OSD CONFIG REFERENCE

You can configure Ceph OSD Daemons in the Ceph configuration file, but Ceph OSD Daemons can use the default values and a very minimal configuration. A minimal Ceph OSD Daemon configuration sets `osd journal size` and `host`, and uses default values for nearly everything else.

Ceph OSD Daemons are numerically identified in incremental fashion, beginning with `0` using the following convention.

```
osd.0
osd.1
osd.2
```

In a configuration file, you may specify settings for all Ceph OSD Daemons in the cluster by adding configuration settings to the `[osd]` section of your configuration file. To add settings directly to a specific Ceph OSD Daemon (e.g., `host`), enter it in an OSD-specific section of your configuration file. For example:

```
[osd]
        osd journal size = 1024

[osd.0]
        host = osd-host-a

[osd.1]
        host = osd-host-b
```

GENERAL SETTINGS

The following settings provide an Ceph OSD Daemon's ID, and determine paths to data and journals. Ceph deployment scripts typically generate the UUID automatically. We **DO NOT** recommend changing the default paths for data or journals, as it makes it more problematic to troubleshoot Ceph later.

The journal size should be at least twice the product of the expected drive speed multiplied by `filestore max sync interval`. However, the most common practice is to partition the journal drive (often an SSD), and mount it such that Ceph uses the entire partition for the journal.

`osd uuid`

- **Description:** The universally unique identifier (UUID) for the Ceph OSD Daemon.
- **Type:** UUID
- **Default:** The UUID.
- **Note:** The `osd uuid` applies to a single Ceph OSD Daemon. The `fsid` applies to the entire cluster.

`osd data`

- **Description:** The path to the OSDs data. You must create the directory when deploying Ceph. You should mount a drive for OSD data at this mount point. We do not recommend changing the default.
- **Type:** String
- **Default:** `/var/lib/ceph/osd/$cluster-$id`

`osd max write size`

- **Description:** The maximum size of a write in megabytes.
- **Type:** 32-bit Integer
- **Default:** `90`

`osd max object size`

- **Description:** The maximum size of a RADOS object in bytes.
- **Type:** 32-bit Unsigned Integer
- **Default:** 128MB

`osd client message size cap`

Description: The largest client data message allowed in memory.
Type: 64-bit Unsigned Integer
Default: 500MB default. `500*1024L*1024L`

`osd class dir`

Description: The class path for RADOS class plug-ins.
Type: String
Default: `$libdir/rados-classes`

FILE SYSTEM SETTINGS

Ceph builds and mounts file systems which are used for Ceph OSDs.

`osd mkfs options {fs-type}`

Description: Options used when creating a new Ceph OSD of type {fs-type}.
Type: String
Default for xfs:
`-f -i 2048`
Default for other file systems:
{empty string}

For example::

```
osd mkfs options xfs = -f -d agcount=24
```

`osd mount options {fs-type}`

Description: Options used when mounting a Ceph OSD of type {fs-type}.
Type: String
Default for xfs:
`rw,noatime,inode64`
Default for other file systems:
`rw, noatime`

For example::

```
osd mount options xfs = rw, noatime, inode64, logbufs=8
```

JOURNAL SETTINGS

By default, Ceph expects that you will store an Ceph OSD Daemons journal with the following path:

```
/var/lib/ceph/osd/$cluster-$id/journal
```

Without performance optimization, Ceph stores the journal on the same disk as the Ceph OSD Daemons data. An Ceph OSD Daemon optimized for performance may use a separate disk to store journal data (e.g., a solid state drive delivers high performance journaling).

Ceph's default `osd journal size` is 0, so you will need to set this in your `ceph.conf` file. A journal size should find the product of the `filestore max sync interval` and the expected throughput, and multiply the product by two (2):

```
osd journal size = {2 * (expected throughput * filestore max sync interval)}
```

The expected throughput number should include the expected disk throughput (i.e., sustained data transfer rate), and network throughput. For example, a 7200 RPM disk will likely have approximately 100 MB/s. Taking the `min()` of the disk and network throughput should provide a reasonable expected throughput. Some users just start off with a 10GB journal size. For example:

```
osd journal size = 10000
```

`osd journal`

Description: The path to the OSD's journal. This may be a path to a file or a block device (such as a partition of an SSD). If it is a file, you must create the directory to contain it. We recommend using a drive separate from the `osd data` drive.
Type: String
Default: `/var/lib/ceph/osd/$cluster-$id/journal`

`osd journal size`

Description: The size of the journal in megabytes. If this is 0, and the journal is a block device, the entire block device is used. Since v0.54, this is ignored if the journal is a block device, and the entire block device is used.
Type: 32-bit Integer
Default: `5120`
Recommended: Begin with 1GB. Should be at least twice the product of the expected speed multiplied by `filestore max sync interval`.

See Journal Config Reference for additional details.

MONITOR OSD INTERACTION

Ceph OSD Daemons check each other's heartbeats and report to monitors periodically. Ceph can use default values in many cases. However, if your network has latency issues, you may need to adopt longer intervals. See Configuring Monitor/OSD Interaction for a detailed discussion of heartbeats.

DATA PLACEMENT

See Pool & PG Config Reference for details.

SCRUBBING

In addition to making multiple copies of objects, Ceph insures data integrity by scrubbing placement groups. Ceph scrubbing is analogous to `fsck` on the object storage layer. For each placement group, Ceph generates a catalog of all objects and compares each primary object and its replicas to ensure that no objects are missing or mismatched. Light scrubbing (daily) checks the object size and attributes. Deep scrubbing (weekly) reads the data and uses checksums to ensure data integrity.

Scrubbing is important for maintaining data integrity, but it can reduce performance. You can adjust the following settings to increase or decrease scrubbing operations.

`osd max scrubs`

Description: The maximum number of simultaneous scrub operations for a Ceph OSD Daemon.
Type: 32-bit Int
Default: `1`

`osd scrub begin hour`

Description: The time of day for the lower bound when a scheduled scrub can be performed.
Type: Integer in the range of 0 to 24
Default: `0`

`osd scrub end hour`

Description: The time of day for the upper bound when a scheduled scrub can be performed. Along with `osd scrub begin hour`, they define a time window, in which the scrubs can happen. But a scrub will be performed no matter the time window allows or not, as long as the placement group's scrub interval exceeds `osd scrub max interval`.
Type: Integer in the range of 0 to 24
Default: 24

`osd scrub during recovery`

Description: Allow scrub during recovery. Setting this to `false` will disable scheduling new scrub (and deep–scrub) while there is active recovery. Already running scrubs will be continued. This might be useful to reduce

load on busy clusters.

Type: Boolean
Default: `true`

`osd scrub thread timeout`

Description: The maximum time in seconds before timing out a scrub thread.
Type: 32-bit Integer
Default: `60`

`osd scrub finalize thread timeout`

Description: The maximum time in seconds before timing out a scrub finalize thread.
Type: 32-bit Integer
Default: `60*10`

`osd scrub load threshold`

Description: The maximum load. Ceph will not scrub when the system load (as defined by `getloadavg()`) is higher than this number. Default is `0.5`.
Type: Float
Default: `0.5`

`osd scrub min interval`

Description: The minimal interval in seconds for scrubbing the Ceph OSD Daemon when the Ceph Storage Cluster load is low.
Type: Float
Default: Once per day. `60*60*24`

`osd scrub max interval`

Description: The maximum interval in seconds for scrubbing the Ceph OSD Daemon irrespective of cluster load.
Type: Float
Default: Once per week. `7*60*60*24`

`osd scrub chunk min`

Description: The minimal number of object store chunks to scrub during single operation. Ceph blocks writes to single chunk during scrub.
Type: 32-bit Integer
Default: 5

`osd scrub chunk max`

Description: The maximum number of object store chunks to scrub during single operation.
Type: 32-bit Integer
Default: 25

`osd scrub sleep`

Description: Time to sleep before scrubbing next group of chunks. Increasing this value will slow down whole scrub operation while client operations will be less impacted.
Type: Float
Default: 0

`osd deep scrub interval`

Description: The interval for “deep” scrubbing (fully reading all data). The `osd scrub load threshold` does not affect this setting.
Type: Float
Default: Once per week. `60*60*24*7`

`osd scrub interval randomize ratio`

Description: Add a random delay to `osd scrub min interval` when scheduling the next scrub job for a placement group. The delay is a random value less than `osd scrub min interval` * `osd scrub interval randomized ratio`. So the default setting practically randomly spreads the scrubs out in the allowed time window of `[1, 1.5]` * `osd scrub min interval`.
Type: Float
Default: `0.5`

`osd deep scrub stride`

Description: Read size when doing a deep scrub.
Type: 32-bit Integer
Default: 512 KB. 524288

OPERATIONS

Operations settings allow you to configure the number of threads for servicing requests. If you set `osd op threads` to `0`, it disables multi-threading. By default, Ceph uses two threads with a 30 second timeout and a 30 second complaint time if an operation doesn't complete within those time parameters. You can set operations priority weights between client operations and recovery operations to ensure optimal performance during recovery.

`osd op threads`

Description: The number of threads to service Ceph OSD Daemon operations. Set to `0` to disable it. Increasing the number may increase the request processing rate.
Type: 32-bit Integer
Default: 2

`osd op queue`

Description: This sets the type of queue to be used for prioritizing ops in the OSDs. Both queues feature a strict sub-queue which is dequeued before the normal queue. The normal queue is different between implementations. The original PrioritizedQueue (`prio`) uses a token bucket system which when there are sufficient tokens will dequeue high priority queues first. If there are not enough tokens available, queues are dequeued low priority to high priority. The WeightedPriorityQueue (`wpq`) dequeues all priorities in relation to their priorities to prevent starvation of any queue. WPQ should help in cases where a few OSDs are more overloaded than others. The new mClock based OpClassQueue (`mclock_opclass`) prioritizes operations based on which class they belong to (recovery, scrub, snaptrim, client op, osd subop). And, the mClock based ClientQueue (`mclock_client`) also incorporates the client identifier in order to promote fairness between clients. See QoS Based on mClock. Requires a restart.
Type: String
Valid Choices: prio, wpq, mclock_opclass, mclock_client
Default: `prio`

`osd op queue cut off`

Description: This selects which priority ops will be sent to the strict queue verses the normal queue. The `low` setting sends all replication ops and higher to the strict queue, while the `high` option sends only replication acknowledgement ops and higher to the strict queue. Setting this to `high` should help when a few OSDs in the cluster are very busy especially when combined with `wpq` in the `osd op queue` setting. OSDs that are very busy handling replication traffic could starve primary client traffic on these OSDs without these settings. Requires a restart.
Type: String
Valid Choices: low, high
Default: `low`

`osd client op priority`

Description: The priority set for client operations. It is relative to `osd recovery op priority`.
Type: 32-bit Integer
Default: 63
Valid Range: 1-63

`osd recovery op priority`

Description: The priority set for recovery operations. It is relative to `osd client op priority`.
Type: 32-bit Integer
Default: 3
Valid Range: 1-63

`osd scrub priority`

Description: The priority set for scrub operations. It is relative to `osd client op priority`.
Type: 32-bit Integer
Default: 5
Valid Range: 1-63

`osd snap trim priority`

Description: The priority set for snap trim operations. It is relative to `osd client op priority`.
Type: 32-bit Integer
Default: 5
Valid Range: 1-63

`osd op thread timeout`

Description: The Ceph OSD Daemon operation thread timeout in seconds.
Type: 32-bit Integer
Default: 15

`osd op complaint time`

Description: An operation becomes complaint worthy after the specified number of seconds have elapsed.
Type: Float
Default: `30`

`osd disk threads`

Description: The number of disk threads, which are used to perform background disk intensive OSD operations such as scrubbing and snap trimming.
Type: 32-bit Integer
Default: 1

`osd disk thread ioprio class`

Description: Warning: it will only be used if both `osd disk thread ioprio class` and `osd disk thread ioprio priority` are set to a non default value. Sets the ioprio_set(2) I/O scheduling `class` for the disk thread. Acceptable values are `idle`, be or `rt`. The `idle` class means the disk thread will have lower priority than any other thread in the OSD. This is useful to slow down scrubbing on an OSD that is busy handling client operations. be is the default and is the same priority as all other threads in the OSD. `rt` means the disk thread will have precendence over all other threads in the OSD. Note: Only works with the Linux Kernel CFQ scheduler. Since Jewel scrubbing is no longer carried out by the disk iothread, see osd priority options instead.
Type: String
Default: the empty string

`osd disk thread ioprio priority`

Description: Warning: it will only be used if both `osd disk thread ioprio class` and `osd disk thread ioprio priority` are set to a non default value. It sets the ioprio_set(2) I/O scheduling `priority` of the disk thread ranging from 0 (highest) to 7 (lowest). If all OSDs on a given host were in class `idle` and compete for I/O (i.e. due to controller congestion), it can be used to lower the disk thread priority of one OSD to 7 so that another OSD with priority 0 can have priority. Note: Only works with the Linux Kernel CFQ scheduler.
Type: Integer in the range of 0 to 7 or -1 if not to be used.
Default: `-1`

`osd op history size`

Description: The maximum number of completed operations to track.
Type: 32-bit Unsigned Integer

Default: 20

`osd op history duration`

Description: The oldest completed operation to track.
Type: 32-bit Unsigned Integer
Default: 600

`osd op log threshold`

Description: How many operations logs to display at once.
Type: 32-bit Integer
Default: 5

QOS BASED ON MCLOCK

Ceph's use of mClock is currently in the experimental phase and should be approached with an exploratory mindset.

CORE CONCEPTS

The QoS support of Ceph is implemented using a queueing scheduler based on the dmClock algorithm. This algorithm allocates the I/O resources of the Ceph cluster in proportion to weights, and enforces the constraits of minimum reservation and maximum limitation, so that the services can compete for the resources fairly. Currently the *mclock_opclass* operation queue divides Ceph services involving I/O resources into following buckets:

- client op: the iops issued by client
- osd subop: the iops issued by primary OSD
- snap trim: the snap trimming related requests
- pg recovery: the recovery related requests
- pg scrub: the scrub related requests

And the resources are partitioned using following three sets of tags. In other words, the share of each type of service is controlled by three tags:

1. reservation: the minimum IOPS allocated for the service.
2. limitation: the maximum IOPS allocated for the service.
3. weight: the proportional share of capacity if extra capacity or system oversubscribed.

In Ceph operations are graded with "cost". And the resources allocated for serving various services are consumed by these "costs". So, for example, the more reservation a services has, the more resource it is guaranteed to possess, as long as it requires. Assuming there are 2 services: recovery and client ops:

- recovery: (r:1, l:5, w:1)
- client ops: (r:2, l:0, w:9)

The settings above ensure that the recovery won't get more than 5 requests per second serviced, even if it requires so (see CURRENT IMPLEMENTATION NOTE below), and no other services are competing with it. But if the clients start to issue large amount of I/O requests, neither will they exhaust all the I/O resources. 1 request per second is always allocated for recovery jobs as long as there are any such requests. So the recovery jobs won't be starved even in a cluster with high load. And in the meantime, the client ops can enjoy a larger portion of the I/O resource, because its weight is "9", while its competitor "1". In the case of client ops, it is not clamped by the limit setting, so it can make use of all the resources if there is no recovery ongoing.

Along with *mclock_opclass* another mclock operation queue named *mclock_client* is available. It divides operations based on category but also divides them based on the client making the request. This helps not only manage the distribution of resources spent on different classes of operations but also tries to insure fairness among clients.

CURRENT IMPLEMENTATION NOTE: the current experimental implementation does not enforce the limit values. As a first approximation we decided not to prevent operations that would otherwise enter the operation sequencer from doing so.

SUBTLETIES OF MCLOCK

The reservation and limit values have a unit of requests per second. The weight, however, does not technically have a unit and the weights are relative to one another. So if one class of requests has a weight of 1 and another a weight of 9, then the latter class of requests should get 9 executed at a 9 to 1 ratio as the first class. However that will only happen once the reservations are met and those values include the operations executed under the reservation phase.

Even though the weights do not have units, one must be careful in choosing their values due how the algorithm assigns weight tags to requests. If the weight is *W*, then for a given class of requests, the next one that comes in will have a weight tag of *1/W* plus the previous weight tag or the current time, whichever is larger. That means if *W* is sufficiently large and therefore *1/W* is sufficiently small, the calculated tag may never be assigned as it will get a value of the current time. The ultimate lesson is that values for weight should not be too large. They should be under the number of requests one expects to ve serviced each second.

CAVEATS

There are some factors that can reduce the impact of the mClock op queues within Ceph. First, requests to an OSD are sharded by their placement group identifier. Each shard has its own mClock queue and these queues neither interact nor share information among them. The number of shards can be controlled with the configuration options `osd_op_num_shards`, `osd_op_num_shards_hdd`, and `osd_op_num_shards_ssd`. A lower number of shards will increase the impact of the mClock queues, but may have other deliterious effects.

Second, requests are transferred from the operation queue to the operation sequencer, in which they go through the phases of execution. The operation queue is where mClock resides and mClock determines the next op to transfer to the operation sequencer. The number of operations allowed in the operation sequencer is a complex issue. In general we want to keep enough operations in the sequencer so it's always getting work done on some operations while it's waiting for disk and network access to complete on other operations. On the other hand, once an operation is transferred to the operation sequencer, mClock no longer has control over it. Therefore to maximize the impact of mClock, we want to keep as few operations in the operation sequencer as possible. So we have an inherent tension.

The configuration options that influence the number of operations in the operation sequencer are `bluestore_throttle_bytes`, `bluestore_throttle_deferred_bytes`, `bluestore_throttle_cost_per_io`, `bluestore_throttle_cost_per_io_hdd`, and `bluestore_throttle_cost_per_io_ssd`.

A third factor that affects the impact of the mClock algorithm is that we're using a distributed system, where requests are made to multiple OSDs and each OSD has (can have) multiple shards. Yet we're currently using the mClock algorithm, which is not distributed (note: dmClock is the distributed version of mClock).

Various organizations and individuals are currently experimenting with mClock as it exists in this code base along with their modifications to the code base. We hope you'll share you're experiences with your mClock and dmClock experiments in the ceph-devel mailing list.

`osd push per object cost`

Description: the overhead for serving a push op
Type: Unsigned Integer
Default: 1000

`osd recovery max chunk`

Description: the maximum total size of data chunks a recovery op can carry.
Type: Unsigned Integer
Default: 8 MiB

`osd op queue mclock client op res`

Description: the reservation of client op.
Type: Float
Default: 1000.0

`osd op queue mclock client op wgt`

Description: the weight of client op.
Type: Float
Default: 500.0

`osd op queue mclock client op lim`

Description: the limit of client op.
Type: Float
Default: 1000.0

`osd op queue mclock osd subop res`

Description: the reservation of osd subop.
Type: Float
Default: 1000.0

`osd op queue mclock osd subop wgt`

Description: the weight of osd subop.
Type: Float
Default: 500.0

`osd op queue mclock osd subop lim`

Description: the limit of osd subop.
Type: Float
Default: 0.0

`osd op queue mclock snap res`

Description: the reservation of snap trimming.
Type: Float
Default: 0.0

`osd op queue mclock snap wgt`

Description: the weight of snap trimming.
Type: Float
Default: 1.0

`osd op queue mclock snap lim`

Description: the limit of snap trimming.
Type: Float
Default: 0.001

`osd op queue mclock recov res`

Description: the reservation of recovery.
Type: Float
Default: 0.0

`osd op queue mclock recov wgt`

Description: the weight of recovery.
Type: Float
Default: 1.0

`osd op queue mclock recov lim`

Description: the limit of recovery.
Type: Float
Default: 0.001

`osd op queue mclock scrub res`

Description: the reservation of scrub jobs.
Type: Float
Default: 0.0

`osd op queue mclock scrub wgt`

Description: the weight of scrub jobs.
Type: Float
Default: 1.0

`osd op queue mclock scrub lim`

Description: the limit of scrub jobs.
Type: Float
Default: 0.001

BACKFILLING

When you add or remove Ceph OSD Daemons to a cluster, the CRUSH algorithm will want to rebalance the cluster by moving placement groups to or from Ceph OSD Daemons to restore the balance. The process of migrating placement groups and the objects they contain can reduce the cluster's operational performance considerably. To maintain operational performance, Ceph performs this migration with 'backfilling', which allows Ceph to set backfill operations to a lower priority than requests to read or write data.

`osd max backfills`

Description: The maximum number of backfills allowed to or from a single OSD.
Type: 64-bit Unsigned Integer
Default: 1

`osd backfill scan min`

Description: The minimum number of objects per backfill scan.
Type: 32-bit Integer
Default: 64

`osd backfill scan max`

Description: The maximum number of objects per backfill scan.
Type: 32-bit Integer
Default: 512

`osd backfill retry interval`

Description: The number of seconds to wait before retrying backfill requests.
Type: Double
Default: 10.0

OSD MAP

OSD maps reflect the OSD daemons operating in the cluster. Over time, the number of map epochs increases. Ceph provides some settings to ensure that Ceph performs well as the OSD map grows larger.

`osd map dedup`

Description: Enable removing duplicates in the OSD map.
Type: Boolean
Default: true

`osd map cache size`

Description: The number of OSD maps to keep cached.
Type: 32-bit Integer
Default: 500

`osd map cache bl size`

Description: The size of the in-memory OSD map cache in OSD daemons.
Type: 32-bit Integer
Default: 50

`osd map cache bl inc size`

Description: The size of the in-memory OSD map cache incrementals in OSD daemons.
Type: 32-bit Integer

Default: 100

`osd map message max`

Description: The maximum map entries allowed per MOSDMap message.
Type: 32-bit Integer
Default: 100

RECOVERY

When the cluster starts or when a Ceph OSD Daemon crashes and restarts, the OSD begins peering with other Ceph OSD Daemons before writes can occur. See Monitoring OSDs and PGs for details.

If a Ceph OSD Daemon crashes and comes back online, usually it will be out of sync with other Ceph OSD Daemons containing more recent versions of objects in the placement groups. When this happens, the Ceph OSD Daemon goes into recovery mode and seeks to get the latest copy of the data and bring its map back up to date. Depending upon how long the Ceph OSD Daemon was down, the OSD's objects and placement groups may be significantly out of date. Also, if a failure domain went down (e.g., a rack), more than one Ceph OSD Daemon may come back online at the same time. This can make the recovery process time consuming and resource intensive.

To maintain operational performance, Ceph performs recovery with limitations on the number recovery requests, threads and object chunk sizes which allows Ceph perform well in a degraded state.

`osd recovery delay start`

Description: After peering completes, Ceph will delay for the specified number of seconds before starting to recover objects.
Type: Float
Default: 0

`osd recovery max active`

Description: The number of active recovery requests per OSD at one time. More requests will accelerate recovery, but the requests places an increased load on the cluster.
Type: 32-bit Integer
Default: 3

`osd recovery max chunk`

Description: The maximum size of a recovered chunk of data to push.
Type: 64-bit Unsigned Integer
Default: 8 << 20

`osd recovery max single start`

Description: The maximum number of recovery operations per OSD that will be newly started when an OSD is recovering.
Type: 64-bit Unsigned Integer
Default: 1

`osd recovery thread timeout`

Description: The maximum time in seconds before timing out a recovery thread.
Type: 32-bit Integer
Default: 30

`osd recover clone overlap`

Description: Preserves clone overlap during recovery. Should always be set to `true`.
Type: Boolean
Default: `true`

`osd recovery sleep`

Description: Time in seconds to sleep before next recovery or backfill op. Increasing this value will slow down

recovery operation while client operations will be less impacted.
Type: Float
Default: `0`

`osd recovery sleep hdd`

Description: Time in seconds to sleep before next recovery or backfill op for HDDs.
Type: Float
Default: `0.1`

`osd recovery sleep ssd`

Description: Time in seconds to sleep before next recovery or backfill op for SSDs.
Type: Float
Default: `0`

`osd recovery sleep hybrid`

Description: Time in seconds to sleep before next recovery or backfill op when osd data is on HDD and osd journal is on SSD.
Type: Float
Default: `0.025`

TIERING

`osd agent max ops`

Description: The maximum number of simultaneous flushing ops per tiering agent in the high speed mode.
Type: 32-bit Integer
Default: `4`

`osd agent max low ops`

Description: The maximum number of simultaneous flushing ops per tiering agent in the low speed mode.
Type: 32-bit Integer
Default: `2`

See cache target dirty high ratio for when the tiering agent flushes dirty objects within the high speed mode.

MISCELLANEOUS

`osd snap trim thread timeout`

Description: The maximum time in seconds before timing out a snap trim thread.
Type: 32-bit Integer
Default: `60*60*1`

`osd backlog thread timeout`

Description: The maximum time in seconds before timing out a backlog thread.
Type: 32-bit Integer
Default: `60*60*1`

`osd default notify timeout`

Description: The OSD default notification timeout (in seconds).
Type: 32-bit Unsigned Integer
Default: `30`

`osd check for log corruption`

Description: Check log files for corruption. Can be computationally expensive.

Type: Boolean
Default: `false`

`osd remove thread timeout`

Description: The maximum time in seconds before timing out a remove OSD thread.
Type: 32-bit Integer
Default: `60*60`

`osd command thread timeout`

Description: The maximum time in seconds before timing out a command thread.
Type: 32-bit Integer
Default: `10*60`

`osd command max records`

Description: Limits the number of lost objects to return.
Type: 32-bit Integer
Default: `256`

`osd auto upgrade tmap`

Description: Uses `tmap` for `omap` on old objects.
Type: Boolean
Default: `true`

`osd tmapput sets users tmap`

Description: Uses `tmap` for debugging only.
Type: Boolean
Default: `false`

`osd fast fail on connection refused`

Description: If this option is enabled, crashed OSDs are marked down immediately by connected peers and MONs (assuming that the crashed OSD host survives). Disable it to restore old behavior, at the expense of possible long I/O stalls when OSDs crash in the middle of I/O operations.
Type: Boolean
Default: `true`