How etcd Works: Consensus, Quorum, and Failure Handling

medium.com/@pragalva.sapkota/how-etcd-works-consensus-quorum-and-failure-handling-3c97e9aaaa8f

Pragalva Sapkota December 17, 2025

Introduction

etcd is a core component in distributed systems, yet it is often overlooked by beginners. Despite its critical role in maintaining consistency and state, it is frequently treated as a black box.

In this blog, I explain what etcd is, how it works, and why its design decisions matter. The explanation is grounded in real world analogies and reinforced with practical contexts, rather than abstract theory.

The goal is to make etcd easier to reason about, not just to describe what it does, but to understand how and why it behaves the way it does in real systems.

What is etcd?

It is a distributed key-value store that provides a reliable way to store data that needs to be accessed by a distributed system or clusters. etcd achieves this by using a consensus algorithm so that even if some nodes fail, the remaining nodes still agree on the data. In practical terms, etcd exists to make sure that critical system information is never ambiguous, duplicated, or silently lost. In Kubernetes, it serves as the single source of truth that the control plane relies on for all cluster state.

Raft consensus

Raft is a consensus algorithm, its goal is not just leader selection, but keeping multiple replicas of data consistent.

Imagine a system that stores the price of apples.
If you have one database and it goes down, you lose the price completely.
If you have multiple databases, but each one updates independently, different users may see different prices, which is equally bad.

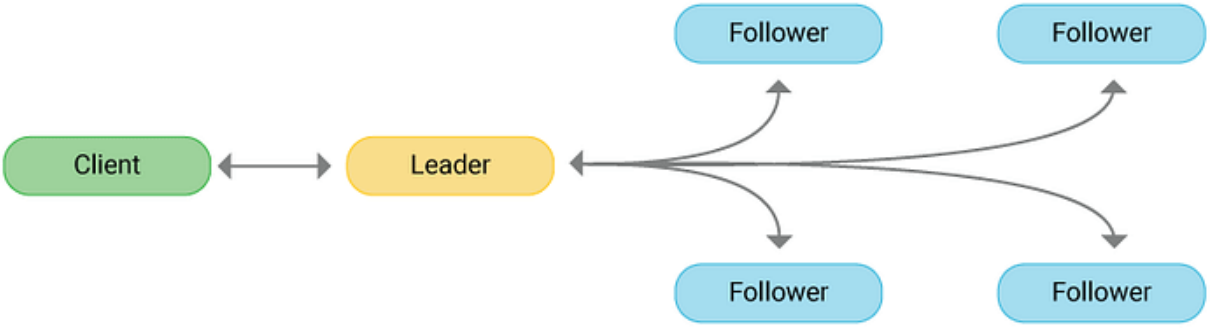
Raft solves this by enforcing a single leader model.
Only one node, the leader, is allowed to accept writes. All changes are first written to the leader, and then replicated to the followers. This ensures every node eventually has the same data in the same order.

The leader sends regular heartbeats to followers to signal that it is alive. If followers stop receiving heartbeats, they assume the leader has failed and start a new election.

Each node has its own randomized election timeout. The node whose timeout expires first becomes a candidate and asks others for votes. This randomness prevents multiple nodes from starting elections at the same time.

A follower will vote for a candidate only if the candidate's log is at least as up to date as its own. This rule prevents stale nodes from becoming leader and overwriting newer data.

Once a candidate receives votes from a majority (quorum) of nodes, it becomes the new leader and resumes serving writes.



From this diagram, we can see that once a leader is elected, all write operations are first handled by the leader and then replicated to the followers.

Quorum: why etcd sometimes refuses to work

In simple terms, a quorum is the minimum number of members required for a group, such as a board, committee, or parliament, to officially conduct business or make decisions. If a quorum is not met, the meeting is considered inquorate and any decisions taken are considered invalid.

Distributed systems like etcd use the same idea of quorum to ensure decisions are only made when a majority of nodes agree, preventing inconsistent or conflicting state.

Quorum formula is ⌊N / 2⌋ + 1
where N is the total total amount of etcd cluster

etcd clusters are deployed with an odd number of nodes because quorum requires a strict majority to make decisions. With an odd number of members, the system can tolerate the maximum number of failures while still maintaining quorum.

If N = 2:
⌊2 / 2⌋ + 1 = 1 + 1 = 2

So the quorum is 2
This means both nodes must agree for any write or leader election.

Which means there is zero failure tolerance.

What happens when the quorum is lost?

When etcd loses quorum it stops accepting write operations. This protects the system from a split brain scenario where multiple nodes could accept conflicting updates. It mostly becomes read only until the quorum is restored.

Quorum focuses on **correctness over availability**, which is rather important for high level systems such as Kubernetes which depends on consistent and authoritative state.

How etcd represents and transmits state (Protobuf)

Once etcd reaches agreement on what the system state should be, that state must be stored and transmitted efficiently and consistently between nodes. For this purpose, etcd uses Protocol Buffers (Protobuf) as its serialization format. Protobuf provides a compact, binary representation of structured data that is both fast to encode and decode and deterministic across implementations.

Deterministic serialization is important in a consensus system because the same data must be replicated exactly between the leader and followers. Protobuf also reduces network and storage overhead compared to text based formats such as JSON or YAML, since Protobuf messages are binary, smaller, and faster to process. Protobuf does not influence consensus decisions themselves, it only defines how agreed upon state is encoded and transmitted.

Disaster recovery in etcd

etcd is designed to withstand machine and process failures without losing critical system state. While consensus mechanisms such as quorum prevent further damage when parts of the cluster fail, disaster recovery relies on persisted data to restore the system. etcd achieves this through snapshots and the write ahead log, which together allow the cluster to be reconstructed without losing agreed upon data.

WAL (Write Ahead Log)

The write ahead log records every state change before it is applied. You can think of it as writing decisions into a notebook before acting on them. If a node crashes or restarts, etcd replays the WAL to recover the exact state it had before the failure, ensuring no committed data is lost.

Snapshots

Snapshots are periodic, point in time copies of the current state. Instead of replaying the entire write ahead log from the beginning, etcd can load the latest snapshot and replay only the changes that occurred after it. This makes recovery faster and prevents the log from growing indefinitely. A snapshot is like making a photocopy of the notebook so recovery does not have to start from the first page.

Conclusion

In this blog, we learned what etcd is, how it works, and why it is crucial in distributed systems like Kubernetes. We also focused on the Raft consensus algorithm, how it maintains consistency, and how etcd handles failure and disaster scenarios.

A solid understanding of etcd makes it easier to reason about cluster state, failure modes, and recovery behavior. This becomes increasingly important as systems scale, where reliability and correctness are not optional but foundational.