# APACHE ZOOKEEPER

## What is Apache ZooKeeper?

- Apache ZooKeeper is a centralized, open-source service used for distributed coordination — providing naming, configuration management, synchronization, and group services that are critical for systems like Kafka, Hadoop, and HBase.
- It ensures reliability, fault tolerance, and ordered operations across clusters.

## Key Features of Apache ZooKeeper

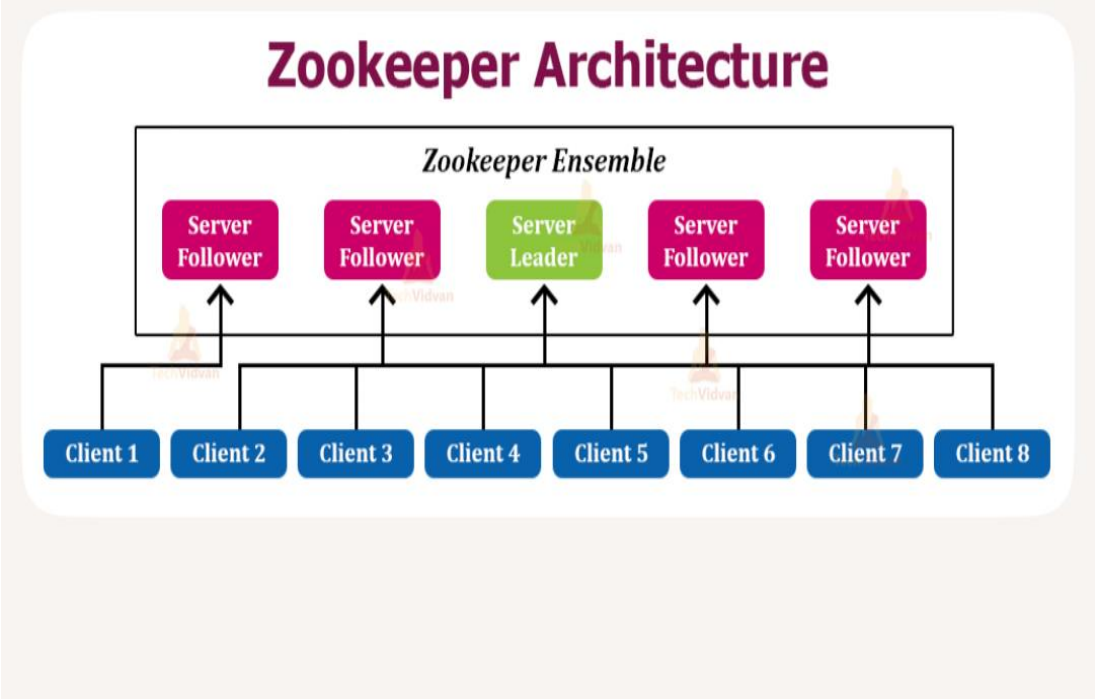
- Centralized coordination: Manages configuration, naming, and synchronization for distributed applications.
- High performance: In-memory messaging with sub-millisecond latency, handling tens of thousands of requests per second.
- Fault tolerance: Replicated across an ensemble of servers, surviving node failures without data loss.
- Ordered operations: Every update is stamped with a monotonically increasing transaction ID (zxid), ensuring total ordering.
- Hierarchical namespace: Modeled after a file system, making coordination logic intuitive.

## Apache ZooKeeper architecture

- Apache ZooKeeper's architecture is designed to provide reliable coordination for distributed systems, ensuring consistency and fault tolerance across clusters.

### *Core Components of ZooKeeper Architecture*

**Ensemble**: A group of ZooKeeper servers (usually 3, 5, or 7) that work together. One acts as the leader, while others are followers.
**Leader**: Handles all write requests and coordinates updates.
**Followers:** Handle read requests and forward writes to the leader.
**Observers**: Non-voting members that improve read scalability without participating in leader elections.
**ZNodes**: Data nodes in ZooKeeper's hierarchical namespace (like a filesystem). They can be persistent or ephemeral.
**Sessions**: Client connections tracked by ZooKeeper; ephemeral znodes exist only as long as the session is active.
**Watches**: Notification mechanism that alerts clients when data changes.




**Workflow**

**Client connection**

- A client connects to any ZooKeeper server in the ensemble.
- The server establishes a session and assigns a unique session ID.

**Read requests**

- Reads are served directly by the connected server (leader or follower).
- Since ZooKeeper maintains a replicated state, reads are consistent.

**Write requests**

- All writes (create, update, delete) are forwarded to the leader.
- The leader assigns a transaction ID (zxid) to ensure global ordering.

**Proposal phase**

- The leader broadcasts the proposed update to all followers using the Zab protocol (ZooKeeper Atomic Broadcast).
- Each follower logs the proposal to disk and sends an acknowledgment.

**Commit phase**

- Once a majority (quorum) of followers acknowledge, the leader commits the transaction.
- The leader then informs followers to apply the change.

**Replication**

- Followers update their local state to match the leader.
- This ensures strong consistency across the ensemble.

**Client notification**

- If the client had set a watch on the znode, ZooKeeper sends a notification about the change.
- Clients can react immediately (e.g., reconfigure services, elect leaders).

## ZooKeeper Zab protocol

- The ZooKeeper Atomic Broadcast (Zab) protocol is the heart of ZooKeeper's architecture — it ensures total order, reliability, and fault tolerance for all updates across the ensemble.
- It is a leader-based consensus protocol, similar to Paxos, but optimized for continuous streams of operations and recovery.

## Core Principles of Zab

- Reliable delivery: If one server delivers a message, all servers will eventually deliver it.
- Total order: All servers deliver messages in the same sequence.
- Causal order: If message B depends on A, then A is always delivered before B.
- Prefix ordering: Ensures that followers' histories are consistent with the leader's, even during recovery.

## Workflow of Zab Protocol

**Leader election**
- One server becomes the leader; others act as followers.
- Leader is responsible for proposing updates.

**Proposal phase**
- Leader assigns a zxid (ZooKeeper transaction ID) to each update.
- Sends a proposal packet to followers.

**Acknowledgment phase**
- Followers log the proposal and send back an ACK.
- A quorum (majority) of ACKs is required to proceed.

**Commit phase**
- Once quorum is reached, the leader broadcasts a COMMIT.
- Followers apply the update to their state.

**Delivery phase**
- Clients are notified of success.
- Watches trigger if relevant znodes were updated.

## Zab Packet Types

**Proposal packet**
- Sent by the leader to followers when a new transaction (write request) is initiated.
- Contains the zxid (ZooKeeper transaction ID) and the proposed update.
- Followers log this proposal before acknowledging.

**ACK packet**
- Sent by followers back to the leader after logging the proposal.
- Confirms that the follower is ready to commit the transaction.
- A quorum of ACKs is required before moving forward.

**Commit packet**
- Sent by the leader once a majority of ACKs are received.
- Instructs followers to apply the transaction to their state.

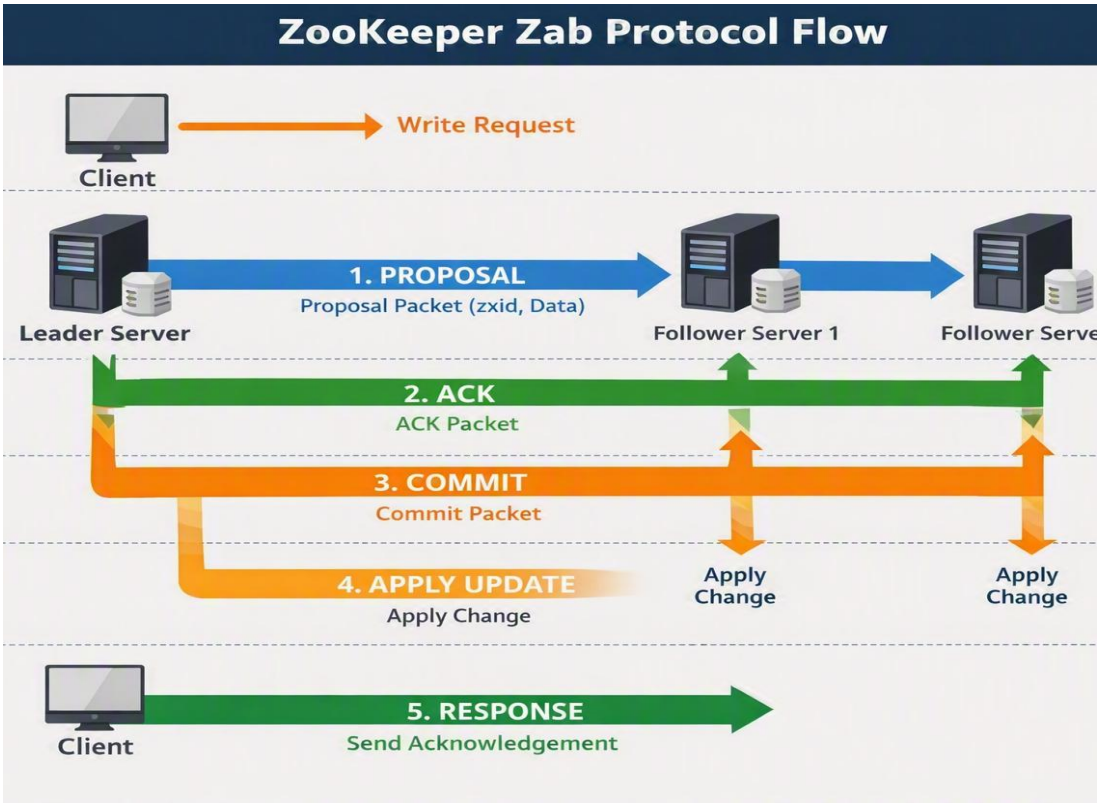
- Ensures atomicity — either all servers commit or none do.

**Inform packet**

- Used in some implementations to notify observers (non-voting servers).
- Helps scale read operations without affecting quorum-based writes.

**Sync packet**

- Sent during leader recovery or when a new follower joins.
- Ensures the follower's state is consistent with the leader's history.




## ZooKeeper Architecture Layers

**Client Layer**

- Applications and services that interact with ZooKeeper.
- They send read/write requests and set watches for notifications.

**Service Layer**

- The ensemble of ZooKeeper servers (leader, followers, observers).
- Handles client requests, leader election, and coordination.

**Consensus Layer**

- Implements the Zab protocol (ZooKeeper Atomic Broadcast).
- Ensures ordered updates, fault tolerance, and leader election.

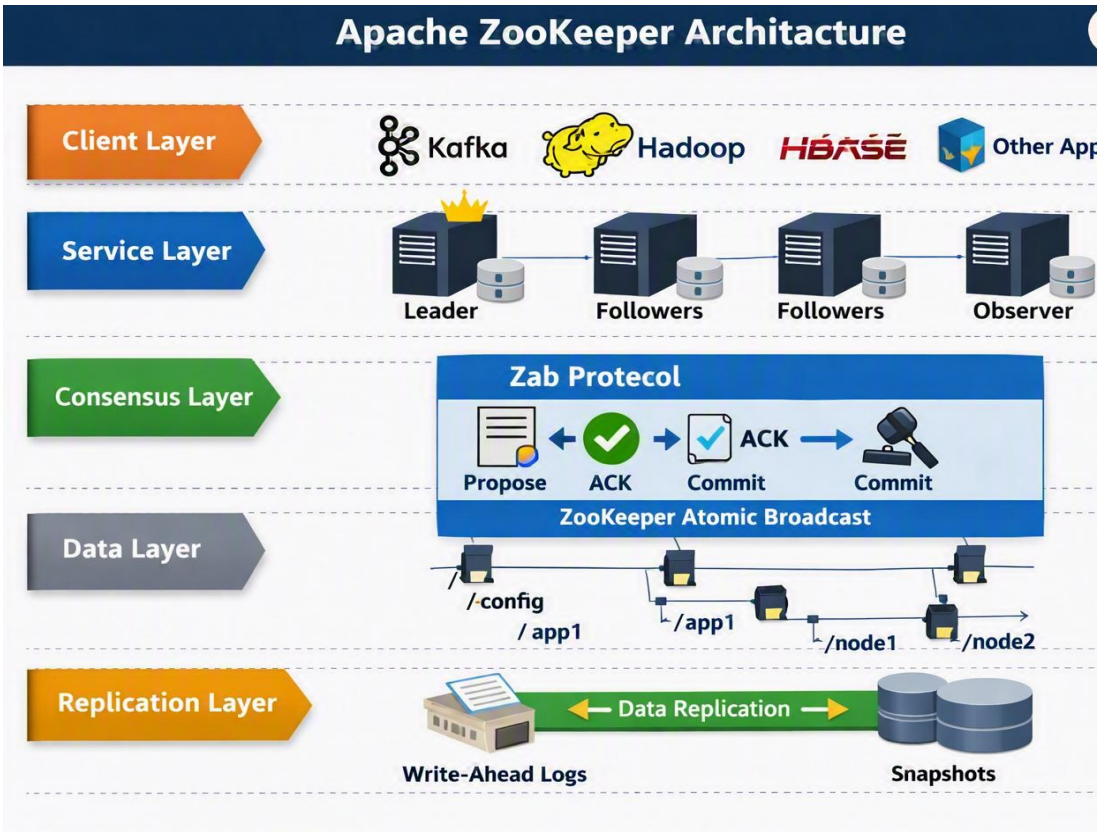
**Data Layer**

- Hierarchical namespace of ZNodes (like a filesystem).
- Stores metadata, configuration, and coordination information.

**Replication Layer**

- Ensures all servers have the same state.

- Uses write-ahead logs and snapshots for durability.




## Core Benefits

- Centralized coordination
- Provides a single, reliable service for managing configuration, naming, and synchronization across distributed applications.
- High availability
- Uses ensemble replication and quorum-based consensus to ensure the system continues functioning even if some nodes fail.
- Consistency and reliability
- The Zab protocol guarantees ordered and atomic updates, ensuring all servers maintain the same state.
- Scalability
- Supports large-scale distributed environments with thousands of clients and high request throughput.
- Fault tolerance
- Automatically recovers from node failures through leader election and state synchronization.
- Simplified synchronization
- Provides primitives like locks, barriers, and queues, making distributed coordination easier to implement.
- Dynamic configuration management
- Enables real-time updates to configuration data without restarting services.
- Service discovery

- Helps distributed applications locate and register active services dynamically.
- Ease of integration
- Works seamlessly with systems like Kafka, Hadoop, HBase, and Pulsar for cluster coordination and metadata management.

**Disadvantages**

- Operational complexity → Requires careful setup and management of ensembles.
- Write bottleneck → All writes go through the leader, which can limit throughput.
- Resource intensive → Needs dedicated servers and careful tuning for performance.
- Not for large data storage → Designed only for metadata and coordination, not bulk data.
- Dependency overhead → Some systems (like older Kafka versions) depend on ZooKeeper, adding extra moving parts.
- Latency in recovery → Leader failure triggers election and synchronization, which can temporarily slow operations.

**Real-World Examples of ZooKeeper Usage**

**1.Apache Kafka**

- ZooKeeper manages broker metadata, topic configurations, and partition leadership.
- Ensures high availability by electing a new leader if a broker fails.
- Newer Kafka versions (post 2.8) are moving toward a ZooKeeper-free architecture, but many production clusters still depend on it.

**2. Apache Hadoop**

- Used for NameNode failover in Hadoop Distributed File System (HDFS).
- Provides coordination between active and standby NameNodes.
- Ensures seamless recovery during failures.

**3. Apache HBase**

- ZooKeeper tracks region servers and manages cluster membership.
- Provides automatic failover and load balancing.
- Ensures clients always connect to the correct region server.

**4. Apache Cassandra**

- Manages ring topology and node configuration.
- Helps with replication factor settings and data center assignments.
- Improves fault tolerance and scalability in distributed NoSQL clusters.

**5. Neo4j**

- Used in high availability components for write-master election.
- Coordinates read slaves and ensures consistency in graph database clusters.

**6. Spring Cloud Zookeeper**

- Provides service discovery and configuration management for microservices.
- Integrates with Spring Boot applications for dynamic cluster coordination.