Cloudera Streams Messaging Operator 1.0.0

Kafka Operations

Date published: 2024-06-11
Date modified: 2024-06-11

CLOUDERA

Legal Notice

Contents

Managing topics

Topics can be managed declarativaly with the Strimzi Topic Operator.

The Strimzi Topic Operator runs in unidirectional mode. With this mode, you can create topics with the `KafkaTopic` resource, which are then managed by the Strimzi Topic Operator. Since the Strimzi Topic Operator only manages topics which have corresponding `KafkaTopic` resources, if a managed topic's configuration is changed directly in the Kafka cluster, those changes will be reverted.

Note: Bidirectional mode for the Strimzi Topic Operator is deprecated.

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: my-topic
  labels:
    strimzi.io/cluster: my-cluster
spec:
  partitions: 1
  replicas: 1
  config:
    retention.ms: 7200000
    segment.bytes: 1073741824
```

You can list topic resources with kubectl get.

```
kubectl get kafkatopics --output wide --namespace [***NAMESPACE***]
```

Related Information

Using the Topic Operator to manage Kafka topics | Strimzi

KafkaTopic schema reference | Strimzi API reference

Scaling brokers

Learn how to manually upscale or downscale the number of brokers in your Kafka cluster.

Scaling Kafka brokers in CSM Operator is a two-step manual process that involves the following tasks:

- Scaling the number of broker replicas in your `KafkaNodePool` resources.
- Moving your data by rebalancing with Cruise Control.

The order in which you scale and rebalance is different for upscale and downscale operations. In case of an upscale, you scale brokers first and then rebalance. In case of a downscale, you rebalance first and scale brokers afterward.

When you scale a `KafkaNodePool` resource, the Strimzi Cluster Operator automatically adds or deletes broker nodes. Initiating a rebalance process with Cruise Control automatically moves data between brokers based on a proposal generated by Cruise Control.

Upscaling Kafka brokers

Complete these steps to upscale the number of brokers in your Kafka cluster

Before you begin

- Ensure that Cruise Control is deployed in your cluster. See Deploying Cruise Control.
- You can control the IDs of newly added brokers with the strimzi.io/next-node-ids annotation. See Configuring Kafka broker node IDs.

Procedure

1. Add new brokers to your cluster.

   This is done by updating the replica count in your `KafkaNodePool` resources, which you can do in the following two ways.

   - Update the value of spec.replicas directly in the resource and apply your changes.
   - Scale the resource with kubectl   scale.

   ```
   kubectl scale kafkanodepool [***NODE POOL NAME***] \
      --namespace [***NAMESPACE***] \
      --replicas=[***COUNT***]
   ```

2. Wait until readiness checks are complete and all new brokers are in a Ready state.

   Use the following command to monitor cluster state.

   ```
   kubectl get pods --namespace [***NAMESPACE***] --output wide --watch
   ```

   Note: New brokers are assigned the next available node IDs. If you configured an ID range to use, the IDs are selected from the configured range.

3. Rebalance your cluster with Cruise Control.

   When upscaling a cluster, rebalancing moves data to newly added brokers.

   a) Create a YAML configuration describing your `KafkaRebalance` resource.

      For example:

      ```
      apiVersion: kafka.strimzi.io/v1beta2
      kind: KafkaRebalance
      metadata:
        name: my-rebalance
        labels:
          strimzi.io/cluster: my-cluster
      spec:
        mode: add-brokers
        brokers: [3]
      ```

      - Note down the resource name you specify in metadata.name. You will need to specify the name in the steps that follow.
      - spec.mode specifies the mode to use for rebalancing. The add-brokers mode is used after upscaling a cluster. This mode moves replicas to newly added brokers.
      - spec.brokers is a list of the new broker IDs.

   b) Create the resource.

      ```
      kubectl apply --filename [***YAML CONFIG***] --names
      pace [***NAMESPACE***]
      ```

      Creating the resource generates an optimization proposal from Cruise Control.

c) View the generated optimization proposal.

```
kubectl describe kafkarebalance [***RESOURCE NAME***] --names
pace [***NAMESPACE***]
```

The generated optimization proposal will be similar to the following example.

```
#...
Status:
  Conditions:
    Last Transition Time:  2023-10-18T09:46:28.921357591Z
    Status:                True
    Type:                  ProposalReady
  Observed Generation:     2
  Optimization Result:
    After Before Load Config Map:  my-rebalance
    Data To Move MB:                    1
    Excluded Brokers For Leadership:
    Excluded Brokers For Replica Move:
    Excluded Topics:
    Intra Broker Data To Move MB:       0
    Monitored Partitions Percentage:    100
    Num Intra Broker Replica Movements: 0
    Num Leader Movements:               22
    Num Replica Movements:              45
    On Demand Balancedness Score After:  80.87946050436929
    On Demand Balancedness Score Before: 63.37577576243774
    Provision Recommendation:
    Provision Status:                   RIGHT_SIZED
    Recent Windows:                     1
  Session Id:                           eee82364-f90e-49cb-b534-629b2
8cef285
```

- Status.Condition.Type shows whether the proposal is ready. ProposalReady means that the proposal is ready.
- Status.Optimization Result describes the recommended optimization.

d) Approve or refresh the proposal.

Refreshing or approving the proposal is done using annotation.

Approve the proposal if your are satisfied with it. Approving the proposal starts the rebalance process. Refresh the proposal if you are not satisfied with it or if it became outdated.

For Approve

```
kubectl annotate kafkarebalance [***RESOURCE NAME***] \
  strimzi.io/rebalance=approve \
  --namespace [***NAMESPACE***]
```

For Refresh

```
kubectl annotate kafkarebalance [***RESOURCE NAME***] \
  strimzi.io/rebalance=refresh \
  --namespace [***NAMESPACE***]
```

e) Monitor the rebalance process.

```
kubectl describe kafkarebalance [***RESOURCE NAME***] --names
pace [***NAMESPACE***]
```

Status.Conditions.Type in the output shows the current status of the rebalance process. While the rebalance is in progress, the type will be Rebalancing. After the rebalance is finished, the status changes to Ready. For example:

```
#...
Status:
  Conditions:
    Last Transition Time:  2023-10-18T09:48:13.561062593Z
    Status:                True
    Type:                  Rebalancing
  Observed Generation:     1
```

f) Delete the `KafkaRebalance` resource.

After the rebalance process completes successfully, you can choose to delete the `KafkaRebalance` resource if you no longer need it. Alternatively, you can keep the resource for later use. If you keep the resource, ensure that you refresh the generated proposal before initiating a new rebalance.

Downscaling Kafka brokers

Complete these steps to downscale the number of brokers in your Kafka cluster.

Before you begin

Ensure that Cruise Control is deployed in your cluster. See Deploying Cruise Control.

Procedure

1. Choose and annotate the broker you want to remove from the cluster.

The ID of the broker that you want to remove must be set with an annotation on the `KafkaNodePool` resource. Annotating the broker ID tells the Strimzi Cluster operator that this is the broker that should be removed when a downscale operation is initiated.

```
kubectl annotate kafkanodepool [***RESOURCE NAME***] \
   strimzi.io/remove-node-ids="[[***BROKER IDS***]]"
```

2. Rebalance your cluster with Cruise Control.

When downscaling the cluster, Cruise Control rebalancing is used to move data from brokers.

a) Create a YAML configuration describing your `KafkaRebalance` resource.

For example:

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaRebalance
metadata:
  name: my-downscale
  labels:
    strimzi.io/cluster: my-cluster
spec:
  mode: remove-brokers
  brokers: [3]
```

• Note down the resource name you specify in metadata.name. You will need to specify the name in the steps that follow.

- spec.mode specifies the mode to use for rebalancing. The remove-brokers mode is used before downscaling a cluster. This mode moves replicas from brokers that will be deleted.
- spec.brokers is a list of the broker IDs that will be removed from the cluster.

b) Create the resource.

```
kubectl apply --filename [***YAML CONFIG***] --names
pace [***NAMESPACE***]
```

Creating the resource generates an optimization proposal from Cruise Control.

c) View the generated optimization proposal.

```
kubectl describe kafkarebalance [***RESOURCE NAME***] --names
pace [***NAMESPACE***]
```

The generated optimization proposal will be similar to the following example.

```
#...
Status:
  Conditions:
    Last Transition Time:  2023-10-18T07:50:28.595242008Z
    Status:                True
    Type:                  ProposalReady
  Observed Generation:     1
  Optimization Result:
    After Before Load Config Map:  my-downscale
    Data To Move MB:                 1
    Excluded Brokers For Leadership:
    Excluded Brokers For Replica Move:
    Excluded Topics:
    Intra Broker Data To Move MB:        0
    Monitored Partitions Percentage:     100
    Num Intra Broker Replica Movements:  0
    Num Leader Movements:                3
    Num Replica Movements:               54
    On Demand Balancedness Score After:  87.91899658253526
    On Demand Balancedness Score Before: 74.87703685014604
    Provision Recommendation:
    Provision Status:                    RIGHT_SIZED
    Recent Windows:                      1
  Session Id:                            5b58f7d4-260d-4047-bdbe-253
4e395783c
```

- Status.Condition.Type shows whether the proposal is ready. ProposalReady means that the proposal is ready.
- Status.Optimization Result describes the recommended optimization.

d) Approve or refresh the proposal.

Refreshing or approving the proposal is done using annotation.

Approve the proposal if your are satisfied with it. Approving the proposal starts the rebalance process. Refresh the proposal if you are not satisfied with it or if it became outdated.

For Approve

```
 kubectl annotate kafkarebalance [***RESOURCE NAME***] \
    strimzi.io/rebalance=approve \
    --namespace [***NAMESPACE***]
```

For Refresh

```
 kubectl annotate kafkarebalance [***RESOURCE NAME***] \
```

```
        strimzi.io/rebalance=refresh \
        --namespace [***NAMESPACE***]
```

e) Monitor the rebalance process.

```
kubectl describe kafkarebalance [***RESOURCE NAME***] --names
pace [***NAMESPACE***]
```

The rebalance is finished once Status.Conditions.Type is Ready.

```
#...
Status:
  Conditions:
    Last Transition Time:   2023-10-18T09:49:28.612243136Z
    Status:                 True
    Type:                   Ready
```

⚠️ **Warning:** Before you continue with the next step, ensure that all data is moved from the brokers that will be deleted.

3. Remove the Kafka brokers from your cluster.

This is done by updating the replica count in your `KafkaNodePool` resources, which you can do in the following two ways.

- Update the value of spec.replicas directly in the resource and apply your changes.
- Scale the resource with kubectl   scale.

```
kubectl scale kafkanodepool [***NODE POOL NAME***] \
    --namespace [***NAMESPACE***] \
    --replicas=[***COUNT***]
```

The Strimzi Cluster Operator blocks the downscale operation if there are still replicas on the broker targeted for removal. If required, you can bypass this blocking mechanism.

4. Remove the annotation from the `KafkaNodePool` resource.

This annotation was added in a previous step and was used to influence which node should be removed from the cluster.

```
 kubectl annotate kafkanodepool [***NODE POOL NAME***] \
    --namespace [***NAMESPACE***] \
    strimzi.io/remove-node-ids-
```

Related Information
Skipping checks on scale-down operations | Strimzi

Log collection

Cloudera requires that the logs of the operator components are stored long term for diagnostic and supportability purposes. Learn about the settings for platform level log collection recommended by Cloudera.

Logs can be collected using the log collector feature of the specific Kubernetes platform. Ensuring that log collection is correctly set up is your responsibility. Cloudera recommends at least one week of retention time for the collected logs.

Using kubectl logs is not sufficient in some cases. This is because pods are created and destroyed dynamically by operator applications. The logs of destroyed pods are deleted, which makes them inaccessible. Log collection can ensure that the logs of already deleted pods are retained.

The following collects the recommended and required logging practices for specific Kubernetes platforms.

Openshift

Latest OpenShift versions support the Vector log collector. Log collection and forwarding can be configured using a ClusterLogging resource.

Ensure the following if you are on Openshift:

*   The `ClusterLogging` resource includes all namespaces and pods used by the operators.
*   Use a log sink that supports time-based retention. The `ClusterLogging` resource supports a number of log sinks. Cloudera recommends using a sink that supports time-based retention to limit storage costs. Additionally, the selected sink should allow easy access to the collected logs when a diagnostic investigation requires them.

Related Information
The Vector documentation
About log collection and forwarding | OpenShift
5.6 Logging API reference | OpenShift

RKE2 with Rancher

Rancher relies on the Logging operator for log collection. Log collection can be configured using Flow, ClusterFlow, Output, and ClusterOutput resources.

Ensure the following if you are on RKE2 with Rancher:

*   When using a `Flow` resource, ensure that the `Flow` resource includes all namespaces and pods used by the operators.
*   Use a log sink that supports time-based retention. `Output` and `ClusterOutput` resources support a number of log sinks. Cloudera recommends using a sink that supports time-based retention to limit storage costs. Additionally, the selected sink should allow easy access to the collected logs when a diagnostic investigation requires them.

Related Information
Rancher Integration with Logging Services | Rancher
Logging operator documentation

Monitoring with Prometheus

Learn about the example files Cloudera provides related to Prometheus monitoring. Additionally, learn about recommended metrics, alerts, and the Kafka Exporter.

Cloudera provides various example files related to the setup and configuration of Prometheus monitoring. These example files configure Kafka and other cluster components to expose recommended metrics. Additionally, they set up a Prometheus instance that scrapes exposed metrics and publishes recommended alerts.

The example files are hosted on the Cloudera Archive. They are located in /csm-operator/1.0/examples/metrics/.

The example files related to Prometheus are as follows.

*   kafka-metrics.yaml – A configuration file that includes `Kafka`, `KafkaNodePool`, and `ConfigMap` resource examples. You can use this configuration example to deploy a Kafka cluster that exposes the recommended metrics that Prometheus can scrape.

    If you already have a cluster and want to configure your existing Kafka to expose metrics, review the `ConfigMap` manifest in this example file. The `ConfigMap` specifies what Kafka and ZooKeeper metrics are

exposed. For comprehensive steps on how to configure Kafka and ZooKeeper to expose Prometheus compatible metrics, see *Configuring Kafka for Prometheus monitoring*.

- /prometheus-install/

    - alertmanager.yaml – An example `AlertManager` resource for deploying and configuring the Prometheus Alertmanager.
    - prometheus.yaml – A configuration file that you can use to deploy a Prometheus server.
    - prometheus-rules.yaml – An example `PrometheusRule` resource that includes alert rules recommended by Cloudera.
    - strimzi-pod-monitor.yaml – Includes examples of `PodMonitor` resources. These resources define Prometheus jobs that scrape metrics directly from pods. `Podmonitor` resources are used to scrape metrics data directly from Kafka, ZooKeeper, Operator, Kafka Bridge and Cruise Control pods.

- /prometheus-additional-properties/

    - prometheus-additional.yaml – A `Secret` resource that stores additional Prometheus configuration for scraping CPU, memory, and disk volume usage metrics. These metrics are reported by the Kubernetes cAdvisor agent and kubelet on the nodes.

- /prometheus-alertmanager-config/

    - alert-manager-config.yaml – A `Secret` resource containing additional configuration for the Prometheus Operator. The configuration specifies hook definitions for sending notifications from your Kafka cluster through the Alertmanager.

> **Note:** Configuration examples located in the prometheus-install, prometheus-additional-properties, and prometheus-alertmanager-config directories are Prometheus Operator resource examples. You must have a running instance of the Prometheus Operator in your cluster to use them.

Related Information

Configuring Kafka for Prometheus monitoring

Cloudera Archive

Prometheus Operator

Prometheus metrics

To expose metrics recommended by Cloudera, set up and configure both Prometheus and Kafka instances using the example configuration files provided by Cloudera.

The example files are hosted on the Cloudera Archive. They are located in /csm-operator/1.0/examples/metrics/. When you deploy using these examples, your deployment exposes and monitors the metrics recommended by Cloudera.

The specific metrics that you need to monitor will depend on your use case and operational objectives. As a result, any metric can be useful and there are no metrics that can be highlighted. Start out with the provided example files and make changes as necessary.

Prometheus alerts

Cloudera provides a Prometheus alert configuration example that contains recommended alert rules. Learn about the highlighted alerts defined in this example. Additionally, learn about configuring custom alerts for Kafka components and the Strimzi Cluster Operator.

Default (recommended) alerts

The prometheus-rules.yaml file is an example `PrometheusRule` resource that has various alert rules specified. The alerts specified in this example are generally useful for most use cases and all of them are recommended by Cloudera.

The following is a list of the highlighted alerts defined in the example. Ensure that you always have these alerts configured as they give good insight into the state and health of your cluster.

Kafka

- KafkaRunningOutOfSpace – Kafka is running out of free disk space. Reported for each Persistent Volume Claim.
- UnderReplicatedPartitions – Kafka has underreplicated partitions. Reported for each Kafka pod.
- OfflinePartitions – One or more partitions have no leader on the actual Kafka pods.
- OfflineLogDirectoryCount – Reports the number of offline log directories located on the actual Kafka pod.
- KafkaNetworkProcessorAvgIdle – Less than 30% of network processor capacity available on the actual Kafka pod. You can avoid this alert by increasing the num.network.threads broker property.
- KafkaRequestHandlerAvgIdle – Less than 30% of request handler capacity is available on the actual Kafka pod. You can avoid this alert by increasing the num.io.threads broker property.
- ClusterOperatorContainerDown – The Strimzi Cluster Operator has been down for longer than 90 seconds.

ZooKeeper

- AvgRequestLatency – Zookeeper average request latency on the pod.
- ZookeeperRunningOutOfSpace – Zookeeper is running out of free disk space.

Custom alerts and groups

In addition to the default alerts, you can define custom ones as well. To do this, you extend your `PrometheusRule` resource (prometheus-rules.yaml) with additional alert rules.

Alert rules are grouped and the prometheus-rules.yaml example contains the following default groups.

- kafka
- zookeeper
- entityOperator
- kafkaExporter

For example, to monitor Cruise Control, you must introduce a Cruise Control group that contains valid alert rules. Specifying a new group is also useful if you want to identify host machine related problems. You can find more information on defining alert rules in the Prometheus documentation.

Alerts for the Strimzi Cluster Operator

By default, prometheus-rules.yaml contains a single alert related to the Strimzi Cluster Operator. This alert monitors whether the container of the operator is down. You can define additional alerts using the following metrics.

- strimzi_reconciliations_already_enqueued_total – Number of reconciliations skipped because another reconciliation for the same resource was still running.
- strimzi_reconciliations_duration_seconds – The time reconciliation takes to complete.
- strimzi_reconciliations_duration_seconds_max – Max time of reconciliation takes.
- strimzi_reconciliations_failed_total – Number of failed reconciliations done by the operator for individual resources which failed.
- strimzi_reconciliations_locked_total – Number of reconciliations skipped because another reconciliation for the same resource was still running.
- strimzi_reconciliations_max_batch_size – Max size recorded for a single event batch.
- strimzi_reconciliations_periodical_total – Number of periodical reconciliations done by the operator.
- strimzi_reconciliations_successful_total – Number of reconciliations done by the operator for individual resources which were successful.
- strimzi_reconciliations_total – Number of reconciliations done by the operator for individual resources.

- strimzi_resources – Number of custom resources the operator sees.
- strimzi_resources_paused – Number of custom resources with paused reconciliations.

Related Information
Alerting rules | Prometheus

Kafka Exporter

You can use Kafka Exporter to publish additional Kafka metrics related to brokers and clients.

Kafka Exporter is an open source project to enhance the monitoring of Apache Kafka brokers and clients. Kafka Exporter extracts additional metrics data from Kafka brokers related to offsets, consumer groups, consumer lag, and topics.

If Kafka Exporter is deployed, it is typically deployed with its default configuration (spec.kafkaExporter: {}). Cloudera recommends that you deploy Kafka Exporter and customize its configuration based on your cluster and operational objectives.

Cloudera recommends that at minimum you capture additional metrics for your mission critical topics and groups. Additional metrics include metrics related to latest offsets, consumer lags, and others.

The following example configures the Kafka Exporter to collect additional metrics from all topics and groups.

```
#...
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafkaExporter:
    topicRegex: ".*"
    groupRegex: ".*"
```

This configuration snippet is included in the kafka-metrics.yaml example provided by Cloudera, which is the recommended baseline example for a Kafka deployment that has metric collection enabled.

Related Information
Kafka Exporter | GitHub
KafkaExporterSpec schema reference | Strimzi

Diagnostics

Learn about collecting diagnostics information, the diagnostic tools shipped with CSM Operator, as well as a number of useful kubectl commands that you can use to gather diagnostic information.

Cloudera provides various command line tools that you can use to capture diagnostic bundles, thread dumps, and other types of information about your CSM Operator installation. You use these tools when contacting Cloudera support or when troubleshooting issues.

There are three tools available.

- report.sh – A diagnostic bundle tool that captures various information about your CSM Operator installation.
- java_thread_dump.sh – A thread dump capturing tool that collects thread dumps of containers in a specified pod.
- kafka_shell.sh – An administrative tool that sets up a pod where you can easily run Kafka command line tools.

Diagnostic tools are not downloaded, deployed, or installed when you install CSM Operator and its components. You must download and run them separately. All tools are available for download from the Cloudera Archive. They are located in the /csm-operator/1.0/tools/ directory.

In addition to the tools provided by Cloudera, you can also use kubectl to gather diagnostics and troubleshooting data.

Related Information

Cloudera Archive

Capturing a diagnostic bundle with report.sh

Use report.sh to capture diagnostic information about your deployment.

About this task

CSM Operator diagnostic bundles are captured using the report.sh command line tool. The bundle that the tool captures is used as the baseline when contacting Cloudera support for assistance with CSM Operator. The bundle captures all available, cluster-wide information about CSM Operator.

Before you begin

- Ensure that you have access to your Cloudera credentials (username and password).
- Ensure that the environment where you run the tool has the following:

  - Bash 4 or higher
  - GNU utilities:

    - echo
    - grep
    - sed
    - date
  - base64
  - kubectl or oc
  - kubeconfig configured to target Kubernetes cluster
  - zip

Procedure

1. Download the tool.

   ```
   curl --user [***USERNAME***] \
     https://archive.cloudera.com/p/csm-operator/1.0/tools/report.sh \
     --output report.sh \
   && chmod +x report.sh
   ```

   Replace *[*** USERNAME***]* with your Cloudera username. Enter your Cloudera password when prompted.
2. Capture a diagnostic bundle.

   ```
   ./report.sh
   ```

   The tool prints the resources it collects information on. Afterward it generates a diagnostic bundle ZIP (report file). The path to the generated ZIP is printed on the standard output.

   > **Tip:** Use the --help option to view additional options and information on tool usage.

Capturing a thread dump of a pod with java_thread_dump.sh

Use java_thread_dump.sh to capture a thread dump of a pod.

About this task

Some types of issues require investigating the threads of the components running in a CSM Operator installation. You can use the java_thread_dump.sh command line tool to capture the thread dumps of all containers of a specific pod with the specified number of samples and frequency.

Before you begin

- Ensure that you have access to your Cloudera credentials (username and password).
- Ensure that the environment where you run the tool has the following:

  - Bash 4 or higher
  - GNU utilities:

    - echo
    - grep
    - sed
    - date
  - kubectl or oc
  - kubeconfig configured to target Kubernetes cluster
  - zip

Procedure

1. Download the tool.

   ```
   curl --user [***USERNAME***] \
     https://archive.cloudera.com/p/csm-operator/1.0/tools/java_thread_dump.
   sh \
     --output java_thread_dump.sh \
   && chmod +x java_thread_dump.sh
   ```

   Replace *[***USERNAME***]* with your Cloudera username. Enter your Cloudera password when prompted.

2. Capture a thread dump of a pod.

   ```
   ./dump.sh --namespace=[***POD NAMESPACE***] \
     --pod=[***POD NAME***] \
     --dumps=[***NUMBER OF THREAD DUMPS***] \
     --interval=[***DUMP INTERVAL IN SECONDS***]
   ```

   The tool collects the specified number of thread dumps for the specified pod with the specified interval. Afterward, it generates a ZIP (report file) containing the thread dumps.

   > **Tip:** Use the --help option to view additional options and information on tool usage.

Using kafka_shell.sh

Use kafka_shell.sh to set up a pod where Kafka CLI tools are readily available.

About this task

Kafka is shipped with a number of useful CLI tools. Easy access to these tools is essential for administering and troubleshooting your cluster. The kafka_shell.sh command line tool creates a pod where all Kafka CLI tools are readily available, and full Kafka admin client configurations are prepared.

The pod created by kafka_shell.sh:

- Uses the Kafka docker image. This means that Kafka CLI tools are readily accessible within the pod.

- Has both a truststore and keystore present that give you administrative privileges.
- Has a ready-to-use client configuration file available at /tmp/client.properties.
- Has bootstrap server configuration available in the BOOTSTRAP_SERVERS environment variable.

You can use the tool in two ways. You either use it interactively, or run one-off commands using pipe.

Before you begin

⚠️ **Caution:** This tool gives administrative access to the Kafka cluster. Cloudera advises caution when using this tool.

- Ensure that you have access to your Cloudera credentials (username and password).
- Ensure that the environment where you run the tool has the following:
  - Bash 4 or higher
  - GNU utilities:
    - echo
    - grep
    - sed
    - date
    - cut
    - head
  - kubectl or oc
  - kubeconfig configured to target Kubernetes cluster
  - zip

Procedure

1. Download the tool.

```
curl --user [***USERNAME***] \
  https://archive.cloudera.com/p/csm-operator/1.0/tools/kafka_shell.sh \
  --output kafka_shell.sh \
&& chmod +x kafka_shell.sh
```

Replace *[***USERNAME***]* with your Cloudera username. Enter your Cloudera password when prompted.

2. Use the tool.

You have two choices. You can either use the tool interactively. In this case, you run the tool which opens an interactive shell window where you run your Kafka CLI commands. Alternatively, you can use pipe (|) to run Kafka CLI commands one at a time.

> **For Interactive**
>
> **a.** Run the tool.
>
> ```
> ./kafka_shell.sh \
>   --namespace=[***KAFKA CLUSTER NAMESPACE***] \
>   --cluster=[***KAFKA CLUSTER NAME***]
> ```
>
> **b.** Run your Kafka CLI command within the shell that opens.
>
> For example, you can list your topics with the following command.
>
> ```
> bin/kafka-topics.sh \
>   --list \
>   --command-config /tmp/client.properties \
> ```

```
      --bootstrap-server $BOOTSTRAP_SERVERS
```

The kafka-shell pod is deleted after you exit the interactive shell.

For Pipe

To run one-off commands, pipe them into kafka_admin_shell.sh. For example:

```
echo 'bin/kafka-topics.sh \
  --list \
  --command-config /tmp/client.properties \
  --bootstrap-server $BOOTSTRAP_SERVERS' \
| ./kafka_shell.sh --namespace=[***KAFKA CLUSTER NAMESPACE***] \
  --cluster=[***KAFKA CLUSTER NAME***]
```

The kafka-shell pod is deleted after you run your command.

> **Tip:** Use the --help option to view additional options and information on tool usage.

Monitoring pod status during reconciliation

You can check the status of the pods after applying a change to the deployment configuration using kubectl get pods.

```
kubectl get pods --namespace [***NAMESPACE***] --output wide --watch
```

Reading Strimzi Cluster Operator logs

The Strimzi Cluster Operator log contains useful information about the tasks that the operator performs and details for failed operations. You can check the Strimzi Cluster Operator logs with kubectl logs.

```
kubectl logs [***STRIMZI CLUSTER OPERATOR POD***] --names
pace [***NAMESPACE***]
```

Reading effective generated Kafka broker properties

You can get the effective Kafka properties of a broker using kubectl exec. Broker properties are generated by the Strimzi Cluster Operator.

```
kubectl exec -it \
  --namespace [***NAMESPACE***] \
  [***KAFKA BROKER POD***] \
  --container kafka \
  -- /bin/bash -c "cat /tmp/strimzi.properties"
```

Reading Kafka broker logs

You can check the Kafka broker logs with kubectl logs.

```
kubectl logs [***KAFKA BROKER POD***] --namespace [***NAMESPACE***] -f
```

Sizing and performance considerations

Learn about ways you can size your Kafka broker resource allocations for optimal performance.

Kafka broker performance primarily depends on the IO bandwidth of the nodes and disks. Because of this, Cloudera recommends using SSDs with high IOPS and throughput for large workloads. When optimizing for large workloads, avoid using HDDs and storage replication services, such as Longhorn. JBOD can also lead to throughput improvements when the node IO bandwidth can support multiple disks.

Depending on the characteristics of the workload, brokers might require a large memory pool to be able to serve fetch requests from the cache. Brokers might also require an increased CPU allocation to support compressed messages

Zookeeper requires a small resource pool in most workloads, and scaling the cluster to more than three nodes usually provides no benefit.

Recommended minimum setup

Learn about the minimum number of broker and nodes, as well as minimum resource allocations for a cluster that enable optimal cluster performance.

As a baseline for small and medium workloads, Cloudera recommends the following cluster sizing:

* Three brokers, each with eight CPU cores and 20 GB memory.
* Three Zookeper nodes, each with four CPU cores and 16 GB memory

Updating a license

CSM 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