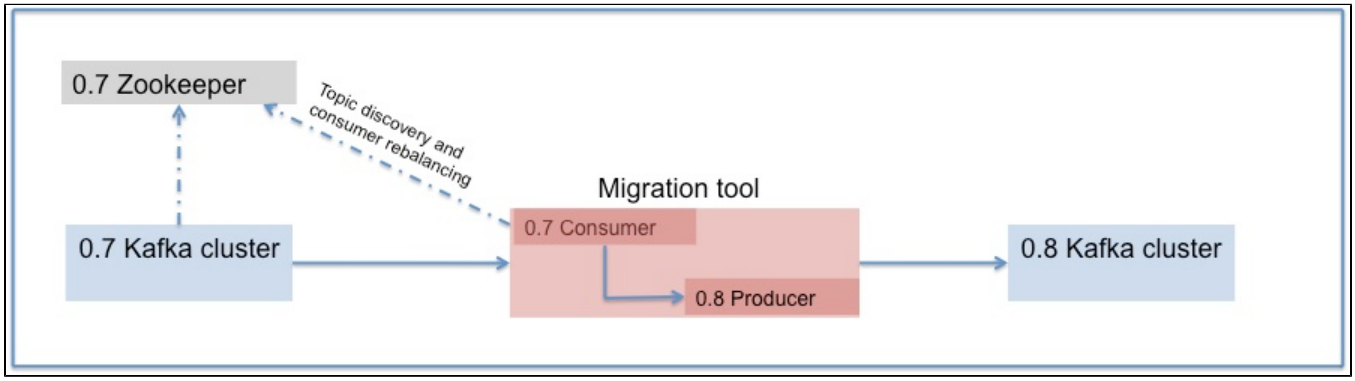
Migrating from 0.7 to 0.8

Kafka provides the ability to seamlessly migrate from 0.7 to 0.8 using a migration tool. The tool uses a 0.7 Kafka consumer to consume messages from the 0.7 source cluster and re-publishes those messages to the 0.8 target cluster using an embedded Kafka producer.



Steps to move from 0.7 to 0.8

1. Setup a separate 0.8 cluster (Do not modify anything in the existing 0.7 cluster). More info on setup can be found here - https://cwiki.apache.org/confluence/display/KAFKA/Kafka+0.8+Quick+Start
2. Setup the migration tool to consume messages from the 0.7 cluster and re-publish them to the 0.8 cluster (Details below)
3. Move the 0.7 consumers consuming from the 0.7 cluster to 0.8
4. Move the 0.7 producers to 0.8

The rest of the document describes how the migration tool can be setup followed by some FAQ.

How to set up the Migration tool

The migration tool takes one or more consumer configurations, a producer configuration, kafka 0.7 jar and zkclient.01 jar (Needed by kafka 0.7 consumer) and either a whitelist or a blacklist. You need to point the consumer to the source cluster's ZooKeeper, and the producer to the 0.8 broker.list.

```
kafka-run-class.sh kafka.tools.KafkaMigrationTool --kafka.07.jar kafka-0.7.19.jar --zkclient.01.jar zkclient-
0.2.0.jar --num.producers 16 --consumer.config=sourceCluster2Consumer.config --producer.
config=targetClusterProducer.config --whitelist=.*
```

Import configuration parameters for the migration tool

Whitelist or blacklist

The migration tool accepts exactly one of whitelist or blacklist. These are standard Java regex patterns.

Blocking producer

In order to sustain a higher throughput, you would typically use an asynchronous embedded producer and it should be configured to be in blocking mode (i.e., *queue.enqueueTimeout.ms=-1*). This recommendation is to ensure that messages will not be lost. Otherwise, the default enqueue timeout of the asynchronous producer is zero which means if the producer's internal queue is full, then messages will be dropped due to *QueueFullExceptions*. A blocking producer however, will wait if the queue is full, and effectively throttle back the embedded consumer's consumption rate. You can enable trace logging in the producer to observe the remaining queue size over time. If the producer's queue is consistently full, it indicates that the migration tool is bottle-necked on re-publishing messages to the target cluster and/or flushing messages to disk.

Number of producers

You can use the *--num.producers* option to use a producer pool in the migration tool to increase throughput. This helps because each producer's requests are effectively handled by a single thread on the receiving Kafka broker. i.e., even if you have multiple consumption streams (see next section), the throughput can be bottle-necked at the handling stage of the migration tool's producer requests.

Number of consumption streams

Use the --num.streams option to specify the number of consumer threads to create. Note that if you start multiple migration tool processes, you may want to look at the distribution of partitions on the source cluster. If the number of consumption streams is too high per migration tool process, then some of the consumer threads will be idle by virtue of the consumer rebalancing algorithm (if they do not end up owning any partitions for consumption).

FAQ

Is shallow iteration supported for the Migration tool's consumer config?

No. The message format has changed from 0.7 to 0.8. The migration tool needs to convert the messages to the new format for which it needs to do the deep iteration.

When the 0.8 consumer is made to consume from the 0.8 cluster, what happens to the offsets?

In 0.8, we have moved to logical offsets from physical offsets. This means that the offsets are not compatible. When you try to consume using the 0.7 offsets, you would hit "OffsetOutOfRangeException". The default behavior of the consumer when this happens is based on the config value of "auto.offset.reset". If it is set to "smallest", the consumer will start consuming from the beginning. If it is set to "largest", the consumer will start consuming from the end.