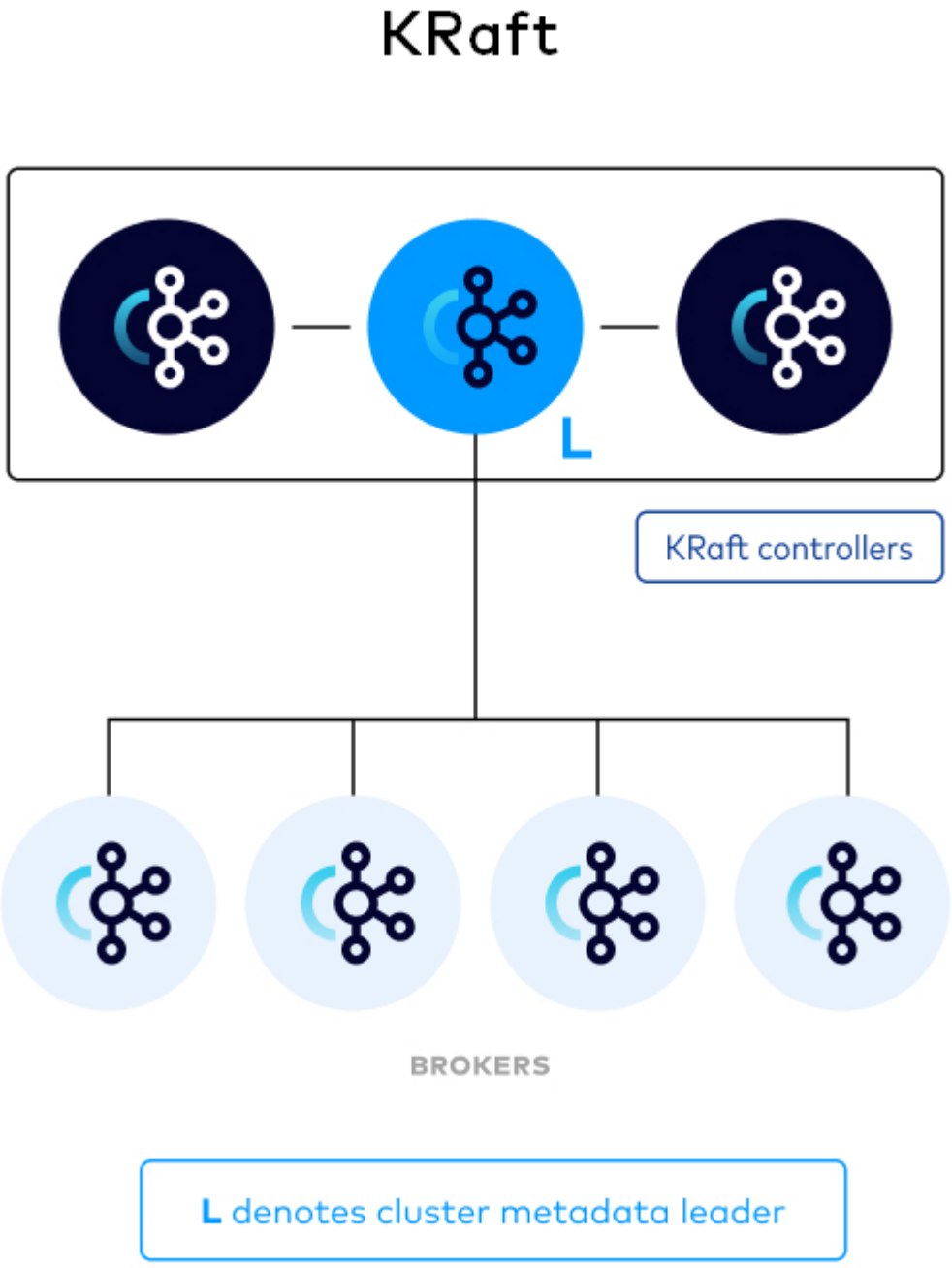
KRaft Overview for Confluent Platform

Starting with Confluent Platform version 8.0, KRaft (pronounced craft) mode is how metadata is managed in Apache Kafka®.

Kafka Raft (KRaft) is the consensus protocol that greatly simplifies Kafka's architecture by consolidating responsibility for metadata into Kafka itself.

The following image provides a simple illustration of Kafka running with KRaft managing metadata for the cluster. Each KRaft controller is a node in a Raft quorum, and each node is a broker that can handle client requests.



The controller quorum

The KRaft controller nodes comprise a Raft quorum which manages the Kafka metadata log. This log contains information about each change to the cluster metadata. Metadata about topics, partitions, ISRs, configurations, and so on, is stored in this log.

Using the Raft consensus protocol, the controller nodes maintain consistency and leader election without relying on any external system. The leader of the metadata log is called the active controller. The active controller handles all RPCs made from the brokers. The follower controllers replicate the data which is written to the active controller, and serve as hot standbys if the active controller should fail. With the concept of a metadata log, brokers use offsets to keep track of the latest metadata stored in the KRaft controllers, which results in more efficient propagation of metadata and faster recovery from controller failovers.

KRaft requires a majority of nodes to be running. For example, a three-node controller cluster can survive one failure. A five-node controller cluster can survive two failures, and so on.

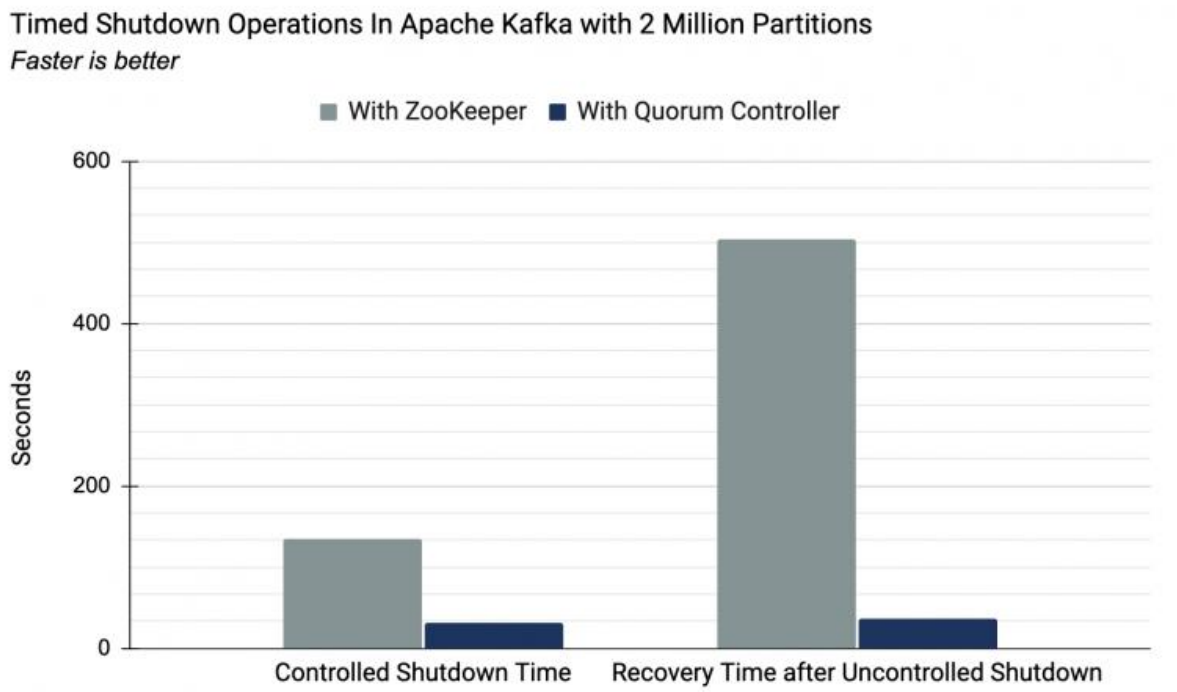
Periodically, the controllers will write out a snapshot of the metadata to disk. This is conceptually similar to compaction, but state is read from memory rather than re-reading the log from disk.

Scaling Kafka with KRaft

There are two properties that determine the number of partitions an Kafka cluster can support: the per-node partition count limit and cluster-wide partition limit.

KRaft mode is designed to handle a large number of partitions per cluster, however Kafka's scalability still primarily depends on adding nodes to get more capacity, so the cluster-wide limit still defines the upper bounds of scalability within the system.

In KRaft, the quorum controller reduces the time taken to move critical metadata in a controller failover scenario. The result of this change is a near-instantaneous controller failover. The following image shows the results of a Confluent lab experiment on a Kafka cluster running 2 million partitions, which is 10 times the maximum number of partitions for a cluster running ZooKeeper. The experiment shows that controlled shutdown time and recovery time after uncontrolled shutdown are greatly improved with a quorum controller versus ZooKeeper.



KRaft Configuration for Confluent Platform

Hardware and JVM requirements

A production KRaft server can cover a wide variety of use cases. In general, you should run KRaft on a server with similar specifications to the following:

- Minimum of 4 GB of RAM
- Dedicated CPU core should be considered when the server is shared
- An SSD disk at least 64 GB in size is highly recommended
- JVM heap size of at least 1 GB is recommended

Currently, it is recommended that you run at least three (3) KRaft controllers in production.

Configuration options

Consider that a KRaft controller is also a Kafka broker processing event records that contain metadata related to the Kafka cluster. However, not all broker properties need to be set on controllers.

There are some settings that must be included for a cluster to run in KRaft mode, and are unique per server, but there are other settings that you configure for a controller because the controller process itself uses that property to perform its controller duties, or the property affects cluster metadata that controllers manage.

Required settings

These entries must be included for each server (controllers and brokers) running in KRaft mode.

process.roles

When you operate Apache Kafka® in KRaft mode, you must set the process.roles property. This property specifies whether the server acts as a controller, broker, or both, although currently both is not supported for production workloads. In KRaft mode, specific Kafka servers are selected to be controllers, storing metadata for the cluster in the metadata log, and other servers are selected to be brokers. The servers selected to be controllers will participate in the metadata quorum. Each controller is either an active or a hot standby for the current active controller.

In a production environment, the controller quorum will be deployed on multiple nodes. This is called an ensemble. An ensemble is a set of 2n + 1 controllers where n is any number greater than 0. The odd number of controllers allows the controller quorum to perform majority elections for leadership. At any given time, there can be up to n failed servers in an ensemble and cluster will keep quorum. For example, with three controllers, the cluster can tolerate one controller failure. If at any time, quorum is lost, the cluster will go down. For production, you should have typically have 3 or 5 controllers, but at least 3.

- Type: string
- Default:
- Importance: required for KRaft mode

process.roles can have the following values:

Value	Result
Not set	The server is assumed to be in ZooKeeper mode. This is not supported in Confluent Platform version 8.0 and later.
broker	The server operates only as a broker.
controller	The server operates in isolated mode as a controller only.

node.id

The unique identifier for this server. Each node ID must be unique across all the brokers and controllers in a particular cluster. No two servers can have the same node ID regardless of

their process.roles value. This identifier replaces broker.id, which was used when operating in ZooKeeper mode.

- Type: int
- Default:
- Importance: required for KRaft mode

controller.quorum.voters

A comma-separated list of quorum voters. All of the servers (controllers and brokers) in a Kafka cluster discover the quorum voters using this property, and you must identify all of the controllers by including them in the list you provide for the property.

Each controller is identified with their ID, host and port information in the format of {id}@{host}:{port}. Multiple entries are separated by commas and might look like the following:

controller.quorum.voters=1@host1:port1,2@host2:port2,3@host3:port3

The node ID supplied in the controller.quorum.voters property must match the corresponding ID on the controller servers. For example, on controller1, node.id must be set to 1. If a server is a broker only, its node ID should not appear in the controller.quorum.voters list.

- Type: string
- Default:
- Importance: required

controller.listener.names

A comma-separated list of listener_name entries for listeners used by the controller. On a node with process.roles=broker, only the first listener in the list will be used by the broker. For KRaft controllers in isolated or combined mode, the node will listen as a KRaft controller on all listeners that are listed for this property, and each must appear in the listeners property. They shouldn't appear in the advertised.listeners property.

- Type: string
- Default: null
- Importance: required

Inter-broker listeners

Listeners are an important part of your configuration. In addition to controller.listener.names described in the previous section, you should configure how KRaft controllers will communicate with brokers. This can be done with the security.inter.broker.protocol property or the inter.broker.listener.name property, but not both.

If inter.broker.listener.name is set then it will be used as a key for lookup in the listener.security.protocol.map property to yield a security protocol, otherwise security.inter.broker.protocol will be used. The default for security.inter.broker.protocol is PLAINTEXT, which is what will be used for communication with brokers if neither property is explicitly set.

Note that controllers do not listen at the inter.broker.listener.name value, but this property defines a listener that the brokers create, and controllers must specify in their security protocol and configuration so it can communicate with the brokers.

Following are descriptions of these properties:

inter.broker.listener.name

The listener name that is used for inter-broker communication. If this is not set, inter-broker communication is defined by the security.inter.broker.protocol property. Set one of these, but not both, or an error will occur. This property must be set on KRaft brokers, but note that you must also set this property for KRaft controllers because controllers sometimes need to talk to Kafka brokers in Confluent Platform.

The inter-broker listener name for a controller node **must not** appear in controller.listener.names property, and this applies regardless of whether the node is a controller in isolated or combined mode. Following is an example configuration file for a KRaft controller that shows how to configure this property as well as the security map for the listener.

process.roles=controller

node.id=100

controller.quorum.voters=100@node1:9093,101@node2:9093,102@node3:9093

controller.listener.names=CONTROLLER

listeners=CONTROLLER://:9093

inter.broker.listener.name=BROKER

listener.security.protocol.map=CONTROLLER:SSL,BROKER:SSL

 # Define the controller's listener and how we will use it.

listener.name.controller.ssl.keystore.location=/some/keystore/path

listener.name.controller.ssl.truststore.location=/some/truststore/path

 # etc...

 # Define how we will use the broker's listener.

 # No keystore needed since the controller isn't listening here; only need a truststore.

listener.name.broker.ssl.truststore.location=/some/truststore/path

 # etc...

listener.security.protocol.map

The security protocol to use for inter-broker communication specified by the inter.broker.listener.name property. The security protocol to use for the declared listener names. Note that this includes controller-to-broker communication with the listener identified by the inter.broker.listener.name property for the controller.

security.inter.broker.protocol

Security protocol used to communicate between brokers. Set this property or inter.broker.listener.name, but not both.

Other listeners and logs

Following are additional properties you should be familiar with.

listeners

A comma-separated list of addresses where the socket server listens.

For controllers in isolated mode: Only controller listeners are allowed in this list when process.roles=controller, and this listener should be consistent with controller.quorum.voters value. If not configured, the host name will be equal to the value of java.net.InetAddress.getCanonicalHostName() with the PLAINTEXT listener name, and port 9092.

For controllers in combined mode, you should list the controller listeners as well as the broker listeners.

- Type: string with the format listener_name://host_name:port
- Default: If not configured, the host name will be equal to the value of java.net.InetAddress.getCanonicalHostName(), with PLAINTEXT listener name, and port 9092. Example: listeners=PLAINTEXT://your.host.name:9092
- Importance: high

metadata.log.dir

Use to specify where the metadata log for clusters in KRaft mode is placed after storage is formatted. If not set, the metadata log is placed in the first log directory specified in the log-dirs property described below.

- Type: string
- Default: null
- Importance: high

log.dirs

If metadata.log.dir is not specified, the KRaft metadata log is placed in the first log directory specified by this property after storage is formatted.

- Type: string
- Default: null
- Importance: high

Controller configuration example

You can find the example KRaft configuration files in /etc/kafka/. You will see three different example files in this folder after you install Confluent Platform:

- broker.properties - An example of the settings to use when the server is a broker only.
- controller.properties - An example of the settings to use when the server is a controller only.

- server.properties - An example of the settings to use when the server is both a broker and a controller. This configuration is not supported for production use.

Following is an example excerpt from a properties file for a controller on a system with three controllers.

############################## Server Basics ##############################

 # The role of this server. Setting this puts us in KRaft mode.

process.roles=controller

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

node.id=1

 # The connect string for the controller quorum.

controller.quorum.voters=1@controller1.example.com:9093,2@controller2.example.com:9093,3@controller3.example.com:9093

############################## Socket Server Settings ##############################

 # The address the socket server listens on.

 # Note that only the controller listeners are allowed here when `process.roles=controller`, and this listener should be consistent with `controller.quorum.voters` value.

 # FORMAT:

 # listeners = listener_name://host_name:port

 # EXAMPLE:

 # listeners = PLAINTEXT://your.host.name:9092

listeners=CONTROLLER://controller1.example.com:9093

 # A comma-separated list of the names of the listeners used by the controller.

 # This is required if running in KRaft mode.

controller.listener.names=CONTROLLER

 # How to communicate with brokers.

inter.broker.listener.name=BROKER

 # Maps listener names to security protocols, the default is for them to be the same.

listener.security.protocol.map=CONTROLLER:SSL,BROKER:SSL

############################## Log Basics ##############################

 # A comma separated list of directories under which to store log files

log.dirs=/tmp/kraft-controller-logs

 # ... # Additional property settings to match broker settings.

Other properties

KRaft controllers need a property only if:

1. The controller process itself uses that property to perform its controller duties, or
2. The property affects cluster metadata that controllers manage

Brokers handle everything else—even if it's cluster-wide configuration. If a property is only consumed by brokers for data-plane operations (like reading or writing data), it does not need to be in the controller's configuration file.

For example:

- confluent.schema.registry.url: Controllers need this because Schema Registry integration affects metadata operations that controllers coordinate. Controllers must validate schema references during topic creation and modification, which is a metadata-layer function.

- confluent.tier.enable: Controllers don’t need this because tiered storage is a broker-only, data-plane feature. Tiered storage operations (uploading and downloading segments to object storage) are only broker responsibilities.

Based on this rule, many properties related to topic management and cluster-wide behavior must be present on the controllers because they affect cluster metadata that controllers manage. This includes security properties such as the truststore locations needed for secure communication with brokers.

The following list provides an example of common settings that are required on KRaft controllers because they relate to cluster metadata. This is not an exhaustive list.

- auto.create.topics.enable
- compression.type
- confluent.metrics.reporter.bootstrap.servers
- confluent.license.topic.replication.factor
- confluent.metadata.topic.replication.factor
- default.replication.factor
- delete.topic.enable
- message.max.bytes
- metrics.reporters
- min.insync.replicas
- num.partitions
- offsets.retention.minutes
- offsets.topic.replication.factor
- transaction.state.log.replication.factor
- transaction.state.log.min.isr
- unclean.leader.election.enable

Settings for other Kafka and Confluent Platform components

You must use current, non-deprecated, configurations settings. The settings to use are described in the following table.

Feature	Allowed with ZooKeeper	Required with KRaft
Clients and services	zookeeper.connect=zookeeper:2181	bootstrap.servers=broker:9092
Schema Registry	kafkastore.connection.url=zookeeper:2181	kafkastore.bootstrap.servers=broker:9092
Administrative tools	kafka-topics --zookeeper zookeeper:2181 (deprecated)	**kafka-topics --bootstrap-server broker:9092 …** --command-config properties to connect to brokers
Retrieve Kafka cluster ID	zookeeper-shell zookeeper:2181 get/cluster/id	From the command line, use kafka-metadata-quorum or confluent cluster describe --url, or view metadata.properties. or http://broker:8090 --output json

Generate and format IDs

Before you start Kafka, you must use the kafka-storage tool with the random-uuid command to generate a cluster ID for each new cluster. You only need one cluster ID, which you will use to format each node in the cluster.

bin/kafka-storage random-uuid

This results in output like the following:

q1Sh-9_ISia_zwGINzRvyQ

Then use the cluster ID to format storage for each node in the cluster with the kafka-storage tool that is provided with Confluent Platform, and the format command like the following example, specifying the properties file for a controller.

bin/kafka-storage format -t q1Sh-9_ISia_zwGINzRvyQ -c etc/kafka/controller.properties

Previously, Kafka would format blank storage directories automatically and generate a new cluster ID automatically. One reason for the change is that auto-formatting can sometimes obscure an error condition. This is particularly important for the metadata log maintained by the controller and broker servers. If a majority of the controllers were able to start with an empty log directory, a leader might be able to be elected with missing committed data. To configure the log directory, either set metadata.log.dir or log.dirs.

Tools for debugging KRaft mode

Kafka provides tools to help you debug a cluster running in KRaft-mode.

Describe runtime status

You can describe the runtime state of the cluster metadata partition using the kafka-metadata-quorum tool and specify either a Kafka broker with the --bootstrap-server option or a KRaft controller with the --bootstrap-controller option.

For example, the following command specifies a broker and displays a summary of the metadata quorum:

bin/kafka-metadata-quorum --bootstrap-server host1:9092 describe --status

Output might look like the following:

```
ClusterId:              fMCL8kv1SWm87L_Md-I2hg
LeaderId:               3002
LeaderEpoch:            2
HighWatermark:          10
MaxFollowerLag:         0
MaxFollowerLagTimeMs:   -1
CurrentVoters:          [3000,3001,3002]
CurrentObservers:       [0,1,2]
```

You can specify a controller with the --bootstrap-controller option. This is useful when the brokers are not accessible.

bin/kafka-metadata-quorum --bootstrap-controller host1:9093 describe --status

Debug log segments

The kafka-dump-log tool tool can be used to debug the log segments and snapshots for the cluster metadata directory. The tool will scan the provided files and decode the metadata records. For example, the following command decodes and prints the records in the first log segment:

bin/kafka-dump-log --cluster-metadata-decoder --files tmp/kraft-controller-logs/_cluster_metadata-0/00000000000000023946.log

Inspect the metadata partition

You can use the kafka-metadata-shell to inspect the metadata partition.

The Kafka version of the kafka-metadata-shell tool enables you to interactively examine the metadata stored in a KRaft cluster. To analyze metadata, point the kafka-metadata-shell tool to a log directory, run the kafka-metadata-shell.sh command, using the --directory flag to specify the path to your cluster metadata log.

./kafka-metadata-shell.sh --directory /tmp/kraft-combined-logs/__cluster_metadata-0/

Once the shell loads, you can use commands like ls and cat to explore the metadata records.

The Confluent Platform version of kafka-metadata-shell includes additional options for connecting directly to a running controller, which is necessary for clusters using security features like SSL/TLS.

The following is the help output for the kafka-metadata-shell tool.

kafka-metadata-shell.sh --help

usage: kafka-metadata-shell.sh [-h] [--cluster-id CLUSTER_ID] [--offset OFFSET] [--config CONFIG] (--directory DIRECTORY |

--controllers CONTROLLERS) [command [command ...]]

The Apache Kafka metadata shell tool

positional arguments:

command The command to run.

optional arguments:

-h, --help show this help message and exit

--cluster-id CLUSTER_ID, -t CLUSTER_ID

The cluster id. Required when using --controllers

--directory DIRECTORY, -d DIRECTORY

The __cluster_metadata-0 directory to read.

--controllers CONTROLLERS, -q CONTROLLERS

The controller.quorum.voters.

--offset OFFSET, -o OFFSET

The (exclusive) offset to read up to

--config CONFIG Path to property file containing a Kafka configuration

You use the --config flag to point to a client properties file that contains your security settings.

Example: Connecting to a controller over SSL

The following example shows how to connect to a controller that's configured to require SSL on its listener.

1. Create a client configuration file. This file contains the properties the shell needs to authenticate with the controller.

 # client.properties

 # Maps the listener name from the server to the SSL security protocol.

listener.security.protocol.map=CONTROLLER:SSL,PLAINTEXT:PLAINTEXT

 # Provides the location and password for your client's truststore.

ssl.truststore.location=/path/to/your/truststore.jks

ssl.truststore.password=your-password

ssl.truststore.type=JKS

2. Run the kafka-metadata-shell command, pointing to your controller, cluster ID, and the new client configuration file:

```
./bin/kafka-metadata-shell \
 --cluster-id your-cluster-id \
 --controllers controller-host:9593 \
 --config /path/to/your/client.properties
```

The shell uses the properties in the config file to establish a secure, trusted connection to the controller.

Check migration status

You can check the migration status of a KRaft cluster using Confluent-provided kafka-migration-check tool. This tool is included in Confluent Platform 7.9.2 or later, and can be found in the bin directory of your Confluent Platform installation.

Limitations and known issues

- Combined mode, where a Kafka node acts as a broker and also a KRaft controller, is not currently supported by Confluent. There are key security and feature gaps between combined mode and isolated mode in Confluent Platform.
- Versions of Confluent Platform older than Confluent Platform 7.9 do not support quorum reconfiguration, meaning you cannot add more KRaft controllers, or remove existing ones in your cluster. Confluent Platform 7.9 adds this feature. Three (3) or five (5) controllers are generally recommended for production, but you should have at least 3.
- You cannot currently use Schema Registry Topic ACL Authorizer for Confluent Platform for Schema Registry with Confluent Platform in KRaft mode. As an alternative, you can use Schema Registry ACL Authorizer for Confluent Platform or Configure Role-Based Access Control for Schema Registry in Confluent Platform.
- Currently, Health+ reports KRaft controllers as brokers and as a result, alerts may not function as expected.