Cloudera Streams Messaging Operator 1.2.0

Kafka Operations

Date published: 2024-06-11
Date modified: 2024-12-02

https://docs.cloudera.com/

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 Cloudera Streams Messaging - Kubernetes 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

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