Cloudera Runtime 7.3.2

Migrating Kafka clusters from ZooKeeper to KRaft in Cloudera on cloud

Date published: 2026-03-31
Date modified: 2026-03-31

https://docs.cloudera.com/

Legal Notice

Contents

Migrating Kafka from ZooKeeper to KRaft

Learn about migrating an existing Zookeeper-based Kafka cluster to KRaft. Migration is performed in Cloudera Manager using Kafka service actions, and requires minimal user interaction. Additionally, migration is done in a rolling fashion requiring no downtime.

Apache Kafka Raft (KRaft) is a consensus protocol used for metadata management that was developed as a replacement for Apache ZooKeeper. Using KRaft for managing Kafka metadata instead of ZooKeeper offers various benefits including a simplified architecture and a reduced operational footprint.

Important: Cloudera encourages you to migrate existing clusters to KRaft as soon as you have completed your upgrade to Cloudera Runtime 7.3.2. This is the only Cloudera Runtime version where migration is possible. Neither previous or future major, minor, and maintenance versions support migration.

Migration at glance

Migration is completed in two steps, start and finalize, with the option to revert at any time before finalization. You manage a migration (start, finalize, or revert) using Kafka service actions available in Cloudera Manager.

Most configuration changes needed for migration are handled by Cloudera Manager when you run migration actions. Before starting migration, you are only required to deploy KRaft Controller service roles and configure required policies in Ranger. All other changes are automated by the migration actions.

The migration actions in Cloudera Manager are as follows:

- Migrate Kafka to KRaft – Starts migration of Kafka brokers from ZooKeeper to KRaft for the specified Kafka service. Migrates the cluster up to a sate where reverting is still possible.

  When this action finishes, brokers will run in KRaft mode and will be disconnected from ZooKeeper. KRaft controllers will still be connected to ZooKeeper and will continue to write metadata to ZooKeeper, but are ready to disconnect (dual-write mode).
- Finalize KRaft Migration – Finalizes migration by disconnecting KRaft controllers from ZooKeeper. Reverting to ZooKeeper is not possible once this action starts.
- Revert KRaft Migration – Reverts migration of Kafka brokers from ZooKeeper to KRaft for the specified Kafka service using rolling restarts.
- Revert KRaft Migration (force restart) – Reverts migration of Kafka brokers from ZooKeeper to KRaft for the specified Kafka service with normal restarts (non-rolling). Expect service downtime when using this action.

You can only start these actions if your cluster meets the required prerequisites. If prerequisites are not met, the actions will be disabled.

Cluster health during migration

During migration and until migration is finalized with the Finalize KRaft Migration action, the health of the Kafka service will be in concerning (yellow) health. This is because the Migrate Kafka to KRaft action stops, starts, and updates the Kafka service roles during the migration process.

The Kafka service will remain in concerning health even after the Migrate Kafka to KRaft action completes successfully. This is because the KRaft Controller roles stay in concerning health until migration is finalized. The KRaft Controllers display one of two health test states that indicate the migration progress:

- **KRaft migration state is: Pre-Migration** – Controllers are in migration mode and waiting for brokers to join the migration. This is a transitional state during migration.
- **KRaft migration state is: Migration** – Migration is in progress and controllers are running in dual-write mode (writing to both ZooKeeper and KRaft). The Kafka Controllers will remain in this state until you run the Finalize KRaft Migration action.

Note: KRaft controller roles will have a health state test of **KRaft migration state is: Migration** before the Migrate Kafka to KRaft action finishes.

Related Information

Troubleshooting ZooKeeper to KRaft migration

Migrating a Kafka cluster to KRaft

You migrate an existing ZooKeeper-based Kafka cluster to KRaft by creating KRaft controllers, configuring required Ranger policies, and using the Migrate Kafka to KRaft Kafka service action in Cloudera Manager.

Before you begin

- Migrating an existing ZooKeeper-based Kafka cluster KRaft is only available if you are on Cloudera Manager 7.13.2 or later and Cloudera Runtime 7.3.2.

  Migration is only possible with this combination of versions. This is because:

  - Earlier Cloudera Manager versions do not include the necessary Kafka service actions.
  - Cloudera Runtime 7.3.2 is the only version where migration is possible. Neither previous or future major, minor, and maintenance versions support migration.
- Ensure that all Kafka Broker and ZooKeeper Server roles are running. In addition, ensure that the Kafka and ZooKeeper services do not have stale configurations.

  If you have stale configurations, resolve staleness by restarting the service or reverting configuration changes.
- Ensure that both the Kafka and ZooKeeper services are in a healthy (green) state.

  Migration is blocked if the Kafka service is in concerning (yellow) or bad (red) health.
- The Kafka service must run with inter-broker protocol version 3.9.

  Verify the version by checking the value of the Kafka Inter-Broker Protocol Version property in Cloudera Manager Kafka service Configuration . The value of the property must be 3.9 or empty. An empty value means that version is set to the default, which is 3.9 in Cloudera Runtime 7.3.2.

Tip: If the migration command is disabled in Cloudera Manager, your cluster does not satisfy requirements for migration. Check that all prerequisites are met.

Procedure

1. Add the kraft user to required Ranger policies and restrict access to the __cluster_metadata topic.
   a) In the Ranger Admin Web UI, select the Kafka resource-based service (default cm_kafka).
   b) Add the kraft user to all policies that include the kafka user.

      The kraft user must have the same permission in all policies as the kafka user.

      At minimum, you must add the kraft user to the following default policies:

      - all - consumergroup
      - all - topic
      - all - transactionalid
      - all - cluster
      - all - delegationtoken
      - connect internal - topic

      

      Tip: The kafka user/principal is the default for Kafka. If you have a custom Kerberos principals configured for the Kafka service, the user/principal might be different. Check the Cloudera Manager Kafka Configuration Kerberos Principal property. Its value is the default user/principal Kafka runs as.
   c) Create a new policy that restricts access to the __cluster_metadata topic with the following permissions:

      - kraft user – All permissions
      - kafka user – **Describe** (describe), **Describe Configs**(describe_configs), and **Consume** (consume).

      Policy example in JSON:

```
{
  "isEnabled": true,
  "service": "cm_kafka",
  "name": "kraft internal - topic",
  "policyType": 0,
  "policyPriority": 0,
  "description": "Policy for kraft internal - topic",
  "isAuditEnabled": true,
  "resources": {
    "topic": {
      "values": [
        "__cluster_metadata"
      ],
      "isExcludes": false,
      "isRecursive": false
    }
  },
  "policyItems": [
    {
      "accesses": [
        {
          "type": "create",
          "isAllowed": true
        },
        {
          "type": "delete",
          "isAllowed": true
        },
        {
          "type": "configure",
          "isAllowed": true
        },
        {
          "type": "alter",
          "isAllowed": true
        },
```

```
        {
          "type": "alter_configs",
          "isAllowed": true
        },
        {
          "type": "describe",
          "isAllowed": true
        },
        {
          "type": "describe_configs",
          "isAllowed": true
        },
        {
          "type": "consume",
          "isAllowed": true
        },
        {
          "type": "publish",
          "isAllowed": true
        }
      ],
      "users": [
        "kraft"
      ],
      "delegateAdmin": false
    },
    {
      "accesses": [
        {
          "type": "describe",
          "isAllowed": true
        },
        {
          "type": "describe_configs",
          "isAllowed": true
        },
        {
          "type": "consume",
          "isAllowed": true
        }
      ],
      "users": [
        "kafka"
      ],
      "delegateAdmin": false
    }
  ],
  "serviceType": "kafka",
  "isDenyAllElse": true
}
```

2. Add KRaft Controller service role instances to your cluster.
   a) In Cloudera Manager, select the Kafka service.
   b) Go to Configuration.
   c) Click Actions Add Role Instances .
   d) Click Select hosts under **KRaft Controller**.
   e) Select at least three cluster hosts and click OK.
   f) Click Continue.
   g) On the Review Changes page, configure the service role based on your cluster and requirements.
   h) Click Finish.

   Adding new roles causes configuration staleness in the Kafka service.

3. Restart the Kafka Broker and Kafka Connect service roles.

   Important: Do not start or restart the Kafka Controller roles you added.

4. Click Actions Migrate Kafka to KRaft .

   This action migrates your Kafka cluster to KRaft.

5. Wait until the action finishes.

Results

ZooKeeper to KRaft migration for Kafka succeeded. However, migration is not finalized. Brokers are running in KRaft mode and are disconnected from ZooKeeper. KRaft controllers are still connected to ZooKeeper and continue to write metadata to ZooKeeper (dual-write mode).

The Kafka service will be in a concerning (yellow) health state. This is because the KRaft Controller roles will stay in a concerning health with the **KRaft migration state is: Migration** health test state until the migration is finalized. This is normal and expected behavior. This health test state indicates that migration is not yet finalized.

What to do next

Finalize or revert the migration.

Related Information

Troubleshooting ZooKeeper to KRaft migration

Finalizing a KRaft migration

You finalize ZooKeeper to KRaft migration with the Finalize KRaft Migration Kafka service action in Cloudera Manager.

Important: Once you finalize migration, you cannot revert your cluster to use ZooKeeper. To revert your cluster, see Reverting a Kafka cluster to ZooKeeper on page 9.

Before you begin

- Ensure that the Migrate Kafka to KRaft action has successfully finished.

  If your KRaft Controller roles are in a concerning health state with the **KRaft migration state is: Migration** health test state, you can safely proceed with finalization. This health test state indicates that the service is ready for finalization.
- Ensure that all applications or services that connect to Kafka operate normally and can produce or consume messages.

Procedure

1. In Cloudera Manager select the Kafka service.
2. Click Actions Finalize KRaft Migration .
3. Wait until the action finishes.

Results

KRaft migration is finalized. It is not possible to revert this operation. KRaft roles are restarted and disconnected from ZooKeeper. The Kafka service is now running in KRaft mode.

Related Information

Troubleshooting ZooKeeper to KRaft migration

Reverting a Kafka cluster to ZooKeeper

You revert migration with the Revert KRaft Migration or Revert KRaft Migration (force restart) Kafka service actions in Cloudera Manager.

Before you begin

Reverting to ZooKeeper is only possible if the migration to KRaft is not finalized. A migration is not finalized if the KRaft Controller roles are in a concerning health state with the **KRaft migration state is: Migration** health test state.

Procedure

1. In Cloudera Manager, select the Kafka service.
2. Click the Revert KRaft Migration or Revert KRaft Migration (force restart) action.

   Both of the actions revert a cluster to using ZooKeeper and undo the changes made by the Migrate Kafka to KRaft action.

   The difference between the two actions is the type of restart used. The Revert KRaft Migration action uses rolling restarts. The Revert KRaft Migration (force restart) action uses normal (non-rolling) restarts.

   Cloudera recommends that you use Revert KRaft Migration. If the action fails or rolling restarts are not required, use Revert KRaft Migration (force restart).
3. Wait until the command finishes.

Results

Migration is successfully rolled back. The Kafka cluster uses ZooKeeper for metadata management

Related Information

Troubleshooting ZooKeeper to KRaft migration

Troubleshooting ZooKeeper to KRaft migration

Common issues and solutions for troubleshooting problems that might occur during the migration of Kafka clusters from ZooKeeper to KRaft mode.

KRaft Controllers were started without migration configuration

Condition

When KRaft Controller roles are added to the cluster in Cloudera Private Cloud Base, by default they are stopped, but it is possible to start them without enabling migration mode. If this is done, then the controllers start the KRaft cluster in full KRaft mode. This is a problem because the internal KRaft migration state will be initialized in the terminal KRaft cluster state. Migration will not happen and cannot be initiated in this state.

Confirm this by looking at Cloudera Manager command history and the health history of the controller instances. If KRaft Controllers are running but there was no migration started, then controllers were started manually.

Further confirmation can be done by reviewing the kafka_zk_migration_state metric. Use the following query in Charts Chart Builder to review the metric.

```
SELECT kafka_zk_migration_state
```

A metric value of 0 indicates that controllers were started without enabling migration mode and the cluster is started in full KRaft mode.

Cause

KRaft Controller role instances were added to the cluster and started manually without migration configuration. That is, they were started manually and not through the Migrate Kafka to KRaft action.

Remedy

Procedure

1. Stop the KRaft Controller roles.
2. Wipe the log directories of the KRaft Controller roles.
   This resets the controllers to a state where migration can be initiated.
3. Start migration with the Migrate Kafka to KRaft.

Newly provisioned KRaft controllers cannot maintain leader or quorum

Condition

The KRaft controllers cannot maintain a stable leader election or quorum, the cluster's core metadata management is at risk. The active controller chart is frequently updated with different controllers assuming leadership.

Cause

Networking (high likelihood) or disk issues (low likelihood).

Remedy

Collect diagnostic data and troubleshoot potential issues in your cluster. Look for possible networking or disk issues. In stretch clusters, you will most likely need to adjust buffer sizes.

Broker connectivity issues

Condition

Brokers fail to correctly register with the KRaft controller quorum after their configuration is updated and started in migration mode.

Cause

Brokers show concerning (yellow) or bad (red) health in Cloudera Manager. Additionally, the logs show that brokers cannot register to the KRaft quorum.

Remedy

Determine if the issue is a misconfiguration or a bug in the migration integration. Likely causes of misconfiguration can be a custom listener setup that the migration integration was not prepared for, or authentication issues.

The remedy is highly dependent on the specific problem, and you will need to review logs to find the root cause. Before you review logs, ensure that brokers are connected to ZooKeeper and serve traffic normally.

Resume migration if the issue is resolved. Alternatively, revert the migration.

Performance degradation

Condition

Experiencing unexpected or significant performance drops, high latency, or increased resource utilization (CPU/memory) on the brokers or controllers. Relevant charts in Cloudera Manager or in any other metrics monitoring applications show the performance degradation of brokers, controllers, or clients.

Cause

This issue might surface during migration, but it might be unrelated to it. Consumer and Connect group rebalances might cause a performance degradation with unbalanced clusters.

Remedy

Revert migration and troubleshoot performance issues in your cluster. If the issues remain even after a revert, then it is highly likely that the performance degradation is caused by an inherent cluster imbalance.

Metadata consistency issues

Condition

Observing errors in metadata synchronization between the KRaft quorum and ZooKeeper during the dual-write phase and during migration to KRaft mode.

You might experience issues like:

- Brokers have inconsistent information about partition leaders, causing partitions to fall out of the ISR.
- Topic configurations might differ for topics in Zookeeper and in KRaft.
- Brokers might be different in Zookeeper and KRaft metadata.

Cause

As a result of network issues, bugs, unsupported configurations, or misconfiguration, metadata consistency issues might happen when the brokers are being restarted in KRaft mode. During this phase, a part of the cluster already reads metadata from KRaft, while the other parts still use ZooKeeper. Parts of the cluster that still use ZooKeeper might learn about updates much later if there is a network glitch or a bug.

Remedy

- Do network diagnostics to ensure that buffers are set up for the latency of the network (especially in stretch clusters).
- Check DNS response times. Specifically check if Kafka spends a lot of time resolving DNS addresses. Sometimes DNS resolution problems can cause unstable messaging in Kafka.
- Check for authentication or encryption issues. A slow or inconsistent Key Distribution Center (KDC) might cause connections or authentication requests to lag. This in turn slows down in-sync replicas (ISR) and UpdateMetadata requests which may affect metadata propagation.
- Consider reverting the migration, fixing any network issues, then restart the migration.

Client compatibility

Condition

A critical application or tooling component (especially older ones) relies directly on ZooKeeper paths or functionality that is broken during the dual-write phases. Such errors have a very low chance since the last Java client that directly accesses ZooKeeper has been removed in Kafka 0.10, but third-party clients not supported by Cloudera might still work with ZooKeeper connections. Ideally, this problem should only surface in development or test environments.

Cause

Client applications cannot communicate with the brokers. The crash happens when the brokers enter their dual-write phase.

Remedy

Revert the migration and upgrade your clients so that they are compatible with KRaft. Restart migration afterwards.

Migration fails with `NoAuthException`

Condition

Zookeeper Access Control List (ACL) synchronization is done by the migration command as a preparatory step. Any subsequent manual changes to the ACLs can potentially break the migration. Such issues can happen if you use Kafka CLI tools to modify topic configuration or change the topics. Essentially, changing the ZooKeeper structure with any CLI commands can break migration.

Kafka or KRaft migration fails in any step and the Kafka or KRaft logs include `NoAuthException` stack traces, similar to the following example:

```
Exception in thread "main" org.apache.zookeeper.KeeperException$NoAuthExcept
ion: KeeperErrorCode = NoAuth for /kafka-specific-node
```

Cause

ZooKeeper ACLs were changed during migration. Kafka command line tools were used to update topics or their configurations.

Remedy

Procedure

1. Apply the same ACLs to the node that is printed in the exception as all the others.
2. Resume migration.

The kraft user is not authorized to perform actions

Condition

Authorization errors are included in the Kafka and KRaft logs that say that the kraft user is not authorized to perform actions.

Cause

Ranger policies are not updated for KRaft, or are misconfigured.

Remedy

Procedure

1. Add the kraft user to required Ranger policies.
   a) In the Ranger Admin Web UI, select the Kafka resource-based service (default cm_kafka).
   b) Add the kraft user to all policies that include the kafka user.

      The kraft user must have the same permission in all policies as the kafka user.

      At minimum, you must add the kraft user to the following default policies:

      - all - consumergroup
      - all - topic
      - all - transactionalid
      - all - cluster
      - all - delegationtoken
      - connect internal - topic

      

      Tip: The kafka user/principal is the default for Kafka. If you have a custom Kerberos principals configured for the Kafka service, the user/principal might be different. Check the Cloudera Manager Kafka Configuration Kerberos Principal property. Its value is the default user/principal Kafka runs as.
2. Create a new policy that restricts access to the __cluster_metadata topic with the following permissions:

   - kraft user – All permissions
   - kafka user – **Describe** (describe), **Describe Configs**(describe_configs), and **Consume** (consume).

   Policy example in JSON:

```
{
  "isEnabled": true,
  "service": "cm_kafka",
  "name": "kraft internal - topic",
  "policyType": 0,
  "policyPriority": 0,
  "description": "Policy for kraft internal - topic",
  "isAuditEnabled": true,
  "resources": {
    "topic": {
      "values": [
        "__cluster_metadata"
      ],
      "isExcludes": false,
      "isRecursive": false
    }
  },
  "policyItems": [
    {
      "accesses": [
        {
          "type": "create",
          "isAllowed": true
        },
        {
          "type": "delete",
          "isAllowed": true
        },
        {
```

```
          "type": "configure",
          "isAllowed": true
        },
        {
          "type": "alter",
          "isAllowed": true
        },
        {
          "type": "alter_configs",
          "isAllowed": true
        },
        {
          "type": "describe",
          "isAllowed": true
        },
        {
          "type": "describe_configs",
          "isAllowed": true
        },
        {
          "type": "consume",
          "isAllowed": true
        },
        {
          "type": "publish",
          "isAllowed": true
        }
      ],
      "users": [
        "kraft"
      ],
      "delegateAdmin": false
    },
    {
      "accesses": [
        {
          "type": "describe",
          "isAllowed": true
        },
        {
          "type": "describe_configs",
          "isAllowed": true
        },
        {
          "type": "consume",
          "isAllowed": true
        }
      ],
      "users": [
        "kafka"
      ],
      "delegateAdmin": false
    }
  ],
  "serviceType": "kafka",
  "isDenyAllElse": true
}
```