US HEADQUARTERS

Sunnyvale, CA
525 Almanor Ave, 4th Floor
Sunnyvale, CA 94085
+1-650-963-9828 Phone
+1-650-968-2997 Fax

Mirantis OpenStack®

The most open, flexible, commercially supported OpenStack distribution

Ceph Best Practices Manual

Version 1.0
21-Feb-2016

Authors:
Christian Huebner (Storage Architect)
Pawel Stefanski (Senior Deployment Engineer)
Kostiantyn Danilov (Principal Software Engineer)
Igor Fedotov (Senior Software Engineer)

Table of contents

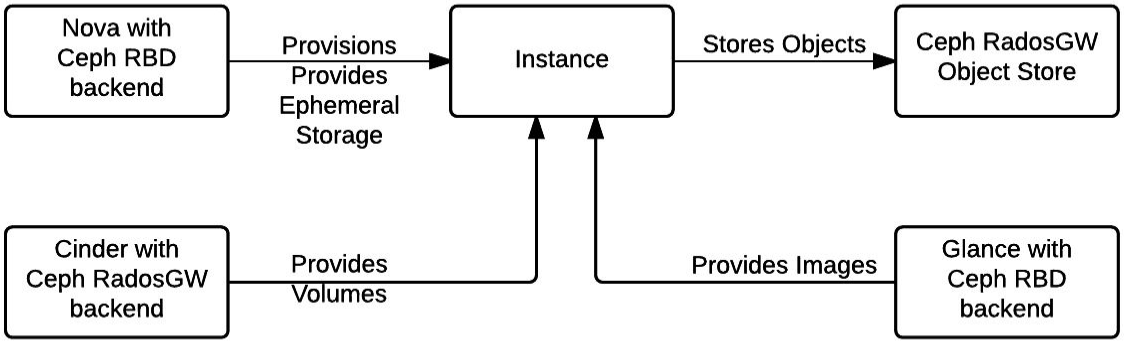
1 Introduction

The purpose of this manual is to provide the best practices for Ceph configuration, deployment, operation, and troubleshooting. It is aimed to help deployment and operations engineers, as well as storage administrators, to recognize and fix a majority of common Ceph operational issues.

2 Deployment considerations

2.1 Ceph and OpenStack integration

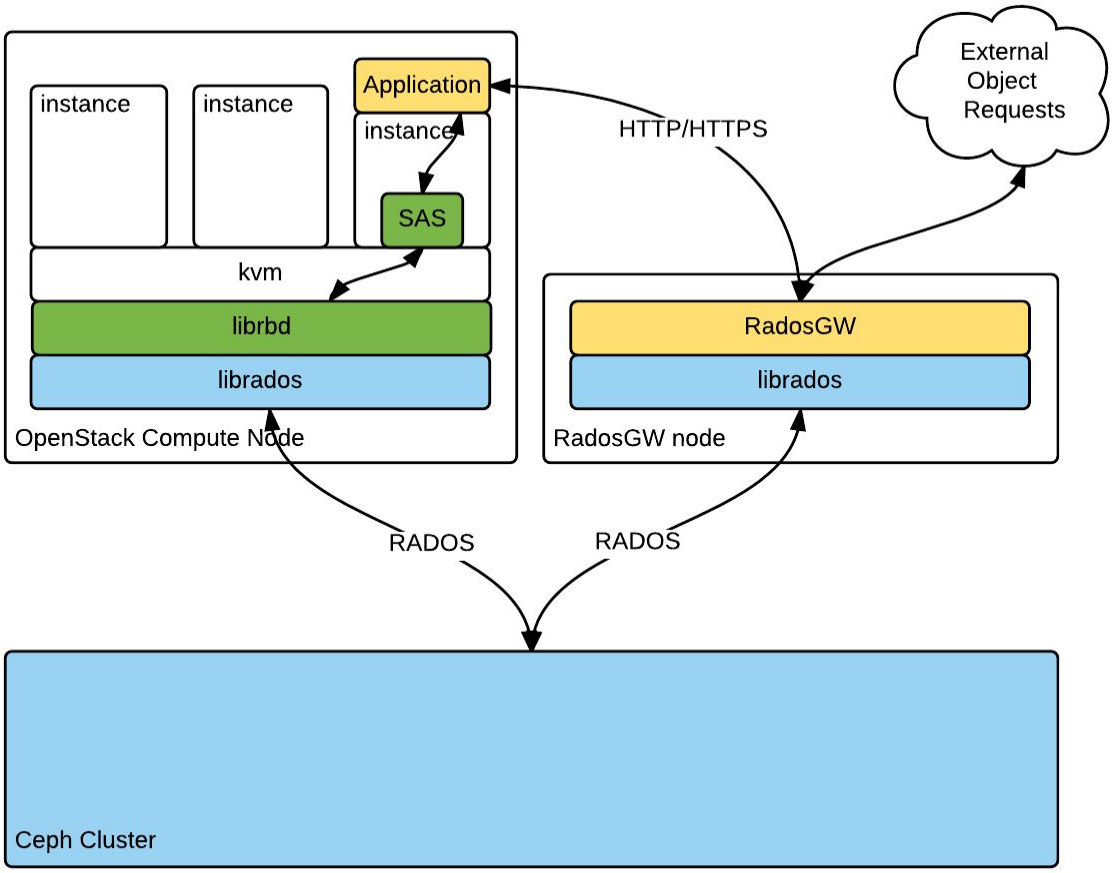
When you deploy Ceph software-defined storage with Fuel and MOS, Cinder uses it to provide volumes and Glance to provide image service. Ceph RadosGW object storage can be used by any other service like an object store.



Ceph integration in OpenStack

The diagram above shows all data flows that Ceph is involved with. It simply does a back end for the Cinder, replacing any legacy storage array; replaces Swift for the Object Service as a back end for Glance, and provides ephemeral storage for Nova directly.

Ceph is integrated into OpenStack Nova and Cinder by Rados Block Device (RBD). This overlay interface to Rados is using block addressing and it is supported by QEMU and Libvirt as a native storage back end.



Ceph communication inside openStack

The main advantage of Cinder with Ceph over Cinder with LVM is that Ceph is distributed and network-available. Ceph also provides redundancy by data replication and allows the use of commodity hardware. A properly defined CRUSH map is rack and host aware with full cluster and HA based on quorum rule.

Another feature that can be used is Copy-on-write. It allows using an existing volume as a source for unmodified data of another volume. Copy-on-write significantly accelerates provisioning and consumes less space for new VMs based on templates or snapshots.

With network-available and distributed storage, the Live Migration feature is available even for ephemeral disks. This can be used to evacuate failing hosts or to implement non-disruptive upgrades of infrastructure.

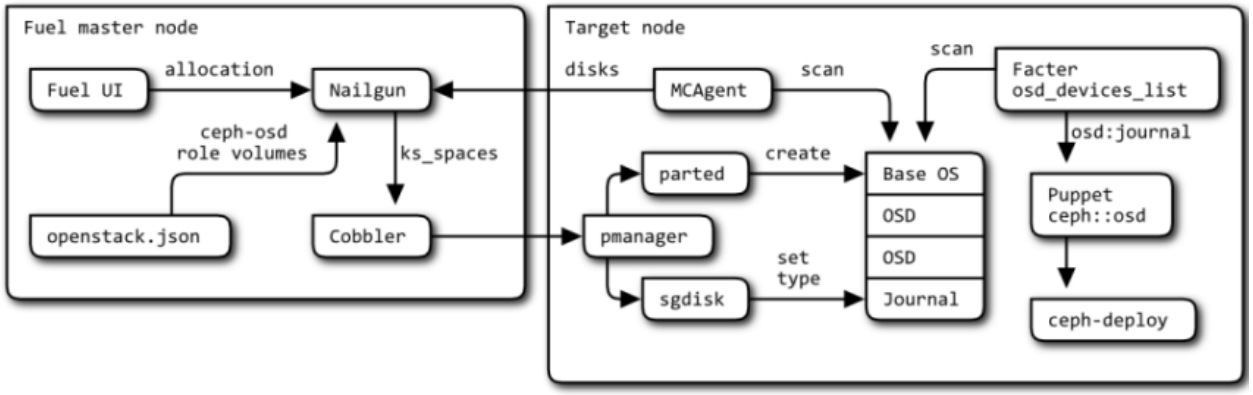
The level of integration into QEMU also gives a possibility to use Cinder QoS feature to limit uncontrollable VMs and prevent them from consuming all IOPS and storage throughput.

2.2 Fuel Ceph deployment

Fuel uses the native Ceph tool *ceph-deploy* to help with successful and clean node deployments.
Fuel relies on adding a role to the host. For Ceph, Fuel provides the Ceph/OSD role. Network and interface configurations should be changed to meet the requirements of an environment.

Note: By default, Ceph monitors reside on OpenStack controllers, and we do not have a specific role for the monitors.

When a deployment change action is triggered by the Fuel UI or CLI, the node is deployed and the `ceph::osd` manifests are applied; the disk is prepared and tagged with its UUIDs by Cobbler just before the ceph-deploy is used to populate configuration and finish making a new OSD daemon instance. The last part done by ceph-deploy is to place the new OSD into CRUSH map and make it available to extended cluster.



2.3 Ceph default configuration

Fuel deploys a cluster with some best-practice and entry configuration, but it should be tuned and corrected according to specific expectations, workload and hardware configuration. The parameters can change depending on the cluster size. Fuel deploys standard values considerable up to mid-range installations.

The Ceph configuration file can be found in:

```
/etc/ceph/ceph.conf
```

This file is managed by the `ceph-deploy` tool. When a new node is deployed, `ceph-deploy` pulls this file from the controller. Any manual changes should also be populated

on all nodes to maintain consistency of configuration files.

The configuration file is divided into the following sections: `global`, `client`, `osd`, `mon`, `client.radosgw`, `mds` for each Ceph daemon.

The main section is `global`. It contains general configuration options and default values:

```
[global]
fsid = a7825190-4cb0-4168-9d5e-7353e56c8b01 # clusterid
mon_initial_members = node-17 # initial mons to connect
mon_host = 192.168.0.4 # mon host list
auth_cluster_required = cephx #when cephx is used
auth_service_required = cephx #when cephx is used
auth_client_required = cephx #when cephx is used
filestore_xattr_use_omap = true #for ext4 and other fs
log_to_syslog_level = info
log_to_syslog = True
osd_pool_default_size = 3 #default replica number for new pool
osd_pool_default_min_size = 1 #default mandatory replica count for
osd_pool_default_pg_num = 256 #default pg number for new pools

public_network = 192.168.0.4/24 #network for client communication
log_to_syslog_facility = LOG_LOCAL0
osd_journal_size = 2048 #default journal size (MB)
auth_supported = cephx #when cephx is used
osd_pool_default_pgp_num = 256 #default pg number for new pools
osd_mkfs_type = xfs #default fs for ceph-deploy
cluster_network = 192.168.1.2/24 #inter cluster data network
osd_recovery_max_active = 1 #recovery throttling
osd_max_backfills = 1 #recovery and resize throttling
```

Example of `client` section:

```
[client]
rbd_cache_writethrough_until_flush = True
rbd_cache = True
```

RBD Cache is used to accelerate IO operations on instances, the default is to use write-back cache mode, though it can be disabled by setting the additional option "`rbd cache max dirty`" to 0. This option should comply Nova and libvirt settings on OpenStack side.
The `rbd_cache_write through_until_flush` option is used to start operations as a write-through and then switch to write-back mode to comply older clients.

Example of `RadosGW` section:

```
[client.radosgw.gateway]
rgw_keystone_accepted_roles = _member_, Member, admin, swiftoperator
keyring = /etc/ceph/keyring.radosgw.gateway
rgw_socket_path = /tmp/radosgw.sock
rgw_keystone_revocation_interval = 1000000
rgw_keystone_url = 192.168.0.2:35357
rgw_keystone_admin_token = _keystone_admin_token
host = node-17
rgw_dns_name = *.domain.tld
rgw_print_continue = True
rgw_keystone_token_cache_size = 10
rgw_data = /var/lib/ceph/radosgw
user = www-data
```

This whole section is describing the RadosGW configuration. All rgw_keystone_ prefixed parameters are set to support Keystone user authentication. Keystone admin user is used, and the Keystone vip is pointed.

The option `rgw_print_continue = True` should only be used when HTTP gateway understands this HTTP response code and supports it. Fuel deploys the Inktank version of Apache2 and FastCGI module that supports it.

The option `rgw_dns_name = *.domain.tld` should be modified to proper domain value when using container names as a domain suffixes. It should resolve as a CNAME or A record to the RGW host.

For example in BIND zone style:

```
*     IN  CNAME  rgw.domain.tld.
rgw   IN  A      192.168.0.2
```

The configuration file can include many other options. For available options, refer to original Ceph documentation.

2.4 Adding nodes to the cluster

The new nodes are discovered and appear in the Fuel interface. When the node is added to the "Ceph OSD" role, disk allocation can be reviewed in UI or CLI. There are two types of disks for Ceph:

- OSD Data, that holds the data
- OSD Journal, that only stores all written data into a journal

The partitions are marked with different UUIDs, so they can be recognized later.

```
JOURNAL_UUID = '45b0969e-9b03-4f30-b4c6-b4b80ceff106'
OSD_UUID     = '4fbd7e29-9d25-41b8-afd0-062c0ceff05d'
```

If there is one Journal device, it will be evenly allocated to OSDs on the host.

Next, the Puppet manifests are started to use `ceph-deploy` as the main tool to create the new OSD. The Puppet script automatically adjusts the CRUSH map with the new OSD(s). After the disks are cataloged by UUID and prepared (`ceph mkfs`), the new OSD is activated and a daemon is launched. The cluster map, which contains the CRUSH map, is automatically disseminated to the new cluster nodes from the monitors when the nodes attach to the cluster.

2.5 Removing nodes from the cluster

When a node is removed from a deployed cluster, Ceph configuration stays untouched. So Ceph treats the node as if it had gone down. To completely remove the node from Ceph, manual intervention is needed. The procedure is covered in the *Remove an OSD* subsection in the *Procedures* section.

2.6 Time-consuming operations on cluster

There are several cluster-wide operations that are IO consuming, and the administrator should be aware of the impact these operations can cause before starting those tasks. Most of these operations have no option to cancel them or tune options to lessen the impact on cluster performance. Therefore, great care must be taken to only execute the operations when safe.
Cluster remapping is the most affecting performance operation. It occurs when making any changes in cluster size or placement. When the cluster is changed, the CRUSH algorithm recalculates placement group positions, which causes data migrations inside the cluster.
Prior to the Hammer version of Ceph, there is a very limited possibility to throttle those operations, and any cluster changes cause harmful operations and performance impact.

In the Fuel-deployed default configuration, there are two options that help to address this issue:

```
osd_recovery_max_active = 1
osd_max_backfills = 1
```

Both options prevent the OSD from executing more than one recovery/backfill PG operation at a time. By reducing the parallelism of operations, the overall internal load of the cluster is reduced to a reasonable level. These options adversely affect the speed of recovery and backfill operations because the operations are severely throttled.
The Ceph documentation also recommends tuning IO thread priorities. Ceph Hammer is the first release to provide these options.

Another type of time-consuming operation is the peering phase after OSD process is (re)started or brought up. The OSD process scans the whole disk just after is starts. When the OSD has to scan a lot of files and directories, it takes a long time to gather the full tree (especially on slow 7.2k HDD drives).

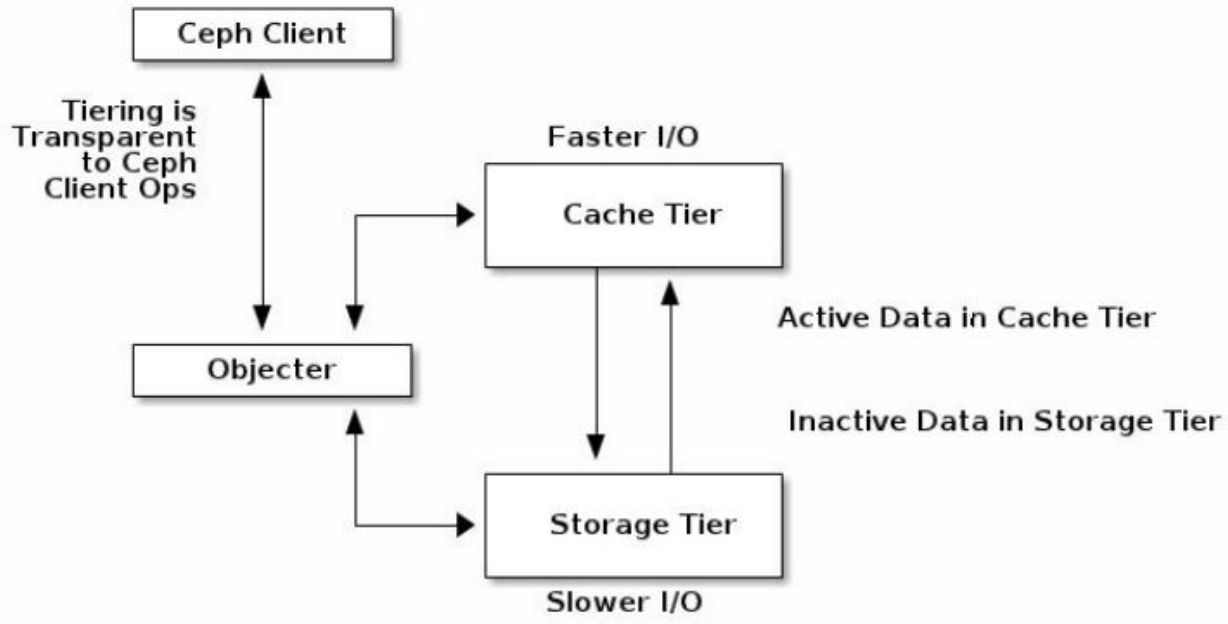
2.7 Cache configuration

Starting with Firefly, Ceph provides a feature that allows fronting a large amount of OSDs with spinning drives with a cache layer, most commonly executed in SSD. The SSD cache can be deployed on nodes that provide regular OSDs or on specific cache nodes.

2.7.1 Design

Architecture

In a Ceph cache tier design, the underlying Ceph infrastructure remains unchanged. An “Objecter” instance is created to manage the tiering infrastructure and communicate with both the cache and the OSD back end.



Ceph cache tier architecture overview. Source: Ceph manual

Ceph provides two modes of caching: writeback and read-only. In the read-only mode, Ceph maintains a set of the most requested rados objects in the cache for quick read. It can be paired with SSD journaling for a moderate write performance increase. In the writeback mode, Ceph writes to the cache tier first and provides a mechanism for the data in the cache tier to be written to disk in an orderly fashion.

In writeback mode, the data in the cache tier must be replicated or erasure-coded to ensure data is not lost if a cache tier component fails before the cluster can write the data to disk.

2.7.2 Implementation

Deployment

As a cache tier is not deployed out of the box by Mirantis OpenStack, Mirantis recommends deploying the cache nodes as regular Ceph nodes. Upon completion of the deployment, specific rules are created in the crush map to place the cache pools on the SSD backed OSDs and the regular pools on HDD backed OSDs.

Once the cache pools are established, they are added to the regular storage pools with the **ceph osd tier** command set.

No changes are necessary on the client side, as long as the current CRUSH map is available to all clients, which it must be for the Ceph cluster to function.

Cached Pools

As cache tiers are a perpool property, a separate cache pool must be created on the SSD infrastructure for each pool that requires caching.
The pools that benefit most from caching are pools that either have high performance SLAs and experience heavy read and especially write traffic.
The **backup** pool should only be cached if performance requirements can not be met with hard disk based storage.

Copy-on-Write

When Copy-on-Write is utilized, the direct image URL must be exposed. Glance cache management should be disabled to avoid double caching. An extensive explanation and a step-by-step guide is available in the Block Devices and OpenStack section of the Ceph documentation. As caching is necessary for both the copy-on-write source and destination, both the images and compute volumes must be cached.

See also:

- Ceph Cache Tiering documentation:
  http://docs.ceph.com/docs/master/rados/operations/cache-tiering/
- Placing different pools on different OSDs
  http://docs.ceph.com/docs/master/rados/operations/crush-map/#placing-different-pools-on-different-osds
- Block devices and OpenStack
  http://docs.ceph.com/docs/master/rbd/rbd-openstack/

2.8 Cache tiering How-To

2.8.1 Create buckets

Before starting, verify that all OSD devices which are supposed to be caching OSDs are marked "out".

1. List of OSDs may be retrieved with the command:

```
#ceph osd tree
```

2. Finding out which OSDs need to be moved to caching bucket, mark them as "out":

```
#ceph osd out <osd_number>
```

Watching at **ceph -w** output, find out when the process of replication of the placement groups is finished.

3. Create 2 buckets for regular OSDs and cache OSDs and move them to root:

```
ceph osd crush add-bucket regular datacenter
ceph osd crush add-bucket cache datacenter
ceph osd crush move cache root=default
ceph osd crush move regular root=default
```

4. Move those hosts with regular OSDs to the bucket "regular", and all hosts with fast OSDs to the bucket "cache" (execute for every host):

```
# ceph osd crush move <hostname>  datacenter=<regular/cache>
```

5. Verify that the structure of the OSDs is correct with **ceph osd tree** command.

2.8.2 CRUSH map modifications

1. Get current map and decompile it:

```
ceph osd getcrushmap -o crushmap.compiled
```

```
crushtool -d crushmap.compiled -o crushmap.decompiled
```

2. Change the first CRUSH rule and add one more for caching pools:

```
……...
# rules
rule replicated_ruleset {
            ruleset 0
            type = replicated
            min_size 1
            max_size 10
            step take regular
            step chooseleaf firstn 0 type host
            step emit
}
rule cache_ruleset {
           ruleset 1
           type replicated
           min_size 1
           max_size 10
           step take cache
           step chooseleaf firstn 0 type host
           step emit
}
```

3. Save the map, compile and upload it:

```
crushtool -c crushmap.decompiled -o crushmap_modified.compiled
ceph osd set crushmap -i crushmap_modified.compiled
```

Watching at **ceph -w** output, find out when the process of replication of the placement groups is finished.

4. Bring all inactive OSDs back in, so that they become active:

```
# ceph osd in <osd_number>
```

2.8.3 Create new caching pools

1. Create caching pools using **cache_ruleset** CRUSH rule and creating 512 placement groups per pool (the number is calculated for 16 OSDs):

```
ceph osd pool create cache-images 512 cache_ruleset
ceph osd pool create cache-volumes 512 cache_ruleset
ceph osd pool create cache-compute 512 cache_ruleset
```

   Watching at ceph -w output, find out when the process of replication of the placement groups is finished.

2. Update ACLs for existing CEPH users, so that they can use new pools:

```
ceph auth caps client.compute mon 'allow r' osd 'allow class-read
object_prefix rbd_children, allow rwx pool=volumes, allow rx
pool=images, allow rwx pool=computes, allow rwx pool=cache-volumes,
allow rx pool=cache-images, allow rwx pool=cache-compute'

ceph auth caps client.volumes mon 'allow r' osd 'allow class-read
object_prefix rbd_children, allow rwx pool=volumes, allow rx
pool=images, allow rwx pool=cache-volumes, allow rx pool=cache-images'

ceph auth caps client.images mon 'allow r' osd 'allow class-read
object_prefix rbd_children, allow rwx pool=images, allow rwx
pool=cache-images'
```

2.8.4 Set up caching

Now, turn on caching. Caching pools have to be set up as the overlays for the regular pools.

```
ceph osd tier add compute cache-compute
ceph osd tier cache-mode cache-compute writeback
ceph osd tier set-overlay compute cache-compute

ceph osd tier add compute cache-volumes
ceph osd tier cache-mode cache-volumes writeback
ceph osd tier set-overlay volumes cache-volumes

ceph osd tier add compute cache-images
ceph osd tier cache-mode cache-images writeback
ceph osd tier set-overlay images cache-images
```

Setting up the pools will require some specific parameters of the cache to be set up.

```
ceph osd pool set cache-compute hit_set_type bloom
ceph osd pool set cache-volumes hit_set_type bloom
ceph osd pool set cache-images hit_set_type bloom
```

```
ceph osd pool set cache-compute cache_target_dirty_ratio 0.4
ceph osd pool set cache-compute cache_target_dirty_high_ratio 0.6
ceph osd pool set cache-compute cache_target_full_ratio 0.8

ceph osd pool set cache-volumes cache_target_dirty_ratio 0.4
ceph osd pool set cache-volumes cache_target_dirty_high_ratio 0.6
ceph osd pool set cache-volumes cache_target_full_ratio 0.8

ceph osd pool set cache-images cache_target_dirty_ratio 0.4
ceph osd pool set cache-images cache_target_dirty_high_ratio 0.6
ceph osd pool set cache-images cache_target_full_ratio 0.8
```

At this time, caching has to be ready to work.

2.8.5 Turn cache down

To turn off caching of a particular pool, execute the following set of commands:

```
ceph osd tier cache-mode <cache pool> forward
rados -p <cache pool> cache-flush-evict-all
ceph osd tier remove-overlay <regular pool>
ceph osd tier remove <storage pool> <cache pool>
```

3 Operations

3.1 Procedures

3.1.1 Remove an OSD

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.

1. Remove the old OSD from the cluster:

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

2. Wait till data migration complete

   ```
   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.

3. Stop the OSD:

```
service ceph stop osd.{osd-num}
```

4. Remove the OSD from the CRUSH map:

```
ceph osd crush remove osd.{osd-num}
```

5. Delete the authentication key:

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

6. Remove the OSD from cluster:

```
ceph osd rm {osd-num}
```

Note: If an OSD is removed from the CRUSH map, a new OSD subsequently created will be assigned the same number if `ceph osd create` is called without parameters.

7. Remove entry for the OSD from */etc/ceph/ceph.conf* if present.
8. Optional. If a device is to be replaced, add the new OSD using the procedure described in the *Add an OSD* subsection in the *Procedures* section.

   Note: Replication of the data to the new OSD will be performed here. If multiple OSDs are to be replaced, add new OSDs gradually to prevent excessive replication load.

3.1.2 Add an OSD

1. List disks in a node:

```
ceph-deploy disk list {node}
```

2. The Ceph-deploy tool can be used with one 'create' command or with two steps, as a safer option, or while preparing disks manually.
   a. Create a new OSD using one command:

```
ceph-deploy osd create {node}:{devicename}[:{journaldevice}]
```

   b. Use the two-step method:

```
ceph-deploy osd prepare {node}:{devicename}
ceph-deploy osd activate {node}:{devicename}
```

   c. If you are trying to add an OSD with the journal on the separate partition:

```
ceph-deploy osd prepare {node}:{devicename}:{journal_dev_name}
ceph-deploy osd activate {node}:{devicename}
```

   Note: Avoid simultaneous activation of multiple OSDs with default Ceph settings

as it can severely impact cluster performance. Backfilling (osd_max_backfills) and recovery settings (osd_recovery_max_active) can be tuned to lessen the impact of addition of multiple OSDs at once. Alternatively, multiple OSDs can be added at a lower weight and gradually increased, though this approach prolongs the addition process.

3. You may want to replace any physical device on the Ceph-OSD node (in case it is broken). In this case, same journaling partition may be used, and the steps for the drive replacement may be the following:
    a. Shut down the ceph-osd daemon if it is still running:

```
stop ceph-osd <ID>
```

    b. Figure out which device was used as a journal (it is a soft link to **/var/lib/ceph/osd/ceph-<ID>/journal**).
    c. Remove the OSD from the CRUSH map (see above).
    d. Shut down the node and replace the physical drive.
    e. Bring the node up and add the new **ceph-osd** instance with the new drive following the steps above.

4. Clean the previously used drive and prepare it for the new OSD:

```
ceph-deploy disk zap {node-name}:{devicename}
ceph-deploy --overwrite-conf osd prepare {node-name}:{devicename}
```

Important: This will DELETE all data on {devicename} disk.

5. Verify the new device is placed inside the CRUSH tree and recovery started:

```
ceph osd tree
ceph -s
```

3.1.3 Remove Ceph monitor from healthy cluster

Remove the monitor from the cluster:

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

Important: This operation is extremely dangerous to a working cluster, use with care.

3.1.4 Decreasing recovery and backfilling performance impact

The main settings which affect recovery are:

- osd max backfills - integer, default 10, the maximum number of backfills allowed to or from a single OSD.
- osd recovery max active - integer, default 15, the number of active recovery requests per OSD at one time.
- osd recovery threads - integer, default 1. The number of threads for recovering data.

Increasing these settings value will increase recovery/backfill performance, but decrease client performance and vice versa.

3.1.5 Remove Ceph monitor(s) from downed cluster

1. Find the most recent monmap:

```
ls $mon_data/monmap
```

2. Copy the monmap to a temporary location and remove all nodes that are damaged or failed:

```
cp $mon_map/monmap/{node_number} ~/newmonmap
monmaptool ~/newmonmap --rm {node-name}
...
```

3. Verify that ceph-mon is not running on the affected node(s):

```
service ceph stop mon
```

4. Inject the modified map on all surviving nodes:

```
ceph-mon -i a --inject-monmap /tmp/foo
```

5. Start surviving monitors:

```
service ceph start mon
```

6. Remove the old monitors from the *ceph.conf*.

3.1.6 Add Ceph monitor to cluster

1. Procure the monmap:

```
ceph mon getmap -o ~/monmap
```

2. Export the monkey:

```
ceph auth export mon. -o ~/monkey
```

3. Add a section for the new monitor to the */etc/ceph/ceph.conf*.
4. Add **mon_addr** for the new monitor to the new section with IP and port.
5. Create the ceph monitor:

```
ceph-mon -i {mon-name} --mkfs --monmap ~/monmap --keyring ~/monkey
```

6. The monitor will automatically join the cluster.

3.2 Failure Scenarios

3.2.1 Failed OSD device

1. Determine the failed OSD:

```
ceph osd tree |grep -i down

Example output:
# id  weighttype name   up/down     reweight
0     0.06              osd.0 down  1
```

2. Set “noout”.
3. Remove the failed OSD(0/osd.0 in the example) from the cluster. See the *Remove OSD* subsection in the *Procedures* section.

   The cluster will start to replicate data to recover the potentially lost copies.
4. Examine the node holding disk and eventually replace the drive.
5. If the drive is lost, create a new OSD and add it to the cluster. See the *Add OSD* subsection in the *Procedures* section.

What will happen to my data if one of the OSDs fails?

If an OSD is failed, Ceph starts a countdown trigger (mon_osd_down_interval), and when it expires (default is 5 minutes), recovery is commenced by replicating data to achieve assumed data replication ratio even with a failed OSD. As data is replicated across multiple OSDs, data loss only occurs if all OSDs containing a replica of the data are lost at the same time.

3.2.2 Lost journal device

As the journal device is also a physical drive, it can fail. All OSDs, which use the failed journal device, will also fail and must be recovered. After finding the root cause of failure, the whole OSD should be recreated to preserve data safety. For this, follow the steps described in the *Remove an OSD* and *Add an OSD* subsections in the *Procedures* section.

3.2.3 Failed storage node

1. Determine which node has failed with cluster health commands and tree.
2. Remove all OSDs from the cluster if they are still present. For details, see the *Remove OSD* subsection in the *Procedures* section.
3. Deploy a replacement node.
4. Add Ceph OSDs from the node as appropriate. See the *Add OSD* subsection in the *Procedures* section.

3.2.4 Failed Ceph monitor

1. Remove the monitor from the healthy cluster. For details, see the *Remove monitor from healthy cluster* subsection in the *Procedures* section.
2. Add a new monitor to the cluster. For details, see the *Add monitor* subsection in the *Procedures* section.

3.2.5 Ceph monitor quorum not met

1. Remove the monitor from the downed cluster. For details, see the *Remove monitor from downed cluster* subsection in the *Procedures* section.
2. Add a sufficient number of new monitors to a cluster to form quorum. For details, see the *Add monitor* subsection in the *Procedures* section.

3.2.6 Client loses connection

1. Repair Client network connectivity.
2. Client must be able to communicate with all Ceph monitors and OSDs.
3. Verify Ceph cluster access is restored.

3.2.7 Network issue in Ceph cluster environment

1. Repair inter-cluster connectivity.
2. Ceph monitors and OSD nodes should have working intra-cluster communication. Two networks can be used there.
3. Verify Ceph cluster access is restored.

3.2.8 Time synchronization issue

There are strong requirements to keep all cluster nodes in sync. Most important is to have every MON node synchronized with some arbitrary time source. The Paxos algorithm relays on timestamped maps that are created and marked down relying on cluster state. If there is any time difference, Paxos can mark healthy process as down, become unreliable, partition a cluster, or even die. Internal MON check verifies that the maximal time difference between running nodes is not higher than `0.05`sec (default value).

It is therefore strongly recommended using NTP, ideally a common NTP source, to keep all cluster nodes and clients in time sync. However, if the cluster is not configured with NTP,

monitoring should be configured to react to the following warning:

```
HEALTH_WARN clock skew detected on mon.XXXX
```

The solution is to check clock accuracy on all MON nodes and to correct any errors.

After fixing this issue, verify that the warning disappears. It can take up to 300 seconds for Ceph to recheck clock skew.

3.2.9 Object Service failure

If the Object Service fails, the troubleshooting procedure should be engaged. First, perform the following steps:

1. Check the Apache service availability on controllers:
   ```
   curl -i http://localhost:6780/
   ```
   This should give a *200* result code with standard AWS response about empty bucket list.
2. If there is a *500* response, restart RadosGW as a simplest possible solution:
   ```
   /etc/init.d/radosgw restart
   ```

3.2.10 Complete cluster restart/power failure

Ceph monitors must be started first, then OSDs:

1. Make sure that the network connectivity is restored (wait for the switch to boot).
2. Start MON nodes/daemons.
3. Wait for the quorum to be established.
4. Start the OSD daemons.

Peering state can take significant time on big clusters with OSD daemons full of data placed on HDD drives. This can even cause timeouts or a daemon irresponsibility timeout finished with a daemon suicide procedure. During this stage, OSD flapping can be observed when a heavily busy OSD daemon is losing and recovering connectivity to MON and other OSDs. When an OSD dies during this phase, it should be restarted. The second or another peering try should be significantly faster because of the file, descriptors, and directory tree cache.

If starting a cluster causes a massive amount of timeouts and the daemon suicides all the time, a couple of options can be changed to help OSD daemons wait a little bit more.

Increasing those options will help to establish a stable OSD, and will decrease the amount of daemon suicides caused by timeouts.

```
osd heartbeat grace = 240     # default is 20
mon osd report timeout = 1800 # default is 900
osd heartbeat interval = 12   # default is 6
```

During a complete cluster startup, client operations will be enabled when at first MON quorum is established and a minimal amount of replicas is available (active PGs).

3.2.11 Out of disk space on MON

It is crucial to have enough free space in MON work directories. LevelDB, used as core MON internal state database, is very fragile to out of disk space situations. At the level of 5% of disk space available, MON will exit and will not start.

MON database is growing during normal operations, this is the standard behaviour of the LevelDB store. To reclaim this space, perform one of the following:

- Add the following option to the ceph.conf file and restart mon:

```
mon compact on start = true
```

- Or run the command:

```
ceph tell mon.XXX compact
```

Note: During the compacting procedure, the LevelDB database will grow even more at first. Then compacting will replace many files with one with actual data and thus reclaim the free space.

3.2.12 Out of disk space on OSD

Ceph prevents writing to an almost full OSD device. By default it stops at 95% of used disk space. A warning message appears at the level of 85%.

1. Analyse the situation, check “ceph health detail” to find an almost full OSD.
2. Add more space to logical volume, or
   Add new OSD devices.
3. Wait for rebalance and refill the OSDs.

3.3 Tuning

Ceph clusters can be parametrized after deployment to better fit the requirements of the workload. Some configuration options can affect data redundancy and have significant implications on stability and safety of data. Tuning should be performed on test environment prior issuing any command and configuration changes on production.

All changes should be documented and reviewed by experienced staff. Before and after the change, the full set of tests should be executed.

3.3.1 Using ceph-deploy to distribute configuration over cluster

The ceph-deploy tool can also be used to distribute configuration changes over the cluster. It is recommended implementing any fixes on one node and then distributing the new configuration file to the rest of the nodes.
The following can be executed on a node with changed configuration:

```
ceph-deploy config push nodename1 nodename2 nodename3
```

Another useful tip is to use this tool with name expansion format as:

```
ceph-deploy config push nodename{1,2,3}
```

which will be equal to the example above.

3.3.2 Changes

3.3.2.1 Changes in a config file

All changes made in configuration file will be read and implemented during the daemon startup. Thus, after making any changes in the ceph.conf configuration file, the daemons need to be restarted to take the changes in effect.

3.3.2.2 Online changes with monitor

Changes can be injected online through monitor to the daemon communication channel:

```
ceph tell osd.0 injectargs --debug-osd 20
```

3.3.2.3 Online changes with admin socket

Changes can also be implemented by using admin socket communication to daemon while MON is unreachable or when it is more convenient:

```
ceph --admin-daemon /var/run/ceph/ceph-osd.0.asok config set debug_osd
20/20
```

Important: Any online changes will not be saved during daemon restart. To make the changes permanent, configuration file change is mandatory.

3.3.3 Common tuning parameters

For a production cluster, any changes in configuration should be tested in a test environment. However, there are some situations when a production cluster can respond differently to changes and make a regression. Extreme caution is required when performing any tuning.

The most commonly changed parameters:

```
public_network = 192.168.0.4/24 #points to client network
cluster_network = 192.168.1.2/24 #points to inter OSD communication
osd_recovery_max_active = 1 #
osd_max_backfills = 1
```

3.3.4 Performance measurement best practice

For best measurement results, follow these rules while testing:

1. Change one option at a time.
2. Understand what is changing.
3. Choose the right performance test for the changed option.
4. Retest at least ten times.
5. Run tests for hours, not seconds.
6. Trace for any errors.
7. Decisively look at results.
8. Always try to estimate results and see at standard difference to eliminate spikes and false tests.

3.4 Ongoing operations

3.4.1 Background activities

The Ceph cluster is constantly monitoring itself with scrub and deep scrub. Scrub verifies attributes and object sizes. It is very fast and not very resource-hungry - ideal for daily checks. Deep scrub checks each rados object's checksum by CRC32 algorithm, and every difference in replicas is reported as inconsistent.
Scrub and deep scrub operations are very IO-consuming and can affect cluster performance. However, these operations should be enabled to ensure data integrity and availability. Ceph tries to execute scrub and deep scrub when a cluster is not overloaded. But once executed, scrub is running till it finishes checking of all the PGs.

To disable scrub and deep scrub, run the following commands:

```
ceph set noscrub
ceph set nodeep-scrub
```

To restore standard options, run:

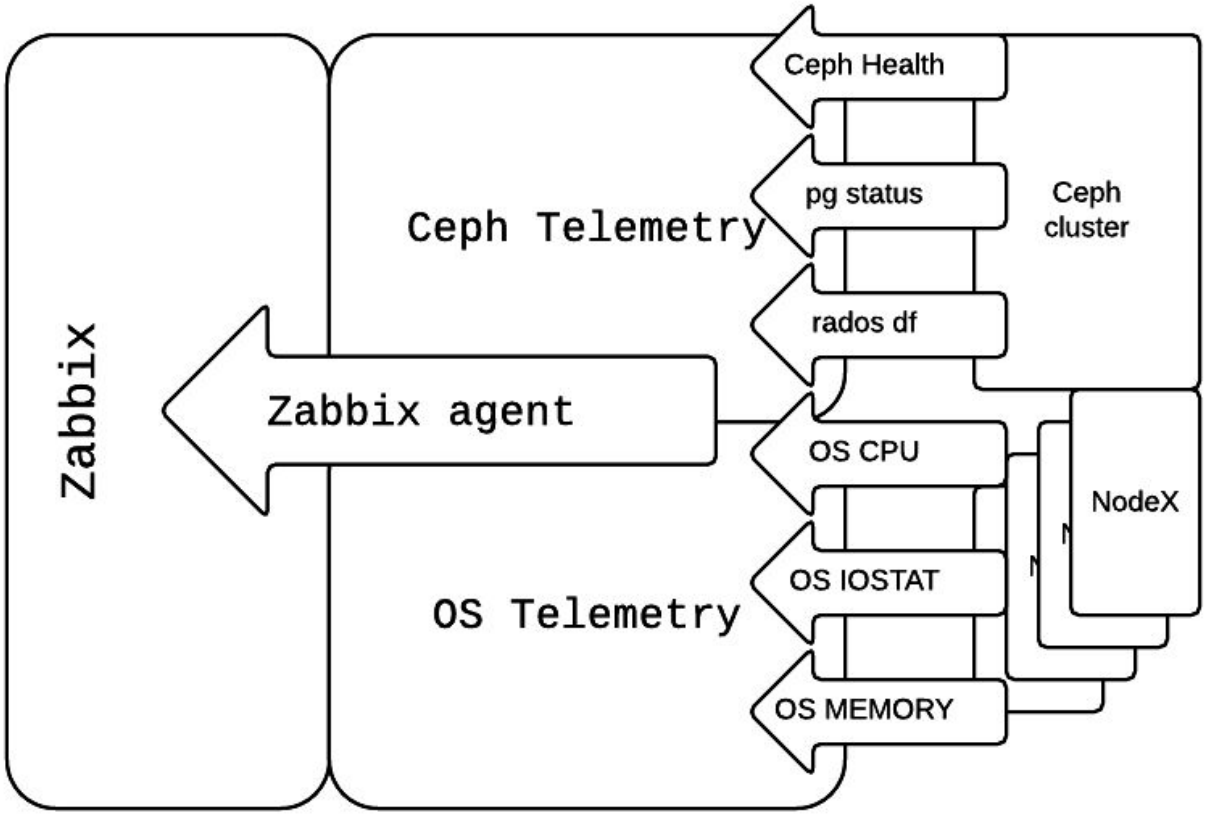
```
ceph unset noscrub
ceph unset nodeep-scrub
```

To fine-tune the scrub processes, use the following configuration options (default values provided):

```
osd_scrub_begin_hour = 0 # begin at this hour
osd_scrub_end_hour = 24 # start last scrub at
osd_scrub_load_threshold = 0.05 #scrub only below load
osd_scrub_min_interval = 86400 # not more often than 1 day
osd_scrub_max_interval = 604800 # not less often than 1 week
osd_deep_scrub_interval = 604800 # scrub deeply once a week
```

3.4.2 Monitoring

Monitoring of a Ceph cluster should include utilization, saturation and errors. Ceph itself has a number of tools to check cluster health - beginning with simple CLI tools, ending with API methods to gather health status. Several methods for observing cluster performance and health can be deployed. It is most common to include health checks in some dedicated monitoring software like Zabbix and Nagios. Standard Ceph messages have two severities - WARN and ERR, both of them should be noted by cluster operators and both should be treated as significant signals to check the cluster.
Basic metrics of the Ceph cluster and of the OS on the Ceph nodes should also be monitored to see actual cluster utilization and predict possible performance issues.



In a Fuel deployed Ceph cluster, Zabbix is configured as the main monitoring software. It gathers and stores all information into an internal database. All parameters are configured as items. When an item arrives, it is also compared with configured triggers. When the value of an item is beyond the trigger rule, some action can be executed. Zabbix provides a dashboard with graphs and plots trends for easy to use day to day monitoring.
The amount of gathered data depends on the Zabbix agent and options configuration. The administrator can add or modify items to better fit monitoring requirements.

Special attention should be given to disk space management, since stability and overall cluster health depends on many database or metadata operations.

The Ceph cluster consists of a number of separate, dedicated daemon processes. Monitoring software should check for the correct number of MON, OSD's and RadosGW processes. It is also useful to monitor the memory allocated and used by processes to quickly find any memory leaks - especially from an OSD process.

Process table

Name	Purpose	Process name	Count on host	Open Ports	System Memory Per OSD
Ceph MON	cluster coordination	ceph-mon	1, at least 3 summary	6789	~1GB
Ceph OSD	data daemon	ceph-osd	as many as devices	6800, 6801, 6802, 6803	~1-3GB
RadosGW	http rest interface	radosgw	as necessary on HTTP gateways	fastcgi socket	<1GB

3.4.3 Dumping memory heap

Every Ceph process can be instructed to dump memory heap. This can be very helpful to debug any memory leaks and process-related problems.

Memory heap dumping procedure:

1. Start profiler:

```
ceph osd tell osd.0 heap start_profiler
```

2. Dump heap:

```
ceph tell osd.0 heap dump

Example output:
osd.0dumping heap profile now.
------------------------------------------------
MALLOC:       41200288 (   39.3 MiB) Bytes in use by application
MALLOC: +       581632 (    0.6 MiB) Bytes in page heap freelist
MALLOC: +      5124544 (    4.9 MiB) Bytes in central cache freelist
MALLOC: +      2752512 (    2.6 MiB) Bytes in transfer cache freelist
MALLOC: +      6964128 (    6.6 MiB) Bytes in thread cache freelists
MALLOC: +      1323160 (    1.3 MiB) Bytes in malloc metadata
MALLOC:   ------------
MALLOC: =     57946264 (   55.3 MiB) Actual memory used (physical + swap)
MALLOC: +            0 (    0.0 MiB) Bytes released to OS (aka unmapped)
MALLOC:   ------------
MALLOC: =     57946264 (   55.3 MiB) Virtual address space used
MALLOC:
MALLOC:           2544              Spans in use
MALLOC:             86              Thread heaps in use
MALLOC:           8192              Tcmalloc page size
------------------------------------------------
Call ReleaseFreeMemory() to release freelist memory to the OS (via madvise()).
Bytes released to the
```

3. Stop profile:

```
ceph tell osd.0 heap stop_profiler
```

3.4.4 Maintenance

The Ceph cluster is mostly auto-managed and auto-healing on the software side, but the hardware underlay should be maintained, reviewed, and repaired as needed. Ceph reacts to any hardware failure and prevents data loss by making additional replicas when any OSD is lost.
The administrator should apply available upgrades, especially revision releases, after careful consideration and planning. Security and critical patches are backported to long-term support releases (as Firefly) just after implementation to current release.

For Fuel deployed environment, refer to the Mirantis OpenStack Operations Guide.

4 Troubleshooting

4.1 Overall Ceph cluster health

To observe overall Ceph cluster health, use simple bundled commands:

Simple status:

```
ceph -s
```

Detailed Ceph health with additional informations and debug output:

```
ceph health detail
```

Report on config and runtime statuses and counters (very extensive):

```
ceph report
```

4.2 Logs

Logs for all Ceph core components are stored in the `/var/log/ceph` directory. Logs are rotated daily.
Default debug levels are not sufficient in many times. Ceph daemons enable changing the log level through a live argument injection to daemon process. While most of the log facilities are running with semi-high (5) logging level in memory, the log-level for on-disk storing is enabled at very low level only for some services.
Debug level can be changed online or in `/etc/ceph/ceph.conf` file for boot time read.

Debug level => 0 - turned off => 1 - terse => 20 - verbose

The simplest way to change debug level online:

```
ceph tell osd.0 injectargs --debug-osd 5/5
```

Note: This method requires MON connectivity. If you have issues with that, then use the next method (configuration change or socket connection).

Changing debug level by connecting to local socket (has to be run on daemon's machine):

```
ceph --admin-daemon /var/run/ceph/ceph-osd.0.asok config set debug_osd 5/5
```

The most common service list to debug Ceph issues is: rados, crush, osd, filestore, ms, mon, auth.

Ceph logging subsystem is very extensive and resource consuming, it can generate a lot of data in a very short time. Be aware of free disk space for verbose logging.
Procedure entry into log and debug routines are also very time-consuming, you should be aware that best performance results can be archived without any debug options and log levels set to turned-off. It is recommended that you keep reasonable low level of debugging during normal operations and set it higher only for troubleshooting.

4.3 Failed MON

The MON instances are most important to the cluster, so troubleshooting and recovery should begin with those instances.

Use the following command to display the current state of quorum, MONs, and PAXOS algorithm status:

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

If a client can not connect to MON, there can be problems with:

1. Connectivity and firewall rules. Verify that the TCP port 6789 is allowed on monitor hosts.
2. Disk space. There should be safe free disk space margin for LevelDB internal database operation on every MON node.
3. MON process that is not working or is out of quorum. Check `quorum_status`, `mon_status` and `ceph -s` output to identify failed MON and try to restart it or deploy a new one instead.

If the methods above fail, try to increase debug level on `debug_mon` to 10/10 via inject args, or admin socket as described in 3.1 to find the root cause of failure.
If a daemon is failing on LevelDB operation or another assertion, file a bug report for Ceph.

4.4 Failed OSD

It is important to continuously monitor cluster health as there can be many different root causes that cause OSD processes to die. Some of them may be caused by hardware failures, including hard to determine and unpredictable firmware and physical failures. There is also a possibility to experience a software bug that can cause bad assertion and abnormal OSD exit.
Ceph administrator has several ways to determine Ceph cluster health, most common is to observe admin commands output first, then to deeply debug failed devices and equipment. Ceph cluster can be monitored from any node involved into cluster operations, but good practice is to check it from MON nodes, as they are the closest to local MON daemons.

```
ceph -s
or
ceph health
```

In case of any concerns, warnings, and errors, the troubleshooting procedure should be engaged. The health of the cluster is crucial to the data safety and for the operations reliability. This is an example of command output while one of OSD daemons is down:

```
cluster f4ad6d65-6d37-4318-9e5c-a5f59d6e6ad7
health HEALTH_WARN 767 pgs stale; 1 requests are blocked > 32 sec; 1/4 in
osds are down
monmap e1: 1 mons at {node-36=192.168.0.1:6789/0}, election epoch 1, quorum 0
node-36
osdmap e74: 4 osds: 3 up, 4 in
pgmap v1452: 3008 pgs, 14 pools, 12860 kB data, 51 objects
8403 MB used, 245 GB / 253 GB avail
                 767 stale+active+clean
                2241 active+clean
client io 0 B/s rd, 0 B/s wr, 0 op/s
```

If a problem with an OSD is identified (by looking at **HEALTH_WARN** and the number of the OSDs that are down and up OSD count), perform the procedure for replacing the failed OSD. See the *Failed OSD device* subsection in the *Failure Scenarios* section of this document.

Possible causes (most common) are:

- Hard disk failure. It can be determined by system messages, or SMART activity. Some defective disks are very slow because of extensive TLER activity.
- Network connectivity issues. You can use ordinary network check tools like ping, tracepath, iperf to debug this.
- Out of disk space on filestore. When you are running out of space, Ceph is triggering alarms with HEALTH_WARN on 85% full and HEALTH_ERR on 95% full. Then it stops to prevent fulfillment of whole disk. Note that it is not just filestore, it holds also indexed

metadata and metadata for files. It is very important to keep enough free space for smooth operations.

- Running out of system resources or hitting limits cap. There should be enough system memory to hold all OSD processes on machine, and system limits for open files and maximal number of threads should be big enough.
- OSD process heartbeats limits causes processes to suicide. A default process and communication timeouts can be not enough to perform IO-hungry operations - especially during recovery after failure. This can be also observed as OSD flapping.

4.4.1 OSD is flapping during peering state, after restart or recovery

You can stabilize IO-hungry operations causing timeouts by turning on “nodown”, “noup” options for the cluster:

```
ceph set nodown
ceph set noup
ceph set noout
```

When the whole cluster is healthy and stable, restore this to default values by running:

```
ceph unset nodown
ceph unset noup
ceph unset noout
```

4.4.2 How to determine that a drive is failing

Logs should contain extensive information regarding the failing device. There should also be some sign in SMART.

To check the logs, the administrator can execute:

```
dmesg | egrep sd[a-z]
```

Examine the suspicious device with `smartctl` to extract informations and perform tests:

```
smartctl -a /dev/sdX
```

The drive that is going to fail can be also determined by watching response (seek) times and overall disk responsiveness. Any drive that shows sustained unusual values can be about to fail:

```
iostat -x /dev/sdX
```

It is also necessary to watch for **`avgqu-sz`** values (should be lower than device queue) and **`util`** parameters. They should be more or less equal on all the devices of the same type.

4.5 Failed node

First examination should determine connectivity issues and network-related problems. If SSH connection to node is working, and simple ping tests ensure that network layers are OK, further examination should focus on:

1. Failed node hardware failure that can cause:
    - Connectivity issues
    - OSD to die on long-lasting IO operation
    - OSD to die on IO error or EOT
    - Many unpredictable issues with OS or OSD daemons
2. Failed node software (OS, Ceph) issue

4.6 Issues with Placement Groups (PGs)

4.6.1 PG Status

The optimum PG state is 100% active + clean. This means that all Placement Groups are accessible, and assumed replica number is avaliable for all PGs. If Ceph also reports other states, it is a warning or an error status (beside `scrub` or `deep-scrub` operations).
PG status quick reference (for a complete one, refer to the official Ceph documentation):

State	Description
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.
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 (for example objects are wrong size, objects are missing from one replica *after* recovery finished, and so on).
Peering	The placement group is undergoing the peering process.
Recovering	Ceph is migrating/synchronizing objects and their replicas.
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.

4.6.2 PG stuck in some state for a long time

When a new *Pool* is created and after a reasonable time does not get an active+clean status, there is most likely an issue with configuration, CRUSH map, or there are too few resources to achieve configured replication level.
Debugging should start with examination of cluster state and PG statuses:

```
ceph osd pg dump# to find PG number for any status
ceph pg {pg_id} query #to see verbose information about PG in JSON
```

While analysing query output, special attention should be paid on “info”, “peer_info”, and “recovery_status” sections.

The monitor warns about PGs that are stuck in the same status for some time. They can be listed with:

```
ceph pg dump_stuck stale
ceph pg dump_stuck inactive
ceph pg dump_stuck unclean
```

4.6.3 Default ruleset constraints

The Ceph data distribution algorithm is working according to the rulesets that are stored in the monmap encoded file. The monmaps are replicated and versioned to maintain cluster consistency and condition.
If there is an issue with the syntax of the ruleset, it should be found during monmap compilation, but there can be logical mistakes that will pass analyze before compilation and will cause the CRUSH algorithm to not distribute data as was assumed, or it will prevent the cluster from getting the active+clean state on all PGs.

The first thing to verify is whether the default replication ratio condition is met by checking min_size and size in conjunction with the hardware configuration of cluster. The default Ceph configuration is prepared for replication against hosts, not OSDs.

4.6.4 Inconsistent PG after scrub or deep-scrub

The scrub operation is used to check the availability and health of objects. PGs are scrubbed while a cluster is not running any IO intensive operations, for example recovery (scrubbing already started will continue, though). If this task finds any object with broken or mismatched data (checksum is checked), it will mark this object as unusable and manual intervention and recovery is needed.
Ceph prior 0.90 version does not store object checksum information while it is written. Checksums are calculated on OSD write operations, and Ceph cannot arbitrarily decide which one is the correct one. For a simple example with 3 replicas and one different checksum it is easy to guess which one is wrong and should be corrected (recovered from other replica), but when there are 3 different checksums, or we got some bit rot, or malfunction of the controller on two nodes - we cannot arbitrarily say which one is good. It is not an end-to-end data correction check.

Manual repair of a broken PG is necessary:

1. First find a broken PG with inconsistent objects:

```
ceph pg dump | grep inconsistent
```
or
```
ceph health detail
```

2. Then instruct to repair (when the primary copy is our good data), or repair manually, by moving/deleting wrong files on OSD disk:

```
ceph pg repair {pgnum}
```

Important: The repair process is very tricky when the primary copy is broken. Current repair behavior with replicated PGs is to copy the primary's data to the other nodes. This makes the Ceph cluster self-consistent, but might cause problems for consumers if the primary had the wrong data.

4.6.5 Incomplete PG

This warning is issued when the actual replica number is less that min_size.

4.6.6 Unfound objects

When Ceph cluster health command returns information about unfound objects, it means that there are some parts of data that are not accessible in even one copy.

```
ceph health detail
```

Thy following command displays a PG name with unfound objects. Then the PG should be examined for any missing parts:

```
ceph pg {pgname} list_missing
```

4.6.7 Stale PG

Simply restart an affected OSD. This issue occurs when an OSD cannot map all objects that it holds. To find the OSD, run the following command:

```
ceph pg dump_stuck stale
```

Then map the PG:

```
ceph pg map {pgname}
```

Alternatively, the information can be acquired with:

```
ceph health detail
```

This command will display defective OSDs as "`last acting`" ones.

Those daemons should be restarted and deeply debugged.

4.6.8 Peering and down PGs

When any peering and down PGs are lasting for a long time after any cluster change (recovery, adding new OSD, map or ruleset changes).

The following command will display affected PGs, then we should identify the issue causing peer:

```
ceph health detail
```

Thy following command will display in the ["recovery_state"][“blocked”] section why the peering is stopped:

```
ceph pg {pgname} query
```

There will be information about some OSD being down in most of cases.

When the OSD cannot be brought up again, it should be marked as “lost”, and the recovery process will begin:

```
ceph osd lost {osd_number}
```

4.7 Resolving issues with CRUSH maps

After making any changes in CRUSH maps, the new version should be tested to confirm compliance with OSD layout and to review any issues with new data placement. It is also good to review it and look at the amount of data that will be remapped with new placement reorder.

```
crushtool -i crush.map --test --show-bad-mappings \
   --rule 1 \
   --num-rep 9 \
   --min-x 1 --max-x $((1024 * 1024))
```

Placement statistics can be checked with the following command:

```
crushtool -i crush.map --test --show-utilization --rule 1
```

4.8 Object service RadosGW troubleshooting

RadosGW is the object storage service component of Ceph. It provides an S3 and Swift compatible RESTful interface to the Ceph RADOS back-end store.
The RadosGW daemon is connected through FastCGI interface with an Apache HTTPD server, which acts as an HTTP gateway to the outside.

4.8.1 RadosGW logs

RadosGW logs are separated and stored in:

```
/var/log/radosgw/ceph-client.radosgw.gateway.log
```

The logs are rotated daily like the rest of Ceph logs.

To debug the rgw service, the log level can be increased:

```
debug rgw = 10/10  # (Representing Log Level and Memory Level)
```

This setting will provide extensive information output and a significant amount of data to analyse. The default settings are ‘1/5’.

For the Apache HTTPd daemon the logs are stored in:

```
/var/log/apache
```

The main error.log in this directory is useful for debugging most of the issues.

4.8.2 RadosGW daemon pools

Object storage is using several pools. And to troubleshoot any performance and availability issue, debugging of underlying Rados pools is essential:

1. Check overall cluster health and PG statuses.
2. Check Ceph cluster performance with simple checks.
3. Check RadosGW process and logs.

Default pool names to consider health status:

```
.rgw.root
.rgw.control
.rgw
.rgw.gc
.users.uid
.users
.rgw.buckets.index
.rgw.buckets
```

4.8.3 Authorization issues

RadosGW connects to OpenStack Identity service (Keystone) for authorization. If Keystone is not available, this will result in constant authorization failures and *403 Access Forbidden* responses to the clients.
Connection and availability of the Identity service should be checked.

To check whether a user is available:

```
radosgw-admin user info --uid {radosgwuser}
```

The user should not be `suspended` to use this service and should have an S3 key and a Swift key to use both service endpoints.

4.8.4 Remapping index of RadosGW buckets

While PGs holding bucket index are remapped (during for example cluster expansion or osd failure), significant delays and slow queries can occur.

4.8.5 Quick functional check for RadosGW service

The s3curl tool can be used to perform simple and quick tests for the RadosGW. A modified version is available to test out RGW, while the original version was written for the AWS S3 service.

```
apt-get install libdigest-hmac-perl
git clone https://github.com/rzarzynski/s3curl.git
cd s3curl
chmod 755 s3curl.pl

#to get user credentials (keys)
radosgw-admin user info --uid={rgwuid}

#bucket creation
./s3curl.pl --debug --id {accesskey} --key {secretkey} --endpoint
--createBucket -- http://localhost:6780/test

#put object into test bucket
./s3curl.pl --debug --id {accesskey} --key {secretkey} --endpoint --put
/etc/hostname -- http://localhost:6780/test/hostname

#list objects in test bucket
./s3curl.pl --debug --id {accesskey} --key {secretkey} --endpoint --
http://localhost:6780/test/

#put object into test bucket
./s3curl.pl --debug --id {accesskey} --key {secretkey} --endpoint --delete --
http://localhost:6780/test/hostname
```

All test should pass and response correct HTTP response codes.