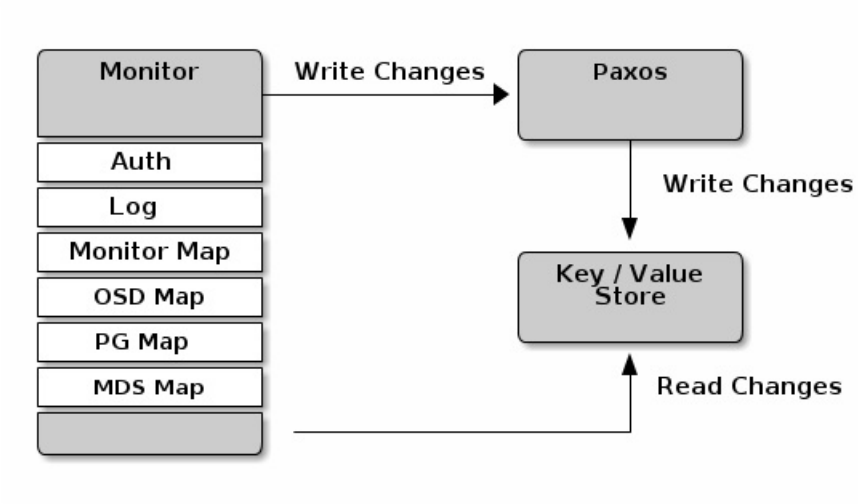
MONITOR CONFIG REFERENCE

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 Configuring ceph 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.).

`mon force quorum join`

Description:	Force monitor to join quorum even if it has been previously removed from the map
Type:	Boolean
Default:	`False`

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 are not 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 is not 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.

Monitors can also be found by clients using DNS SRV records. See Monitor lookup through DNS 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. As leveldb is using `mmap()` for writing the data, Ceph Monitors flush their data from memory to disk very often, which can interfere with Ceph OSD Daemon workloads if the data store is co-located with the OSD Daemons.

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`

`mon data size warn`

Description: Issue a `HEALTH_WARN` in cluster log when the monitor's data store goes over 15GB.
Type: Integer
Default: 15*1024*1024*1024*

`mon data avail warn`

Description: Issue a `HEALTH_WARN` in cluster log when the available disk space of monitor's data store is lower or equal to this percentage.
Type: Integer
Default: 30

`mon data avail crit`

Description: Issue a `HEALTH_ERR` in cluster log when the available disk space of monitor's data store is lower or equal to this percentage.
Type: Integer
Default: 5

`mon warn on cache pools without hit sets`

Description: Issue a `HEALTH_WARN` in cluster log if a cache pool does not have the hitset type set set. See hit set type for more details.
Type: Boolean
Default: True

`mon warn on crush straw calc version zero`

Description: Issue a `HEALTH_WARN` in cluster log if the CRUSH's `straw_calc_version` is zero. See CRUSH map tunables for details.
Type: Boolean
Default: True

`mon warn on legacy crush tunables`

Description: Issue a `HEALTH_WARN` in cluster log if CRUSH tunables are too old (older than `mon_min_crush_required_version`)
Type: Boolean
Default: True

`mon crush min required version`

Description: The minimum tunable profile version required by the cluster. See CRUSH map tunables for details.
Type: String
Default: `firefly`

`mon warn on osd down out interval zero`

Description: Issue a `HEALTH_WARN` in cluster log if `mon osd down out interval` is zero. Having this option set to zero on the leader acts much like the `noout` flag. It's hard to figure out what's going wrong with clusters witout the `noout` flag set but acting like that just the same, so we report a warning in this case.
Type: Boolean
Default: True

`mon cache target full warn ratio`

Description: Position between pool's `cache_target_full` and `target_max_object` where we start warning
Type: Float
Default: `0.66`

`mon health data update interval`

Description: How often (in seconds) the monitor in quorum shares its health status with its peers. (negative number disables it)
Type: Float
Default: `60`

`mon health to clog`

Description: Enable sending health summary to cluster log periodically.
Type: Boolean
Default: True

`mon health to clog tick interval`

Description: How often (in seconds) the monitor send health summary to cluster log (a non-positive number disables it). If current health summary is empty or identical to the last time, monitor will not send it to cluster log.
Type: Integer
Default: 3600

`mon health to clog interval`

Description: How often (in seconds) the monitor send health summary to cluster log (a non-positive number disables it). Monitor will always send the summary to cluster log no matter if the summary changes or not.
Type: Integer
Default: 60

STORAGE CAPACITY

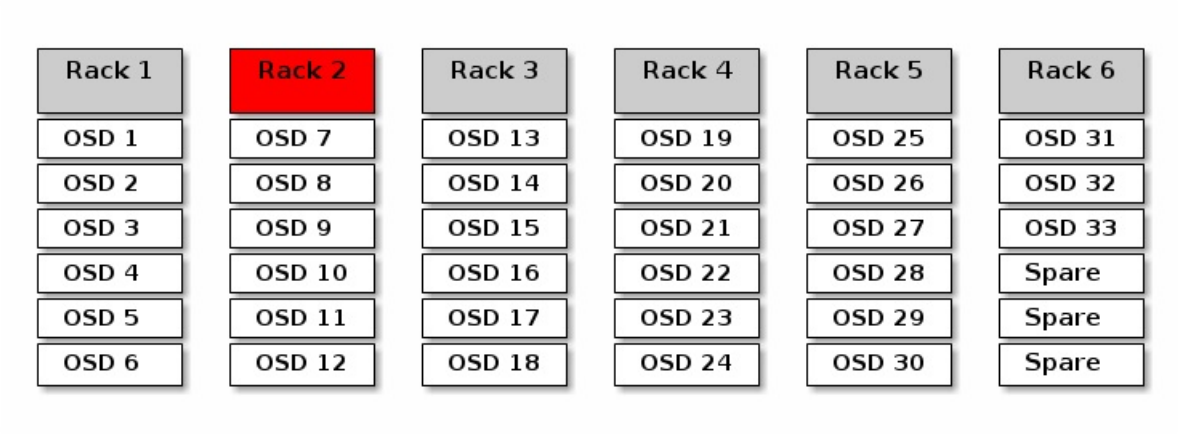
When a Ceph Storage Cluster gets close to its maximum capacity (i.e., `mon osd full ratio`), Ceph prevents you from writing to or reading from Ceph OSD Daemons as a safety measure to prevent data loss. Therefore, letting a production Ceph Storage Cluster approach its full ratio is not a good practice, 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 backfillfull ratio = .75
        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 backfillfull ratio`

Description: The percentage of disk space used before an OSD is considered too `full` to backfill.
Type: Float
Default: `.90`

`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

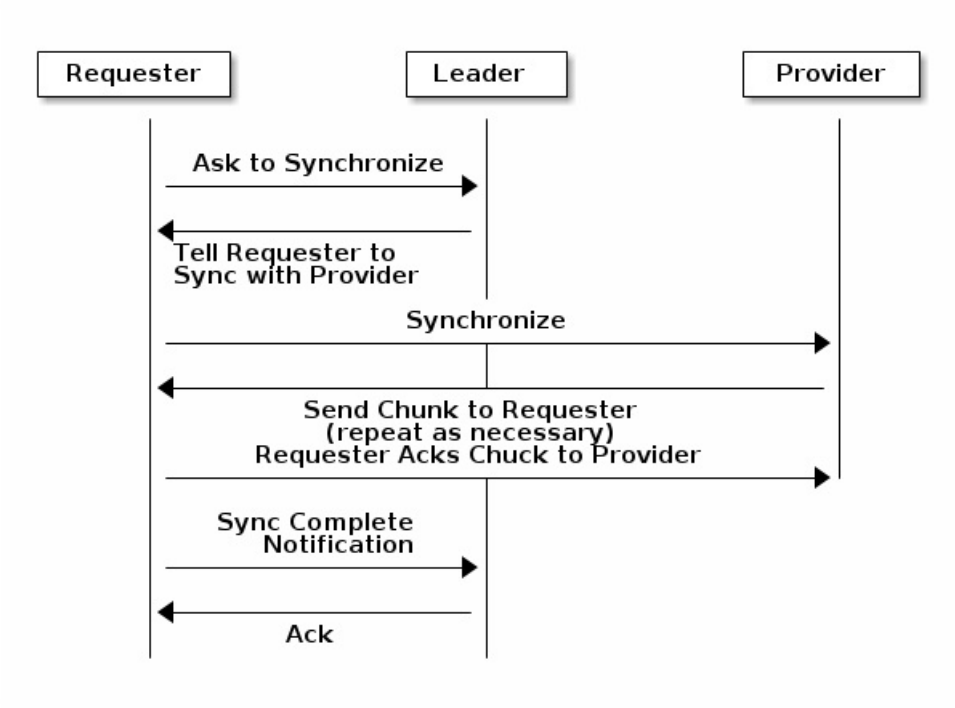
Ceph monitors know about the cluster by requiring reports from each OSD, and by receiving reports from OSDs about the status of their neighboring OSDs. Ceph provides reasonable default settings for monitor/OSD interaction; however, you may modify them as needed. See Monitor/OSD Interaction for details.

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.



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: Number of seconds the monitor will wait for the next update message from its sync provider before it gives up and bootstrap again.
Type: Double
Default: `60.0`

`mon sync max retries`

Description:
Type: Integer
Default: `5`

`mon sync max payload size`

Description: The maximum size for a sync payload (in bytes).
Type: 32-bit Integer
Default: `1045676`

`paxos max join drift`

Description: The maximum Paxos iterations before we must first sync the monitor data stores. When a monitor finds that its peer is too far ahead of it, it will first sync with data stores before moving on.
Type: Integer
Default: `10`

`paxos stash full interval`

Description: How often (in commits) to stash a full copy of the PaxosService state. Current this setting only affects `mds`, `mon`, `auth` and `mgr` PaxosServices.
Type: Integer
Default: 25

`paxos propose interval`

Description: Gather updates for this time interval before proposing a map update.
Type: Double
Default: `1.0`

`paxos min`

Description: The minimum number of paxos states to keep around
Type: Integer
Default: 500

`paxos min wait`

Description: The minimum amount of time to gather updates after a period of inactivity.
Type: Double
Default: `0.05`

`paxos trim min`

Description: Number of extra proposals tolerated before trimming
Type: Integer
Default: 250

`paxos trim max`

Description: The maximum number of extra proposals to trim at a time
Type: Integer
Default: 500

`paxos service trim min`

Description: The minimum amount of versions to trigger a trim (0 disables it)
Type: Integer
Default: 250

`paxos service trim max`

Description: The maximum amount of versions to trim during a single proposal (0 disables it)
Type: Integer
Default: 500

`mon max log epochs`

Description: The maximum amount of log epochs to trim during a single proposal
Type: Integer
Default: 500

`mon max pgmap epochs`

Description: The maximum amount of pgmap epochs to trim during a single proposal
Type: Integer
Default: 500

`mon mds force trim to`

Description: Force monitor to trim mdsmaps to this point (0 disables it. dangerous, use with care)
Type: Integer
Default: 0

`mon osd force trim to`

Description: Force monitor to trim osdmaps to this point, even if there is PGs not clean at the specified epoch (0 disables it. dangerous, use with care)
Type: Integer
Default: 0

`mon osd cache size`

Description: The size of osdmaps cache, not to rely on underlying store’s cache
Type: Integer
Default: 10

`mon election timeout`

Description: On election proposer, maximum waiting time for all ACKs in seconds.
Type: Float
Default: 5

`mon lease`

Description: The length (in seconds) of the lease on the monitor's versions.
Type: Float
Default: `5`

`mon lease renew interval factor`

Description: `mon lease` * `mon lease renew interval factor` will be the interval for the Leader to renew the other monitor's leases. The factor should be less than `1.0`.
Type: Float
Default: `0.6`

`mon lease ack timeout factor`

Description: The Leader will wait `mon lease` * `mon lease ack timeout factor` for the Providers to acknowledge the lease extension.
Type: Float
Default: `2.0`

`mon accept timeout factor`

Description: The Leader will wait `mon lease` * `mon accept timeout factor` for the Requester(s) to accept a Paxos update. It is also used during the Paxos recovery phase for similar purposes.
Type: Float
Default: `2.0`

`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`

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 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 is not 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`

`mon timecheck skew interval`

Description: The time check interval (clock drift check) in seconds when in presence of a skew in seconds for the Leader.
Type: Float
Default: `30.0`

CLIENT

`mon client hunt 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`

POOL SETTINGS

Since version v0.94 there is support for pool flags which allow or disallow changes to be made to pools.

Monitors can also disallow removal of pools if configured that way.

`mon allow pool delete`

Description: If the monitors should allow pools to be removed. Regardless of what the pool flags say.
Type: Boolean
Default: `false`

`osd pool default flag hashpspool`

Description: Set the hashpspool flag on new pools
Type: Boolean
Default: `true`

`osd pool default flag nodelete`

Description: Set the nodelete flag on new pools. Prevents allow pool removal with this flag in any way.
Type: Boolean
Default: `false`

`osd pool default flag nopgchange`

Description: Set the nopgchange flag on new pools. Does not allow the number of PGs to be changed for a pool.
Type: Boolean
Default: `false`

`osd pool default flag nosizechange`

Description: Set the nosizechange flag on new pools. Does not allow the size to be changed of pool.
Type: Boolean
Default: `false`

For more information about the pool flags see Pool values.

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 subscribe interval`

Description: The refresh interval (in seconds) for subscriptions. The subscription mechanism enables obtaining the cluster maps and log information.
Type: Double
Default: `86400`

`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`

`mon osd prime pg temp`

Description: Enables or disable priming the PGMap with the previous OSDs when an out OSD comes back into the cluster. With the `true` setting the clients will continue to use the previous OSDs until the newly in OSDs as that PG peered.
Type: Boolean
Default: `true`

`mon osd prime pg temp max time`

Description: How much time in seconds the monitor should spend trying to prime the PGMap when an out OSD comes back into the cluster.
Type: Float
Default: `0.5`

`mon osd prime pg temp max time estimate`

Description: Maximum estimate of time spent on each PG before we prime all PGs in parallel.
Type: Float
Default: `0.25`

`mon osd allow primary affinity`

Description: allow `primary_affinity` to be set in the osdmap.
Type: Boolean
Default: False

`mon osd pool ec fast read`

Description: Whether turn on fast read on the pool or not. It will be used as the default setting of newly created erasure pools if `fast_read` is not specified at create time.
Type: Boolean
Default: False

`mon mds skip sanity`

Description: Skip safety assertions on FSMap (in case of bugs where we want to continue anyway). Monitor terminates if the FSMap sanity check fails, but we can disable it by enabling this option.
Type: Boolean
Default: False

`mon max mdsmap epochs`

Description: The maximum amount of mdsmap epochs to trim during a single proposal.
Type: Integer
Default: 500

`mon config key max entry size`

Description: The maximum size of config-key entry (in bytes)
Type: Integer
Default: 4096

`mon scrub interval`

Description: How often (in seconds) the monitor scrub its store by comparing the stored checksums with the computed ones of all the stored keys.
Type: Integer
Default: 3600*24

`mon scrub max keys`

Description: The maximum number of keys to scrub each time.
Type: Integer
Default: 100

`mon compact on start`

Description: Compact the database used as Ceph Monitor store on `ceph-mon` start. A manual compaction helps to shrink the monitor database and improve the performance of it if the regular compaction fails to work.
Type: Boolean
Default: False

`mon compact on bootstrap`

Description: Compact the database used as Ceph Monitor store on on bootstrap. Monitor starts probing each other for creating a quorum after bootstrap. If it times out before joining the quorum, it will start over and bootstrap itself again.
Type: Boolean
Default: False

`mon compact on trim`

Description: Compact a certain prefix (including paxos) when we trim its old states.
Type: Boolean
Default: True

`mon cpu threads`

Description: Number of threads for performing CPU intensive work on monitor.
Type: Boolean
Default: True

`mon osd mapping pgs per chunk`

Description: We calculate the mapping from placement group to OSDs in chunks. This option specifies the number of placement groups per chunk.
Type: Integer
Default: 4096

`mon osd max split count`

Description: Largest number of PGs per “involved” OSD to let split create. When we increase the `pg_num` of a pool, the placement groups will be splitted on all OSDs serving that pool. We want to avoid extreme multipliers on PG splits.
Type: Integer
Default: 300

`mon session timeout`

Description: Monitor will terminate inactive sessions stay idle over this time limit.
Type: Integer

Default: 300