KIP-372: Naming Repartition Topics for Joins and Grouping

Status

Current state: *Accepted*

Discussion thread: *here*

Voting thread*:* *here*

JIRA: *here*

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

Motivation

To ensure uniqueness when naming operators in the topology, Kafka Streams adds an incrementing number to the operator name. This includes repartition topics which inherit this incrementing numbering scheme.

When a user updates her topology, this can cause the names of the operators to change due to an increase or decrease in the size of the topology, hence altering the names of the repartition topics. This name change of repartition topics is problematic as the user can no longer to a rolling re-deployment. All streams instances must be stopped, new instances installed then restarted.

We'd like to improve on the user experience by allowing users to name some operators explicitly and by extension any repartition topics resulting from the operation. By giving names to repartition topics, any changes to a topology will affect those names and will enable rolling re-deployments which is essential for those users who don't want a production outage for changes in the topology.

We also need to consider how repartition topics come into play with respect to naming so some additional contextual information would be helpful.

Today we have the following scenarios which will introduce a repartition topic:

1. KStream involved in joins (including Stream-Stream and Stream-Table): if the KStream is ever generated from key changing operators, we cannot guarantee if it's still partitioned by the key on its source topic, and hence need to repartition.
2. KStream involved in aggregates, following a groupBy function: as long as it is not groupByKey, we must repartition; and even if it is groupByKey, if the KStream has key changing operations preceded similar to 1), we also need to repartition.
3. KTable involved in aggregations, following a groupBy function: we will always repartition since we do not have KTable#groupByKey at all.

   * Note that since there is no other key-changing operations for KTables except groupBy, KTable joins (Table-Table) never need to require repartitioning as its joining KTables are always safe to assume they are partitioned by key from their source topics (or from the previous repartition topics).

Scope Overlaps

There is an existing KIP-230 the goal of which is to name repartition topics for windowed stream-stream joins. Also KIP-221 aims to provide repartition hints in the Streams DSL, including repartition topic name, number of partitions, etc.

This KIP will subsume KIP-230, while having some overlaps but not completely subsume KIP-221.

Public Interfaces

This KIP will introduce an new class Grouped. There will be changes made to the Joined class as well as the KStream, and KTable interfaces described below.

Proposed Changes

As mentioned above this KIP will add a new class Grouped. Also since Serialized is only used by "groupBy" and "groupByKey", the Serialized class will be deprecated.

Adding Grouped

The new Grouped class can contain a Serde for the key, Serde for the value and a name for the operation.

a. Note that a unique name must be provided for each Grouped instance provided in the topology or a ToplogyException will result from the duplicated names when the topology is built.

Grouped

```
public class Grouped<K, V> {

  public static Grouped as(final String name)


  public static <K> Grouped keySerde(final Serde<K> keySerde)


  public static <V> Grouped valueSerde(final Serde<V> valueSerde)


  public static <K, V> Grouped<K, V> with(final String name,
                                          final Serde<K> keySerde,
                                          final Serde<V> valueSerde)


  public static <K, V> Grouped<K, V> with(final Serde<K> keySerde,
                                          final Serde<V> valueSerde)

  public Grouped withName(final String name)

  public  Grouped<K, V> withKeySerde(final Serde<K> keySerde)

  public  Grouped<K, V> withValueSerde(final Serde<V> valueSerde)
}
```

Serialized Changes

The Serialized class will be deprecated

Serialized Changes

```
@Deprecated
public class Serialized<K, V> {
  // variables and methods left out for clarity
}
```

Joined Changes

Joined will be updated to accept a "name" parameter.

a. The provided name must be unique for each Joined instance provided in the topology or a TopologyException will be thrown when the topology is built.

b. The provided name will not affect the naming of the internal processors.

c. The name used for internal repartition topics will follow the following format "application.id-name-left|right-repartition". The use of "left" or "right" depends on the Stream use in the join. The calling or left hand side is named "left" and the Stream passed into the join method is named "right".

d. If the name is not provided the current naming procedures will be followed.

Joined changes

```
public static <K, V, VO> Joined<K, V, VO> with(final Serde<K> keySerde,
                                               final Serde<V> valueSerde,
                                               final Serde<VO> otherValueSerde,
                                               final String name)


public static <K, V, VO> Joined<K, V, VO> named(final String name)


public Joined<K, V, VO> withName(final String name)


public String name()
```

KStream Changes

There will be new methods added to KStream interfaces accepting a Grouped instance as a parameter.

a. The current methods accepting a Serialized instance will be deprecated.

b. The provided name will not affect the naming of the internal processors.

c. The name is used for internal repartition topics if needed. Named repartitioned topics will follow the following fomat "application.id-name-repartition".

d. If no name is provided then the current naming conventions are used.

e. The provided name must be unique for each Grouped instance provided in the topology or a TopologyException will be thrown when the topology is built.

KStream Added Methods and Deprectations

```
KGroupedStream<K, V> groupByKey(final Grouped<K, V> grouped);


KGroupedStream<KR, V> groupBy(final KeyValueMapper<? super K, ? super V, KR> selector,
                              final Grouped<KR, V> grouped);



@Deprecated
KGroupedStream<K, V> groupByKey(final Serialized<K, V> serialized);


@Deprecated
KGroupedStream<KR, V> groupBy(final KeyValueMapper<? super K, ? super V, KR> selector,
                              final Serialized<KR, V> serialized);
```

KTable Changes

There will be a new method added to the KTable interface accepting a Grouped instance as a parameter.

a. The current methods accepting a Serialized instance will be deprecated.

b. The provided name will not affect the naming of the internal processors.

c. The name is used for internal repartition topics if needed. Named repartitioned topics will follow the following fomat "application.id-name-repartition".

d. If no name is provided then the current naming conventions are used.

e. The provided name must be unique for each Grouped instance provided in the topology or a TopologyException will be thrown when the topology is built.

KTable Added Method and Deprecation

```
KGroupedTable<KR, VR> groupBy(final KeyValueMapper<? super K, ? super V, KeyValue<KR, VR>> selector,
                              final Grouped<KR, VR> grouped);


@Deprecated
KGroupedTable<KR, VR> groupBy(final KeyValueMapper<? super K, ? super V, KeyValue<KR, VR>> selector,
                              final Serialized<KR, VR> serialized);
```

Compatibility, Deprecation, and Migration Plan

- Since these changes (and naming) are optional for users to implement, there is no impact on existing code.
- With an optimization where we reduce the number of repartition topics, if the user has not named the repartition topics, we'll generate a name for the optimized repartition topic. In the case where users have provided names for the repartition topics,
  we'll reuse the name from the first of the named repartition topics. In either case, electing to optimize a topology where the existing number of repartition topics will change, will in all likelihood require an application reset.
- In the case where users have an existing topology and wish to specifically name the repartition topics there exist two options
  - Giving new names to the repartition topics. This approach will require an application reset and a rolling bounce will not be possible. However by naming the repartition topics subsequent changes to the topology should be eligible for rolling bounce re-deployments as the repartition topic names will not change. However, this assumes users have also explicitly named all state stores, thus the changelog topics name will remain stable.
  - Using the existing names of the repartition topics. By using the existing names of the repartition topics, performing a rolling upgrade should be possible. This approach needs to be vetted and will results of this approach will be posted here.
- For the cases where users are updating from an older version to a newer version say 2.1 to 2.2 there are a couple of scenarios to consider
  - You have a 2.1 application where you have written explicit topology names and you want to upgrade to 2.2 and enable optimizations. If we re-use an existing repartition topic name, it should be safe to to a rolling bounce. However, since we don't know which optimizations will be available in 2.2, it's hard to say with complete certainty wether rolling upgrades will be an option or not.

- The deprecated methods will be removed at a later release TBD.

Rejected Alternatives

N/A