Apache Kafka 3

Mirroring data between Kafka clusters

Date of Publish: 2018-08-30

http://docs.hortonworks.com

Contents

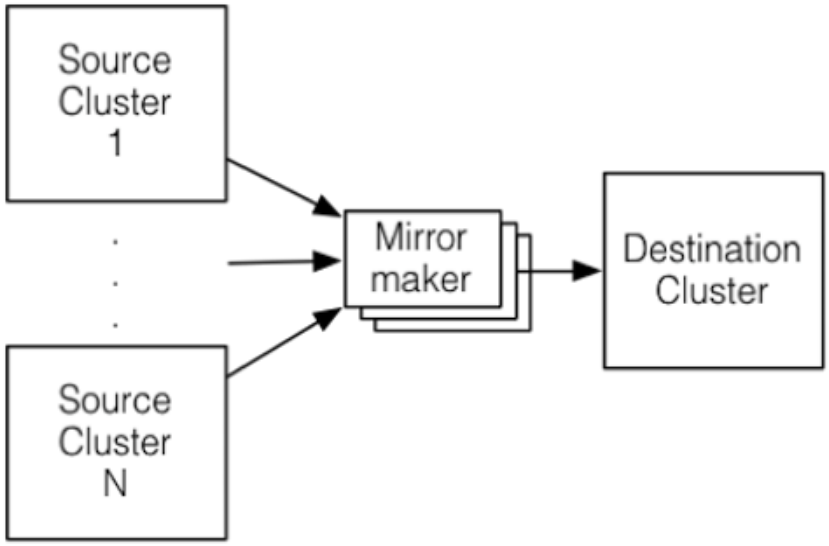
Mirroring Data Between Clusters: Using the MirrorMaker Tool.....3

Running MirrorMaker.....3
Checking Mirroring Progress.....5
Avoiding Data Loss.....5
Running MirrorMaker on Kerberos-Enabled Clusters.....6

Mirroring Data Between Clusters: Using the MirrorMaker Tool

The process of replicating data between Kafka clusters is called "mirroring", to differentiate cross-cluster replication from replication among nodes within a single cluster. A common use for mirroring is to maintain a separate copy of a Kafka cluster in another data center.

Kafka's MirrorMaker tool reads data from topics in one or more source Kafka clusters, and writes corresponding topics to a destination Kafka cluster (using the same topic names):



To mirror more than one source cluster, start at least one MirrorMaker instance for each source cluster.

You can also use multiple MirrorMaker processes to mirror topics within the same consumer group. This can increase throughput and enhance fault-tolerance: if one process dies, the others will take over the additional load.

The source and destination clusters are completely independent, so they can have different numbers of partitions and different offsets. The destination (mirror) cluster is not intended to be a mechanism for fault-tolerance, because the consumer position will be different. (The MirrorMaker process will, however, retain and use the message key for partitioning, preserving order on a per-key basis.) For fault tolerance we recommend using standard within-cluster replication.

Running MirrorMaker

Prerequisite: The source and destination clusters must be deployed and running.

To set up a mirror, run kafka.tools.MirrorMaker. The following table lists configuration options.

At a minimum, MirrorMaker requires one or more consumer configuration files, a producer configuration file, and either a whitelist or a blacklist of topics. In the consumer and producer configuration files, point the consumer to the ZooKeeper process on the source cluster, and point the producer to the ZooKeeper process on the destination (mirror) cluster, respectively.

Table 1: MirrorMaker Options

Parameter	Description	Examples

--consumer.config	Specifies a file that contains configuration settings for the source cluster. For more information about this file, see the "Consumer Configuration File" subsection.	--consumer.config hdp1-consumer.properties
--producer.config	Specifies the file that contains configuration settings for the target cluster. For more information about this file, see the "Producer Configuration File" subsection.	--producer.config hdp1-producer.properties
--whitelist --blacklist	(Optional) For a partial mirror, you can specify exactly one comma-separated list of topics to include (--whitelist) or exclude (--blacklist). In general, these options accept Java regex patterns. For caveats, see the note after this table.	--whitelist my-topic
--num.streams	Specifies the number of consumer stream threads to create.	--num.streams 4
--num.producers	Specifies the number of producer instances. Setting this to a value greater than one establishes a producer pool that can increase throughput.	--num.producers 2
--queue.size	Queue size: number of messages that are buffered, in terms of number of messages between the consumer and producer. Default = 10000.	--queue.size 2000
--help	List MirrorMaker command-line options.	

Note:

- A comma (',') is interpreted as the regex-choice symbol ('|') for convenience.
- If you specify --white-list=".*", MirrorMaker tries to fetch data from the system-level topic __consumer-offsets and produce that data to the target cluster. This can result in the following error:

  Producer cannot send requests to __consumer-offsets

  Workaround: Specify topic names, or to replicate all topics, specify --blacklist="__consumer-offsets".

The following example replicates topic1 and topic2 from sourceClusterConsumer to targetClusterProducer:

```
/usr/hdp/current/kafka-broker/bin/kafka-run-class.sh kafka.tools.MirrorMaker
 --consumer.config sourceClusterConsumer.properties  --producer.config
 targetClusterProducer.properties --whitelist="topic1, topic"
```

Consumer Configuration File

The consumer configuration file must specify the ZooKeeper process in the source cluster.

Here is a sample consumer configuration file:

```
zk.connect=hdp1:2181/kafka
zk.connectiontimeout.ms=1000000
consumer.timeout.ms=-1
groupid=dp-MirrorMaker-test-datap1
shallow.iterator.enable=true
mirror.topics.whitelist=app_log
```

Producer Configuration File

The producer configuration should point to the target cluster's ZooKeeper process (or use the broker.list parameter to specify a list of brokers on the destination cluster).

Here is a sample producer configuration file:

```
zk.connect=hdp1:2181/kafka-test
producer.type=async
compression.codec=0
serializer.class=kafka.serializer.DefaultEncoder
max.message.size=10000000
queue.time=1000
queue.enqueueTimeout.ms=-1
```

Checking Mirroring Progress

You can use Kafka's Consumer Offset Checker command-line tool to assess how well your mirror is keeping up with the source cluster. The Consumer Offset Checker checks the number of messages read and written, and reports the lag for each consumer in a specified consumer group.

The following command runs the Consumer Offset Checker for group KafkaMirror, topic test-topic. The --zkconnect argument points to the ZooKeeper host and port on the source cluster.

```
/usr/hdp/current/kafka/bin/kafka-run-class.sh
 kafka.tools.ConsumerOffsetChecker --group KafkaMirror --zkconnect source-
cluster-zookeeper:2181 --topic test-topic

Group        Topic          Pid Offset     logSize     Lag     Owner
KafkaMirror  test-topic     0   5          5           0       none
KafkaMirror  test-topic     1   3          4           1       none
KafkaMirror  test-topic     2   6          9           3       none
```

Table 2: Consumer Offset Checker Options

--group	(Required) Specifies the consumer group.
--zkconnect	Specifies the ZooKeeper connect string. The default is localhost:2181.
--broker-info	Lists broker information
--help	Lists offset checker options.
--topic	Specifies a comma-separated list of consumer topics. If you do not specify a topic, the offset checker will display information for all topics under the given consumer group.

Avoiding Data Loss

If for some reason the producer cannot deliver messages that have been consumed and committed by the consumer, it is possible for a MirrorMaker process to lose data.

To prevent data loss, use the following settings. (Note: these are the default settings.)

- For consumers:
  - auto.commit.enabled=false
- For producers:
  - max.in.flight.requests.per.connection=1
  - retries=Int.MaxValue
  - acks=-1
  - block.on.buffer.full=true
- Specify the --abortOnSendFail option to MirrorMaker

The following actions will be taken by MirrorMaker:

- MirrorMaker will send only one request to a broker at any given point.
- If any exception is caught in the MirrorMaker thread, MirrorMaker will try to commit the acked offsets and then exit immediately.
- On a RetriableException in the producer, the producer will retry indefinitely. If the retry does not work, MirrorMaker will eventually halt when the producer buffer is full.
- On a non-retriable exception, if --abort.on.send.fail is specified, MirrorMaker will stop.

  If --abort.on.send.fail is not specified, the producer callback mechanism will record the message that was not sent, and MirrorMaker will continue running. In this case, the message will not be replicated in the target cluster.

Running MirrorMaker on Kerberos-Enabled Clusters

To run MirrorMaker on a Kerberos/SASL-enabled cluster, configure producer and consumer properties as follows:

Procedure

1. Choose or add a new principal for MirrorMaker. Do not use kafka or any other service accounts. The following example uses principal mirrormaker.
2. Create client-side Kerberos keytabs for your MirrorMaker principal. For example:

```
sudo kadmin.local -q "ktadd -k /tmp/mirrormaker.keytab mirrormaker/
HOSTNAME@EXAMPLE.COM"
```

3. Add a new Jaas configuration file to the node where you plan to run MirrorMaker:

```
-Djava.security.auth.login.config=/usr/hdp/current/kafka-broker/config/
kafka_mirrormaker_jaas.conf
```

4. Add the following settings to the KafkaClient section of the new Jaas configuration file. Make sure the principal has permissions on both the source cluster and the target cluster.

```
KafkaClient {
     com.sun.security.auth.module.Krb5LoginModule required
     useKeyTab=true
     keyTab="/tmp/mirrormaker.keytab"
     storeKey=true
     useTicketCache=false
     serviceName="kafka"
     principal="mirrormaker/HOSTNAME@EXAMPLE.COM";
    };
```

5. Run the following ACL command on the source and destination Kafka clusters:

```
bin/kafka-acls.sh
--topic test-topic
--add
--allow-principal user:mirrormaker
--operation ALL
--config /usr/hdp/current/kafka-broker/config/server.properties
```

6. In your MirrorMaker consumer.config and producer.config files, specify security.protocol=SASL_PLAINTEXT.
7. Start MirrorMaker. Specify the new.consumer option in addition to your other options. For example:

```
/usr/hdp/current/kafka-broker/bin/kafka-run-class.sh
 kafka.tools.MirrorMaker
--consumer.config consumer.properties
--producer.config target-cluster-producer.properties
```

```
--whitelist my-topic
--new.consumer
```