Cloudera Streams Messaging - Kubernetes Operator 1.3.0

Configuring Strimzi

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

https://docs.cloudera.com/

Legal Notice

Contents

Configuring Strimzi Cluster Operator

Learn how to configure an existing deployment of the Strimzi Cluster Operator with helm upgrade.

The Strimzi Cluster Operator is deployed when you install Strimzi to your Kubernetes cluster. During installation you can configure the Cluster Operator. If required, you can make configuration updates following installation. This is done with helm upgrade command using the --reuse-values option together with the -f (--values) or --set options.

```
helm upgrade [***RELEASE***] [***CHART***] \
  --namespace [***NAMESPACE***] \
  (--set '[***KEY***]=[***VALUE***]' | -f [***MY-VALUES.YAML***]) \
  --reuse-values
```

- Ensure that *[***RELEASE***]* and *[***CHART***]* are the same as the ones you used during installation. You can use helm list to list currently installed releases and charts.
- Use --set if you want to update properties directly from the command line. Helm supports various --set options like --set-file, --set-string, and others. You can use any of these options.
- Use -f (--values) if you want to pass a file containing your configuration updates.
- The --reuse-values option instructs Helm to merge existing values with new ones. You use this option when you want to update an existing configuration

Configurable properties

The Strimzi Cluster Operator accepts various configuration properties. You can find a comprehensive list of these properties in the *Helm chart configuration reference*. Alternatively, you can list available properties with helm show readme.

```
helm show readme [***CHART***]
```

Related Information

Helm chart configuration reference

Helm List | Helm

Helm Upgrade | Helm

Helm Show Readme | Helm

Configuring Strimzi Cluster Operator watched namespaces

By default, the Strimzi Cluster Operator watches and manages the Kafka clusters that are deployed in the same namespace as the Strimzi Cluster Operator. However, you can configure it to watch selected namespaces or watch all namespaces. This allows you to manage multiple Kafka clusters deployed in different namespaces using a single Strimzi Cluster Operator installation.

If you have a specific list of namespaces that you want the Strimzi Cluster Operator to watch, specify them in the watchNamespaces property. Delimit each namespace with a comma.

```
helm upgrade [***RELEASE***] [***CHART***] \
  --namespace [***NAMESPACE***] \
  --set 'watchNamespaces={[***NAMESPACE A***],[***NAMESPACE B***]}' \
  --reuse-values
```

Tip: The namespace where the Strimzi Cluster Operator is deployed is always watched. You do not need to add it to the list in watchNamespaces.

If you want the Strimzi Cluster Operator to watch all namespaces in your cluster, set watchAnyNamespace to true.

```
helm upgrade [***RELEASE***] [***CHART***] \
  --namespace [***NAMESPACE***] \
  --set 'watchAnyNamespace=true' \
  --reuse-values
```

Increasing the Strimzi Cluster Operator's memory

If you want a single installation of the Strimzi Cluster Operator to watch and manage more than 20 Kafka clusters, you must increase the memory and heap allocated for the Strimzi Cluster Operator. Otherwise, you will encounter out of memory and heap related errors. To do this, you configure memory-related properties.

To configure the memory allocated to the Strimzi Cluster Operator, set -XX:MinRAMPercentage and -XX:Max RAMPercentage Java parameters. Additionally, configure resources.limits.memory and resources.requests.memory Helm properties.

The following example contains memory settings recommended by Cloudera for a deployment with more than 20 Kafka clusters. You can fine-tune your setting as needed.

```
helm upgrade [***RELEASE***] [***CHART***] \
  --namespace [***NAMESPACE***] \
  --set 'extraEnvs[0].name=JAVA_OPTS' \
  --set 'extraEnvs[0].value=-XX:MinRAMPercentage=25 -XX:MaxRAMPercentage=
70' \
  --set 'resources.limits.memory=600Mi' \
  --set 'resources.requests.memory=600Mi' \
  --reuse-values
```

The default values for the properties configured in the example are as follows.

- -XX:MinRAMPercentage=15
- -XX:MaxRAMPercentage=20
- resources.limits.memory=384Mi
- resources.requests.memory=384Mi

Running the Strimzi Cluster Operator with a restricted profile

You run the Strimzi Cluster Operator with a restricted profile by configuring the securityContext Helm properties.

By default, the Strimzi Cluster Operator runs with the baseline profile. However, the Helm templates allow customizing the security context of the Strimzi Cluster Operator with the securityContext properties. You run the Strimzi Cluster Operator with a restricted profile by specifying appropriate privileges with helm upgrade.

```
helm upgrade [***RELEASE***] [***CHART***] --namespace [***NAMESPACE***] \
  --set watchAnyNamespace=true
  --set securityContext.allowPrivilegeEscalation=false \
  --set securityContext.capabilities.drop={ALL} \
  --set securityContext.runAsNonRoot=true \
  --set securityContext.seccompProfile.type=RuntimeDefault \
  --reuse-values
```

Configuring Strimzi Cluster Operator's log level

The Strimzi Cluster Operator uses log4j2 configuration for logging. By default the log level is set to INFO. You can update the log level by setting the loglevel property with helm upgrade.

```
helm upgrade [***RELEASE***] [***CHART***] \
  --namespace [***NAMESPACE***] \
  --set logLevel=[***LOG LEVEL***] \
  --reuse-values
```

Configuring pod security providers

Pod Security Providers allow you to manage the security context for all pods and containers managed by the Strimzi Cluster Operator from a single location. That is, a Security Provider defines the default security context of the pods and containers that the Strimzi Cluster Operator creates and manages.

The following two providers are available.

Baseline

The Baseline Provider is based on the Kubernetes baseline security profile. This is a minimally restrictive profile that prevents privilege escalations and defines other standard access controls and limitations.

Restricted

The Restricted Provider is based on the Kubernetes restricted security profile. This is a highly restrictive profile that is aimed for use in environments where high levels of security is critical.

By default, the Strimzi Cluster Operator uses the Baseline Provider. To use the Restricted Provider, set the STRI MZI_POD_SECURITY_PROVIDER_CLASS environment variable of the Strimzi Cluster Operator to restricted.

```
helm upgrade csm-operator [***HELM CHART***] --namespace [***NAMESPACE***] \
--set extraEnvs[0].name=STRIMZI_POD_SECURITY_PROVIDER_CLASS \
--set extraEnvs[0].value=resticted \
--reuse-values
```

Configuring additional volumes and volume mounts

Additional volumes and volume mounts enable a way to attach extra files to all pods handled by Strimzi. With the help of these you can attach Secrets, ConfigMaps, empty directories, or PersistentVolumeClaims the pods as volumes. Once attached, pods are able to read and use the data available in the volumes.

You can use additional volumes and volume mounts when components or processes running in pods require access to additional data. For example, many Kafka Connect connectors access external systems like databases. Access to these systems might require credentials or TLS certificates for access. Using additional volumes and volume mounts, you can store TLS certificates in a `Secret` and mount that `Secret` to the Kafka Connect pods. This makes the certificates available to connectors. Once mounted, data from additional volumes can be referenced in resources using configuration providers.

Supported components

The following table collects the components that support additional volumes and volume mounts as well as their corresponding resources.

Table 1: Supported components for additional volumes and volume mounts

Component	Resource
Cruise Control	`Kafka`
Kafka	`Kafka` and `KafkaNodePool`
Kafka Connect	`Kafka`
KafkaExporter	`Kafka`
Entity Operator • Topic Operator • User Operator	`Kafka`
ZooKeeper	`Kafka`

Supported volume types

The following volume types are supported.

- `Secret`
- `ConfigMap`
- `EmptyDir`
- `PersistentVolumeClaim`

For `Secrets` or `ConfigMaps`, the contents of the resource's data field will be presented in a volume as files using the keys in the data field as the file names.

The empty directory represents an empty (ephemeral) directory for a pod.

For `PersistentVolumeClaims` (PVC), you reference the name of an existing PVC when defining the additional volume. The PVC must be created by you. When the PVC is referenced, it finds the bound `PersistentVolume` and mounts the volume for the pod.

Adding additional volumes and volume mounts

You mount additional volumes and volume mounts using pod and container template properties in the spec of a supported resource.

The following examples add a `Secret`, `ConfigMap`, an empty directory, as well as a `PersistentVolumeClaim` to a `Kafka` and `KafkaNodePool` resource. Additional volumes are defined the same way in other supported components and resources.

Important: All additional mounted paths must be located inside the /mnt path. If you mount a volume outside of this path, the `Kafka` resource remains in a NotReady state, the Kafka pods are not created, and a related warning is logged in the Strimzi Cluster Operator log.

For Kafka

```
#...
kind: Kafka
spec:
  kafka:
    template:
      pod:
        volumes:
          - name: example-secret
            secret:
              secretName: secret-name
          - name: example-configmap
```

```
            configMap:
              name: config-map-name
          - name: temp
            emptyDir: {}
          - name: example-pvc-volume
            persistentVolumeClaim:
              claimName: myclaim
      kafkaContainer:
        volumeMounts:
          - name: example-secret
            mountPath: /mnt/secret-volume
          - name: example-configmap
            mountPath: /mnt/cm-volume
          - name: temp
            mountPath: /mnt/temp
          - name: example-pvc-volume
            mountPath: /mnt/data
```

For KafkaNodePool

```
#...
kind: KafkaNodePool
spec:
  template:
    pod:
      volumes:
        - name: example-secret
          secret:
            secretName: secret-name
        - name: example-configmap
          configMap:
            name: config-map-name
        - name: temp
          emptyDir: {}
        - name: example-pvc-volume
          persistentVolumeClaim:
            claimName: myclaim
    kafkaContainer:
      volumeMounts:
        - name: example-secret
          mountPath: /mnt/secret-volume
        - name: example-configmap
          mountPath: /mnt/cm-volume
        - name: temp
          mountPath: /mnt/temp
        - name: example-pvc-volume
          mountPath: /mnt/data
```

Note: When additional volumes are defined in both `Kafka` and `KafkaNodePool` resources, the definition in the `KafkaNodePool` resource takes precedence.

Related Information

Additional Volumes | Strimzi

AdditionalVolume schema reference | Strimzi API reference

ContainerTemplate schema properties | Strimzi API reference

VolumeMount v1 core | Kubernetes

Updating a license

Cloudera Streams Messaging - Kubernetes Operator requires a valid license to function. You must update expired licenses, otherwise, cluster resources will break down over time.

About this task

You register your initial license during installation by setting the clouderaLicense.fileContent Helm chart property. When this property is set, a Kubernetes secret is automatically generated that stores your license. The name of the secret is csm-op-license.

If the license expires, it must be updated. You update the license by updating the secret that stores the license with your new license. Specifically, you update the value of the data.license property in the secret with your new license.

Licenses can be updated at any point in time. If your license is already expired and you update your license, restrictions on functionality are lifted immediately after the license is updated.

Updating a license does not carry any risks and does not result in cluster downtime.

Before you begin

Important: Ensure that the start date of your new license is the current or a past date. Licenses become valid on their start date. Updating your old license with a new license that is not yet valid is the equivalent of registering an expired license. The start date of a license is specified in the startDate property of the license.

Procedure

1. Create a manifest in YAML format that defines the license secret.

   Add your new license to stringData.license. Ensure that you add the full contents of the license as it is in the license file you received from Cloudera.

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: csm-op-license
   type: Opaque
   stringData:
     license: |
       [***YOUR LICENSE***]
   ```

2. Replace your old secret with the new one.

   ```
   kubectl replace --namespace [***NAMESPACE***] -filename [***LICENSE SECRET
    YAML***]
   ```

3. Verify that the license is updated.

   ```
   kubectl get secret csm-op-license \
     --namespace [***NAMESPACE***] \
     --output jsonpath="{.data.license}" \
   | base64 --decode
   ```

   The output of this command should be identical with the contents of the license file you received from Cloudera.

Related Information

Licensing