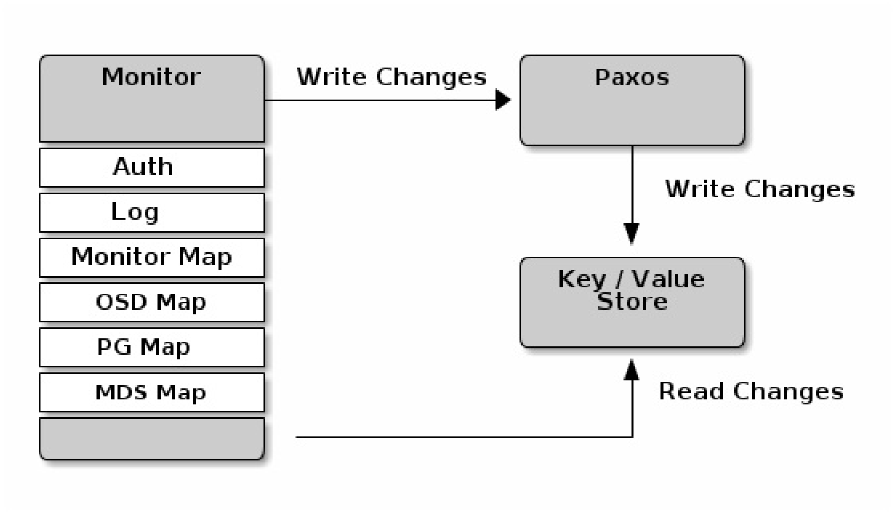
MONITOR CONFIG REFERENCE

Understanding how to configure a Ceph monitor is an important part of building a reliable Ceph cluster. **All Ceph 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 for details.

BACKGROUND

Monitors maintain a "master copy" of the cluster map, which means a client can determine the location of all monitors, OSDs, and metadata servers just by connecting to one monitor and retrieving a current cluster map. Before Ceph clients can read from or write to OSDs or metadata servers, they must connect to a monitor first. With a current copy of the cluster map and the CRUSH algorithm, a client can compute the location for any object. The ability to compute object locations allows a client to talk directly to OSDs, which is a very important aspect of Ceph's high scalability and performance.

The primary role of the monitor is to maintain a master copy of the cluster map. 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 cluster; which processes that are `in` the 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., an OSD 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 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 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 details.

MONITOR QUORUM

Our 5-minute Quick Start 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 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 cluster runs multiple 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 monitor always refers to the local copy of the monmap when discovering other monitors in the 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 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. Maintaining a history enables 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 configuration 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.

BOOTSTRAPPING MONITORS

In most configuration and deployment cases, tools that deploy Ceph may help bootstrap the monitors by generating a monitor map for you (e.g., `mkcephfs`, `ceph-deploy`, etc). A monitor requires four 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., `mkcephfs` or `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 `mkcephfs` or `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 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 cluster with at least three 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 monitors store data. For optimal performance in a production cluster, we recommend running monitors on separate hosts and drives from OSDs. Monitors do lots of `fsync()`, which can interfere with OSD workloads.

In Ceph versions 0.58 and earlier, 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, monitors store their data as key/value pairs. Monitors require ACID transactions. Using a data store prevents recovering 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 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`

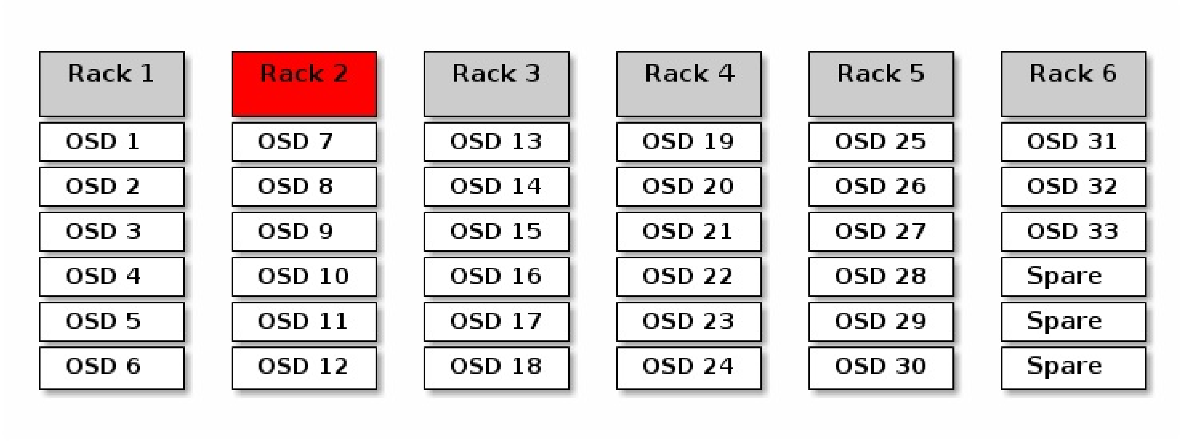
STORAGE CAPACITY

When a Ceph cluster gets close to its maximum capacity (i.e., `mon osd full ratio`), Ceph prevents you from writing to or reading from OSDs as a safety measure to prevent data loss. Therefore, letting a production 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 an OSD from the cluster to watch the cluster rebalance; then, removing another OSD, and so on until the cluster eventually reaches the full ratio and locks up. We recommend a bit of capacity planning even with a test cluster so that you can gauge how much spare capacity you will need to maintain for high availability. Ideally, you want to plan for a series of OSD failures where the cluster can recover to an `active + clean` state without replacing those OSDs 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 cluster containing 33 hosts with one OSD per host, each OSD having a 3TB capacity. So this exemplary cluster has a maximum actual capacity of 99TB. With a `mon osd full ratio` of `0.95`, if the cluster falls to 5TB of remaining capacity, the cluster will not allow Ceph clients to read and write data. So its 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

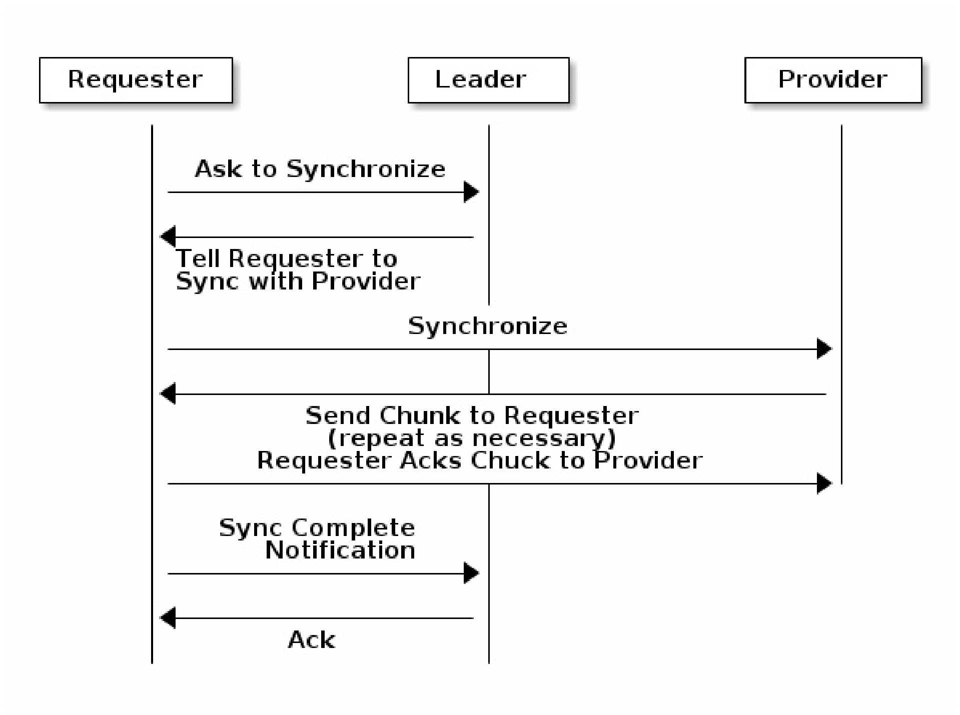
Ceph monitors know about the cluster by requiring reports from each OSD, and by receiving reports from OSDs about the status of their neighboring OSDs. Ceph provides reasonable default settings for monitor/OSD interaction; however, you may modify them as needed. See Monitor/OSD Interaction for details.

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

`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`