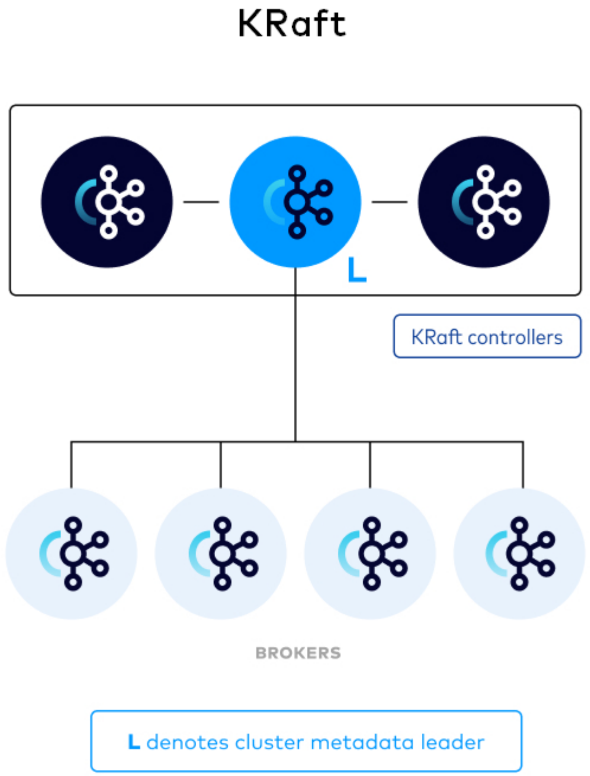
Introduction to KRaft in Confluent Platform

Simplifying Kafka Metadata Management

- **Kafka Raft (KRaft) Overview:** KRaft, short for Kafka Raft, is the consensus protocol that replaces ZooKeeper in Apache Kafka® starting from Confluent Platform 8.0. It consolidates metadata management directly into Kafka for improved simplicity and reliability.
- **Architectural Shift:** KRaft removes the external ZooKeeper dependency by embedding metadata management in Kafka itself. Each KRaft controller participates in a Raft quorum, ensuring strong consistency and streamlined control.
- **Motivation for Adoption:** The move to KRaft mode reduces operational complexity, enhances metadata propagation speed, and enables faster controller failover with fewer moving parts.
- **Key Benefit:** By eliminating ZooKeeper, Confluent simplifies deployment, increases fault tolerance, and supports larger Kafka clusters with lower maintenance overhead.

KRaft Architecture

How the Raft Quorum Manages Kafka Metadata



KRaft Architecture

How the Raft Quorum Manages Kafka Metadata

Controller Quorum and Metadata Log
KRaft controllers form a Raft quorum that manages the Kafka metadata log, storing all topic, partition, ISR, and configuration changes. This ensures consistency and durability across controllers.

Active and Follower Controllers
The active controller handles all broker RPCs, while follower controllers replicate metadata updates and serve as hot standbys, ensuring immediate recovery during failover.

Leader Election and Consensus
Using the Raft protocol, KRaft achieves consensus without external coordination. Majority voting enables leader election, fault tolerance, and fast recovery.

Snapshot and Recovery Mechanism
Controllers periodically write metadata snapshots to disk, improving recovery times and reducing replay overhead during restarts.

Scaling Kafka with KRaft

Performance, Partition Limits, and Failover Improvements

- **Partition and Cluster Limits:** Kafka scalability in KRaft mode depends on both the per-node partition count limit and the cluster-wide partition limit. Adding more nodes remains the primary way to increase capacity.
- **Enhanced Metadata Propagation:** KRaft's quorum controller minimizes metadata propagation delay, enabling faster synchronization across brokers and reducing cluster downtime.
- **Rapid Failover:** Experiments by Confluent demonstrate near-instantaneous controller failover in KRaft. In a 2-million-partition cluster, KRaft achieved 10× the partition scale of ZooKeeper-based clusters.
- **High Availability through Majority Consensus:** A three-node quorum can survive one controller failure; five nodes can survive two. Majority voting ensures operational continuity during failures.

KRaft Configuration Essentials

Core Settings Required for Confluent Platform Deployment

- **process.roles:** Specifies whether the Kafka server acts as a controller, broker, or both (though combined mode is not supported for production). This setting activates KRaft mode.
- **node.id:** Assigns a unique identifier to each node. No two servers—controller or broker—may share the same ID within a cluster.
- **controller.quorum.voters:** Defines the set of controllers participating in the Raft quorum using the format {id}@{host}:{port}. Ensures all nodes are aware of the quorum members.
- **controller.listener.names:** Lists controller listeners and ensures they match the listeners property. Essential for isolated or combined mode configurations.

Listeners, Logs, and Security in KRaft Mode

Ensuring Secure and Reliable Metadata Communication

- **Inter-Broker Communication:** Defined via either `inter.broker.listener.name` or `security.inter.broker.protocol`. These settings ensure secure communication between brokers and controllers.
- **Listener Configuration:** Controllers must list their own listeners under `controller.listener.names`, consistent with the quorum voters list. Brokers and controllers use mapped security protocols for communication.
- **Metadata Log Directories:** The `metadata.log.dir` property specifies where controller metadata is stored. If unset, it defaults to the first directory in `log.dirs`.
- **Security and Truststores:** Secure communication is achieved by defining SSL keystores and truststores per listener. Controllers must include truststore configurations for broker communication.

Tools for Managing and Debugging KRaft Mode

Command-Line Utilities for Cluster Insight and Diagnostics

kafka-metadata-quorum
Describes the runtime status of the KRaft metadata partition. Can connect via `--bootstrap-server` or `--bootstrap-controller` to inspect quorum state and leadership.

kafka-dump-log
Decodes and prints metadata records from KRaft log segments and snapshots, enabling deep inspection of stored metadata.

kafka-metadata-shell
Provides an interactive shell for exploring metadata partitions. Supports secure connections to running controllers via SSL.

kafka-migration-check
Assesses migration status and readiness for KRaft clusters. Useful for validating Confluent Platform transitions post-ZooKeeper.

Migration, Limitations & Best Practices

Transitioning from ZooKeeper and Ensuring KRaft Stability

Migration Process
Use the `kafka-migration-check` tool to validate migration readiness. Ensure each node’s storage is formatted with a consistent cluster ID using the `kafka-storage format` command.

Unsupported Combined Mode
Running a Kafka node as both broker and controller (combined mode) is not supported in production due to security and operational gaps.

Quorum Reconfiguration
Prior to Confluent Platform 7.9, KRaft clusters could not add or remove controllers dynamically. Later versions introduce controlled quorum reconfiguration.

Monitoring Limitations
Certain tools, such as Health+, may still misreport KRaft controllers as brokers, impacting alert accuracy. Administrators should account for this discrepancy.

Conclusion: The Benefits of KRaft Architecture

Simpler, Faster, and More Reliable Kafka Metadata Management

- **Unified Metadata Management:** KRaft embeds metadata handling directly into Kafka, eliminating ZooKeeper and reducing operational overhead.
- **Improved Fault Tolerance:** Raft-based consensus and majority voting ensure resilient controller failover and minimal downtime.
- **Operational Efficiency:** Simplified configuration and deployment enable easier scaling and reduced administrative complexity.
- **Enhanced Scalability:** Supports up to 10× more partitions than ZooKeeper-based clusters, allowing massive deployments to operate efficiently.