Cloudera Streams Messaging Operator for Kubernetes 1.7.0

Configuring Strimzi

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

https://docs.cloudera.com/

Legal Notice

Contents

Configuring the 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`

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

Strimzi Drain Cleaner

Strimzi Drain Cleaner coordinates safe rolling restarts of Strimzi-managed Kafka Pods during node drains by intercepting eviction requests and delegating Pod management to the Strimzi Cluster Operator.

Strimzi-managed Kafka `Pods` may be evicted when Kubernetes worker nodes are cordoned and drained. This happens during cluster maintenance, Kubernetes upgrades, or pod rescheduling. Platform teams typically initiate draining with kubectl drain or cluster maintenance tooling.

Without Strimzi Drain Cleaner, Kubernetes handles eviction according to its own rules and `PodDisruptionBudget` settings. The Strimzi Cluster Operator has no chance to coordinate when and how Kafka `Pods` restart. Strimzi Drain Cleaner changes this behavior. It intercepts eviction requests and hands control to the Strimzi Cluster Operator so Kafka `Pods` are rolled in a controlled sequence instead of being evicted directly by Kubernetes.

After installation, Strimzi Drain Cleaner runs continuously in the background. It requires no day-to-day operator action and responds automatically when `Pod` evictions occur. You do not trigger Strimzi Drain Cleaner as part of routine Kafka operations. Platform teams drain nodes as part of maintenance, and Strimzi Drain Cleaner handles matching Strimzi Kafka `Pods` on those nodes.

How it works

Strimzi Drain Cleaner registers a validating admission webhook that receives pods/eviction CREATE requests from the Kubernetes API. When something tries to evict a matching Strimzi-managed Kafka `Pod`, the webhook adds the strimzi.io/manual-rolling-update annotation to that `Pod`. With default chart values, the webhook then denies the eviction request.

Denying the eviction request prevents Kubernetes from restarting the `Pod` on its own. The Strimzi Cluster Operator detects the annotation during reconciliation and performs a controlled rolling update of the `Pod`. The Cluster Operator rolls `Pods` one at a time using its own algorithms. This helps keep partition replicas in sync and Kafka available while worker nodes are drained.

When a worker node is drained, the drain command may fail or report errors when the webhook denies eviction of Kafka `Pods` on that node. This is expected. Strimzi Drain Cleaner and the Strimzi Cluster Operator handle those `Pods` automatically. The Cluster Operator rolls each affected `Pod` off the node one at a time. The node stays cordoned during this process.

Whoever runs the drain command, typically the platform team using kubectl drain or cluster maintenance tooling, may need to run it again on the same cordoned node after rolled `Pods` have left it. If multiple affected `Pods` were on the node, more than one retry may be needed as the Cluster Operator rolls them in sequence. The drain completes when no matching Kafka `Pods` remain on the node.

Affected pods

Strimzi Drain Cleaner acts on Strimzi-managed Kafka `Pods`. This includes broker `Pods` and KRaft controller `Pods` deployed by the Strimzi Cluster Operator.

The webhook matches `Pods` labeled with strimzi.io/kind=Kafka and a strimzi.io/name label whose value ends with -kafka. These are the Kafka broker and controller `Pods` created from your `Kafka` and `KafkaNodePool` resources.

Strimzi Drain Cleaner does not act on other Strimzi workloads, such as Kafka Connect, the Entity Operator, or Kafka Exporter `Pods`. It also does not manage `Pods` that are not deployed and labeled by Strimzi.

Related Information

Installing Strimzi Drain Cleaner with Helm

Strimzi Drain Cleaner Helm reference

Deploying a Kafka cluster

Default webhook configuration

With default Helm chart values, Strimzi Drain Cleaner registers a validating admission webhook automatically. No manual TLS configuration is required when cert-manager is installed.

Strimzi Drain Cleaner installs a cluster-scoped `ValidatingWebhookConfiguration` resource named strimzi-drain-cleaner. The resource registers the strimzi-drain-cleaner.strimzi.io webhook with the Kubernetes API.

The webhook intercepts pods/eviction CREATE operations in namespaced scope. When a matching eviction request is received, the Kubernetes API forwards the request to the Strimzi Drain Cleaner `Service` on port 443 at the /dra iner path.

The following example shows the key webhook settings deployed by the Strimzi Drain Cleaner Helm chart with default values:

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  name: strimzi-drain-cleaner
webhooks:
  - name: strimzi-drain-cleaner.strimzi.io
    rules:
      - apiGroups: [""]
        apiVersions: ["v1"]
        operations: ["CREATE"]
        resources: ["pods/eviction"]
        scope: Namespaced
    clientConfig:
      service:
        namespace: strimzi-drain-cleaner
        name: strimzi-drain-cleaner
        path: /drainer
        port: 443
    admissionReviewVersions: ["v1"]
    sideEffects: None
    failurePolicy: Ignore
    timeoutSeconds: 5
```

With default chart values, cert-manager creates TLS certificates for the webhook and injects the CA bundle into the `ValidatingWebhookConfiguration` automatically. You do not need to configure TLS manually after installation.

Testing Strimzi Drain Cleaner with a node drain

Walk through a node drain to see how Strimzi Drain Cleaner annotates Strimzi-managed Kafka `Pods` and how the Strimzi Cluster Operator performs a rolling update.

About this task

After installation, Strimzi Drain Cleaner requires no day-to-day operator action. It responds automatically when `Pod` evictions occur.

Use this procedure once, or when validating a new installation, to understand the eviction, annotation, and rolling update sequence. In production, platform teams drain nodes as part of maintenance. Strimzi Drain Cleaner handles Strimzi-managed Kafka `Pods` automatically. You do not need to repeat these steps each time a node is drained.

Before you begin

- Strimzi Drain Cleaner is installed. For installation steps, see Installing Strimzi Drain Cleaner with Helm.

- A Kafka cluster is deployed with appropriate replication settings, including a replication factor greater than 1 and min.insync.replicas lower than the replication factor. For an example, see Deploying a Kafka cluster.
- If you use anti-affinity rules for Kafka `Pods`, consider adding a spare worker node to your cluster. Spare capacity helps ensure that rolled `Pods` can be rescheduled while a node is drained.
- Kafka `Pods` must use a `StorageClass` backed by network-attached storage, such as cloud block storage or a distributed storage solution. If Kafka `Pods` use node-local storage, their `PersistentVolumes` are bound to the specific node where the data resides. In that case, evicted `Pods` cannot be rescheduled on another node and the drain cannot complete.

Procedure

1. Drain a Kubernetes worker node that hosts Strimzi-managed Kafka `Pods`.

   ```
   kubectl get nodes
   ```

   ```
   kubectl drain [***NODE NAME***] --delete-emptydir-data --ignore-daemons
   ets --timeout=6000s --force
   ```

   The first drain attempt may fail or report that the validating admission webhook denied eviction of Kafka `Pods`. This is expected behavior.
2. Check the Strimzi Drain Cleaner log for eviction webhook and annotation events.

   Look for log lines similar to the following examples:

   ```
   INFO ... Received eviction webhook for Pod my-cluster-kafka-0 in namespace
    my-project
   INFO ... Pod my-cluster-kafka-0 in namespace my-project will be annotated
   for restart
   INFO ... Pod my-cluster-kafka-0 in namespace my-project found and annota
   ted for restart
   ```
3. Check the Strimzi Cluster Operator log for rolling update reconciliation events.

   Look for log lines similar to the following examples:

   ```
   INFO  PodOperator:68 - Reconciliation #13(timer) Kafka(my-project/my-clu
   ster): Rolling Pod my-cluster-kafka-0
   INFO  AbstractOperator:500 - Reconciliation #13(timer) Kafka(my-project/my
   -cluster): reconciled
   ```
4. Retry the drain command on the same node after affected `Pods` have been rolled off the node.

   ```
   kubectl drain [***NODE NAME***] --delete-emptydir-data --ignore-daemons
   ets --timeout=6000s --force
   ```

   Wait until the Strimzi Cluster Operator has finished rolling affected `Pods` off the node, then run the drain command again on the same cordoned node. If the node hosted multiple affected `Pods`, repeat this step until the drain completes.

Updating a license

Cloudera Streams Messaging Operator for Kubernetes 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