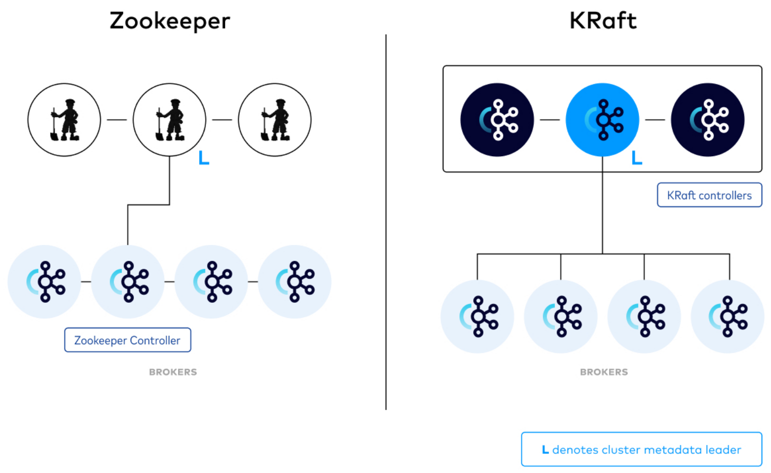
KRaft: Apache Kafka Without ZooKeeper

See how Apache Kafka®'s architecture has been greatly simplified by the introduction of Apache Kafka Raft (KRaft).

Apache Kafka Raft (KRaft) is the consensus protocol that was introduced in KIP-500 to remove Apache Kafka's dependency on ZooKeeper for metadata management. This greatly simplifies Kafka's architecture by consolidating responsibility for metadata into Kafka itself, rather than splitting it between two different systems: ZooKeeper and Kafka. KRaft mode makes use of a new quorum controller service in Kafka which replaces the previous controller and makes use of an event-based variant of the Raft consensus protocol.

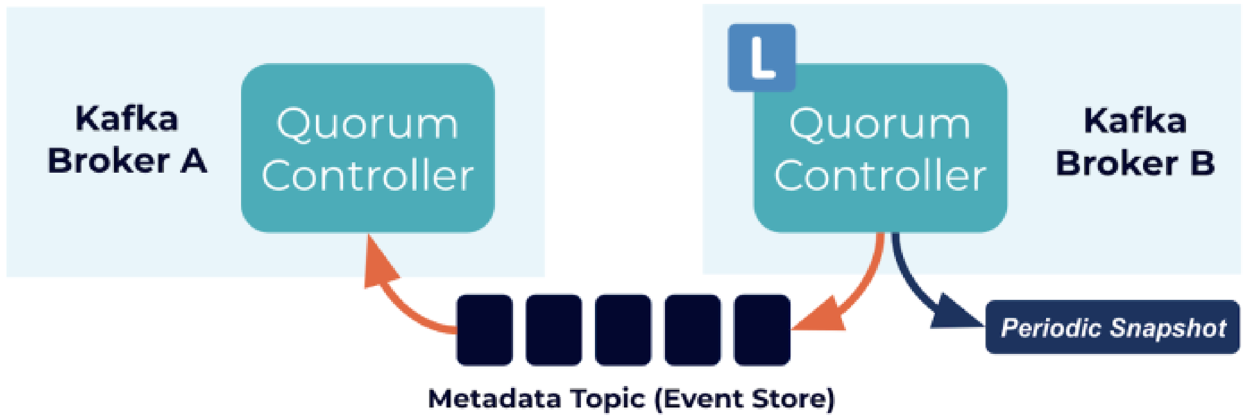


Benefits of Kafka’s new quorum controller

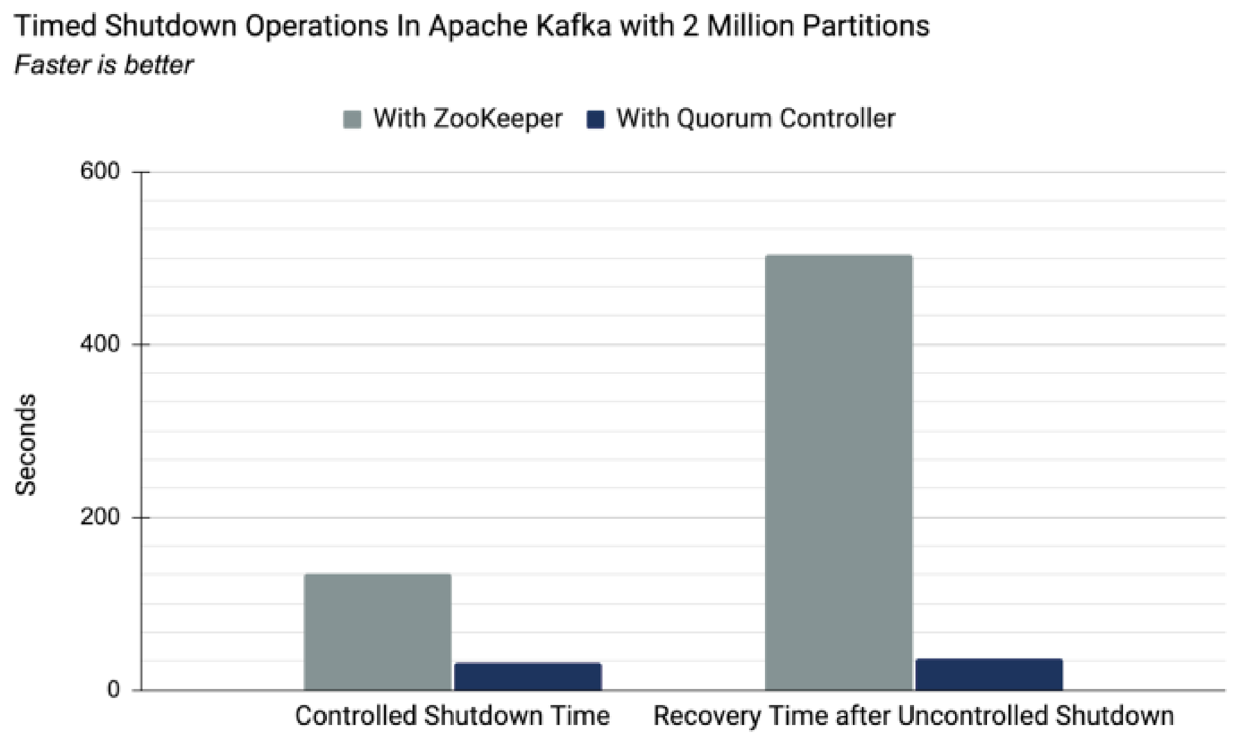
1. KRaft enables *right-sized* clusters, meaning clusters that are sized with the appropriate number of brokers and compute to satisfy a use case’s throughput and latency requirements, with the potential to scale up to millions of partitions
2. Improves stability, simplifies the software, and makes it easier to monitor, administer, and support Kafka
3. Allows Kafka to have a single security model for the whole system
4. Unified management model for configuration, networking setup, and communication protocols
5. Provides a lightweight, single-process way to get started with Kafka
6. Makes controller failover near-instantaneous

How it works

The quorum controllers use the new KRaft protocol to ensure that metadata is accurately replicated across the quorum. The quorum controller stores its state using an event-sourced storage model, which ensures that the internal state machines can always be accurately recreated. The event log used to store this state (also known as the metadata topic) is periodically abridged by snapshots to guarantee that the log cannot grow indefinitely. The other controllers within the quorum follow the active controller by responding to the events that it creates and stores in its log. Thus, should one node pause due to a partitioning event, for example, it can quickly catch up on any events it missed by accessing the log when it rejoins. This significantly decreases the unavailability window, improving the worst-case recovery time of the system.



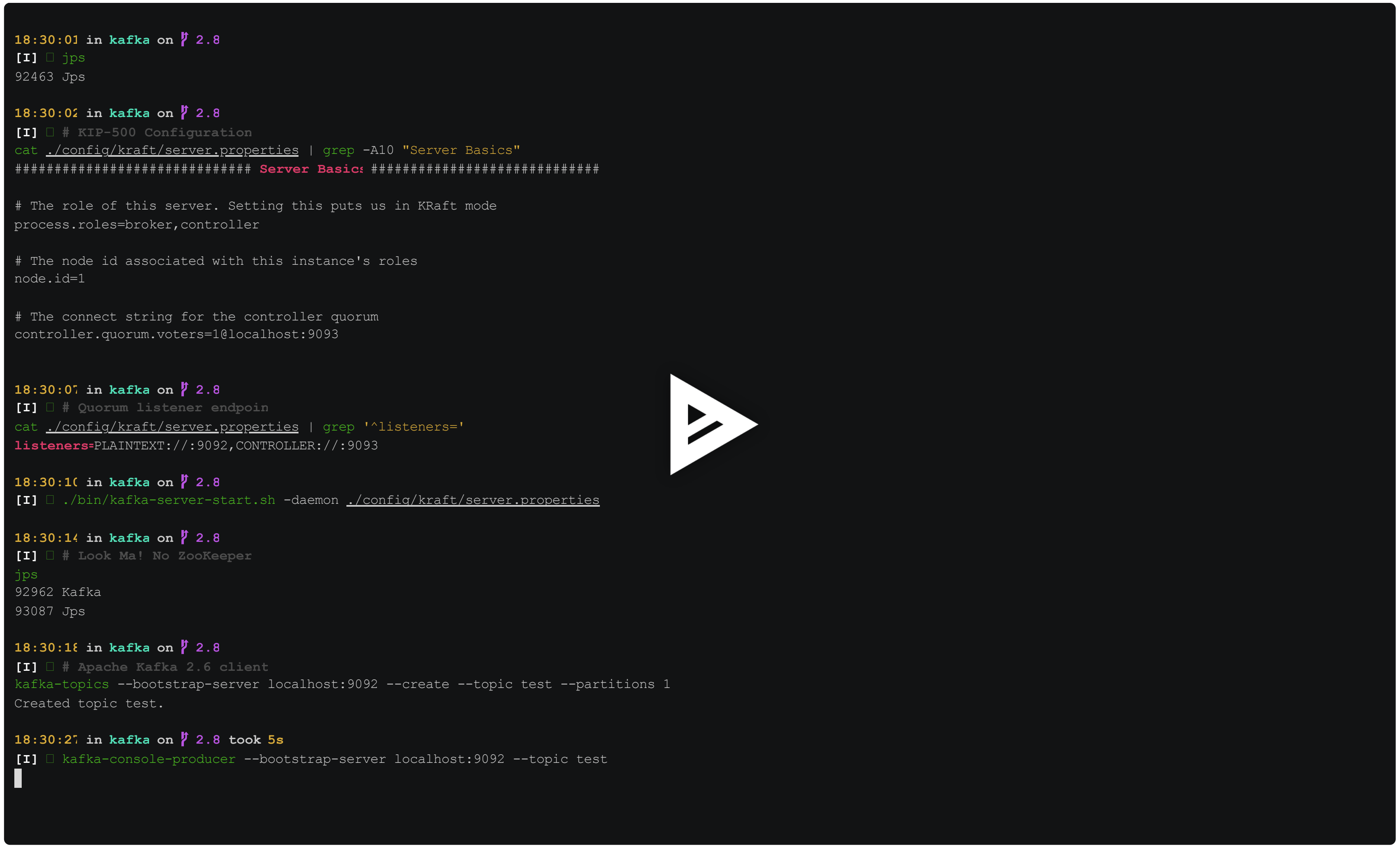
The event-driven nature of the KRaft protocol means that, unlike the ZooKeeper-based controller, the quorum controller does not need to load state from ZooKeeper before it becomes active. When leadership changes, the new active controller already has all of the committed metadata records in memory. What's more, the same event-driven mechanism used in the KRaft protocol is used to track metadata across the cluster. A task that was previously handled with RPCs now benefits from being event-driven as well as using an actual log for communication.



Try it!

KRaft mode is production ready for new clusters as of Apache Kafka 3.3. The development progress for additional features like migration from ZooKeeper is tracked in KIP-833.

If you want to try it yourself, follow the Kafka Local Quickstart. This quick start runs in KRaft combined mode, meaning that one process acts as both the broker and controller. Combined mode is only appropriate for local development and testing. Refer to the documentation here for details on configuring KRaft for production in isolated mode, meaning controllers run independently from brokers.



```
18:30:01 in kafka on ᚠ 2.8
[I] ▯ jps
92463 Jps

18:30:02 in kafka on ᚠ 2.8
[I] ▯ # KIP-500 Configuration
cat ./config/kraft/server.properties | grep -A10 "Server Basics"
############################# Server Basics #############################

# The role of this server. Setting this puts us in KRaft mode
process.roles=broker,controller

# The node id associated with this instance's roles
node.id=1

# The connect string for the controller quorum
controller.quorum.voters=1@localhost:9093


18:30:07 in kafka on ᚠ 2.8
[I] ▯ # Quorum listener endpoin
cat ./config/kraft/server.properties | grep '^listeners='
listeners=PLAINTEXT://:9092,CONTROLLER://:9093

18:30:10 in kafka on ᚠ 2.8
[I] ▯ ./bin/kafka-server-start.sh -daemon ./config/kraft/server.properties

18:30:14 in kafka on ᚠ 2.8
[I] ▯ # Look Ma! No ZooKeeper
jps
92962 Kafka
93087 Jps

18:30:18 in kafka on ᚠ 2.8
[I] ▯ # Apache Kafka 2.6 client
kafka-topics --bootstrap-server localhost:9092 --create --topic test --partitions 1
Created topic test.

18:30:27 in kafka on ᚠ 2.8 took 5s
[I] ▯ kafka-console-producer --bootstrap-server localhost:9092 --topic test
▮
```

Recorded with **asciinema**

Right now you can swing your tools from getting metadata from ZooKeeper over to getting metadata from the brokers instead, as described in Preparing Your Clients and Tools for KIP-500: ZooKeeper Removal from Apache Kafka.

	WITH ZOOKEEPER	WITH KRAFT (NO ZOOKEEPER)
Configuring Clients and Services	`zookeeper.connect=zookeeper:2181`	`bootstrap.servers=broker:9092`
Configuring Schema Registry	`kafkastore.connection.url=zookeeper:2181`	`kafkastore.bootstrap.servers=broker:9092`
Kafka administrative tools	`kafka-topics --zookeeper zookeeper:2181`	`kafka-topics --bootstrap-server broker:9092 ... --command-config properties to connect to brokers`
REST Proxy API	v1	v2 or v3
Getting the Kafka Cluster ID	`zookeeper-shell zookeeper:2181 get/cluster/id`	`kafka-metadata-quorum` or view `metadata.properties` or `confluent cluster describe --url http://broker:8090 --output json`

Learn More

- For more details on what KRaft is, see the KRaft Overview documentation
- For hardware requirements and how to configure and monitor KRaft see the Configure KRaft in Production documentation
- For steps to migrate from ZooKeeper to KRaft (early access), see the Migrate from ZooKeeper to KRaft (EA) documentation
- For how to configure ZooKeeper for production use, see the Configure ZooKeeper for Production documentation
- Read Guozhang Wang's blog post for a deep dive on why ZooKeeper is being replaced: Why ZooKeeper Was Replaced with KRaft – The Log of All Logs

Apache Kafka Quick Start

Apache Flink Quick Start