KIP-310: Add a Kafka Source Connector to Kafka Connect

Status

Current state: *Discarded. (KIP-382 supersedes this with a much more complete vision)*

Discussion thread: *here*

JIRA: *KAFKA-6963*

Motivation

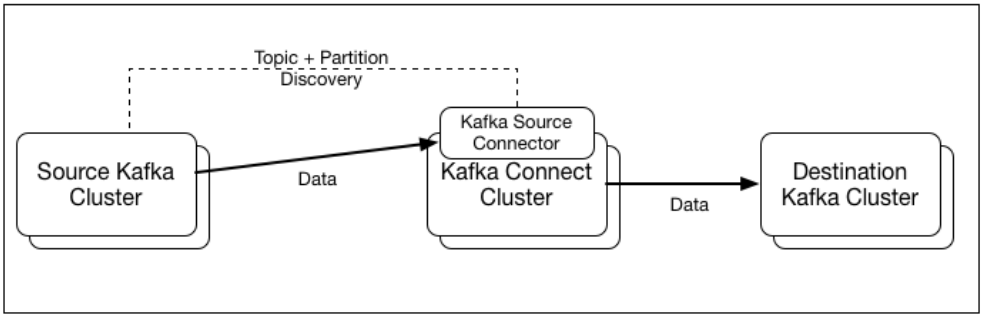
Copying messages between Kafka clusters is common enough that Kafka has had its own standalone MirrorMaker tool early on in the project. Kafka Connect was introduced to provide a standard framework for moving data into and out of Kafka, and it has been quickly adopted by the community with a large number of connectors available for other platforms. One such connector that does not exist is a Kafka connector. While MirrorMaker does satisfy the basic functional requirement copying messages between clusters, its standalone nature means that there is additional work required in maintaining the MirrorMaker cluster, especially in environments where Kafka Connect will also be used.

Public Interfaces

This proposal introduces a new Kafka Connect Source Connector. The source connector is bundled with Kafka Connect alongside the existing file connector, and can be used to copy data from a Source Kafka Cluster defined in the task configuration, to the Destination Kafka Cluster, denied by the Kafka Connect configuration.

Proposed Changes

Implementation of a Kafka Source Connector for Kafka Connect. The connector will enable users to implement MirrorMaker like cluster mirroring via Kafka Connect.



Key Requirements

- At least once delivery to the destination cluster must be supported
- Can specify a topic whitelist as a regular expression (Same as MirrorMaker)

Basic Implementation Components

An existing implementation of a Kafka Source Connector is available and provides a solid base for implementation in Kafka. The key components of this solution are:

Partition Monitor

Starts an AdminClient in a separate thread when the connector is first started (prior to determining the number of tasks to run) to determine the topics and partitions available on the source cluster which match the supplied topic list. After the first check of the available partitions, periodically runs (at a configurable interval) to check if the matching topics or partitions on the source cluster have changed. Triggers a task reconfiguration if it detects that there has been a change in the matching topic partitions (or, leaders - if option is selected to reconfigure on leader change)

Source Task

Each source task will be started with a list of topic partitions to subscribe to. On task start will start a KafkaConsumer and manually **assign** *(rather than the high level subscribe())* these topic partitions to the consumer. On task start, will check the Kafka connect offset storage for an existing offset for each topic partition - if found then it will resume reading the topic partition at this offset, otherwise it will begin from the beginning or the end of the topic (configurable). Each time the source task's **poll()** method is called, it will poll the Kafka consumer for available messages, and return them to the connect framework to be delivered to the Destination cluster.

Note that offset checkpointing is managed by Kafka Connect. As it is common to monitor consumer lag using consumer groups, the connector will also commit the last batch of offsets to the source Kafka cluster before polling the consumer for new data, unless this is turned off via configuration (consumer.enable.auto.commit = false).

Configuration Options

Standard Options

These are the most common options that are required when configuring this connector:

Configuration Parameter	Example	Description
source.bootstrap.servers	source.broker1:9092,source.broker2:9092	Mandatory. Comma separated list of boostrap servers for the source Kafka cluster
source.topic.whitelist	topic, topic-prefix*	Java regular expression to match topics to mirror. For convenience, comma (',') is interpreted as the regex-choice symbol ('|').
source.auto.offset.reset	latest	If there is no stored offset for a partition, indicates where to start consuming from. Options are *"earliest"* or *"latest"*. Default: *earliest*
source.group.id	kafka-connect	Group ID used when writing offsets back to source cluster (for offset lag tracking)
destination.topics.prefix	aggregate.	Prefix to add to source topic names when determining the Kafka topic to publish data to

Advanced Options

Some use cases may require modifying the following default connector options.

Configuration Parameter	Default	Description
include.message.headers	true	Indicates whether message headers from source records should be included when delivered to the destination cluster.
topic.list.timeout.ms	60000	Amount of time (in milliseconds) the partition monitor thread should wait for the source kafka cluster to return topic information before logging a timeout error.
topic.list.poll.interval.ms	300000	Amount of time (in milliseconds) the partition monitor will wait before re-querying the source cluster for a change in the topic partitions to be consumed
reconfigure.tasks.on.leader.change	false	Indicates whether the partition monitor should request a task reconfiguration when partition leaders have changed. In some cases this may be a minor optimization as when generating task configurations, the connector will try to group partitions to be consumed by each task by the leader node. The downside to this is that it may result in additional rebalances.

poll.loop.timeout.ms	1000	Maximum amount of time (in milliseconds) the connector will wait in each poll loop without data before returning control to the kafka connect task thread.
max.shutdown.wait.ms	2000	Maximum amount of time (in milliseconds) to wait for the connector to gracefully shut down before forcing the consumer and admin clients to close. Note that any values greater than the kafka connect parameter *task.shutdown.graceful.timeout.ms* will not have any effect.
source.max.poll.records	500	Maximum number of records to return from each poll of the internal KafkaConsumer. When dealing with topics with very large messages, the connector may sometimes spend too long processing each batch of records, causing lag in offset commits, or in serious cases, unnecessary consumer rebalances. Reducing this value can help in these scenarios. Conversely, when processing very small messages, increasing this value may improve overall throughput.
source.key.deserializer	org.apache.kafka.common.serialization.ByteArrayDeserializer	Key deserializer to use for the kafka consumers connecting to the source cluster.
source.value.deserializer	org.apache.kafka.common.serialization.ByteArrayDeserializer	Value deserializer to use for the kafka consumers connecting to the source cluster.
source.enable.auto.commit	true	If true the consumer's offset will be periodically committed to the source cluster in the background. Note that these offsets are not used to resume the connector (They are stored in the Kafka Connect offset store), but may be useful in monitoring the current offset lag of this connector on the source cluster

Overriding the internal KafkaConsumer and AdminClient Configuration

Note that standard Kafka parameters can be passed to the internal KafkaConsumer and AdminClient by prefixing the standard configuration parameters with *"source."*.

For cases where the configuration for the KafkaConsumer and AdminClient diverges, you can use the more explicit *"connector.consumer."* and *"connector.admin."* configuration parameter prefixes to fine tune the settings used for each.

Compatibility, Deprecation, and Migration Plan

There is no impact on existing code.

Rejected Alternatives

Sink Connector - This could also be implemented as a Sink Connector. A source connector was chosen as a pull based mode is more common in the environments the author has encountered. A sink connector should be considered for a follow up release to support the cases where the "owner" of the Kafka connect instance wishes to push messages into another cluster.