Cloudera Streams Messaging Operator for Kubernetes 1.7.0

Upgrading Cloudera Streams Messaging Operator for Kubernetes

Date published: 2024-06-11
Date modified: 2026-07-31

https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2026. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Upgrading and rolling back Cloudera Streams Messaging Operator for Kubernetes

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

Upgrades

Important: Before upgrading to this release, convert all Strimzi custom resources to the v1 API (kafka.strimzi.io/v1) using the Strimzi API conversion tool. The v1alpha1, v1beta1, and v1beta2 API versions are removed in this release. Custom resources still using old API versions will stop working after upgrading. For instructions, see Strimzi API conversion tool.

Upgrading Cloudera Streams Messaging Operator for Kubernetes 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 Operator for Kubernetes 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 Operator for Kubernetes

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

Before you begin

- Convert all Strimzi custom resources to the v1 API using the Strimzi API conversion tool. Run the following command to convert all resources across all namespaces:

  ```
  bin/v1-api-conversion.sh convert-resource --all-namespaces
  ```

  Verify that the conversion completed successfully before proceeding. For full instructions and alternative execution modes (including running the tool as a Kubernetes Job), see Converting Strimzi custom resources to the v1 API.

- Upgrade the CRD storage version to v1 using the Strimzi v1 API Conversion Tool. Run the following command after the resource conversion is complete:

  ```
  bin/v1-api-conversion.sh crd-upgrade
  ```

  This removes v1beta2 from the CRD status.storedVersions field. Without this step the CRD update in step 1 will be rejected by Kubernetes. For full instructions and alternative execution modes (including running the tool as a Kubernetes Job), see Strimzi API conversion tool.
- 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 Operator for Kubernetes. 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 Operator for Kubernetes version you are upgrading to.

  Cloudera Streams Messaging Operator for Kubernetes 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 spec.metadataVersion during the upgrade to keep Kafka in backward compatible state during the upgrade. This is only necessary if the metadata 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/1.7/install.

   ```
   curl -s --user [***USERNAME***] \
     https://archive.cloudera.com/p/csm-operator/1.7.0/install/strimzi-crd
   s-1.0.1-csmop-1.7.0-b248.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. Check for cross-namespace Entity Operator watchedNamespace configuration.

   To do this, run the following command to list each Kafka cluster, its namespace, and the watchedNamespace values set on the Topic and User Operators.

   ```
   kubectl get kafka \
     --all-namespaces \
     --output custom-columns="KAFKA_NAME:.metadata.name,\
   CLUSTER_NAMESPACE:.metadata.namespace,\
   TOPIC_WATCH:.spec.entityOperator.topicOperator.watchedNamespace,\
   USER_WATCH:.spec.entityOperator.userOperator.watchedNamespace"
   ```

   If the TOPIC_WATCH or USER_WATCH columns show a namespace value that differs from CLUSTER_NAMESPACE, you have cross-namespace watching configured. If these columns show <none> or match CLUSTER_NAMESPACE, the operators watch the cluster namespace by default.

6. 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.

   Follow the Standard upgrade tab if cross-namespace watching is not configured. Follow the Cross namespace tab if cross-namespace watching is configured.

   For Standard upgrade

   a. Run helm upgrade

   ```
   helm upgrade strimzi-cluster-operator \
     --namespace [***STRIMZI CLUSTER OPERATOR NAMESPACE***] \
     --atomic \
     oci://container.repository.cloudera.com/cloudera-helm/csm-operator/
   strimzi-kafka-operator \
     --version 1.7.0-b248
   ```

- 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.

For Cross namespace

a. Create a file (for example, entity-operator-values.yaml) with the following content.

```
extraEnvs:
  - name: STRIMZI_ENTITY_OPERATOR_WATCHED_NAMESPACE_ENABLED
    value: "true"
```

b. Run the helm upgrade command with the --reset-then-reuse-values option.

```
helm upgrade strimzi-cluster-operator \
  --namespace [***STRIMZI CLUSTER OPERATOR NAMESPACE***] \
  --atomic \
  --reset-then-reuse-values \
  --values entity-operator-values.yaml \
  oci://container.repository.cloudera.com/cloudera-helm/csm-operator/
strimzi-kafka-operator \
  --version 1.7.0-b248
```

- 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.
- --reset-then-reuse-values ensures that other previously set values are not overwritten.
- --values specifies the file containing the configuration to enable cross-namespace watching.
- 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.

7. 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***]
```

8. 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) Set spec.metadataVersion to the old metadata version.

This is needed to keep the Kafka cluster in a backward-compatible state after an upgrade.

```
#...
kind: Kafka
spec:
  kafka:
    version: 4.2.0-csmop-1.7
    metadataVersion: 4.1-IV1
```

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-
```

9. 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
```

10. 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-
```

11. 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
```

12. Finalize the upgrade.

Important: A rollback is no longer possible after you complete this step.

To do this, remove the spec.metadataVersion from your `Kafka` resource.

Rolling back Cloudera Streams Messaging Operator for Kubernetes

To roll back Cloudera Streams Messaging Operator for Kubernetes 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 Operator for Kubernetes version that you upgraded to.
  - The Kafka upgrade is not finalized. A Kafka upgrade is finalized if spec.metadataVersion for the Kafka cluster is set to the current metadata version.
- 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 Operator for Kubernetes 1.7 ships two Kafka base images. One for Kafka 4.2.0-csmop-1.7 (latest/current) and one for Kafka 4.1.1.1.6 (previous). In order to rollback, you need a custom image based on the 4.1.1.1.6 image shipped with 1.7.

    The base image used to build the custom image must be an image shipped in the Cloudera Streams Messaging Operator for Kubernetes version you upgraded to. Even though earlier Cloudera Streams Messaging Operator for Kubernetes versions might have shipped the same Kafka version, you cannot use a base image from a previous Cloudera Streams Messaging Operator for Kubernetes 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
```

Upgrading and rolling back Cloudera Surveyor for Apache Kafka

Learn about upgrading and rolling back Cloudera Surveyor. Upgrading and rolling back Cloudera Surveyor is done with helm commands. Use helm upgrade to upgrade to a new release, and helm rollback to roll back to a previous release. Both operations are seamless and require no downtime, ensuring uninterrupted service.

Upgrading Cloudera Surveyor

You upgrade Cloudera Surveyor using helm upgrade. The upgrade is a two-step process. First, you upgrade the Cloudera Surveyor server, afterward, you upgrade the Cloudera Surveyor UI. You also use helm history and take note of revisions before and during the upgrade.

Before you begin

During the upgrade, when the server is upgraded but the UI is not, you might encounter minor UI issues. For example, the configuration of some topics might fail to load. This is temporary, all data is loaded correctly once the upgrade finishes.

Procedure

1. List available Helm revisions and take note of the currently deployed revision.

```
helm history cloudera-surveyor \
  --namespace [***SURVEYOR NAMESPACE***]
```

   Taking note of the revision now makes future rollbacks easier, as it can be difficult to identify the correct revision later. The rollback steps refer to this revision as *[***ORIGINAL REVISION***]*.

2. Upgrade the Cloudera Surveyor server.

```
helm upgrade cloudera-surveyor \
  --namespace [***SURVEYOR NAMESPACE***] \
  --atomic \
  --reset-then-reuse-values \
  --set=image.tag=0.2.0-csmop-1.7.0-b248 \
  --set=image.uiTag=0.1.0.1.6.0-b99 \
```

```
  oci://container.repository.cloudera.com/cloudera-helm/csm-operator/surv
eyor \
  --version 1.7.0-b248
```

- The string cloudera-surveyor 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 Cloudera Surveyor.
- The --atomic option makes the command wait for the upgrade to complete or roll back if the upgrade fails.
- The --reset-then-reuse-values option is used to ensure that previously set configurations are preserved. This option is required because the upgrade command also uses --set options.
- The --set options specify the exact image tags to use for the server and UI. In this step, only the server image tag (image.tag=0.2.0-csmop-1.7.0-b248), is updated. The UI image tag (image.uiTag=0.1.0.1.6.0-b99) remains unchanged. This ensures that only the server is upgraded.

3. Take note of the Helm revision.

   The revision is printed in the output of the helm upgrade command. The revision is needed in case you want to roll back the upgrade. The rollback steps refer to this revision as *[***SERVER UPGRADE REVISION***]*.

4. Upgrade the Cloudera Surveyor UI.

```
helm upgrade cloudera-surveyor \
  --namespace [***SURVEYOR NAMESPACE***] \
  --atomic \
  --reset-then-reuse-values \
  --set=image.tag=0.2.0-csmop-1.7.0-b248 \
  --set=image.uiTag=0.2.0-csmop-1.7.0-b248 \
  oci://container.repository.cloudera.com/cloudera-helm/csm-operator/surv
eyor \
  --version 1.7.0-b248
```

   In this step, both the server (image.tag=0.2.0-csmop-1.7.0-b248) and UI (image.uiTag=0.2.0-csmop-1.7.0-b248) image tags are updated, completing the upgrade for the entire application.

5. Verify that the upgrade is successful.

   To do this, check that the Cloudera Surveyor `Deployment` is in a healthy state.

```
kubectl get deployments cloudera-surveyor \
  --namespace [***SURVEYOR NAMESPACE***]
```

   In addition, you can check the image version that the `Pods` are using.

```
kubectl get pods \
  --namespace=kafka \
  --output custom-columns="NAME:.metadata.name,IMAGE:.spec.containers[*
].image"
```

   On successful upgrade, the `Pods` will use the new version of the Cloudera Surveyor image.

```
NAME                         IMAGE
cloudera-surveyor-558595d999-dt2dj docker-private.infra.cloudera.com/clo
udera/surveyor:0.2.0-csmop-1.7.0-b248
```

What to do next

Reload the browser page where you access the UI to ensure that the latest version of the UI is displayed. The browser may continue to show a previous version of the UI until you refresh.

Rolling back Cloudera Surveyor

You roll back Cloudera Surveyor using helm rollback. A rollback involves listing and identifying the upgrade revisions that you want to roll back to. Afterward, you complete a two-step rollback process where you roll back the Cloudera Surveyor UI and the Cloudera Surveyor server.

Before you begin

- During the rollback, when the server is still upgraded but the UI is already rolled back, you might encounter minor UI issues. For example, the configuration of some topics might fail to load. This is temporary, all data is loaded correctly once the upgrade finishes.
- If you took note of your revisions during the upgrade process, skip steps 1 on page 13 and 2 on page 13.

Procedure

1. List available Helm revisions.

```
helm history cloudera-surveyor \
  --namespace [***SURVEYOR NAMESPACE***]
```

2. Identify the revision that upgraded the Cloudera Surveyor server as well as the original revision you upgraded from.

   For example, if you just completed an upgrade, you need the previous two revisions. These are revisions 1 and 2 in the following example.

```
REVISION  #...  STATUS      CHART                  DESCRIPTION
1         #...  superseded  surveyor-1.6.0-b99   Install complete
2         #...  superseded  surveyor-1.7.0-b248   Upgrade complete
3         #...  deployed    surveyor-1.7.0-b248   Upgrade complete
```

   You can also identify revisions by looking at the chart version.

   - The revision that upgraded the server is the first revision that has the latest chart version. This is revision 2 in the example.
   - The original revision you upgraded from is the last revision that has the old chart version. This is revision 1 in the example.

3. Roll back the Cloudera Surveyor UI.

```
helm rollback cloudera-surveyor [***SERVER UPGRADE REVISION***] \
  --namespace [***SURVEYOR NAMESPACE***]
```

   Replace *[***SERVER UPGRADE REVISION***]* with the revision that upgraded the Cloudera Surveyor server.

4. Roll back the Cloudera Surveyor server.

```
helm rollback cloudera-surveyor [***ORIGINAL REVISION***] \
  --namespace [***SURVEYOR NAMESPACE***]
```

   Replace *[***ORIGINAL REVISION***]* with the revision that you originally upgraded from.

5. Verify that the rollback is successful.

   To do this, check that the Cloudera Surveyor `Deployment` is in a healthy state.

```
kubectl get deployments cloudera-surveyor \
  --namespace [***SURVEYOR NAMESPACE***]
```

In addition, you can check the image version that the Cloudera Surveyor `Pods` are using.

```
kubectl get pods \
  --namespace [***SURVEYOR NAMESPACE***] \
  --output custom-columns="NAME:.metadata.name,IMAGE:.spec.containers[*].
image"
```

On successful upgrade, the `Pods` will use the old version of the Cloudera Surveyor image.

```
NAME                        IMAGE
cloudera-surveyor-558595d999-dt2dj docker-private.infra.cloudera.com/clo
udera/surveyor:0.1.0.1.6.0-b99
```

What to do next

Reload the browser page where you access the UI to ensure that the latest version of the UI is displayed. The browser may continue to show a previous version of the UI until you refresh.

Converting Strimzi custom resources to the v1 API

The Strimzi API conversion tool converts custom resources and CRDs from deprecated API versions to v1. You must complete resource conversion and CRD storage upgrade before upgrading to Cloudera Streams Messaging Operator for Kubernetes 1.7.

The v1alpha1, v1beta1, and v1beta2 API versions for Strimzi Custom Resource Definitions (CRDs) are removed in Cloudera Streams Messaging Operator for Kubernetes 1.7. Only the kafka.strimzi.io/v1 API is supported. Custom resources that still use old API versions stop working after upgrading.

The Strimzi API conversion tool handles most of the migration of custom resources and CRDs to the v1 API automatically, but some changes must be applied manually before or after running the tool. You have two methods when running the tool:

- **Running the conversion tool locally** **on page 15** — download and run the tool from your machine. Use this method for GitOps and file-based workflows, or when you prefer to run the tool from your machine against a cluster you have kubectl access to. In this method you:
  1. Convert resources using the convert-file command to convert local YAML files, or the convert-resource command to convert resources straight in a running cluster.
  2. Finalize the CRD storage version using the crd-upgrade command.

  Note: Running the tool locally requires Java to be installed and available in your PATH. If Java is not available on your local machine, run the tool as a Kubernetes Job instead.
- **Running the conversion tool as a Kubernetes Job** **on page 17** — run the tool inside the cluster using the Strimzi Docker image. Use this method when you cannot run the conversion tool from your local machine. This approach does not require you to download the tool to your local machine. In this method you:
  1. Convert resources using the convert-resource command as a Kubernetes Job.
  2. Finalize the CRD storage version using the crd-upgrade command as a Kubernetes Job by editing and reapplying the same manifest.

Note: While KafkaBridge and KafkaMirrorMaker2 resources are not supported in Cloudera Streams Messaging Operator for Kubernetes, their Custom Resource Definitions (CRDs) must be upgraded to the v1 API. The crd-upgrade command automatically upgrades these CRDs as well. If you have live instances of these resources that need API conversion, see the Strimzi documentation for KafkaBridge and KafkaMirrorMaker2 v1 API changes.

Related Information

Cloudera Archive

Running the conversion tool locally

Converting custom resource YAML files

Convert Strimzi custom resources from local YAML files to the v1 API using the convert-file command. This approach is suitable for GitOps workflows or environments where direct cluster access is restricted.

Before you begin

- Java is installed and available in your PATH.
- Download and extract the Strimzi API conversion tool from the Cloudera Archive. https://archive.cloudera.com/p/csm-operator/1.7/tools/.
- Back up all custom resources.
- You have reviewed the manual changes required for your custom resource types. Some manual changes must be applied before running the tool. See Strimzi v1 API configuration changes on page 20.
- The Cloudera Streams Messaging Operator for Kubernetes version deployed in the cluster is 1.6 or later.

Procedure

1. Run convert-file with the --file option to specify the input file.

   Tip: Each input file can contain multiple resource definitions. The conversion operation converts all Strimzi custom resources in the file while leaving other Kubernetes resources unchanged.

   Choose one of the following output options:

   - Use --in-place to overwrite the source file.
   - Use --output *[***OUTPUT FILE***]* to write the converted resource to a new file.
   - Omit both options to print the converted resource to standard output.

   Command examples:

   Convert a single file and print output with INFO logging.

   ```
   bin/v1-api-conversion.sh convert-file --file [***INPUT FILE***] --log-l
   evel INFO
   ```

   Convert a file and overwrite it in place.

   ```
   bin/v1-api-conversion.sh convert-file --file [***INPUT FILE***] --in-place
   ```

   Convert a file and write the output to a new file.

   ```
   bin/v1-api-conversion.sh convert-file --file [***INPUT FILE***] --outpu
   t [***OUTPUT FILE***]
   ```

   Convert multiple input files and print the output.

   ```
   bin/v1-api-conversion.sh convert-file --file [***INPUT FILE 1***] --f
   ile [***INPUT FILE 2***]
   ```

2. Review the log output to ensure that all custom resources converted successfully and address any errors.
3. Inspect the output YAML and confirm that Strimzi resources use the kafka.strimzi.io/v1 API version.

4. Apply the converted files to the cluster using kubectl apply or kubectl replace or commit them to your GitOps repository.
5. Ensure that all conversions complete successfully before proceeding.

What to do next

Upgrade the CRD storage version. See Upgrading CRD storage version on page 17.

Converting custom resources in a Kubernetes cluster

Convert existing Strimzi custom resources directly in a live Kubernetes cluster to the v1 API using the convert-resource command. This command directly updates the resources in the cluster without requiring local YAML files.

Before you begin

- Java is installed and available in your PATH.
- Download and extract the Strimzi API conversion tool from the Cloudera Archive. https://archive.cloudera.com/p/csm-operator/1.7/tools/.
- Back up all custom resources.
- You have reviewed the manual changes required for your custom resource types. Some manual changes must be applied before running the tool. See Strimzi v1 API configuration changes on page 20.
- The user or service account running the tool has list, get, and replace permissions for all Strimzi custom resources in the target namespaces.
- The Cloudera Streams Messaging Operator for Kubernetes version deployed in the cluster is 1.6 or later.

Procedure

1. Run convert-resource with the appropriate scope options.

   Choose from the following scope options:

   - --namespace *[***NAMESPACE***]* to convert resources in a specific namespace.
   - --all-namespaces to convert resources in all namespaces.
   - Omit both options to convert resources in the current namespace.

   Use the following filtering options to limit the scope of conversion:

   - --kind *[***KIND***]* to convert only resources of a specific kind, for example Kafka or KafkaConnect.
   - --name *[***NAME***]* with --kind to convert a single resource by name.

   Command examples:

   Convert all Strimzi resources in the current namespace with INFO logging.

   ```
   bin/v1-api-conversion.sh convert-resource --log-level INFO
   ```

   Convert all Strimzi resources in all namespaces.

   ```
   bin/v1-api-conversion.sh convert-resource --all-namespaces
   ```

   Convert all Strimzi resources in a specific namespace.

   ```
   bin/v1-api-conversion.sh convert-resource --namespace [***NAMESPACE***]
   ```

   Convert only resources of a specific kind in all namespaces.

   ```
   bin/v1-api-conversion.sh convert-resource --all-namespaces --kind Kafka
   ```

Convert resources of multiple kinds in all namespaces.

```
bin/v1-api-conversion.sh convert-resource --all-namespaces --kind Kafka --
kind KafkaConnect
```

Convert a single resource by name in a specific namespace.

```
bin/v1-api-conversion.sh convert-resource --namespace [***NAMESPACE***] --
kind Kafka --name [***CLUSTER NAME***]
```

2. Review the log output to ensure that all custom resources converted successfully and address any errors.
3. Ensure that all conversions complete successfully before proceeding.

What to do next

Upgrade the CRD storage version. See Upgrading CRD storage version on page 17.

Upgrading CRD storage version

Upgrade Strimzi Custom Resource Definitions (CRDs) to the v1 storage version using the crd-upgrade command as the final step of the conversion before upgrading to Cloudera Streams Messaging Operator for Kubernetes 1.7.

Before you begin

Important: Run crd-upgrade only after all custom resources have been converted to v1. See Converting custom resource YAML files on page 15 or Converting custom resources in a Kubernetes cluster on page 16.

- Java is installed and available in your PATH.
- Download and extract the Strimzi API conversion tool from the Cloudera Archive. https://archive.cloudera.com/p/csm-operator/1.7/tools/.
- The user or service account running the tool has patch permissions for Custom Resource Definitions.
- The Cloudera Streams Messaging Operator for Kubernetes version deployed in the cluster is 1.6 or later.

Procedure

1. Run crd-upgrade:

```
bin/v1-api-conversion.sh crd-upgrade
```

2. Verify that all Strimzi CRDs use the v1 API as the storage version:

```
kubectl get crd -o name | grep kafka.strimzi.io | while read crd; do
  echo -n "$crd  "
  kubectl get "$crd" -o jsonpath='{.status.storedVersions}{"\n"}'
done
```

All CRDs must return [v1] as the stored version.

3. After upgrading the CRDs, use only the properties supported in the v1 API in all custom resources.

What to do next

Upgrade Cloudera Streams Messaging Operator for Kubernetes

Running the conversion tool as a Kubernetes Job

Run the Strimzi API conversion tool as a Kubernetes Job to convert custom resources in a cluster without running the tool locally.

About this task

Running the conversion tool as a Kubernetes Job is an alternative to running it locally. The tool is embedded in the Strimzi Docker image and can be run as a Kubernetes `Job`.

Use this approach when you cannot run the conversion tool from your local machine, or when you prefer to run the conversion entirely within the cluster. You still need access to apply manifests to the cluster (for example, using kube ctl).

Before you begin

- You have kubectl access to the cluster and can apply manifests.
- All custom resources are backed up.
- You have reviewed the manual changes required for your custom resource types. Some manual changes must be applied before running the tool. See Strimzi v1 API configuration changes on page 20.
- In air-gapped environments, ensure the Strimzi Docker image is available in your own registry before proceeding. The image you must mirror is:

```
container.repository.cloudera.com/cloudera/kafka-operator:1.0.1-
csmop-1.7.0-b248
```

- The Cloudera Streams Messaging Operator for Kubernetes version deployed in the cluster is 1.6 or later.

Procedure

1. Create a manifest that defines the resources required to run the conversion tool in the cluster.

   The manifest must include the following:

   - A ServiceAccount that the Job runs under.
   - A ClusterRole and ClusterRoleBinding (RBAC permissions) that grant the required access to Strimzi custom resources and CRDs.
   - A Job resource that runs the conversion tool using those permissions.

   If you are running the Job in a specific namespace, ensure that the namespace name is consistent across the Serv iceAccount, ClusterRoleBinding, and Job definitions.

   The following example combines all required resources into a single YAML file. Replace *[***NAMESPACE***]* with the namespace where the Job should run.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: strimzi-v1-api-conversion
  namespace: [***NAMESPACE***]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: strimzi-v1-api-conversion
rules:
  - apiGroups:
      - kafka.strimzi.io
    resources:
      - kafkas
      - kafkanodepools
      - kafkaconnects
      - kafkaconnectors
      - kafkabridges
      - kafkamirrormaker2s
      - kafkarebalances
      - kafkatopics
```

```
      - kafkausers
    verbs:
      - get
      - list
      - patch
      - update
  - apiGroups:
      - core.strimzi.io
    resources:
      - strimzipodsets
    verbs:
      - get
      - list
      - patch
      - update
  - apiGroups:
      - apiextensions.k8s.io
    resources:
      - customresourcedefinitions
      - customresourcedefinitions/status
    resourceNames:
      - kafkabridges.kafka.strimzi.io
      - kafkaconnectors.kafka.strimzi.io
      - kafkaconnects.kafka.strimzi.io
      - kafkamirrormaker2s.kafka.strimzi.io
      - kafkanodepools.kafka.strimzi.io
      - kafkarebalances.kafka.strimzi.io
      - kafkas.kafka.strimzi.io
      - kafkatopics.kafka.strimzi.io
      - kafkausers.kafka.strimzi.io
      - strimzipodsets.core.strimzi.io
    verbs:
      - get
      - list
      - patch
      - update
  - apiGroups:
      - ""
    resources:
      - configmaps
    verbs:
      - get
      - list
      - create
      - patch
      - update
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: strimzi-v1-api-conversion
subjects:
  - kind: ServiceAccount
    name: strimzi-v1-api-conversion
    namespace: [***NAMESPACE***]
roleRef:
  kind: ClusterRole
  name: strimzi-v1-api-conversion
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: batch/v1
kind: Job
metadata:
  name: strimzi-v1-api-conversion
```

```
  namespace: [***NAMESPACE***]
spec:
  template:
    spec:
      serviceAccountName: strimzi-v1-api-conversion
      containers:
        - name: strimzi-v1-api-conversion
          image: container.repository.cloudera.com/cloudera/kafka-opera
tor:1.0.1-csmop-1.7.0-b248
          command:
            - /opt/v1-api-conversion/bin/v1-api-conversion.sh
            - convert-resource
      restartPolicy: OnFailure
```

Tip: In air-gapped environments, replace the image path in the `Job` definition with the path to your mirrored registry image.

2. Apply the manifest to the cluster:

```
kubectl apply --filename strimzi-v1-api-conversion-job.yaml
```

3. Check that the Job has completed and review the logs.

```
kubectl get jobs --namespace [***NAMESPACE***]
```

```
kubectl logs job/strimzi-v1-api-conversion --namespace [***NAMESPACE***]
```

4. Verify that the converted resources now use the kafka.strimzi.io/v1 API version.

   If direct CLI access is unavailable, check the logs or your monitoring dashboard to confirm that the Job completed successfully.

5. Ensure that all conversions complete successfully before proceeding to upgrade the CRD storage version.
6. Run crd-upgrade to finalize the CRD storage version upgrade.

   To run crd-upgrade as a Job, edit the container command in the manifest and reapply it. Replace convert-resource with crd-upgrade in the command field:

```
#...
kind: Job
spec:
  template:
    spec:
      containers:
        - name: strimzi-v1-api-conversion
          image: container.repository.cloudera.com/cloudera/kafka-operato
r:1.0.1-csmop-1.7.0-b248
          command:
            - /opt/v1-api-conversion/bin/v1-api-conversion.sh
            - crd-upgrade
```

   Monitor the Job to completion as in the previous steps.

Strimzi v1 API configuration changes

Strimzi v1 API configuration changes for custom resources, listing automatic updates applied by the conversion tool and the manual changes you must apply independently.

Manual and automatic changes

Custom resources can be updated for the v1 API in two ways:

- By using the Strimzi API conversion tool to apply supported automatic changes.
- By editing and reapplying resources manually without using the tool.

Even when the conversion tool is used, some changes must still be applied manually because the tool cannot modify or remove unsupported configuration. The reference information below lists which changes the tool applies automatically and which require manual updates.

Manual changes are updates the conversion tool does not apply automatically.

To apply a manual change:

1. Edit the affected custom resource.
2. Update or remove the required properties.
3. Reapply the resource using kubectl apply -f or kubectl replace -f.

Automatic changes are updates the conversion tool applies during conversion. Even when the conversion tool is used, some manual changes are still required.

Kafka

Manual changes required

The conversion tool does not apply the following updates. You must make these changes manually to ensure compatibility:

- Ensure the .spec section is present (required).
- Replace unsupported authentication type: oauth with type: custom.
- In type: custom authentication, remove the secrets property. Mount secrets using the template section instead.
- Replace unsupported authorization types keycloak and opa with type: custom.
- Remove the .spec.kafka.resources property. Define resource requests and limits in `KafkaNodePool` resources instead.

Automatic changes supported

The following updates are applied automatically by the conversion tool. If the conversion tool is not used, the same changes can be made manually:

- Removes the .spec.zookeeper section. ZooKeeper-based clusters are not supported.
- Removes the .spec.jmxTrans section. JMXTrans is not supported.
- Removes the following Kafka properties:
  - .spec.kafka.replicas (replicas are configured through `KafkaNodePool` resources)
  - .spec.kafka.storage (storage is configured through `KafkaNodePool` resources)
  - .spec.kafka.template.statefulset
- Removes the following Cruise Control properties:
  - .spec.cruiseControl.tlsSidecar
  - .spec.cruiseControl.template.tlsSidecarContainer
  - .spec.cruiseControl.brokerCapacity.disk
  - .spec.cruiseControl.brokerCapacity.cpuUtilization
- Removes the .spec.kafkaExporter.template.service property.
- Removes the following Entity Operator properties:
  - .spec.entityOperator.tlsSidecar
  - .spec.entityOperator.template.tlsSidecarContainer
  - .spec.entityOperator.topicOperator.topicMetadataMaxAttempts
  - .spec.entityOperator.topicOperator.zookeeperSessionTimeoutSeconds

    - .spec.entityOperator.userOperator.zookeeperSessionTimeoutSeconds
- Replaces .spec.entityOperator.topicOperator.reconciliationIntervalSeconds with .spec.entityOperator.topicOperator.reconciliationIntervalMs (value multiplied by 1000).
- Replaces .spec.entityOperator.userOperator.reconciliationIntervalSeconds with .spec.entityOperator.userOperator.reconciliationIntervalMs (value multiplied by 1000).
- Removes the following status properties from YAML files that include the status section:
    - .status.registeredNodeIds
    - .status.kafkaMetadataState
    - .status.listeners[].type

Note: Status removal applies only to local YAML files (using the convert-file command). The conversion tool does not modify the status of live resources in the cluster, as their status is actively managed by the Cluster Operator.

KafkaConnect

Manual changes required

The conversion tool does not apply the following updates. You must make these changes manually to ensure compatibility:

- Ensure the .spec section is present (required).
- Replace unsupported authentication type: oauth with type: custom.
- Remove the .spec.externalConfiguration property. Use the .spec.template section to add custom volumes or environment variables instead.

Automatic changes supported

The following updates are applied automatically by the conversion tool. If the conversion tool is not used, the same changes can be made manually:

- Adds the .spec.replicas property if missing, defaulting to 3.
- Renames .spec.build.output.additionalKanikoOptions to .spec.build.output.additionalBuildOptions.
- Removes the .spec.template.deployment property.
- Removes unsupported tracing type: jaeger.
- Adds the following required properties, setting them from existing configuration or default values:
    - .spec.groupId: set from .spec.config.group.id, or defaults to connect-cluster.
    - .spec.configStorageTopic: set from .spec.config.config.storage.topic, or defaults to connect-cluster-configs.
    - .spec.offsetStorageTopic: set from .spec.config.offset.storage.topic, or defaults to connect-cluster-offsets.
    - .spec.statusStorageTopic: set from .spec.config.status.storage.topic, or defaults to connect-cluster-status.

Note: After the group ID and internal topic properties are set by the conversion, remove the corresponding values from .spec.config.

KafkaNodePool

Manual changes required

- Ensure the .spec section is present (required).

Automatic changes supported

- Removes the overrides property from .spec.storage. To configure different storage settings for specific nodes, define separate `KafkaNodePool` resources.

KafkaTopic

Manual changes required

- Ensure the .spec section is present (required).

No automatic changes are applied by the conversion tool for `KafkaTopic` resources.

KafkaUser

Manual changes required

- Ensure the .spec section is present (required).

Automatic changes supported

- Replaces the operation property in .spec.authorization.acls[] with the operations array.

KafkaRebalance

Manual changes required

- Ensure the .spec section is present (required).

No automatic changes are applied by the conversion tool for `KafkaRebalance` resources.

KafkaConnector

Manual changes required

- Ensure the .spec section is present (required).

Automatic changes supported

- Removes the .spec.pause property. If pause: true was set, it is converted to state: paused.

StrimziPodSet

`StrimziPodSet` is an internal resource managed automatically by the Cluster Operator. Conversion is handled by Cloudera Streams Messaging Operator for Kubernetes and does not require user action.