FAQ

Frequently asked questions

etcd, general

What is etcd?

etcd is a consistent distributed key-value store. Mainly used as a separate coordination service, in distributed systems. And designed to hold small amounts of data that can fit entirely in memory.

How do you pronounce etcd?

etcd is pronounced **/'ɛtsi:di:/**, and means "distributed `etc` directory."

Do clients have to send requests to the etcd leader?

Raft is leader-based; the leader handles all client requests which need cluster consensus. However, the client does not need to know which node is the leader. Any request that requires consensus sent to a follower is automatically forwarded to the leader. Requests that do not require consensus (e.g., serialized reads) can be processed by any cluster member.

Configuration

What is the difference between listen-<client,peer>-urls, advertise-client-urls or initial-advertise-peer-urls?

`listen-client-urls` and `listen-peer-urls` specify the local addresses etcd server binds to for accepting incoming connections. To listen on a port for all interfaces, specify `0.0.0.0` as the listen IP address.

`advertise-client-urls` and `initial-advertise-peer-urls` specify the addresses etcd clients or other etcd members should use to contact the etcd server. The advertise addresses must be reachable from the remote machines. Do not advertise addresses like `localhost` or `0.0.0.0` for a production setup since these addresses are unreachable from remote machines.

Why doesn't changing `--listen-peer-urls` or `--initial-advertise-peer-urls` update the advertised peer URLs in `etcdctl member list`?

A member's advertised peer URLs come from `--initial-advertise-peer-urls` on initial cluster boot. Changing the listen peer URLs or the initial advertise peers after booting the member won't affect the exported advertise peer URLs since changes must go through quorum to avoid membership configuration split brain. Use `etcdctl member update` to update a member's peer URLs.

Deployment

System requirements

Since etcd writes data to disk, its performance strongly depends on disk performance. For this reason, SSD is highly recommended. To assess whether a disk is fast enough for etcd, one possibility is using a disk benchmarking tool such as fio. For an example on how to do that, read here. To prevent performance degradation or unintentionally overloading the key-value store, etcd enforces a configurable storage size quota set to 2GB by default. To avoid swapping or running out of memory, the machine should have at least as much RAM to cover the quota. 8GB is a suggested maximum size for normal environments and etcd warns at startup if the configured value exceeds it. At CoreOS, an etcd cluster is usually deployed on dedicated CoreOS Container Linux machines with dual-core processors, 2GB of RAM, and 80GB of SSD *at the very least*. **Note that performance is intrinsically workload dependent; please test before production deployment**. See hardware for more recommendations.

Most stable production environment is Linux operating system with amd64 architecture; see supported platform for more.

Why an odd number of cluster members?

An etcd cluster needs a majority of nodes, a quorum, to agree on updates to the cluster state. For a cluster with n members, quorum is (n/2)+1. For any odd-sized cluster, adding one node

will always increase the number of nodes necessary for quorum. Although adding a node to an odd-sized cluster appears better since there are more machines, the fault tolerance is worse since exactly the same number of nodes may fail without losing quorum but there are more nodes that can fail. If the cluster is in a state where it can't tolerate any more failures, adding a node before removing nodes is dangerous because if the new node fails to register with the cluster (e.g., the address is misconfigured), quorum will be permanently lost.

What is maximum cluster size?

Theoretically, there is no hard limit. However, an etcd cluster probably should have no more than seven nodes. Google Chubby lock service, similar to etcd and widely deployed within Google for many years, suggests running five nodes. A 5-member etcd cluster can tolerate two member failures, which is enough in most cases. Although larger clusters provide better fault tolerance, the write performance suffers because data must be replicated across more machines.

What is failure tolerance?

An etcd cluster operates so long as a member quorum can be established. If quorum is lost through transient network failures (e.g., partitions), etcd automatically and safely resumes once the network recovers and restores quorum; Raft enforces cluster consistency. For power loss, etcd persists the Raft log to disk; etcd replays the log to the point of failure and resumes cluster participation. For permanent hardware failure, the node may be removed from the cluster through runtime reconfiguration.

It is recommended to have an odd number of members in a cluster. An odd-size cluster tolerates the same number of failures as an even-size cluster but with fewer nodes. The difference can be seen by comparing even and odd sized clusters:

Cluster Size	Majority	Failure Tolerance
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2

Cluster Size	Majority	Failure Tolerance
7	4	3
8	5	3
9	5	4

Adding a member to bring the size of cluster up to an even number doesn't buy additional fault tolerance. Likewise, during a network partition, an odd number of members guarantees that there will always be a majority partition that can continue to operate and be the source of truth when the partition ends.

Does etcd work in cross-region or cross data center deployments?

Deploying etcd across regions improves etcd's fault tolerance since members are in separate failure domains. The cost is higher consensus request latency from crossing data center boundaries. Since etcd relies on a member quorum for consensus, the latency from crossing data centers will be somewhat pronounced because at least a majority of cluster members must respond to consensus requests. Additionally, cluster data must be replicated across all peers, so there will be bandwidth cost as well.

With longer latencies, the default etcd configuration may cause frequent elections or heartbeat timeouts. See tuning for adjusting timeouts for high latency deployments.

Operation

How to backup a etcd cluster?

etcdctl provides a `snapshot` command to create backups. See backup for more details.

Should I add a member before removing an unhealthy member?

When replacing an etcd node, it's important to remove the member first and then add its replacement.

etcd employs distributed consensus based on a quorum model; (n/2)+1 members, a majority, must agree on a proposal before it can be committed to the cluster. These proposals include key-value updates and membership changes. This model totally avoids any possibility of split brain inconsistency. The downside is permanent quorum loss is catastrophic.

How this applies to membership: If a 3-member cluster has 1 downed member, it can still make forward progress because the quorum is 2 and 2 members are still live. However, adding a new member to a 3-member cluster will increase the quorum to 3 because 3 votes are required for a majority of 4 members. Since the quorum increased, this extra member buys nothing in terms of fault tolerance; the cluster is still one node failure away from being unrecoverable.

Additionally, that new member is risky because it may turn out to be misconfigured or incapable of joining the cluster. In that case, there's no way to recover quorum because the cluster has two members down and two members up, but needs three votes to change membership to undo the botched membership addition. etcd will by default reject member add attempts that could take down the cluster in this manner.

On the other hand, if the downed member is removed from cluster membership first, the number of members becomes 2 and the quorum remains at 2. Following that removal by adding a new member will also keep the quorum steady at 2. So, even if the new node can't be brought up, it's still possible to remove the new member through quorum on the remaining live members.

Why won't etcd accept my membership changes?

etcd sets `strict-reconfig-check` in order to reject reconfiguration requests that would cause quorum loss. Abandoning quorum is really risky (especially when the cluster is already unhealthy). Although it may be tempting to disable quorum checking if there's quorum loss to add a new member, this could lead to full fledged cluster inconsistency. For many applications, this will make the problem even worse ("disk geometry corruption" being a candidate for most terrifying).

Why does etcd lose its leader from disk latency spikes?

This is intentional; disk latency is part of leader liveness. Suppose the cluster leader takes a minute to fsync a raft log update to disk, but the etcd cluster has a one second election timeout. Even though the leader can process network messages within the election interval (e.g., send heartbeats), it's effectively unavailable because it can't commit any new proposals; it's waiting on the slow disk. If the cluster frequently loses its leader due to disk latencies, try tuning the disk settings or etcd time parameters.

What does the etcd warning “request ignored (cluster ID mismatch)” mean?

Every new etcd cluster generates a new cluster ID based on the initial cluster configuration and a user-provided unique `initial-cluster-token` value. By having unique cluster ID’s, etcd is protected from cross-cluster interaction which could corrupt the cluster.

Usually this warning happens after tearing down an old cluster, then reusing some of the peer addresses for the new cluster. If any etcd process from the old cluster is still running it will try to contact the new cluster. The new cluster will recognize a cluster ID mismatch, then ignore the request and emit this warning. This warning is often cleared by ensuring peer addresses among distinct clusters are disjoint.

What does “mvcc: database space exceeded” mean and how do I fix it?

The multi-version concurrency control data model in etcd keeps an exact history of the keyspace. Without periodically compacting this history (e.g., by setting `--auto-compaction`), etcd will eventually exhaust its storage space. If etcd runs low on storage space, it raises a space quota alarm to protect the cluster from further writes. So long as the alarm is raised, etcd responds to write requests with the error `mvcc: database space exceeded`.

To recover from the low space quota alarm:

1. Compact etcd’s history.
2. Defragment every etcd endpoint.
3. Disarm the alarm.

What does the etcd warning “etcdserver/api/v3rpc: transport: http2Server.HandleStreams failed to read frame: read tcp 127.0.0.1:2379->127.0.0.1:43020: read: connection reset by peer” mean?

This is gRPC-side warning when a server receives a TCP RST flag with client-side streams being prematurely closed. For example, a client closes its connection, while gRPC server has not yet processed all HTTP/2 frames in the TCP queue. Some data may have been lost in server side, but it is ok so long as client connection has already been closed.

Only old versions of gRPC log this. etcd >=v3.2.13 by default log this with DEBUG level, thus only visible with `--log-level=debug` flag enabled.

Performance

How should I benchmark etcd?

Try the benchmark tool. Current benchmark results are available for comparison.

What does the etcd warning “apply entries took too long” mean?

After a majority of etcd members agree to commit a request, each etcd server applies the request to its data store and persists the result to disk. Even with a slow mechanical disk or a virtualized network disk, such as Amazon’s EBS or Google’s PD, applying a request should normally take fewer than 50 milliseconds. If the average apply duration exceeds 100 milliseconds, etcd will warn that entries are taking too long to apply.

Usually this issue is caused by a slow disk. The disk could be experiencing contention among etcd and other applications, or the disk is too simply slow (e.g., a shared virtualized disk). To rule out a slow disk from causing this warning, monitor backend_commit_duration_seconds (p99 duration should be less than 25ms) to confirm the disk is reasonably fast. If the disk is too slow, assigning a dedicated disk to etcd or using faster disk will typically solve the problem.

The second most common cause is CPU starvation. If monitoring of the machine’s CPU usage shows heavy utilization, there may not be enough compute capacity for etcd. Moving etcd to dedicated machine, increasing process resource isolation cgroups, or renicing the etcd server process into a higher priority can usually solve the problem.

Expensive user requests which access too many keys (e.g., fetching the entire keyspace) can also cause long apply latencies. Accessing fewer than a several hundred keys per request, however, should always be performant.

If none of the above suggestions clear the warnings, please open an issue with detailed logging, monitoring, metrics and optionally workload information.

What does the etcd warning “failed to send out heartbeat on time” mean?

etcd uses a leader-based consensus protocol for consistent data replication and log execution. Cluster members elect a single leader, all other members become followers. The elected leader must periodically send heartbeats to its followers to maintain its leadership. Followers infer leader failure if no heartbeats are received within an election interval and trigger an election. If a leader doesn't send its heartbeats in time but is still running, the election is spurious and likely caused by insufficient resources. To catch these soft failures, if the leader skips two heartbeat intervals, etcd will warn it failed to send a heartbeat on time.

Usually this issue is caused by a slow disk. Before the leader sends heartbeats attached with metadata, it may need to persist the metadata to disk. The disk could be experiencing contention among etcd and other applications, or the disk is too simply slow (e.g., a shared virtualized disk). To rule out a slow disk from causing this warning, monitor wal_fsync_duration_seconds (p99 duration should be less than 10ms) to confirm the disk is reasonably fast. If the disk is too slow, assigning a dedicated disk to etcd or using faster disk will typically solve the problem. To tell whether a disk is fast enough for etcd, a benchmarking tool such as fio can be used. Read here for an example.

The second most common cause is CPU starvation. If monitoring of the machine's CPU usage shows heavy utilization, there may not be enough compute capacity for etcd. Moving etcd to dedicated machine, increasing process resource isolation with cgroups, or renicing the etcd server process into a higher priority can usually solve the problem.

A slow network can also cause this issue. If network metrics among the etcd machines shows long latencies or high drop rate, there may not be enough network capacity for etcd. Moving etcd members to a less congested network will typically solve the problem. However, if the etcd cluster is deployed across data centers, long latency between members is expected. For such deployments, tune the `heartbeat-interval` configuration to roughly match the round trip time between the machines, and the `election-timeout` configuration to be at least 5 * `heartbeat-interval`. See tuning documentation for detailed information.

If none of the above suggestions clear the warnings, please open an issue with detailed logging, monitoring, metrics and optionally workload information.

What does the etcd warning "snapshotting is taking more than x seconds to finish ..." mean?

etcd sends a snapshot of its complete key-value store to refresh slow followers and for backups. Slow snapshot transfer times increase MTTR; if the cluster is ingesting data with high throughput, slow followers may livelock by needing a new snapshot before finishing receiving a snapshot. To catch slow snapshot performance, etcd warns when sending a snapshot takes more than thirty seconds and exceeds the expected transfer time for a 1Gbps connection.

Last modified June 3, 2025: Recursively copy the contents of v3.6 to v3.7 (a90b2a6)