Upgrading & Migrating Cloudera Streams Messaging - Kubernetes Operator

Date published: 2024-06-11
Date modified: 2025-02-28

Legal Notice

Contents

Upgrading and rolling back Cloudera Streams Messaging - Kubernetes Operator

Learn about upgrading and rolling back Cloudera Streams Messaging - Kubernetes Operator.

Upgrades

Upgrading Cloudera Streams Messaging - Kubernetes Operator consists of upgrading Strimzi, Kafka, and Kafka Connect.

Upgrading Strimzi involves upgrading the Strimzi Custom Resource Definitions (CRD) and upgrading the Strimzi Cluster Operator using Helm.

Upgrading Kafka and Kafka Connect involves updating your `Kafka` and `KafkaConnect` resources so that they specify the latest supported Kafka version. Upgrading Kafka and Kafka Connect is done following a Strimzi upgrade. Upgrading Kafka and Kafka Connect is:

- Strongly recommended by Cloudera after every Strimzi upgrade.
- Mandatory if you are upgrading from a maintenance version or if you are using a custom Kafka image.

Upgrading Strimzi might affect the `Kafka` and `KafkaConnect` resources in the cluster. All Kafka and Kafka Connect clusters that specify the version of the cluster but not the image are restarted when you upgrade Strimzi. This is due to the fact that the default image of all versions changes with the Strimzi upgrade, triggering a restart. This restart is safe, that is, all healthy topics with a proper replication factor and minimum ISR are kept online during the restart.

Rollbacks

A rollback involves rolling back Kafka and Kafka Connect as well as Strimzi to a previous version.

A rollback is only possible if the Kafka version you are upgrading from is still supported by the Cloudera Streams Messaging - Kubernetes Operator version you are upgrading to. If the Kafka version is not supported, upgrading is possible, but rollback is not. If you are upgrading to a version that does not support your current Kafka version and you want to have the ability to roll back, you might need to carry out multiple upgrades.

Upgrading Cloudera Streams Messaging - Kubernetes Operator

To upgrade Cloudera Streams Messaging - Kubernetes Operator, you upgrade Strimzi as well as your Kafka and Kafka Connect clusters.

Before you begin

- Ensure that your Kubernetes environment meets requirements listed in System requirements.
- Ensure that you have access to your Cloudera credentials (username and password). Credentials are required to access the Cloudera Archive and Cloudera Docker registry where upgrade artifacts are hosted.
- If you are upgrading from a maintenance version, check that the bug fixes provided in the maintenance version are available in the newer Kafka supported by Cloudera Streams Messaging - Kubernetes Operator. If certain fixes are not available, be aware that upgrading will result in regressed functionality.

- If you are using a custom Kafka image, build new custom images that are based on the Kafka images shipped with the Cloudera Streams Messaging - Kubernetes Operator version you are upgrading to.

  Cloudera Streams Messaging - Kubernetes Operator ships with two Kafka images, one for the latest supported version of Kafka, and one for the previously supported version of Kafka. Build new custom images based on both.

  The custom image based on the latest supported version is required for the upgrade. The custom image based on the previously supported version is required for rollbacks.

  Note: If you have Ranger integration set up, you are using a custom image. Building new custom images is required.

- The following steps instruct you to set inter.broker.protocol.version during the upgrade to keep Kafka in backward compatible state during the upgrade. This is only necessary if the protocol version differs between your current and new versions.

  If there is no change in the protocol version, you also do not finalize the upgrade. This means that for these types of upgrades, a rollback is possible even after you completed the upgrade.

  You can find the Kafka protocol version in Component versions.

Procedure

1. Update the Strimzi CRDs.

   You do this by replacing the currently installed CRDs with the new version. The CRDs are published as a single YAML on the Cloudera Archive in /p/csm-operator/1.3/install.

   ```
   curl -s --user [***USERNAME***] \
     https://archive.cloudera.com/p/csm-operator/1.3.0/install/strimzi-crd
   s-0.45.0.1.3.0-b52.yaml \
   | kubectl replace --filename -
   ```

   Enter your Cloudera password when prompted.

2. If you are upgrading from a maintenance version or are using a custom Kafka image, pause reconciliation for all your Kafka clusters.

   Important: During the period when reconciliation is paused, your cluster is not managed by Strimzi. Even if a pod crashes, it is not restarted. Ensure that you prepare for this time window to avoid accidental issues.

   You pause reconciliation by setting the strimzi.io/pause-reconciliation annotation to true in the `Kafka` resource.

   ```
   kubectl annotate kafka [***KAFKA CLUSTER NAME***] /
     --namespace [***KAFKA NAMESPACE***] /
      strimzi.io/pause-reconciliation="true"
   ```

   Pausing reconciliation ensures that the Kafka cluster does not get automatically updated during the upgrade of Strimzi. Reconciliation is only paused temporarily. You re-enable it after the Strimzi upgrade. Ensure that you add the annotation to all your Kafka clusters.

3. If you are upgrading from a maintenance version, or you are using a custom Kafka image, pause reconciliation for all your Kafka Connect clusters.

   Important: During the period when reconciliation is paused, your cluster is not managed by Strimzi. Even if a pod crashes, it is not restarted. Ensure that you prepare for this time window to avoid accidental issues.

   You pause reconciliation by setting the strimzi.io/pause-reconciliation annotation to true in the `KafkaConnect` resource.

   ```
   kubectl annotate kafkaconnect [***KAFKA CONNECT CLUSTER NAME***] /
     --namespace [***KAFKA CONNECT NAMESPACE***] /
   ```

```
    strimzi.io/pause-reconciliation="true"
```

Pausing reconciliation ensures that the Kafka Connect cluster does not get automatically updated during the upgrade of Strimzi. Reconciliation is only paused temporarily. You reenable it after the Strimzi upgrade. Ensure that you add the annotation to all your Kafka Connect clusters.

4. Log in to the Cloudera Docker registry with helm.

```
helm registry login container.repository.cloudera.com
```

Enter your Cloudera credentials when prompted.

5. Upgrade Strimzi using helm upgrade.

This step upgrades the Strimzi Cluster Operator. Under the hood, the Strimzi Cluster Operator deployment is updated by changing the image used by the Strimzi Cluster Operator.

```
helm upgrade strimzi-cluster-operator \
  --namespace [***STRIMZI CLUSTER OPERATOR NAMESPACE***] \
  --atomic \
  oci://container.repository.cloudera.com/cloudera-helm/csm-operator/str
imzi-kafka-operator \
--version 1.3.0-b52
```

- The string strimzi-cluster-operator is the Helm release name of the chart installation. This is an arbitrary, user-defined name. Replace this string if you used a different name when you installed Strimzi.
- The --atomic option makes the command wait for the upgrade to complete or roll back if the upgrade fails.

Note: Usually, you do not need to set new Helm chart properties during upgrade, the properties that you configured during installation are preserved. However, If you set even a single property using the --set options in your helm upgrade command, properties set previously might be ignored. If you decide to set properties during the upgrade, ensure that you do one of the following.

- Set all your properties (including the ones that were set during installation) during upgrade. This ensures that all required properties are set explicitly.
- Set the properties you want to update and use the --reset-then-reuse-values option. Using this option preserves previously set properties. This option is only available in the more recent versions of Helm.

6. Verify that the Strimzi upgrade is successful.

To do this, check that the Strimzi Cluster Operator deployment is in a healthy state.

```
kubectl get deployments strimzi-cluster-operator \
  --namespace [***STRIMZI CLUSTER OPERATOR NAMESPACE***]
```

7. Upgrade Kafka.

To do this, update your `Kafka` resources for each cluster.

```
kubectl edit kafka [***KAFKA CLUSTER NAME***] \
  --namespace [***KAFKA NAMESPACE***]
```

a) Set spec.kafka.version to the latest version.
b) If you are using a custom image, set your new custom image that corresponds with the new Kafka version in spec.kafka.image.
c) Add the old inter.broker.protocol.version in spec.kafka.config.

```
#...
kind: Kafka
spec:
  kafka:
    version: 3.9.0.1.3
    config:
      inter.broker.protocol.version: “3.8”
```

   Specifying inter.broker.protocol.version keeps Kafka in a backward-compatible state following the upgrade.
   d) Save the changes made to the `Kafka` resource.
   e) If you paused reconciliation of the Kafka cluster, remove the strimzi.io/pause-reconciliation annotation.

      This re-enables reconciliation for the cluster.

      ```
      kubectl annotate kafka [***KAFKA CLUSTER NAME***] /
        --namespace [***KAFKA NAMESPACE***] /
        strimzi.io/pause-reconciliation-
      ```

8. Verify that the Kafka upgrade is successful.

   Upgrade is successful once the `Kafka` resources are in a ready state.

   ```
   kubectl get kafkas \
     --namespace [***KAFKA NAMESPACE***] \
     --watch
   ```

9. Upgrade Kafka Connect.

   To do this, update your `KafkaConnect` resources for each cluster.

   ```
   kubectl edit kafkaconnect [***KAFKA CONNECT CLUSTER NAME***] \
     --namespace [***KAFKA CONNECT NAMESPACE***]
   ```

   a) Set spec.version to the latest version.
   b) If you are using a custom image, set your new custom image that corresponds with the new Kafka version in spec.image.
   c) Save the changes made to the `KafkaConnect` resource.
   d) If you paused reconciliation of the Kafka Connect cluster, remove the strimzi.io/pause-reconciliation annotation.

      This re-enables reconciliation for the cluster.

      ```
      kubectl annotate kafkaconnect [***KAFKA CONNECT CLUSTER NAME***] /
        --namespace [***KAFKA CONNECT NAMESPACE***] /
        strimzi.io/pause-reconciliation-
      ```

10. Verify that the Kafka Connect upgrade is successful.

    Upgrade is successful once the `KafkaConnect` resources are in a ready state.

    ```
    kubectl get kafkaconnects \
      --namespace [***KAFKA CONNECT NAMESPACE***] \
      --watch
    ```

11. Finalize the Kafka upgrade by removing the old inter.broker.protocol.version from your `Kafka` resources.

    After this step, rollback is no longer possible.

Rolling back Cloudera Streams Messaging - Kubernetes Operator

To roll back Cloudera Streams Messaging - Kubernetes Operator to a previous version, you roll back your Kafka and Kafka Connect clusters as well as Strimzi.

Before you begin

- Rollbacks are only possible if:
    - The Kafka version you upgraded from is still supported by the Cloudera Streams Messaging - Kubernetes Operator version that you upgraded to.
    - The Kafka upgrade is not finalized. A Kafka upgrade is finalized if inter.broker.protocol.version for the Kafka cluster is set to the current protocol version. That is, the protocol version matches the version of Kafka.
- If you are using a custom Kafka image:
    - Ensure that you have a custom Kafka image that is based on the latest base image with the previous Kafka version. This image is only set and used temporarily during the rollback.

      For example, Cloudera Streams Messaging - Kubernetes Operator 1.3 ships two Kafka base images. One for Kafka 3.9.0.1.3 (latest/current) and one for Kafka 3.8.0.1.2 (previous). In order to rollback, you need a custom image based on the 3.8.0.1.2 image shipped with 1.3.

      The base image used to build the custom image must be an image shipped in the Cloudera Streams Messaging - Kubernetes Operator version you upgraded to. Even though earlier Cloudera Streams Messaging - Kubernetes Operator versions might have shipped the same Kafka version, you cannot use a base image from a previous Cloudera Streams Messaging - Kubernetes Operator release even if the Kafka version is the same.
    - Ensure that you have access to the original custom image that you used before you upgraded. You will be rolling back your clusters to use this image.
    - If you are rolling back to a maintenance version, it can happen that during the rollback your cluster temporarily runs with a Kafka image that does not have all fixes included in the maintenance version.

      This is because a rollback is done in two stages. First, you roll back Kafka and Kafka Connect, then you roll back Strimzi. After rolling back Kafka or Kafka Connect your clusters are automatically restarted and use a Kafka image that includes the previous version of Kafka. However, that image might not contain all fixes that are in the maintenance version you are rolling back to. This is only temporary, once you rollback Strimzi, your cluster will fully revert to using the initial maintenance version you upgraded from.

Procedure

1. Roll back Kafka.

   To do this, edit your `Kafka` resources.

   ```
   kubectl edit kafka [***KAFKA CLUSTER NAME***] \
     --namespace [***KAFKA NAMESPACE***]
   ```

   a) Set spec.kafka.version to the previous version.
   b) If you are using a custom Kafka image, set spec.kafka.image to a custom Kafka image that is based on the latest base image with the previous Kafka version.

      The image you set here is not the original custom image that was used before you completed the upgrade. Strimzi at this stage is still upgraded and is unable to manage your original image. Because of this, you must specify a custom Kafka image that is based on the latest base image with the previous Kafka version.
   c) Save the changes made to the `Kafka` resource.

   After the changes are saved, the Kafka cluster is restarted in a rolling fashion.
2. Roll back Kafka Connect.

   To do this, edit your `KafkaConnect` resources.

   ```
   kubectl edit kafkaconnect [***KAFKA CONNECT CLUSTER NAME***] \
   ```

```
  --namespace [***KAFKA CONNECT NAMESPACE***]
```

a) Set spec.version to the previous version.
b) If you are using a custom Kafka image, set spec.image to a custom Kafka image that is based on the latest base image with the previous Kafka version.

   The image you set here is not the original custom image that was used before you completed the upgrade. Strimzi at this stage is still upgraded and is unable to manage your original image. Because of this, you must specify a custom Kafka image that is based on the latest base image with the previous Kafka version
c) Save the changes made to the `KafkaConnect` resource.

After the changes are saved, the Kafka Connect cluster is restarted in a rolling fashion.

3. If you are using a custom Kafka image, pause reconciliation of the Kafka clusters.

   To do this, add the strimzi.io/pause-reconciliation="true" annotation to your `Kafka` resources.

```
kubectl annotate kafka [***KAFKA CLUSTER NAME***] /
  --namespace [***KAFKA NAMESPACE***] /
   strimzi.io/pause-reconciliation="true"
```

4. If you are using a custom Kafka image, pause reconciliation of the Kafka Connect clusters.

   To do this, add the strimzi.io/pause-reconciliation="true" annotation to your `KafkaConnect` resources.

```
kubectl annotate kafkaconnect [***KAFKA CONNECT CLUSTER NAME***] /
  --namespace [***KAFKA CONNECT NAMESPACE***] /
   strimzi.io/pause-reconciliation="true"
```

5. Roll back Strimzi with helm.

```
helm rollback strimzi-cluster-operator \
  --namespace [***STRIMZI CLUSTER OPERATOR NAMESPACE***]
```

Note: If you did not pause reconciliation, both Kafka and Kafka Connect clusters managed by the Strimzi Cluster Operator are restarted in a rolling fashion when you roll back Strimzi.

6. Verify that the Strimzi rollback is successful.

   To do this, check that the Strimzi Cluster Operator deployment is in a healthy state.

```
kubectl get deployments strimzi-cluster-operator \
  --namespace [***STRIMZI CLUSTER OPERATOR NAMESPACE***]
```

7. If you are using a custom Kafka image, set spec.kafka.image in your `Kafka` resources to the original custom Kafka image that was in use before you completed the upgrade.
8. If you are using a custom Kafka image, set spec.image in your `KafkaConnect` resources to the original custom Kafka image that was in use before you completed the upgrade.
9. If you paused reconciliation for Kafka clusters, remove the strimzi.io/pause-reconciliation annotation from your `Kafka` resources.

   This re-enables reconciliation for the cluster.

```
kubectl annotate kafka [***KAFKA CLUSTER NAME***] /
  --namespace [***KAFKA NAMESPACE***] /
   strimzi.io/pause-reconciliation-
```

10. If you paused reconciliation for Kafka Connect clusters, remove the strimzi.io/pause-reconciliation annotation from your `KafkaConnect` resources.

    This re-enables reconciliation for the cluster.

```
kubectl annotate kafkaconnect [***KAFKA CONNECT CLUSTER NAME***] /
  --namespace [***KAFKA CONNECT NAMESPACE***] /
```

```
    strimzi.io/pause-reconciliation-
```

11. Verify that both Kafka and Kafka Connect rollbacks are successful.

Rollback is successful once your `Kafka` and `KafkaConnect` resources are in a ready state.

```
kubectl get kafkas \
  --namespace [***KAFKA NAMESPACE***] \
  --watch
```

```
kubectl get kafkaconnects \
  --namespace [***KAFKA CONNECT NAMESPACE***] \
  --watch
```

Migrating Kafka clusters from ZooKeeper to KRaft

Learn about migrating an existing Zookeeper-based Kafka cluster to KRaft. Migration is semi-automatic and requires minimal configuration changes. Additionally, migration is done in a rolling fashion and requires 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.

Starting with Cloudera Streams Messaging - Kubernetes Operator 1.3, ZooKeeper is deprecated and will be removed in a future release. Cloudera encourages you to migrate your existing clusters to KRaft.

Migration at glance

Migration is semi-automatic. You manage a migration (start, finalize, or roll back) using the strimzi.io/kraft annotation. You add the annotation to the `Kafka` resource of the cluster.

The majority of configuration changes needed in your cluster are handled by the Strimzi Cluster Operator when you add the strimzi.io/kraft annotation. Other than adding annotation, you are only required to deploy KRaft controllers before starting migration, and clean up unused configuration properties after migration. All other steps are automated.

The annotation has the following valid values.

- migration - Starts the migration process and migrates a cluster up to a state where rollbacks are still possible.
- enabled - Finalizes a migration. Once finalization is triggered, rollbacks to ZooKeeper are no longer possible.
- rollback - Reverts the cluster from Kraft to ZooKeeper. Rollbacks are only possible for clusters where migration is not finalized.
- disabled - The default value of the annotation. Indicates that the cluster uses ZooKeeper. Also used to finalize a rollback.

Cluster metadata states during migration

When you apply strimzi.io/kraft annotation to a Kafka resource, the Strimzi Cluster Operator performs various migration-related actions on the cluster. During a migration, the cluster goes through a number of metadata states. You can monitor these metadata states to see the progress of migration. The metadata state of the cluster is also an indicator that shows whether a cluster can be rolled back.

The metadata states that the cluster goes through during a migration and a rollback are as follows.

Migration

- ZooKeeper - Indicates that the cluster uses ZooKeeper. This is the initial state. Migration is started and the cluster is moved to the next state when the strimzi.io/kraft="migration" annotation is added to the Kafka resource.

- KRaftMigration - KRaft controllers are running, metadata and brokers are getting migrated to KRaft. In this state some brokers are already running in KRaft mode while others are still using ZooKeeper.
- KRaftDualWriting - Metadata migration is finished. However, both brokers and controllers are still connected to ZooKeeper. Metadata is being written by KRaft controllers to ZooKeeper.
- KRaftPostMigration - Brokers are running in KRaft mode and have been disconnected from ZooKeeper. KRaft controllers are connected to ZooKeeper and continue to write metadata to ZooKeeper, but are ready to disconnect.

  This metadata state marks the end of the first migration phase. The Strimzi Cluster Operator only moves the cluster to the next state when you update the strimzi.io/kraft annotation to enab led. This is the last state where rollback is still possible.
- PreKRaft - KRaft controllers are disconnected from ZooKeeper. Zookeeper is ready to be removed.
- KRaft - Migration is finalized. Both Kafka brokers and controllers are running in KRaft mode. ZooKeeper is disconnected from the cluster. ZooKeeper instances are removed.

Rollback

- KRaftPostMigration - Brokers are running in KRaft mode and have been disconnected from ZooKeeper. KRaft controllers are connected to ZooKeeper and continue to write metadata to ZooKeeper, but are ready to disconnect. This is the initial state when you begin a rollback.
- KRaftDualWriting - Both brokers and controllers are connected to ZooKeeper. Metadata is being written by KRaft controllers to ZooKeeper as well. In case of a rollback, the Strimzi Cluster Operator stops at this metadata state and waits until you start finalization by setting the strimzi.io/kraft annotation to disabled.
- ZooKeeper - Indicates that the cluster uses ZooKeeper. This is the final state during a rollback.

Migrating a Kafka cluster to KRaft

You migrate an existing ZooKeeper-based Kafka cluster to KRaft by creating KRaft controllers and then using the strimzi.io/kraft annotation on the Kafka resource of the cluster to initiate and finalize the migration. After finalization, you remove configuration related to ZooKeeper.

Before you begin

Ensure the following:

- You are on Cloudera Streaming Kubernetes Operator 1.3 or higher.
- The Kafka cluster version is 3.9.0.1.3 or higher.

Migration is not supported from lower versions. If you are on a lower version, upgrade to a supported version.

Procedure

1. Create a YAML configuration containing your `KafkaNodePool` resource manifest for KRaft controllers.

   The `KafkaNodePool` configuration must have spec.roles set to controller. Migrating with a Kafka node in combined mode (has both broker and controller roles) is not supported. Ensure that you only set controller as the role.

   Deploy as many controllers as you have ZooKeeper nodes. Additionally, Cloudera strongly recommends that you deploy an uneven number of controllers.

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaNodePool
metadata:
  name: controller
  labels:
    strimzi.io/cluster: my-cluster
spec:
  replicas: 3
  roles:
    - controller
  storage:
    type: jbod
    volumes:
      - id: 0
        type: persistent-claim
        size: 100Gi
        deleteClaim: false
```

2. Deploy the `KafkaNodePool` resource.

This deploys the KRaft controllers.

```
kubectl apply \
  --filename [***NODE POOL YAML***] \
  --namespace [***NAMESPACE***]
```

3. Start migration by annotating your Kafka resource with `strimzi.io/kraft="migration"`.

```
kubectl annotate kafka [***CLUSTER NAME***] \
  --namespace [***NAMESPACE***] \
  --overwrite \
  strimzi.io/kraft="migration"
```

4. Monitor the migration.

```
kubectl get kafka [***CLUSTER NAME***] --namespace [***NAMESPACE***] --w
atch
```

Monitor the METADATA STATE column. The cluster goes through multiple metadata states during migration. This phase of the migration is complete when the cluster is in the KRaftPostMigration state.

```
NAME         #...   READY   METADATA STATE       WARNINGS
my-cluster   #...   True    KRaftPostMigration   True
```

Note: If the Kafka resource includes the inter.broker.protocol.version or log.message.format.version properties, the Strimzi Cluster Operator reports warnings (WARNINGS column is true). This is because neither properties are used or supported by Kafka when it is running in KRaft mode. This is normal and expected behavior. You will be removing these properties as when you finalize migration.

5. Finalize the migration.

Important: Ensure that your cluster is in a healthy and working state before continuing. Once you finalize migration, you cannot rollback your cluster to use ZooKeeper. For rollback instructions skip to Rolling back a cluster to ZooKeeper.

a) Annotate the Kafka resource with strimzi.io/kraft="enabled".

This annotation starts the finalization process and removes ZooKeeper pods.

```
kubectl annotate kafka [***KAFKA NAME***] \
  --namespace [***NAMESPACE***] \
  --overwrite \
  strimzi.io/kraft="enabled"
```

b) Monitor finalization.

```
kubectl get kafka [***CLUSTER NAME***] --namespace [***NAMESPACE***] --w
atch
```

Monitor the METADATA STATE column. The cluster goes through multiple metadata states during finalization. Migration is finalized when the cluster is in the KRaft state.

```
NAME          #...   READY   METADATA STATE   WARNINGS
my-cluster    #...   True    KRaft            True
```

c) Remove inter.broker.protocol.version and log.message.format.version properties from your `Kafka` resource.

d) Remove spec.zookeeper from your Kafka resource.

ZooKeeper is no longer in use for the cluster, keeping its configuration is not necessary.

Rolling back a Kafka cluster to ZooKeeper

You roll back a KRaft-based cluster to Zookeeper using the strimzi.io/kraft="annotation" on the Kafka resource of the cluster. Additionally, you remove KRaft controllers.

Before you begin

Rollbacks to ZooKeeper are only possible if the migration to KRaft is not finalized. A migration is not finalized if the Kafka resource has the strimzi.io/kraft="migration" annotation and is in KRaftPostMigration metadata state. Clusters that have the strimzi.io/kraft="enabled" annotation and are in the PreKraft or KRaft metadata state are finalized and cannot be rolled back.

Procedure

1. Start the rollback by annotating your Kafka resource with strimzi.io/kraft="rollback".

```
kubectl annotate kafka [***CLUSTER NAME***] \
  --namespace [***NAMESPACE***] \]
  --overwrite \
  strimzi.io/kraft="rollback"
```

2. Monitor the rollback.

```
kubectl get kafka [***KAFKA NAME***] --namespace [***NAMESPACE***] --watch
```

Monitor the METADATA STATE column. This stage of the rollback is finished once the cluster reverts to the KRaftDualWriting metadata state.

```
NAME          #...   READY   METADATA STATE      WARNINGS
my-cluster    #...   True    KRaftDualWriting    True
```

Note: The Strimzi Cluster Operator logs a warning notifying you that the strimzi.io/kraft annotation cannot be set to rollback if the cluster metadata state is KRaftDualWriting. This is normal and expected behavior. You update this annotation as part of the finalization process.

3. Finalize the rollback.

a) Annotate your Kafka resource with strimzi.io/kraft="disabled".

```
kubectl annotate kafka [***CLUSTER NAME***] \
  --namespace [***NAMESPACE***] \
  --overwrite \
  strimzi.io/kraft="disabled"
```

Note: The state of the `Kafka` resource will not change to ready. This is because a corresponding `KafkaNodePool` resource has spec.roles set to controller, which is invalid for a ZooKeeper-based cluster. This is normal and expected behavior. You remove invalid node pools in a following step.

b) Delete the `KafkaNodePool` resource you created for your KRaft controllers.

```
kubectl delete kafkanodepool [***NODE POOL NAME***] --names
pace [***NAMESPACE***]
```

c) Monitor rollback finalization.

```
kubectl get kafka [***CLUSTER NAME***] --namespace [***NAMESPACE***] --w
atch
```

The rollback is finalized once the Kafka cluster is in the ZooKeeper metadata state.

```
NAME         #...   READY   METADATA STATE    WARNINGS
my-cluster   #...   True    ZooKeeper         True
```