

# KIP-759: Unneeded repartition canceling


- [Status](#)
- [Motivation](#)
- [Public Interfaces](#)
- [Proposed Change](#)
- [Usage](#)
- [Concerns](#)
- [Compatibility, Deprecation, and Migration Plan](#)
- [Possible Alternatives](#)


## Status

**Current state:** Accepted, targeting 3.7

**Discussion thread:** [here](#)

**JIRA:**

 Unable to render Jira issues macro, execution error.

 Unable to render Jira issues macro, execution error.

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

## Motivation

Each key changing operation in Kafka Streams (`selectKey`, `map`, `transform`, etc.) now leads to automatic repartition before a stateful aggregation. However, repartition is not always necessary if the input stream is already correctly partitioned. In these cases, the automatic repartition brings in additional overhead. As an example, if an input stream comes in partitioned by `key1`, calling the function `selectKey( ... => (key1, metric)).groupByKey` will trigger a repartition today.

In tickets [KAFKA-4835](#) and [KAFKA-10844](#) the option for canceling the unneeded repartition is requested. Repartition canceling is also needed for the efficient usage of `distinct()` operators proposed in [KIP-655](#): `groupBy(...).windowedBy(...).distinct()` will always repartition by default, while this is not always needed in practice.

This KIP proposes a new interface for users to optimize the key changing operations (`selectKey`, `map`, `transform`, etc.) in Kafka Streams.

## Public Interfaces

In accordance with [KStreams DSL Grammar](#), we introduce a new parameterless DSL operation `markAsPartitioned()` on `KStream`.

```

public interface KStream<K, V> {
    /**
     * Marking the {@code KStream} as partitioned signals the stream is partitioned as intended,
     * and does not require further repartitioning in downstream key changing operations.
     *
     * <p><em>
     *     Note that {@link KStream#markAsPartitioned()} SHOULD NOT be used with interactive query(IQ) or
     *     {@link KStream#join}.
     *     For reasons that when repartitions happen, records are physically shuffled by a composite key
     *     defined in the stateful operation.
     *     However, if the repartitions were cancelled, records stayed in their original partition by its
     *     original key. IQ or joins
     *     assumes and uses the composite key instead of the original key.
     * </p></em>
     *
     * @return a new, mutated {@code KStream} that will not repartition in subsequent operations.
     */
    KStream<K, V> markAsPartitioned();
}

```

## Proposed Change

Calling the new `DSLOperation` will return a new, mutated `KStream`. The new instance will not repartition as downstream operations are chained onto it. Whereas the original stream keeps its own internal property to operate in the default way.

## Usage

Example: canceling repartition in a streams aggregation would look like:

```

stream
    .selectKey( ... => (key1, metric))
    .markAsPartitioned()
    .groupByKey()
    .aggregate()

```

Example: fan out from the same stream:

```

KStream myStream = build.stream(...).map(...);

// the aggregation will not repartition as it works on a mutated KStream
myStream.markAsPartiitoned().groupByKey().aggregate(...);

// the join operation will repartition as it left joins with the original KStream
myStream.join(myOtherTable);

```

## Concerns

- The usage of this operation complicates the usage of IQ(Interactive Query) and joins. For reasons that when repartitions happen, records are physically shuffled by a composite key defined in the stateful operation. However, when the repartitions are canceled, records stayed in their original partition by their original key. IQ assumes and uses the composite key instead of the original key. That's when IQ can break downstream. The same applies to joins.
- In the documentation, we specifically advise against using the interface with IQ or joins.
- However, a potential solution to support IQ is to provide a 'reverse mapping' for the composite key that restores the original key, which can then be used for obtaining the metadata. We can follow up with a change when there is request.

## Compatibility, Deprecation, and Migration Plan

No impact on existing users, no migration is needed.

# Possible Alternatives

## Option 1: Composite Key

If we don't want to introduce an unsafe operation, we might discuss introducing composite keys as an alternative.

- `CompositeKey<H, P>` consists of a head and a postfix, and the partition of a composite key is **always** defined by its 'head' only.
- Also, `k` and `CompositeKey(k, v)` must have the same partition for each `k`.
- We will need to introduce `selectCompositeKey` operations that will not lead to repartition.

`CompositeKey` the design will be safe both from the pov of data locality and IQ but adds complexity to the usage.

## Option 2: Optional configuration in Named Operations( `Joined` , `Grouped` , etc)

- It would allow us to hit only the relevant parts of the DSL and be less prone to undesired behaviors when it comes to IQ or joins.
- More generic, can be applied to `KTable` as well. In comparison, the `markAsPartitioned()` approach is targeting the `KStreams` interface only where it focuses on a specific set of overhead/pain points introduced by `repartitionRequired`.
- It touches on a larger surface area of the API.