking issue, user operation mask,
- rgw: fix radosgw-admin buckets list (Yehuda Sadeh)
- rgw: fix usage log scanning for large, untrimmed logs
- rgw: handle deep uri resources
- rgw: infrastructure to support georeplication of bucket and user metadata
- rgw: infrastructure to support georeplication of bucket data
- rgw: multi-region support
- sysvinit: fix enumeration of local daemons
- sysvinit: fix osd crush weight calculation when using -a

- sysvinit: handle symlinks in /var/lib/ceph/osd/*
- use SSE4.2 crc32c instruction if present

11.57 v0.66

11.57.1 Upgrading

- There is now a configurable maximum rados object size, defaulting to 100 GB. If you are using librados and storing objects larger than that, you will need to adjust 'osd max object size', and should consider using smaller objects instead.

11.57.2 Notable changes

- osd: pg log (re)writes are now vastly more efficient (faster peering) (Sam Just)
- osd: fixed problem with front-side heartbeats and mixed clusters (David Zafman)
- mon: tuning, performance improvements
- mon: simplify PaxosService vs Paxos interaction, fix readable/writeable checks
- rgw: fix radosgw-admin buckets list (Yehuda Sadeh)
- mds: support robust lookup by ino number (good for NFS) (Yan, Zheng)
- mds: fix several bugs (Yan, Zheng)
- ceph-fuse, libcephfs: fix truncatation bug on >4MB files (Yan, Zheng)
- ceph/librados: fix resending of commands on mon reconnect
- librados python: fix xattrs > 4KB (Josh Durgin)
- librados: configurable max object size (default 100 GB)
- msgr: fix various memory leaks
- ceph-fuse: fixed long-standing O_NOATIME vs O_LAZY bug
- ceph-fuse, libcephfs: fix request refcounting bug (hang on shutdown)
- ceph-fuse, libcephfs: fix read zeroing at EOF
- ceph-conf: –show-config-value now reflects daemon defaults
- ceph-disk: simpler, more robust locking
- ceph-disk: avoid mounting over an existing osd in /var/lib/ceph/osd/*
- sysvinit: handle symlinks in /var/lib/ceph/osd/*

11.58 v0.65

11.58.1 Upgrading

- Huge revamp of the 'ceph' command-line interface implementation. The `ceph-common` client library needs to be upgrade before `ceph-mon` is restarted in order to avoid problems using the CLI (the old `ceph` client utility cannot talk to the new `ceph-mon`).

- The CLI is now very careful about sending the 'status' one-liner output to stderr and command output to stdout. Scripts relying on output should take care.
- The 'ceph osd tell ...' and 'ceph mon tell ...' commands are no longer supported. Any callers should use:

```
ceph tell osd.<id or *> ...
ceph tell mon.<id or name or *> ...
```

  The 'ceph mds tell ...' command is still there, but will soon also transition to 'ceph tell mds.<id or name or *> ...'

- The 'ceph osd crush add ...' command used to take one of two forms:

```
ceph osd crush add 123 osd.123 <weight> <location ...>
ceph osd crush add osd.123 <weight> <location ...>
```

  This is because the id and crush name are redundant. Now only the simple form is supported, where the osd name/id can either be a bare id (integer) or name (osd.<id>):

```
ceph osd crush add osd.123 <weight> <location ...>
ceph osd crush add 123 <weight> <location ...>
```

- There is now a maximum RADOS object size, configurable via 'osd max object size', defaulting to 100 GB. Note that this has no effect on RBD, CephFS, or radosgw, which all stripe over objects.

11.58.2 Notable changes

- mon, ceph: huge revamp of CLI and internal admin API. (Dan Mick)
- mon: new capability syntax
- osd: do not use fadvise(DONTNEED) on XFS (data corruption on power cycle)
- osd: recovery and peering performance improvements
- osd: new writeback throttling (for less bursty write performance) (Sam Just)
- osd: ping/heartbeat on public and private interfaces
- osd: avoid osd flapping from asymmetric network failure
- osd: re-use partially deleted PG contents when present (Sam Just)
- osd: break blacklisted client watches (David Zafman)
- mon: many stability fixes (Joao Luis)
- mon, osd: many memory leaks fixed
- mds: misc stability fixes (Yan, Zheng, Greg Farnum)
- mds: many backpointer improvements (Yan, Zheng)
- mds: new robust open-by-ino support (Yan, Zheng)
- ceph-fuse, libcephfs: fix a few caps revocation bugs
- librados: new calls to administer the cluster
- librbd: locking tests (Josh Durgin)
- ceph-disk: improved handling of odd device names
- ceph-disk: many fixes for RHEL/CentOS, Fedora, wheezy
- many many fixes from static code analysis (Danny Al-Gaaf)

- daemons: create /var/run/ceph as needed

11.59 v0.64

11.59.1 Upgrading

- New pools now have the HASHPSPOOL flag set by default to provide better distribution over OSDs. Support for this feature was introduced in v0.59 and Linux kernel version v3.9. If you wish to access the cluster from an older kernel, set the 'osd pool default flag hashpspool = false' option in your ceph.conf prior to creating the cluster or creating new pools. Note that the presense of any pool in the cluster with the flag enabled will make the OSD require support from all clients.

11.59.2 Notable changes

- osd: monitor both front and back interfaces
- osd: verify both front and back network are working before rejoining cluster
- osd: fix memory/network inefficiency during deep scrub
- osd: fix incorrect mark-down of osds
- mon: fix start fork behavior
- mon: fix election timeout
- mon: better trim/compaction behavior
- mon: fix units in 'ceph df' output
- mon, osd: misc memory leaks
- librbd: make default options/features for newly created images (e.g., via qemu-img) configurable
- mds: many fixes for mds clustering
- mds: fix rare hang after client restart
- ceph-fuse: add ioctl support
- ceph-fuse/libcephfs: fix for cap release/hang
- rgw: handle deep uri resources
- rgw: fix CORS bugs
- ceph-disk: add '[un]suppress-active DEV' command
- debian: rgw: stop daemon on uninstall
- debian: fix upstart behavior with upgrades

11.60 v0.63

11.60.1 Upgrading

- The 'osd min down {reporters|reports}' config options have been renamed to 'mon osd min down {reporters|reports}', and the documentation has been updated to reflect that these options apply to the monitors (who process failure reports) and not OSDs. If you have adjusted these settings, please update your `ceph.conf` accordingly.

11.60.2 Notable Changes

- librbd: parallelize delete, rollback, flatten, copy, resize
- librbd: ability to read from local replicas
- osd: resurrect partially deleted PGs
- osd: prioritize recovery for degraded PGs
- osd: fix internal heartbeart timeouts when scrubbing very large objects
- osd: close narrow journal race
- rgw: fix usage log scanning for large, untrimmed logs
- rgw: fix locking issue, user operation mask,
- initscript: fix osd crush weight calculation when using -a
- initscript: fix enumeration of local daemons
- mon: several fixes to paxos, sync
- mon: new –extract-monmap to aid disaster recovery
- mon: fix leveldb compression, trimming
- add 'config get' admin socket command
- rados: clonedata command for cli
- debian: stop daemons on uninstall; fix dependencies
- debian wheezy: fix udev rules
- many many small fixes from coverity scan

11.61 v0.62

11.61.1 Notable Changes

- mon: fix validation of mds ids from CLI commands
- osd: fix for an op ordering bug
- osd, mon: optionally dump leveldb transactions to a log
- osd: fix handling for split after upgrade from bobtail
- debian, specfile: packaging cleanups
- radosgw-admin: create keys for new users by default
- librados python binding cleanups
- misc code cleanups

11.62 v0.61.9 “Cuttlefish”

This point release resolves several low to medium-impact bugs across the code base, and fixes a performance problem (CPU utilization) with radosgw. We recommend that all production cuttlefish users upgrade.

11.62.1 Notable Changes

- ceph, ceph-authtool: fix help (Danny Al-Gaaf)
- ceph-disk: partprobe after creating journal partition
- ceph-disk: specific fs type when mounting (Alfredo Deza)
- ceph-fuse: fix bug when compiled against old versions
- ceph-fuse: fix use-after-free in caching code (Yan, Zheng)
- ceph-fuse: misc caching bugs
- ceph.spec: remove incorrect mod_fcgi dependency (Gary Lowell)
- crush: fix name caching
- librbd: fix bug when unpausing cluster (Josh Durgin)
- mds: fix LAZYIO lock hang
- mds: fix bug in file size recovery (after client crash)
- mon: fix paxos recovery corner case
- osd: fix exponential backoff for slow request warnings (Loic Dachary)
- osd: fix readdir_r usage
- osd: fix startup for long-stopped OSDs
- rgw: avoid std::list::size() to avoid wasting CPU cycles (Yehuda Sadeh)
- rgw: drain pending requests during write (fixes data safety issue) (Yehuda Sadeh)
- rgw: fix authenticated users group ACL check (Yehuda Sadeh)
- rgw: fix bug in POST (Yehuda Sadeh)
- rgw: fix sysvinit script ‘status’ command, return value (Danny Al-Gaaf)
- rgw: reduce default log level (Yehuda Sadeh)

For more detailed information, see `the complete changelog`.

11.63 v0.61.8 “Cuttlefish”

This release includes a number of important issues, including rare race conditions in the OSD, a few monitor bugs, and fixes for RBD flush behavior. We recommend that production users upgrade at their convenience.

11.63.1 Notable Changes

- librados: fix async aio completion wakeup
- librados: fix aio completion locking
- librados: fix rare deadlock during shutdown
- osd: fix race when queueing recovery operations
- osd: fix possible race during recovery
- osd: optionally preload rados classes on startup (disabled by default)
- osd: fix journal replay corner condition
- osd: limit size of peering work queue batch (to speed up peering)
- mon: fix paxos recovery corner case
- mon: fix rare hang when monmap updates during an election
- mon: make 'osd pool mksnap ...' avoid exposing uncommitted state
- mon: make 'osd pool rmsnap ...' not racy, avoid exposing uncommitted state
- mon: fix bug during mon cluster expansion
- rgw: fix crash during multi delete operation
- msgr: fix race conditions during osd network reinitialization
- ceph-disk: apply mount options when remounting

For more detailed information, see `the complete changelog`.

11.64 v0.61.7 "Cuttlefish"

This release fixes another regression preventing monitors to start after undergoing certain upgrade sequences, as well as some corner cases with Paxos and support for unusual device names in ceph-disk/ceph-deploy.

11.64.1 Notable Changes

- mon: fix regression in latest full osdmap retrieval
- mon: fix a long-standing bug in a paxos corner case
- ceph-disk: improved support for unusual device names (e.g., /dev/cciss/c0d0)

For more detailed information, see `the complete changelog`.

11.65 v0.61.6 "Cuttlefish"

This release fixes a regression in v0.61.5 that could prevent monitors from restarting. This affects any cluster that was upgraded from a previous version of Ceph (and not freshly created with v0.61.5).

All users are strongly recommended to upgrade.

11.65.1 Notable Changes

- mon: record latest full osdmap
- mon: work around previous bug in which latest full osdmap is not recorded
- mon: avoid scrub while updating

For more detailed information, see `the complete changelog`.

11.66 v0.61.5 "Cuttlefish"

This release most improves stability of the monitor and fixes a few bugs with the ceph-disk utility (used by ceph-deploy). We recommand that all v0.61.x users upgrade.

11.66.1 Upgrading

- This release fixes a 32-bit vs 64-bit arithmetic bug with the feature bits. An unfortunate consequence of the fix is that 0.61.4 (or earlier) ceph-mon daemons can't form a quorum with 0.61.5 (or later) monitors. To avoid the possibility of service disruption, we recommend you upgrade all monitors at once.

11.66.2 Notable Changes

- mon: misc sync improvements (faster, more reliable, better tuning)
- mon: enable leveldb cache by default (big performance improvement)
- mon: new scrub feature (primarily for diagnostic, testing purposes)
- mon: fix occasional leveldb assertion on startup
- mon: prevent reads until initial state is committed
- mon: improved logic for trimming old osdmaps
- mon: fix pick_addresses bug when expanding mon cluster
- mon: several small paxos fixes, improvements
- mon: fix bug osdmap trim behavior
- osd: fix several bugs with PG stat reporting
- osd: limit number of maps shared with peers (which could cause domino failures)
- rgw: fix radosgw-admin buckets list (for all buckets)
- mds: fix occasional client failure to reconnect
- mds: fix bad list traversal after unlink
- mds: fix underwater dentry cleanup (occasional crash after mds restart)
- libcephfs, ceph-fuse: fix occasional hangs on umount
- libcephfs, ceph-fuse: fix old bug with O_LAZY vs O_NOATIME confusion
- ceph-disk: more robust journal device detection on RHEL/CentOS
- ceph-disk: better, simpler locking

- ceph-disk: do not inadvertantely mount over existing osd mounts
- ceph-disk: better handling for unusual device names
- sysvinit, upstart: handle symlinks in /var/lib/ceph/*

For more detailed information, see `the complete changelog`.

11.67 v0.61.4 "Cuttlefish"

This release resolves a possible data corruption on power-cycle when using XFS, a few outstanding problems with monitor sync, several problems with ceph-disk and ceph-deploy operation, and a problem with OSD memory usage during scrub.

11.67.1 Upgrading

- No issues.

11.67.2 Notable Changes

- mon: fix daemon exit behavior when error is encountered on startup
- mon: more robust sync behavior
- osd: do not use sync_file_range(2), posix_fadvise(...DONTNEED) (can cause data corruption on power loss on XFS)
- osd: avoid unnecessary log rewrite (improves peering speed)
- osd: fix scrub efficiency bug (problematic on old clusters)
- rgw: fix listing objects that start with underscore
- rgw: fix deep URI resource, CORS bugs
- librados python binding: fix truncate on 32-bit architectures
- ceph-disk: fix udev rules
- rpm: install sysvinit script on package install
- ceph-disk: fix OSD start on machine reboot on Debian wheezy
- ceph-disk: activate OSD when journal device appears second
- ceph-disk: fix various bugs on RHEL/CentOS 6.3
- ceph-disk: add 'zap' command
- ceph-disk: add '[un]suppress-activate' command for preparing spare disks
- upstart: start on runlevel [2345] (instead of after the first network interface starts)
- ceph-fuse, libcephfs: handle mds session reset during session open
- ceph-fuse, libcephfs: fix two capability revocation bugs
- ceph-fuse: fix thread creation on startup
- all daemons: create /var/run/ceph directory on startup if missing

For more detailed information, see `the complete changelog`.

11.68 v0.61.3 “Cuttlefish”

This release resolves a number of problems with the monitors and leveldb that users have been seeing. Please upgrade.

11.68.1 Upgrading

- There is one known problem with mon upgrades from bobtail. If the ceph-mon conversion on startup is aborted or fails for some reason, we do not correctly error out, but instead continue with (in certain cases) odd results. Please be careful if you have to restart the mons during the upgrade. A 0.61.4 release with a fix will be out shortly.
- In the meantime, for current cuttlefish users, v0.61.3 is safe to use.

11.68.2 Notable Changes

- mon: paxos state trimming fix (resolves runaway disk usage)
- mon: finer-grained compaction on trim
- mon: discard messages from disconnected clients (lowers load)
- mon: leveldb compaction and other stats available via admin socket
- mon: async compaction (lower overhead)
- mon: fix bug incorrectly marking osds down with insufficient failure reports
- osd: fixed small bug in pg request map
- osd: avoid rewriting pg info on every osdmap
- osd: avoid internal heartbeta timeouts when scrubbing very large objects
- osd: fix narrow race with journal replay
- mon: fixed narrow pg split race
- rgw: fix leaked space when copying object
- rgw: fix iteration over large/untrimmed usage logs
- rgw: fix locking issue with ops log socket
- rgw: require matching version of librados
- librbd: make image creation defaults configurable (e.g., create format 2 images via qemu-img)
- fix units in ‘ceph df’ output
- debian: fix prerm/postinst hooks to start/stop daemons appropriately
- upstart: allow uppercase daemons names (and thus hostnames)
- sysvinit: fix enumeration of local daemons by type
- sysvinit: fix osd weight calcuation when using -a
- fix build on unsigned char platforms (e.g., arm)

For more detailed information, see `the complete changelog`.

11.69 v0.61.2 "Cuttlefish"

This release disables a monitor debug log that consumes disk space and fixes a bug when upgrade some monitors from bobtail to cuttlefish.

11.69.1 Notable Changes

- mon: fix conversion of stores with duplicated GV values
- mon: disable 'mon debug dump transactions' by default

For more detailed information, see `the complete changelog`.

11.70 v0.61.1 "Cuttlefish"

This release fixes a problem when upgrading a bobtail cluster that had snapshots to cuttlefish.

11.70.1 Notable Changes

- osd: handle upgrade when legacy snap collections are present; repair from previous failed restart
- ceph-create-keys: fix race with ceph-mon startup (which broke 'ceph-deploy gatherkeys ...')
- ceph-create-keys: gracefully handle bad response from ceph-osd
- sysvinit: do not assume default osd_data when automatically weighting OSD
- osd: avoid crash from ill-behaved classes using getomapvals
- debian: fix squeeze dependency
- mon: debug options to log or dump leveldb transactions

For more detailed information, see `the complete changelog`.

11.71 v0.61 "Cuttlefish"

11.71.1 Upgrading from v0.60

- The ceph-deploy tool is now the preferred method of provisioning new clusters. For existing clusters created via mkcephfs that would like to transition to the new tool, there is a migration path, documented at Transitioning to ceph-deploy.
- The sysvinit script (/etc/init.d/ceph) will now verify (and, if necessary, update) the OSD's position in the CRUSH map on startup. (The upstart script has always worked this way.) By default, this ensures that the OSD is under a 'host' with a name that matches the hostname (`hostname -s`). Legacy clusters create with mkcephfs do this by default, so this should not cause any problems, but legacy clusters with customized CRUSH maps with an alternate structure should set `osd crush update on start = false`.
- radosgw-admin now uses the term zone instead of cluster to describe each instance of the radosgw data store (and corresponding collection of radosgw daemons). The usage for the radosgw-admin command and the 'rgw zone root pool' config options have changed accordingly.

- rbd progress indicators now go to standard error instead of standard out. (You can disable progress with –no-progress.)
- The 'rbd resize ...' command now requires the –allow-shrink option when resizing to a smaller size. Expanding images to a larger size is unchanged.
- Please review the changes going back to 0.56.4 if you are upgrading all the way from bobtail.
- The old 'ceph stop_cluster' command has been removed.
- The sysvinit script now uses the ceph.conf file on the remote host when starting remote daemons via the '-a' option. Note that if '-a' is used in conjunction with '-c path', the path must also be present on the remote host (it is not copied to a temporary file, as it was previously).

11.71.2 Upgrading from v0.56.4 "Bobtail"

Please see Upgrading from Bobtail to Cuttlefish for details.

- The ceph-deploy tool is now the preferred method of provisioning new clusters. For existing clusters created via mkcephfs that would like to transition to the new tool, there is a migration path, documented at Transitioning to ceph-deploy.
- The sysvinit script (/etc/init.d/ceph) will now verify (and, if necessary, update) the OSD's position in the CRUSH map on startup. (The upstart script has always worked this way.) By default, this ensures that the OSD is under a 'host' with a name that matches the hostname (`hostname -s`). Legacy clusters create with mkcephfs do this by default, so this should not cause any problems, but legacy clusters with customized CRUSH maps with an alternate structure should set `osd crush update on start = false`.
- radosgw-admin now uses the term zone instead of cluster to describe each instance of the radosgw data store (and corresponding collection of radosgw daemons). The usage for the radosgw-admin command and the 'rgw zone root pool' config optoins have changed accordingly.
- rbd progress indicators now go to standard error instead of standard out. (You can disable progress with –no-progress.)
- The 'rbd resize ...' command now requires the –allow-shrink option when resizing to a smaller size. Expanding images to a larger size is unchanged.
- Please review the changes going back to 0.56.4 if you are upgrading all the way from bobtail.
- The old 'ceph stop_cluster' command has been removed.
- The sysvinit script now uses the ceph.conf file on the remote host when starting remote daemons via the '-a' option. Note that if '-a' is used in conjuction with '-c path', the path must also be present on the remote host (it is not copied to a temporary file, as it was previously).
- The monitor is using a completely new storage strategy and intra-cluster protocol. This means that cuttlefish and bobtail monitors do not talk to each other. When you upgrade each one, it will convert its local data store to the new format. Once you upgrade a majority, the quorum will be formed using the new protocol and the old monitors will be blocked out until they too get upgraded. For this reason, we recommend not running a mixed-version cluster for very long.
- ceph-mon now requires the creation of its data directory prior to –mkfs, similarly to what happens on ceph-osd. This directory is no longer automatically created, and custom scripts should be adjusted to reflect just that.
- The monitor now enforces that MDS names be unique. If you have multiple daemons start with with the same id (e.g., `mds.a`) the second one will implicitly mark the first as failed. This makes things less confusing and makes a daemon restart faster (we no longer wait for the stopped daemon to time out) but existing multi-mds configurations may need to be adjusted accordingly to give daemons unique names.

- The 'ceph osd pool delete <poolname>' and 'rados rmpool <poolname>' now have safety interlocks with loud warnings that make you confirm pool removal. Any scripts curenty rely on these functions zapping data without confirmation need to be adjusted accordingly.

11.71.3 Notable Changes from v0.60

- rbd: incremental backups
- rbd: only set STRIPINGV2 feature if striping parameters are incompatible with old versions
- rbd: require –allow-shrink for resizing images down
- librbd: many bug fixes
- rgw: management REST API
- rgw: fix object corruption on COPY to self
- rgw: new sysvinit script for rpm-based systems
- rgw: allow buckets with '_'
- rgw: CORS support
- mon: many fixes
- mon: improved trimming behavior
- mon: fix data conversion/upgrade problem (from bobtail)
- mon: ability to tune leveldb
- mon: config-keys service to store arbitrary data on monitor
- mon: 'osd crush add|link|unlink|add-bucket ...' commands
- mon: trigger leveldb compaction on trim
- osd: per-rados pool quotas (objects, bytes)
- osd: tool to export, import, and delete PGs from an individual OSD data store
- osd: notify mon on clean shutdown to avoid IO stall
- osd: improved detection of corrupted journals
- osd: ability to tune leveldb
- osd: improve client request throttling
- osd, librados: fixes to the LIST_SNAPS operation
- osd: improvements to scrub error repair
- osd: better prevention of wedging OSDs with ENOSPC
- osd: many small fixes
- mds: fix xattr handling on root inode
- mds: fixed bugs in journal replay
- mds: many fixes
- librados: clean up snapshot constant definitions
- libcephfs: calls to query CRUSH topology (used by Hadoop)
- ceph-fuse, libcephfs: misc fixes to mds session management

- ceph-fuse: disabled cache invalidation (again) due to potential deadlock with kernel
- sysvinit: try to start all daemons despite early failures
- ceph-disk: new 'list' command
- ceph-disk: hotplug fixes for RHEL/CentOS
- ceph-disk: fix creation of OSD data partitions on >2TB disks
- osd: fix udev rules for RHEL/CentOS systems
- fix daemon logging during initial startup

11.71.4 Notable changes from v0.56 "Bobtail"

- always use installed system leveldb (Gary Lowell)
- auth: ability to require new cephx signatures on messages (still off by default)
- buffer unit testing (Loic Dachary)
- ceph tool: some CLI interface cleanups
- ceph-disk: improve multicluster support, error handling (Sage Weil)
- ceph-disk: support for dm-crypt (Alexandre Marangone)
- ceph-disk: support for sysvinit, directories or partitions (not full disks)
- ceph-disk: fix mkfs args on old distros (Alexandre Marangone)
- ceph-disk: fix creation of OSD data partitions on >2TB disks
- ceph-disk: hotplug fixes for RHEL/CentOS
- ceph-disk: new 'list' command
- ceph-fuse, libcephfs: misc fixes to mds session management
- ceph-fuse: disabled cache invalidation (again) due to potential deadlock with kernel
- ceph-fuse: enable kernel cache invalidation (Sam Lang)
- ceph-fuse: fix statfs(2) reporting
- ceph-fuse: session handling cleanup, bug fixes (Sage Weil)
- crush: ability to create, remove rules via CLI
- crush: update weights for all instances of an item, not just the first (Sage Weil)
- fix daemon logging during initial startup
- fixed log rotation (Gary Lowell)
- init-ceph, mkcephfs: close a few security holes with -a (Sage Weil)
- libcephfs: calls to query CRUSH topology (used by Hadoop)
- libcephfs: many fixes, cleanups with the Java bindings
- libcephfs: new topo API requests for Hadoop (Noah Watkins)
- librados: clean up snapshot constant definitions
- librados: fix linger bugs (Josh Durgin)
- librbd: fixed flatten deadlock (Josh Durgin)

- librbd: fixed some locking issues with flatten (Josh Durgin)
- librbd: many bug fixes
- librbd: optionally wait for flush before enabling writeback (Josh Durgin)
- many many cleanups (Danny Al-Gaaf)
- mds, ceph-fuse: fix bugs with replayed requests after MDS restart (Sage Weil)
- mds, ceph-fuse: manage layouts via xattrs
- mds: allow xattrs on root
- mds: fast failover between MDSs (enforce unique mds names)
- mds: fix xattr handling on root inode
- mds: fixed bugs in journal replay
- mds: improve session cleanup (Sage Weil)
- mds: many fixes (Yan Zheng)
- mds: misc bug fixes with clustered MDSs and failure recovery
- mds: misc bug fixes with readdir
- mds: new encoding for all data types (to allow forward/backward compatbility) (Greg Farnum)
- mds: store and update backpointers/traces on directory, file objects (Sam Lang)
- mon: 'osd crush add|link|unlink|add-bucket ...' commands
- mon: ability to tune leveldb
- mon: approximate recovery, IO workload stats
- mon: avoid marking entire CRUSH subtrees out (e.g., if an entire rack goes offline)
- mon: config-keys service to store arbitrary data on monitor
- mon: easy adjustment of crush tunables via 'ceph osd crush tunables ...'
- mon: easy creation of crush rules vai 'ceph osd rule ...'
- mon: fix data conversion/upgrade problem (from bobtail)
- mon: improved trimming behavior
- mon: many fixes
- mon: new 'ceph df [detail]' command
- mon: new checks for identifying and reporting clock drift
- mon: rearchitected to utilize single instance of paxos and a key/value store (Joao Luis)
- mon: safety check for pool deletion
- mon: shut down safely if disk approaches full (Joao Luis)
- mon: trigger leveldb compaction on trim
- msgr: fix comparison of IPv6 addresses (fixes monitor bringup via ceph-deploy, chef)
- msgr: fixed race in connection reset
- msgr: optionally tune TCP buffer size to avoid throughput collapse (Jim Schutt)
- much code cleanup and optimization (Danny Al-Gaaf)

- osd, librados: ability to list watchers (David Zafman)
- osd, librados: fixes to the LIST_SNAPS operation
- osd, librados: new listsnaps command (David Zafman)
- osd: a few journaling bug fixes
- osd: ability to tune leveldb
- osd: add 'noscrub', 'nodeepscrub' osdmap flags (David Zafman)
- osd: better prevention of wedging OSDs with ENOSPC
- osd: ceph-filestore-dump tool for debugging
- osd: connection handling bug fixes
- osd: deep-scrub omap keys/values
- osd: default to libaio for the journal (some performance boost)
- osd: fix hang in 'journal aio = true' mode (Sage Weil)
- osd: fix pg log trimming (avoids memory bloat on degraded clusters)
- osd: fix udev rules for RHEL/CentOS systems
- osd: fixed bug in journal checksums (Sam Just)
- osd: improved client request throttling
- osd: improved handling when disk fills up (David Zafman)
- osd: improved journal corruption detection (Sam Just)
- osd: improved detection of corrupted journals
- osd: improvements to scrub error repair
- osd: make tracking of object snapshot metadata more efficient (Sam Just)
- osd: many small fixes
- osd: misc fixes to PG split (Sam Just)
- osd: move pg info, log into leveldb (== better performance) (David Zafman)
- osd: notify mon on clean shutdown to avoid IO stall
- osd: per-rados pool quotas (objects, bytes)
- osd: refactored watch/notify infrastructure (fixes protocol, removes many bugs) (Sam Just)
- osd: support for improved hashing of PGs across OSDs via HASHPSPOOL pool flag and feature
- osd: tool to export, import, and delete PGs from an individual OSD data store
- osd: trim log more aggressively, avoid appearance of leak memory
- osd: validate snap collections on startup
- osd: verify snap collections on startup (Sam Just)
- radosgw: ACL grants in headers (Caleb Miles)
- radosgw: ability to listen to fastcgi via a port (Guilhem Lettron)
- radosgw: fix object copy onto self (Yehuda Sadeh)
- radosgw: misc fixes

- rbd-fuse: new tool, package
- rbd: avoid FIEMAP when importing from file (it can be buggy)
- rbd: incremental backups
- rbd: only set STRIPINGV2 feature if striping parameters are incompatible with old versions
- rbd: require –allow-shrink for resizing images down
- rbd: udevadm settle on map/unmap to avoid various races (Dan Mick)
- rbd: wait for udev to settle in strategic places (avoid spurious errors, failures)
- rgw: CORS support
- rgw: allow buckets with '_'
- rgw: fix Content-Length on 32-bit machines (Jan Harkes)
- rgw: fix log rotation
- rgw: fix object corruption on COPY to self
- rgw: fixed >4MB range requests (Jan Harkes)
- rgw: new sysvinit script for rpm-based systems
- rpm/deb: do not remove /var/lib/ceph on purge (v0.59 was the only release to do so)
- sysvinit: try to start all daemons despite early failures
- upstart: automatically set osd weight based on df (Guilhem Lettron)
- use less memory for logging by default

11.72 v0.60

11.72.1 Upgrading

- Please note that the recently added librados 'list_snaps' function call is in a state of flux and is changing slightly in v0.61. You are advised not to make use of it in v0.59 or v0.60.

11.72.2 Notable Changes

- osd: make tracking of object snapshot metadata more efficient (Sam Just)
- osd: misc fixes to PG split (Sam Just)
- osd: improve journal corruption detection (Sam Just)
- osd: improve handling when disk fills up (David Zafman)
- osd: add 'noscrub', 'nodeepscrub' osdmap flags (David Zafman)
- osd: fix hang in 'journal aio = true' mode (Sage Weil)
- ceph-disk-prepare: fix mkfs args on old distros (Alexandre Marangone)
- ceph-disk-activate: improve multicluster support, error handling (Sage Weil)
- librbd: optionally wait for flush before enabling writeback (Josh Durgin)
- crush: update weights for all instances of an item, not just the first (Sage Weil)

- mon: shut down safely if disk approaches full (Joao Luis)
- rgw: fix Content-Length on 32-bit machines (Jan Harkes)
- mds: store and update backpointers/traces on directory, file objects (Sam Lang)
- mds: improve session cleanup (Sage Weil)
- mds, ceph-fuse: fix bugs with replayed requests after MDS restart (Sage Weil)
- ceph-fuse: enable kernel cache invalidation (Sam Lang)
- libcephfs: new topo API requests for Hadoop (Noah Watkins)
- ceph-fuse: session handling cleanup, bug fixes (Sage Weil)
- much code cleanup and optimization (Danny Al-Gaaf)
- use less memory for logging by default
- upstart: automatically set osd weight based on df (Guilhem Lettron)
- init-ceph, mkcephfs: close a few security holes with -a (Sage Weil)
- rpm/deb: do not remove /var/lib/ceph on purge (v0.59 was the only release to do so)

11.73 v0.59

11.73.1 Upgrading

- The monitor is using a completely new storage strategy and intra-cluster protocol. This means that v0.59 and pre-v0.59 monitors do not talk to each other. When you upgrade each one, it will convert its local data store to the new format. Once you upgrade a majority, the quorum will be formed using the new protocol and the old monitors will be blocked out until they too get upgraded. For this reason, we recommend not running a mixed-version cluster for very long.
- ceph-mon now requires the creation of its data directory prior to –mkfs, similarly to what happens on ceph-osd. This directory is no longer automatically created, and custom scripts should be adjusted to reflect just that.

11.73.2 Notable Changes

- mon: rearchitected to utilize single instance of paxos and a key/value store (Joao Luis)
- mon: new 'ceph df [detail]' command
- osd: support for improved hashing of PGs across OSDs via HASHPSPOOL pool flag and feature
- osd: refactored watch/notify infrastructure (fixes protocol, removes many bugs) (Sam Just)
- osd, librados: ability to list watchers (David Zafman)
- osd, librados: new listsnaps command (David Zafman)
- osd: trim log more aggressively, avoid appearance of leak memory
- osd: misc split fixes
- osd: a few journaling bug fixes
- osd: connection handling bug fixes
- rbd: avoid FIEMAP when importing from file (it can be buggy)

- librados: fix linger bugs (Josh Durgin)
- librbd: fixed flatten deadlock (Josh Durgin)
- rgw: fixed >4MB range requests (Jan Harkes)
- rgw: fix log rotation
- mds: allow xattrs on root
- ceph-fuse: fix statfs(2) reporting
- msgr: optionally tune TCP buffer size to avoid throughput collapse (Jim Schutt)
- consume less memory for logging by default
- always use system leveldb (Gary Lowell)

11.74 v0.58

11.74.1 Upgrading

- The monitor now enforces that MDS names be unique. If you have multiple daemons start with with the same id (e.g., `mds.a`) the second one will implicitly mark the first as failed. This makes things less confusing and makes a daemon restart faster (we no longer wait for the stopped daemon to time out) but existing multi-mds configurations may need to be adjusted accordingly to give daemons unique names.

11.74.2 Notable Changes

- librbd: fixed some locking issues with flatten (Josh Durgin)
- rbd: udevadm settle on map/unmap to avoid various races (Dan Mick)
- osd: move pg info, log into leveldb (== better performance) (David Zafman)
- osd: fix pg log trimming (avoids memory bloat on degraded clusters)
- osd: fixed bug in journal checksums (Sam Just)
- osd: verify snap collections on startup (Sam Just)
- ceph-disk-prepare/activate: support for dm-crypt (Alexandre Marangone)
- ceph-disk-prepare/activate: support for sysvinit, directories or partitions (not full disks)
- msgr: fixed race in connection reset
- msgr: fix comparison of IPv6 addresses (fixes monitor bringup via ceph-deploy, chef)
- radosgw: fix object copy onto self (Yehuda Sadeh)
- radosgw: ACL grants in headers (Caleb Miles)
- radosgw: ability to listen to fastcgi via a port (Guilhem Lettron)
- mds: new encoding for all data types (to allow forward/backward compatbility) (Greg Farnum)
- mds: fast failover between MDSs (enforce unique mds names)
- crush: ability to create, remove rules via CLI
- many many cleanups (Danny Al-Gaaf)
- buffer unit testing (Loic Dachary)

- fixed log rotation (Gary Lowell)

11.75 v0.57

This development release has a lot of additional functionality accumulated over the last couple months. Most of the bug fixes (with the notable exception of the MDS related work) has already been backported to v0.56.x, and is not mentioned here.

11.75.1 Upgrading

- The 'ceph osd pool delete <poolname>' and 'rados rmpool <poolname>' now have safety interlocks with loud warnings that make you confirm pool removal. Any scripts curenty rely on these functions zapping data without confirmation need to be adjusted accordingly.

11.75.2 Notable Changes

- osd: default to libaio for the journal (some performance boost)
- osd: validate snap collections on startup
- osd: ceph-filestore-dump tool for debugging
- osd: deep-scrub omap keys/values
- ceph tool: some CLI interface cleanups
- mon: easy adjustment of crush tunables via 'ceph osd crush tunables ...'
- mon: easy creation of crush rules vai 'ceph osd rule ...'
- mon: approximate recovery, IO workload stats
- mon: avoid marking entire CRUSH subtrees out (e.g., if an entire rack goes offline)
- mon: safety check for pool deletion
- mon: new checks for identifying and reporting clock drift
- radosgw: misc fixes
- rbd: wait for udev to settle in strategic places (avoid spurious errors, failures)
- rbd-fuse: new tool, package
- mds, ceph-fuse: manage layouts via xattrs
- mds: misc bug fixes with clustered MDSs and failure recovery
- mds: misc bug fixes with readdir
- libcephfs: many fixes, cleanups with the Java bindings
- auth: ability to require new cephx signatures on messages (still off by default)

11.76 v0.56.7 “bobtail”

This bobtail update fixes a range of radosgw bugs (including an easily triggered crash from multi-delete), a possible data corruption issue with power failure on XFS, and several OSD problems, including a memory “leak” that will affect aged clusters.

11.76.1 Notable changes

- ceph-fuse: create finisher flags after fork()
- debian: fix prerm/postinst hooks; do not restart daemons on upgrade
- librados: fix async aio completion wakeup (manifests as rbd hang)
- librados: fix hang when osd becomes full and then not full
- librados: fix locking for aio completion refcounting
- librbd python bindings: fix stripe_unit, stripe_count
- librbd: make image creation default configurable
- mon: fix validation of mds ids in mon commands
- osd: avoid excessive disk updates during peering
- osd: avoid excessive memory usage on scrub
- osd: avoid heartbeat failure/suicide when scrubbing
- osd: misc minor bug fixes
- osd: use fdatasync instead of sync_file_range (may avoid xfs power-loss corruption)
- rgw: escape prefix correctly when listing objects
- rgw: fix copy attrs
- rgw: fix crash on multi delete
- rgw: fix locking/crash when using ops log socket
- rgw: fix usage logging
- rgw: handle deep uri resources

For more detailed information, see `the complete changelog`.

11.77 v0.56.6 “bobtail”

11.77.1 Notable changes

- rgw: fix garbage collection
- rpm: fix package dependencies

For more detailed information, see `the complete changelog`.

11.78 v0.56.5 "bobtail"

11.78.1 Upgrading

- ceph-disk[-prepare,-activate] behavior has changed in various ways. There should not be any compatibility issues, but chef users should be aware.

11.78.2 Notable changes

- mon: fix recording of quorum feature set (important for argonaut -> bobtail -> cuttlefish mon upgrades)
- osd: minor peering bug fixes
- osd: fix a few bugs when pools are renamed
- osd: fix occasionally corrupted pg stats
- osd: fix behavior when broken v0.56[.0] clients connect
- rbd: avoid FIEMAP ioctl on import (it is broken on some kernels)
- librbd: fixes for several request/reply ordering bugs
- librbd: only set STRIPINGV2 feature on new images when needed
- librbd: new async flush method to resolve qemu hangs (requires Qemu update as well)
- librbd: a few fixes to flatten
- ceph-disk: support for dm-crypt
- ceph-disk: many backports to allow bobtail deployments with ceph-deploy, chef
- sysvinit: do not stop starting daemons on first failure
- udev: fixed rules for redhat-based distros
- build fixes for raring

For more detailed information, see `the complete changelog`.

11.79 v0.56.4 "bobtail"

11.79.1 Upgrading

- There is a fix in the syntax for the output of 'ceph osd tree –format=json'.
- The MDS disk format has changed from prior releases *and* from v0.57. In particular, upgrades to v0.56.4 are safe, but you cannot move from v0.56.4 to v0.57 if you are using the MDS for CephFS; you must upgrade directly to v0.58 (or later) instead.

11.79.2 Notable changes

- mon: fix bug in bringup with IPv6
- reduce default memory utilization by internal logging (all daemons)
- rgw: fix for bucket removal

- rgw: reopen logs after log rotation
- rgw: fix multipat upload listing
- rgw: don't copy object when copied onto self
- osd: fix caps parsing for pools with - or _
- osd: allow pg log trimming when degraded, scrubbing, recoverying (reducing memory consumption)
- osd: fix potential deadlock when 'journal aio = true'
- osd: various fixes for collection creation/removal, rename, temp collections
- osd: various fixes for PG split
- osd: deep-scrub omap key/value data
- osd: fix rare bug in journal replay
- osd: misc fixes for snapshot tracking
- osd: fix leak in recovery reservations on pool deletion
- osd: fix bug in connection management
- osd: fix for op ordering when rebalancing
- ceph-fuse: report file system size with correct units
- mds: get and set directory layout policies via virtual xattrs
- mds: on-disk format revision (see upgrading note above)
- mkcephfs, init-ceph: close potential security issues with predictable filenames

For more detailed information, see `the complete changelog`.

11.80 v0.56.3 "bobtail"

This release has several bug fixes surrounding OSD stability. Most significantly, an issue with OSDs being unresponsive shortly after startup (and occasionally crashing due to an internal heartbeat check) is resolved. Please upgrade.

11.80.1 Upgrading

- A bug was fixed in which the OSDMap epoch for PGs without any IO requests was not recorded. If there are pools in the cluster that are completely idle (for example, the `data` and `metadata` pools normally used by CephFS), and a large number of OSDMap epochs have elapsed since the `ceph-osd` daemon was last restarted, those maps will get reprocessed when the daemon restarts. This process can take a while if there are a lot of maps. A workaround is to 'touch' any idle pools with IO prior to restarting the daemons after packages are upgraded:

```
rados bench 10 write -t 1 -b 4096 -p {POOLNAME}
```

  This will typically generate enough IO to touch every PG in the pool without generating significant cluster load, and also cleans up any temporary objects it creates.

11.80.2 Notable changes

- osd: flush peering work queue prior to start
- osd: persist osdmap epoch for idle PGs
- osd: fix and simplify connection handling for heartbeats
- osd: avoid crash on invalid admin command
- mon: fix rare races with monitor elections and commands
- mon: enforce that OSD reweights be between 0 and 1 (NOTE: not CRUSH weights)
- mon: approximate client, recovery bandwidth logging
- radosgw: fixed some XML formatting to conform to Swift API inconsistency
- radosgw: fix usage accounting bug; add repair tool
- radosgw: make fallback URI configurable (necessary on some web servers)
- librbd: fix handling for interrupted 'unprotect' operations
- mds, ceph-fuse: allow file and directory layouts to be modified via virtual xattrs

For more detailed information, see `the complete changelog`.

11.81 v0.56.2 "bobtail"

This release has a wide range of bug fixes, stability improvements, and some performance improvements. Please upgrade.

11.81.1 Upgrading

- The meaning of the 'osd scrub min interval' and 'osd scrub max interval' has changed slightly. The min interval used to be meaningless, while the max interval would only trigger a scrub if the load was sufficiently low. Now, the min interval option works the way the old max interval did (it will trigger a scrub after this amount of time if the load is low), while the max interval will force a scrub regardless of load. The default options have been adjusted accordingly. If you have customized these in ceph.conf, please review their values when upgrading.
- CRUSH maps that are generated by default when calling `ceph-mon --mkfs` directly now distribute replicas across hosts instead of across OSDs. Any provisioning tools that are being used by Ceph may be affected, although probably for the better, as distributing across hosts is a much more commonly sought behavior. If you use `mkcephfs` to create the cluster, the default CRUSH rule is still inferred by the number of hosts and/or racks in the initial ceph.conf.

11.81.2 Notable changes

- osd: snapshot trimming fixes
- osd: scrub snapshot metadata
- osd: fix osdmap trimming
- osd: misc peering fixes
- osd: stop heartbeating with peers if internal threads are stuck/hung

- osd: PG removal is friendlier to other workloads
- osd: fix recovery start delay (was causing very slow recovery)
- osd: fix scheduling of explicitly requested scrubs
- osd: fix scrub interval config options
- osd: improve recovery vs client io tuning
- osd: improve 'slow request' warning detail for better diagnosis
- osd: default CRUSH map now distributes across hosts, not OSDs
- osd: fix crash on 32-bit hosts triggered by librbd clients
- librbd: fix error handling when talking to older OSDs
- mon: fix a few rare crashes
- ceph command: ability to easily adjust CRUSH tunables
- radosgw: object copy does not copy source ACLs
- rados command: fix omap command usage
- sysvinit script: set ulimit -n properly on remote hosts
- msgr: fix narrow race with message queuing
- fixed compilation on some old distros (e.g., RHEL 5.x)

For more detailed information, see `the complete changelog`.

11.82 v0.56.1 "bobtail"

This release has two critical fixes. Please upgrade.

11.82.1 Upgrading

- There is a protocol compatibility problem between v0.56 and any other version that is now fixed. If your radosgw or RBD clients are running v0.56, they will need to be upgraded too. If they are running a version prior to v0.56, they can be left as is.

11.82.2 Notable changes

- osd: fix commit sequence for XFS, ext4 (or any other non-btrfs) to prevent data loss on power cycle or kernel panic
- osd: fix compatibility for CALL operation
- osd: process old osdmaps prior to joining cluster (fixes slow startup)
- osd: fix a couple of recovery-related crashes
- osd: fix large io requests when journal is in (non-default) aio mode
- log: fix possible deadlock in logging code

For more detailed information, see `the complete changelog`.

11.83 v0.56 "bobtail"

Bobtail is the second stable release of Ceph, named in honor of the *Bobtail Squid*: http://en.wikipedia.org/wiki/Bobtail_squid.

11.83.1 Key features since v0.48 "argonaut"

- Object Storage Daemon (OSD): improved threading, small-io performance, and performance during recovery
- Object Storage Daemon (OSD): regular "deep" scrubbing of all stored data to detect latent disk errors
- RADOS Block Device (RBD): support for copy-on-write clones of images.
- RADOS Block Device (RBD): better client-side caching.
- RADOS Block Device (RBD): advisory image locking
- Rados Gateway (RGW): support for efficient usage logging/scraping (for billing purposes)
- Rados Gateway (RGW): expanded S3 and Swift API coverage (e.g., POST, multi-object delete)
- Rados Gateway (RGW): improved striping for large objects
- Rados Gateway (RGW): OpenStack Keystone integration
- RPM packages for Fedora, RHEL/CentOS, OpenSUSE, and SLES
- mkcephfs: support for automatically formatting and mounting XFS and ext4 (in addition to btrfs)

11.83.2 Upgrading

Please refer to the document Upgrading from Argonaut to Bobtail for details.

- Cephx authentication is now enabled by default (since v0.55). Upgrading a cluster without adjusting the Ceph configuration will likely prevent the system from starting up on its own. We recommend first modifying the configuration to indicate that authentication is disabled, and only then upgrading to the latest version.:

```
auth client required = none
auth service required = none
auth cluster required = none
```

- Ceph daemons can be upgraded one-by-one while the cluster is online and in service.
- The `ceph-osd` daemons must be upgraded and restarted *before* any `radosgw` daemons are restarted, as they depend on some new ceph-osd functionality. (The `ceph-mon`, `ceph-osd`, and `ceph-mds` daemons can be upgraded and restarted in any order.)
- Once each individual daemon has been upgraded and restarted, it cannot be downgraded.
- The cluster of `ceph-mon` daemons will migrate to a new internal on-wire protocol once all daemons in the quorum have been upgraded. Upgrading only a majority of the nodes (e.g., two out of three) may expose the cluster to a situation where a single additional failure may compromise availability (because the non-upgraded daemon cannot participate in the new protocol). We recommend not waiting for an extended period of time between `ceph-mon` upgrades.
- The ops log and usage log for radosgw are now off by default. If you need these logs (e.g., for billing purposes), you must enable them explicitly. For logging of all operations to objects in the `.log` pool (see `radosgw-admin log ...`):

```
rgw enable ops log = true
```

For usage logging of aggregated bandwidth usage (see `radosgw-admin usage ...`):

```
rgw enable usage log = true
```

- You should not create or use "format 2" RBD images until after all `ceph-osd` daemons have been upgraded. Note that "format 1" is still the default. You can use the new `ceph osd ls` and `ceph tell osd.N version` commands to doublecheck your cluster. `ceph osd ls` will give a list of all OSD IDs that are part of the cluster, and you can use that to write a simple shell loop to display all the OSD version strings:

```
for i in $(ceph osd ls); do
    ceph tell osd.${i} version
done
```

11.83.3 Compatibility changes

- The 'ceph osd create [<uuid>]' command now rejects an argument that is not a UUID. (Previously it would take take an optional integer OSD id.) This correct syntax has been 'ceph osd create [<uuid>]' since v0.47, but the older calling convention was being silently ignored.
- The CRUSH map root nodes now have type `root` instead of type `pool`. This avoids confusion with RADOS pools, which are not directly related. Any scripts or tools that use the `ceph osd crush ...` commands may need to be adjusted accordingly.
- The `ceph osd pool create <poolname> <pgnum>` command now requires the `pgnum` argument. Previously this was optional, and would default to 8, which was almost never a good number.
- Degraded mode (when there fewer than the desired number of replicas) is now more configurable on a per-pool basis, with the min_size parameter. By default, with min_size 0, this allows I/O to objects with N - floor(N/2) replicas, where N is the total number of expected copies. Argonaut behavior was equivalent to having min_size = 1, so I/O would always be possible if any completely up to date copy remained. min_size = 1 could result in lower overall availability in certain cases, such as flapping network partitions.
- The sysvinit start/stop script now defaults to adjusting the max open files ulimit to 16384. On most systems the default is 1024, so this is an increase and won't break anything. If some system has a higher initial value, however, this change will lower the limit. The value can be adjusted explicitly by adding an entry to the `ceph.conf` file in the appropriate section. For example:

```
[global]
        max open files = 32768
```

- 'rbd lock list' and 'rbd showmapped' no longer use tabs as separators in their output.
- There is configurable limit on the number of PGs when creating a new pool, to prevent a user from accidentally specifying a ridiculous number for pg_num. It can be adjusted via the 'mon max pool pg num' option on the monitor, and defaults to 65536 (the current max supported by the Linux kernel client).
- The osd capabilities associated with a rados user have changed syntax since 0.48 argonaut. The new format is mostly backwards compatible, but there are two backwards-incompatible changes:
  - specifying a list of pools in one grant, i.e. 'allow r pool=foo,bar' is now done in separate grants, i.e. 'allow r pool=foo, allow r pool=bar'.
  - restricting pool access by pool owner ('allow r uid=foo') is removed. This feature was not very useful and unused in practice.

  The new format is documented in the ceph-authtool man page.

- 'rbd cp' and 'rbd rename' use rbd as the default destination pool, regardless of what pool the source image is in. Previously they would default to the same pool as the source image.
- 'rbd export' no longer prints a message for each object written. It just reports percent complete like other long-lasting operations.
- 'ceph osd tree' now uses 4 decimal places for weight so output is nicer for humans
- Several monitor operations are now idempotent:
  - ceph osd pool create
  - ceph osd pool delete
  - ceph osd pool mksnap
  - ceph osd rm
  - ceph pg <pgid> revert

11.83.4 Notable changes

- auth: enable cephx by default
- auth: expanded authentication settings for greater flexibility
- auth: sign messages when using cephx
- build fixes for Fedora 18, CentOS/RHEL 6
- ceph: new 'osd ls' and 'osd tell <osd.N> version' commands
- ceph-debugpack: misc improvements
- ceph-disk-prepare: creates and labels GPT partitions
- ceph-disk-prepare: support for external journals, default mount/mkfs options, etc.
- ceph-fuse/libcephfs: many misc fixes, admin socket debugging
- ceph-fuse: fix handling for .. in root directory
- ceph-fuse: many fixes (including memory leaks, hangs)
- ceph-fuse: mount helper (mount.fuse.ceph) for use with /etc/fstab
- ceph.spec: misc packaging fixes
- common: thread pool sizes can now be adjusted at runtime
- config: $pid is now available as a metavariable
- crush: default root of tree type is now 'root' instead of 'pool' (to avoid confusiong wrt rados pools)
- crush: fixed retry behavior with chooseleaf via tunable
- crush: tunables documented; feature bit now present and enforced
- libcephfs: java wrapper
- librados: several bug fixes (rare races, locking errors)
- librados: some locking fixes
- librados: watch/notify fixes, misc memory leaks
- librbd: a few fixes to 'discard' support
- librbd: fine-grained striping feature

- librbd: fixed memory leaks
- librbd: fully functional and documented image cloning
- librbd: image (advisory) locking
- librbd: improved caching (of object non-existence)
- librbd: 'flatten' command to sever clone parent relationship
- librbd: 'protect'/'unprotect' commands to prevent clone parent from being deleted
- librbd: clip requests past end-of-image.
- librbd: fixes an issue with some windows guests running in qemu (remove floating point usage)
- log: fix in-memory buffering behavior (to only write log messages on crash)
- mds: fix ino release on abort session close, relative getattr path, mds shutdown, other misc items
- mds: misc fixes
- mkcephfs: fix for default keyring, osd data/journal locations
- mkcephfs: support for formatting xfs, ext4 (as well as btrfs)
- init: support for automatically mounting xfs and ext4 osd data directories
- mon, radosgw, ceph-fuse: fixed memory leaks
- mon: improved ENOSPC, fs error checking
- mon: less-destructive ceph-mon –mkfs behavior
- mon: misc fixes
- mon: more informative info about stuck PGs in 'health detail'
- mon: information about recovery and backfill in 'pg <pgid> query'
- mon: new 'osd crush create-or-move ...' command
- mon: new 'osd crush move ...' command lets you rearrange your CRUSH hierarchy
- mon: optionally dump 'osd tree' in json
- mon: configurable cap on maximum osd number (mon max osd)
- mon: many bug fixes (various races causing ceph-mon crashes)
- mon: new on-disk metadata to facilitate future mon changes (post-bobtail)
- mon: election bug fixes
- mon: throttle client messages (limit memory consumption)
- mon: throttle osd flapping based on osd history (limits osdmap thrashing' on overloaded or unhappy clusters)
- mon: 'report' command for dumping detailed cluster status (e.g., for use when reporting bugs)
- mon: osdmap flags like noup, noin now cause a health warning
- msgr: improved failure handling code
- msgr: many bug fixes
- osd, mon: honor new 'nobackfill' and 'norecover' osdmap flags
- osd, mon: use feature bits to lock out clients lacking CRUSH tunables when they are in use
- osd: backfill reservation framework (to avoid flooding new osds with backfill data)

- osd: backfill target reservations (improve performance during recovery)
- osd: better tracking of recent slow operations
- osd: capability grammar improvements, bug fixes
- osd: client vs recovery io prioritization
- osd: crush performance improvements
- osd: default journal size to 5 GB
- osd: experimental support for PG “splitting” (pg_num adjustment for existing pools)
- osd: fix memory leak on certain error paths
- osd: fixed detection of EIO errors from fs on read
- osd: major refactor of PG peering and threading
- osd: many bug fixes
- osd: more/better dump info about in-progress operations
- osd: new caps structure (see compatibility notes)
- osd: new ‘deep scrub’ will compare object content across replicas (once per week by default)
- osd: new ‘lock’ rados class for generic object locking
- osd: optional ‘min’ pg size
- osd: recovery reservations
- osd: scrub efficiency improvement
- osd: several out of order reply bug fixes
- osd: several rare peering cases fixed
- osd: some performance improvements related to request queuing
- osd: use entire device if journal is a block device
- osd: use syncfs(2) when kernel supports it, even if glibc does not
- osd: various fixes for out-of-order op replies
- rados: ability to copy, rename pools
- rados: bench command now cleans up after itself
- rados: ‘cppool’ command to copy rados pools
- rados: ‘rm’ now accepts a list of objects to be removed
- radosgw: POST support
- radosgw: REST API for managing usage stats
- radosgw: fix bug in bucket stat updates
- radosgw: fix copy-object vs attributes
- radosgw: fix range header for large objects, ETag quoting, GMT dates, other compatibility fixes
- radosgw: improved garbage collection framework
- radosgw: many small fixes, cleanups
- radosgw: openstack keystone integration

- radosgw: stripe large (non-multipart) objects
- radosgw: support for multi-object deletes
- radosgw: support for swift manifest objects
- radosgw: vanity bucket dns names
- radosgw: various API compatibility fixes
- rbd: import from stdin, export to stdout
- rbd: new 'ls -l' option to view images with metadata
- rbd: use generic id and keyring options for 'rbd map'
- rbd: don't issue usage on errors
- udev: fix symlink creation for rbd images containing partitions
- upstart: job files for all daemon types (not enabled by default)
- wireshark: ceph protocol dissector patch updated

11.84 v0.54

11.84.1 Upgrading

- The osd capabilities associated with a rados user have changed syntax since 0.48 argonaut. The new format is mostly backwards compatible, but there are two backwards-incompatible changes:
  - specifying a list of pools in one grant, i.e. 'allow r pool=foo,bar' is now done in separate grants, i.e. 'allow r pool=foo, allow r pool=bar'.
  - restricting pool access by pool owner ('allow r uid=foo') is removed. This feature was not very useful and unused in practice.

  The new format is documented in the ceph-authtool man page.
- Bug fixes to the new osd capability format parsing properly validate the allowed operations. If an existing rados user gets permissions errors after upgrading, its capabilities were probably misconfigured. See the ceph-authtool man page for details on osd capabilities.
- 'rbd lock list' and 'rbd showmapped' no longer use tabs as separators in their output.

11.85 v0.48.3 "argonaut"

This release contains a critical fix that can prevent data loss or corruption after a power loss or kernel panic event. Please upgrade immediately.

11.85.1 Upgrading

- If you are using the undocumented `ceph-disk-prepare` and `ceph-disk-activate` tools, they have several new features and some additional functionality. Please review the changes in behavior carefully before upgrading.
- The .deb packages now require xfsprogs.

11.85.2 Notable changes

- filestore: fix op_seq write order (fixes journal replay after power loss)
- osd: fix occasional indefinitely hung "slow" request
- osd: fix encoding for pool_snap_info_t when talking to pre-v0.48 clients
- osd: fix heartbeat check
- osd: reduce log noise about rbd watch
- log: fixes for deadlocks in the internal logging code
- log: make log buffer size adjustable
- init script: fix for 'ceph status' across machines
- radosgw: fix swift error handling
- radosgw: fix swift authentication concurrency bug
- radosgw: don't cache large objects
- radosgw: fix some memory leaks
- radosgw: fix timezone conversion on read
- radosgw: relax date format restrictions
- radosgw: fix multipart overwrite
- radosgw: stop processing requests on client disconnect
- radosgw: avoid adding port to url that already has a port
- radosgw: fix copy to not override ETAG
- common: make parsing of ip address lists more forgiving
- common: fix admin socket compatibility with old protocol (for collectd plugin)
- mon: drop dup commands on paxos reset
- mds: fix loner selection for multiclient workloads
- mds: fix compat bit checks
- ceph-fuse: fix segfault on startup when keyring is missing
- ceph-authtool: fix usage
- ceph-disk-activate: misc backports
- ceph-disk-prepare: misc backports
- debian: depend on xfsprogs (we use xfs by default)
- rpm: build rpms, some related Makefile changes

For more detailed information, see `the complete changelog`.

11.86 v0.48.2 "argonaut"

11.86.1 Upgrading

- The default search path for keyring files now includes /etc/ceph/ceph.$name.keyring. If such files are present on your cluster, be aware that by default they may now be used.
- There are several changes to the upstart init files. These have not been previously documented or recommended. Any existing users should review the changes before upgrading.
- The ceph-disk-prepare and ceph-disk-active scripts have been updated significantly. These have not been previously documented or recommended. Any existing users should review the changes before upgrading.

11.86.2 Notable changes

- mkcephfs: fix keyring generation for mds, osd when default paths are used
- radosgw: fix bug causing occasional corruption of per-bucket stats
- radosgw: workaround to avoid previously corrupted stats from going negative
- radosgw: fix bug in usage stats reporting on busy buckets
- radosgw: fix Content-Range: header for objects bigger than 2 GB.
- rbd: avoid leaving watch acting when command line tool errors out (avoids 30s delay on subsequent operations)
- rbd: friendlier use of –pool/–image options for import (old calling convention still works)
- librbd: fix rare snapshot creation race (could "lose" a snap when creation is concurrent)
- librbd: fix discard handling when spanning holes
- librbd: fix memory leak on discard when caching is enabled
- objecter: misc fixes for op reordering
- objecter: fix for rare startup-time deadlock waiting for osdmap
- ceph: fix usage
- mon: reduce log noise about "check_sub"
- ceph-disk-activate: misc fixes, improvements
- ceph-disk-prepare: partition and format osd disks automatically
- upstart: start everyone on a reboot
- upstart: always update the osd crush location on start if specified in the config
- config: add /etc/ceph/ceph.$name.keyring to default keyring search path
- ceph.spec: don't package crush headers

For more detailed information, see `the complete changelog`.

11.87 v0.48.1 "argonaut"

11.87.1 Upgrading

- The radosgw usage trim function was effectively broken in v0.48. Earlier it would remove more usage data than what was requested. This is fixed in v0.48.1, but the fix is incompatible. The v0.48 radosgw-admin tool cannot be used to initiate the trimming; please use the v0.48.1 version.
- v0.48.1 now explicitly indicates support for the CRUSH_TUNABLES feature. No other version of Ceph requires this, yet, but future versions will when the tunables are adjusted from their historical defaults.
- There are no other compatibility changes between v0.48.1 and v0.48.

11.87.2 Notable changes

- mkcephfs: use default 'keyring', 'osd data', 'osd journal' paths when not specified in conf
- msgr: various fixes to socket error handling
- osd: reduce scrub overhead
- osd: misc peering fixes (past_interval sharing, pgs stuck in 'peering' states)
- osd: fail on EIO in read path (do not silently ignore read errors from failing disks)
- osd: avoid internal heartbeat errors by breaking some large transactions into pieces
- osd: fix osdmap catch-up during startup (catch up and then add daemon to osdmap)
- osd: fix spurious 'misdirected op' messages
- osd: report scrub status via 'pg ... query'
- rbd: fix race when watch registrations are resent
- rbd: fix rbd image id assignment scheme (new image data objects have slightly different names)
- rbd: fix perf stats for cache hit rate
- rbd tool: fix off-by-one in key name (crash when empty key specified)
- rbd: more robust udev rules
- rados tool: copy object, pool commands
- radosgw: fix in usage stats trimming
- radosgw: misc API compatibility fixes (date strings, ETag quoting, swift headers, etc.)
- ceph-fuse: fix locking in read/write paths
- mon: fix rare race corrupting on-disk data
- config: fix admin socket 'config set' command
- log: fix in-memory log event gathering
- debian: remove crush headers, include librados-config
- rpm: add ceph-disk-{activate, prepare}

For more detailed information, see `the complete changelog`.

11.88 v0.48 “argonaut”

11.88.1 Upgrading

- This release includes a disk format upgrade. Each ceph-osd daemon, upon startup, will migrate its locally stored data to the new format. This process can take a while (for large object counts, even hours), especially on non-btrfs file systems.
- To keep the cluster available while the upgrade is in progress, we recommend you upgrade a storage node or rack at a time, and wait for the cluster to recover each time. To prevent the cluster from moving data around in response to the OSD daemons being down for minutes or hours, you may want to:

```
ceph osd set noout
```

  This will prevent the cluster from marking down OSDs as “out” and re-replicating the data elsewhere. If you do this, be sure to clear the flag when the upgrade is complete:

```
ceph osd unset noout
```

- There is a encoding format change internal to the monitor cluster. The monitor daemons are careful to switch to the new format only when all members of the quorum support it. However, that means that a partial quorum with new code may move to the new format, and a recovering monitor running old code will be unable to join (it will crash). If this occurs, simply upgrading the remaining monitor will resolve the problem.
- The ceph tool’s -s and -w commands from previous versions are incompatible with this version. Upgrade your client tools at the same time you upgrade the monitors if you rely on those commands.
- It is not possible to downgrade from v0.48 to a previous version.

11.88.2 Notable changes

- osd: stability improvements
- osd: capability model simplification
- osd: simpler/safer –mkfs (no longer removes all files; safe to re-run on active osd)
- osd: potentially buggy FIEMAP behavior disabled by default
- rbd: caching improvements
- rbd: improved instrumentation
- rbd: bug fixes
- radosgw: new, scalable usage logging infrastructure
- radosgw: per-user bucket limits
- mon: streamlined process for setting up authentication keys
- mon: stability improvements
- mon: log message throttling
- doc: improved documentation (ceph, rbd, radosgw, chef, etc.)
- config: new default locations for daemon keyrings
- config: arbitrary variable substitutions
- improved ‘admin socket’ daemon admin interface (ceph –admin-daemon ...)

- chef: support for multiple monitor clusters
- upstart: basic support for monitors, mds, radosgw; osd support still a work in progress.

The new default keyring locations mean that when enabling authentication (`auth supported = cephx`), keyring locations do not need to be specified if the keyring file is located inside the daemon's data directory (`/var/lib/ceph/$type/ceph-$id` by default).

There is also a lot of librbd code in this release that is laying the groundwork for the upcoming layering functionality, but is not actually used. Likewise, the upstart support is still incomplete and not recommended; we will backport that functionality later if it turns out to be non-disruptive.

RELEASE TIMELINE

There are approximately three stable releases a year. Every other release is a LTS (Long Term Support). A LTS release is supported for 18 months. A stable release that is not LTS is supported until the next stable release is published. A development / testing release is not supported.

- Long Term Support release : 18 months
- Stable release : until the next stable release is published
- Development / testing release : not supported

Supporting a release means:

- Integration and upgrade tests are run on a regular basis and their results analyzed by Ceph developers.
- Issues fixed in the development branch is scheduled to be backported to the release.
- When an issue found in the release is reported it will be handled by Ceph developers.
- The stable releases and backport team publishes `point releases` including fixes that have been backported to the release.

In the following, the life time of a LTS is calculated to be 18 months after the month of the first release. For instance, Dumpling is published August 2013 and 18 months starting September 2013 is February 2015, therefore by March 2015 Dumpling should be End of Life. The lifetime of a release may be extended a few months. For instance although Dumpling theoritical End of Life was March 2015, it was extended to May 2015.

There is no estimated End of Life for a stable release that is not a LTS because it is set by the release date of the next stable release instead of a fixed duration.

	Dumpling LTS	Emperor Stable	Firefly LTS	Giant Stable	Hammer LTS
First release	August 2013	November 2013	May 2014	October 2014	April 2015
Estimated End of Life	March 2015		January 2016		November 2016
Actual End of Life	May 2015	May 2014		April 2015	

	Development Testing	Dumpling LTS	Emperor Stable	Firefly LTS	Giant Stable	Hammer LTS
April 2015					0.87.2	0.94.1
						0.94
March 2015				0.80.9		
February 2015	0.93				0.87.1	
	0.92					
January 2015	0.91			0.80.8		
December 2014	0.90					
	0.89					
November 2014	0.88					

Continued on next page

Table 12.1 – continued from previous page

October 2014	0.86			0.80.7	0.87	
				0.80.6		
September 2014	0.85	0.67.11				
August 2014	0.84	0.67.10				
July 2014	0.83			0.80.5		
				0.80.4		
				0.80.3		
				0.80.2		
June 2014	0.82					
	0.81					
May 2014		0.67.9		0.80.1		
		0.67.8		0.80		
April 2014	0.79					
March 2014	0.78					
February 2014	0.77	0.67.7				
		0.67.6				
January 2014	0.76					
	0.75					
December 2013	0.74	0.67.5	0.72.2			
	0.73					
November 2013			0.72.1			
			0.72			
October 2013	0.71	0.67.4				
	0.70					
September 2013	0.69					
	0.68	0.67.3				
August 2013		0.67.2				
		0.67.1				
		0.67				

CHAPTER

THIRTEEN

CEPH GLOSSARY

Ceph is growing rapidly. As firms deploy Ceph, the technical terms such as “RADOS”, “RBD,” “RGW” and so forth require corresponding marketing terms that explain what each component does. The terms in this glossary are intended to complement the existing technical terminology.

Sometimes more than one term applies to a definition. Generally, the first term reflects a term consistent with Ceph’s marketing, and secondary terms reflect either technical terms or legacy ways of referring to Ceph systems.

Ceph Project The aggregate term for the people, software, mission and infrastructure of Ceph.

cephx The Ceph authentication protocol. Cephx operates like Kerberos, but it has no single point of failure.

Ceph, Ceph Platform All Ceph software, which includes any piece of code hosted at http://github.com/ceph.

Ceph System, Ceph Stack A collection of two or more components of Ceph.

Ceph Node, Node, Host Any single machine or server in a Ceph System.

Ceph Storage Cluster, Ceph Object Store, RADOS, RADOS Cluster, Reliable Autonomic Distributed Object Store The core set of storage software which stores the user’s data (MON+OSD).

Ceph Cluster Map, cluster map The set of maps comprising the monitor map, OSD map, PG map, MDS map and CRUSH map. See Cluster Map for details.

Ceph Object Storage The object storage “product”, service or capabilities, which consists essentially of a Ceph Storage Cluster and a Ceph Object Gateway.

Ceph Object Gateway, RADOS Gateway, RGW The S3/Swift gateway component of Ceph.

Ceph Block Device, RBD The block storage component of Ceph.

Ceph Block Storage The block storage “product,” service or capabilities when used in conjunction with `librbd`, a hypervisor such as QEMU or Xen, and a hypervisor abstraction layer such as `libvirt`.

Ceph Filesystem, CephFS, Ceph FS The POSIX filesystem components of Ceph.

Cloud Platforms, Cloud Stacks Third party cloud provisioning platforms such as OpenStack, CloudStack, OpenNebula, ProxMox, etc.

Object Storage Device, OSD A physical or logical storage unit (*e.g.*, LUN). Sometimes, Ceph users use the term “OSD” to refer to *Ceph OSD Daemon*, though the proper term is “Ceph OSD”.

Ceph OSD Daemon, Ceph OSD The Ceph OSD software, which interacts with a logical disk (*OSD*). Sometimes, Ceph users use the term “OSD” to refer to “Ceph OSD Daemon”, though the proper term is “Ceph OSD”.

Ceph Monitor, MON The Ceph monitor software.

Ceph Metadata Server, MDS The Ceph metadata software.

Ceph Clients, Ceph Client The collection of Ceph components which can access a Ceph Storage Cluster. These include the Ceph Object Gateway, the Ceph Block Device, the Ceph Filesystem, and their corresponding libraries, kernel modules, and FUSEs.

Ceph Kernel Modules The collection of kernel modules which can be used to interact with the Ceph System (e.g,. `ceph.ko`, `rbd.ko`).

Ceph Client Libraries The collection of libraries that can be used to interact with components of the Ceph System.

Ceph Release Any distinct numbered version of Ceph.

Ceph Point Release Any ad-hoc release that includes only bug or security fixes.

Ceph Interim Release Versions of Ceph that have not yet been put through quality assurance testing, but may contain new features.

Ceph Release Candidate A major version of Ceph that has undergone initial quality assurance testing and is ready for beta testers.

Ceph Stable Release A major version of Ceph where all features from the preceding interim releases have been put through quality assurance testing successfully.

Ceph Test Framework, Teuthology The collection of software that performs scripted tests on Ceph.

CRUSH Controlled Replication Under Scalable Hashing. It is the algorithm Ceph uses to compute object storage locations.

ruleset A set of CRUSH data placement rules that applies to a particular pool(s).

Pool, Pools Pools are logical partitions for storing objects.

PYTHON MODULE INDEX