KIP-303: Add Dynamic Routing in Streams Sink

Status

Current state: *Accepted*

Discussion thread: TBD

JIRA: **KAFKA-4936** - Getting issue details... STATUS

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

Currently, all used output topics must be know beforehand, and thus, it's not possible to send output records to topic in a dynamic fashion.

By allowing users to dynamically choose which topic to send to at runtime based on each record's key value pairs, Streams can help removing the burden of having to update and restart KStreams applications when routing decisions are changed.

Public Interfaces

I propose adding the dynamic routing functionality at both the Topology and the DSL layer. More specifically, the following new APIs will be added:

```
// new class: o.a.k.streams.streams.processor.RecordContext


public interface RecordContext {
    /**
     * @return  The offset of the original record received from Kafka;
     *          could be -1 if it is not available
     */
    long offset();

    /**
     * @return  The timestamp extracted from the record received from Kafka;
     *          could be -1 if it is not available
     */
    long timestamp();

    /**
     * @return  The topic the record was received on;
     *          could be null if it is not available
     */
    String topic();

    /**
     * @return  The partition the record was received on;
     *          could be -1 if it is not available
     */
    int partition();

    /**
     * @return  The headers from the record received from Kafka;
     *          could be null if it is not available
     */
    Headers headers();

}
```

```
// new class: o.a.k.streams.processor.TopicNameExtractor

// since this class will be used in both DSL and PAPI, I propose to put in lower-level processor package; this
is similar to ProcessorSupplier.

@InterfaceStability.Evolving
public interface TopicNameExtractor<K, V> {

    /**
     * Extracts the topic name to send to. The topic name must be pre-existed, since the Kafka Streams library
will not
     * try to automatically create the topic with the extracted name, and will fail with a timeout exception if
the topic
     * does not exist in the Kafka cluster.
     *
     * @param key           the record key
     * @param value         the record value payload
     * @param recordContext current context metadata of the record
     * @return the topic name to send to
     */
    String extract(K key, V value, RecordContext recordContext);
}

// Topology.java

Topology addSink(final String name, final TopicNameExtractor<K, V> topicChooser, final String... parentNames)

Topology addSink(final String name, final TopicNameExtractor<K, V> topicChooser, final StreamPartitioner<?
super K, ? super V> partitioner, final String... parentNames)

Topology addSink(final String name, final TopicNameExtractor<K, V> topicChooser, final Serializer<K>
keySerializer, final Serializer<V> valueSerializer, final String... parentNames)

Topology addSink(final String name, final TopicNameExtractor<K, V> topicChooser, final Serializer<K>
keySerializer, final Serializer<V> valueSerializer, final StreamPartitioner<? super K, ? super V> partitioner,
final String... parentNames)


// KStream.java

void to(final TopicNameExtractor<K, V> topicChooser);

void to(final TopicNameExtractor<K, V> topicChooser, final Produced<K, V> produced);


// KStream.scala

def to(extractor: TopicNameExtractor[K, V])(implicit produced: Produced[K, V]): Unit
```

Also I propose to modify StreamPartitioner#partition to include the topic name, since with dynamic routing the topic name would not always be pre-known anymore:

```
// StreamPartitioner.java

public interface StreamPartitioner<K, V> {

    /**
     * Determine the partition number for a record with the given key and value and the current number of
partitions.
     *
     * @param topic the topic name this record is sent to
     * @param key the key of the record
     * @param value the value of the record
     * @param numPartitions the total number of partitions
     * @return an integer between 0 and {@code numPartitions-1}, or {@code null} if the default partitioning
logic should be used
     */
    Integer partition(String topic, K key, V value, int numPartitions);
}
```

Proposed Changes

With the newly added APIs, on the topology's sink node, for each received record we will apply the extractor to get the topic name to send; if the topic that is sending to does not exist, Streams will fall into the normal fail-fast scenario, in which the metadata refresh will exhaust retries and a fatal RequestTimeout will be thrown.

Compatibility, Deprecation, and Migration Plan

- This KIP should not require changes unless the users function calls may result in ambiguities of overloaded functions (think: KStream.to(null)).

Rejected Alternatives

We have thought about also add the functionality to automatically create destination topics if they don't exist while being sent to. The reason to not include this in the KIP is the following:

- There are security/operational risks where a rogue KStreams app might create unneeded/wrong/... topics. (One potential way to address this risk is to introduce a new quota feature for e.g. how many topics could be created by an application in order to control collateral damage of a rogue app); note current ACL may not be sufficient to secure the destination Kafka cluster of the Streams application since it only allows either a single wildcard or a full topic name.
- Even if the topics can be auto-created, downstream applications would still need to become aware of those topics. Because, by definition, (some of) the topics wouldn't exist at the time the downstream applications have been started themselves.

We have also considered adding a `DynamicStreamPartitioner` to allow topic names in addition to `StreamPartitioner` to not break compatibility. I've tried it out the implementation, and realized that DynamicStreamPartitioner needs to implement two interface functions, with and without the topic name if it is extending from StreamPartitioner; we could also let it to not extend from StreamPartition so it has one function only but then we'd need Produced to have two functions allowing StreamPartitioner and DynamicStreamPartitioner. Thinking about the pros and cons I'm think it may be better to just change the interface of StreamPartitioner itself, since even without dynamic routing, allowing the topic name could let users to give one partitioner implementation that branch on the topic names other than having one partitioner per topic.