CEPH FILESYSTEM CLIENT EVICTION

When a filesystem client is unresponsive or otherwise misbehaving, it may be necessary to forcibly terminate its access to the filesystem. This process is called *eviction*.

Evicting a CephFS client prevents it from communicating further with MDS daemons and OSD daemons. If a client was doing buffered IO to the filesystem, any un-flushed data will be lost.

Clients may either be evicted automatically (if they fail to communicate promptly with the MDS), or manually (by the system administrator).

The client eviction process applies to clients of all kinds, this includes FUSE mounts, kernel mounts, nfs-ganesha gateways, and any process using libcephfs.

AUTOMATIC CLIENT EVICTION

There are two situations in which a client may be evicted automatically:

On an active MDS daemon, if a client has not communicated with the MDS for over `session_autoclose` (a file system variable) seconds (300 seconds by default), then it will be evicted automatically.

During MDS startup (including on failover), the MDS passes through a state called `reconnect`. During this state, it waits for all the clients to connect to the new MDS daemon. If any clients fail to do so within the time window (`mds_reconnect_timeout`, 45 seconds by default) then they will be evicted.

A warning message is sent to the cluster log if either of these situations arises.

MANUAL CLIENT EVICTION

Sometimes, the administrator may want to evict a client manually. This could happen if a client is died and the administrator does not want to wait for its session to time out, or it could happen if a client is misbehaving and the administrator does not have access to the client node to unmount it.

It is useful to inspect the list of clients first:

```
ceph tell mds.0 client ls

[
    {
        "id": 4305,
        "num_leases": 0,
        "num_caps": 3,
        "state": "open",
        "replay_requests": 0,
        "completed_requests": 0,
        "reconnecting": false,
        "inst": "client.4305 172.21.9.34:0/422650892",
        "client_metadata": {
            "ceph_sha1": "ae81e49d369875ac8b569ff3e3c456a31b8f3af5",
            "ceph_version": "ceph version 12.0.0-1934-gae81e49 (ae81e49d369875ac8b569ff3e3c45
            "entity_id": "0",
            "hostname": "senta04",
            "mount_point": "/tmp/tmpcMpF1b/mnt.0",
            "pid": "29377",
            "root": "/"
        }
    }
]
```

Once you have identified the client you want to evict, you can do that using its unique ID, or various other attributes to identify it:

```
# These all work
ceph tell mds.0 client evict id=4305
```

```
ceph tell mds.0 client evict client_metadata.=4305
```

ADVANCED: UN-BLACKLISTING A CLIENT

Ordinarily, a blacklisted client may not reconnect to the servers: it must be unmounted and then mounted anew.

However, in some situations it may be useful to permit a client that was evicted to attempt to reconnect.

Because CephFS uses the RADOS OSD blacklist to control client eviction, CephFS clients can be permitted to reconnect by removing them from the blacklist:

```
ceph osd blacklist ls
# ... identify the address of the client ...
ceph osd blacklist rm <address>
```

Doing this may put data integrity at risk if other clients have accessed files that the blacklisted client was doing buffered IO to. It is also not guaranteed to result in a fully functional client – the best way to get a fully healthy client back after an eviction is to unmount the client and do a fresh mount.

If you are trying to reconnect clients in this way, you may also find it useful to set `client_reconnect_stale` to true in the FUSE client, to prompt the client to try to reconnect.

ADVANCED: CONFIGURING BLACKLISTING

If you are experiencing frequent client evictions, due to slow client hosts or an unreliable network, and you cannot fix the underlying issue, then you may want to ask the MDS to be less strict.

It is possible to respond to slow clients by simply dropping their MDS sessions, but permit them to re-open sessions and permit them to continue talking to OSDs. To enable this mode, set `mds_session_blacklist_on_timeout` to false on your MDS nodes.

For the equivalent behaviour on manual evictions, set `mds_session_blacklist_on_evict` to false.

Note that if blacklisting is disabled, then evicting a client will only have an effect on the MDS you send the command to. On a system with multiple active MDS daemons, you would need to send an eviction command to each active daemon. When blacklisting is enabled (the default), sending an eviction command to just a single MDS is sufficient, because the blacklist propagates it to the others.

BACKGROUND: BLACKLISTING AND OSD EPOCH BARRIER

After a client is blacklisted, it is necessary to make sure that other clients and MDS daemons have the latest OSDMap (including the blacklist entry) before they try to access any data objects that the blacklisted client might have been accessing.

This is ensured using an internal “osdmap epoch barrier” mechanism.

The purpose of the barrier is to ensure that when we hand out any capabilities which might allow touching the same RADOS objects, the clients we hand out the capabilities to must have a sufficiently recent OSD map to not race with cancelled operations (from ENOSPC) or blacklisted clients (from evictions).

More specifically, the cases where an epoch barrier is set are:

- Client eviction (where the client is blacklisted and other clients must wait for a post-blacklist epoch to touch the same objects).
- OSD map full flag handling in the client (where the client may cancel some OSD ops from a pre-full epoch, so other clients must wait until the full epoch or later before touching the same objects).
- MDS startup, because we don’t persist the barrier epoch, so must assume that latest OSD map is always required after a restart.

Note that this is a global value for simplicity. We could maintain this on a per-inode basis. But we don’t, because:

- It would be more complicated.
- It would use an extra 4 bytes of memory for every inode.
- It would not be much more efficient as almost always everyone has the latest. OSD map anyway, in most cases everyone will breeze through this barrier rather than waiting.
- This barrier is done in very rare cases, so any benefit from per-inode granularity would only very rarely be

seen.

The epoch barrier is transmitted along with all capability messages, and instructs the receiver of the message to avoid sending any more RADOS operations to OSDs until it has seen this OSD epoch. This mainly applies to clients (doing their data writes directly to files), but also applies to the MDS because things like file size probing and file deletion are done directly from the MDS.