

# KIP-663: API to Start and Shut Down Stream Threads




- [Status](#)
- [Motivation](#)
- [Public Interfaces](#)
- [Proposed Changes](#)
- [Examples of Adding a Stream Thread in an Uncaught Exception Handler](#)
- [Compatibility, Deprecation, and Migration Plan](#)
- [Rejected Alternatives](#)

## Status

**Current state:** Accepted

**Discussion thread:** [http://mail-archives.apache.org/mod\\_mbox/kafka-dev/202008.mbox/%3Cef875bee-f153-8d06-e338-658640aa539b%40confluent.io%3E](http://mail-archives.apache.org/mod_mbox/kafka-dev/202008.mbox/%3Cef875bee-f153-8d06-e338-658640aa539b%40confluent.io%3E)

**JIRA:**

-  Unable to render Jira issues macro, execution error.
-  Unable to render Jira issues macro, execution error.
-  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

Currently, there is no possibility in Kafka Streams to increase or decrease the number of stream threads after the Kafka Streams client has been started. There are at least two situations where such functionality would be useful:

1. Reacting on an error that killed a stream thread.
2. Adapt the number of stream threads to the current workload without the need to stop and start the Kafka Streams client.

Uncaught exceptions thrown in a stream thread kill the stream thread leaving the Kafka Streams client with less stream threads for processing than when the client was started. The only way to replace the killed stream thread is to restart the whole Kafka Streams client. For transient errors, it might make sense to replace a killed stream thread with a new one while users try to find the root cause of the error. That could be accomplished by starting a new stream thread in the uncaught exception handler of the killed stream thread.

When the workload of a Kafka Streams client increases, it might be beneficial to scale up the Kafka Streams client by increasing the number of stream threads that the client is currently running. On the other hand, too many stream threads might also negatively impact performance, so that decreasing the number of running stream threads could also be beneficial to a Kafka Streams application. Having the possibility to increase and decrease the number of stream threads without restarting a Kafka Streams client would allow to adapt a client to its environment on the fly.

In this KIP, we propose to extend the API of the Kafka Streams client to start and shutdown stream threads.

# Public Interfaces

```
package org.apache.kafka.streams;

public class KafkaStreams implements AutoCloseable {

    /**
     * Adds and starts a stream thread in addition to the stream threads that are already running in this
     * Kafka Streams client.
     *
     * Since the number of stream threads increases, the sizes of the caches in the new stream thread
     * and the existing stream threads are adapted so that the sum of the cache sizes over all stream
     * threads does not exceed the total cache size specified in configuration
     * {@code cache.max.bytes.buffering}.
     *
     * Stream threads can only be added if this Kafka Streams client is in state RUNNING or REBALANCING.
     *
     * @return name of the added stream thread or empty if a new stream thread could not be added
     */
    public Optional<String> addStreamThread();

    /**
     * Removes one stream thread out of the running stream threads from this Kafka Streams client.
     *
     * The removed stream thread is gracefully shut down. This method does not specify which stream
     * thread is shut down.
     *
     * Since the number of stream threads decreases, the sizes of the caches in the remaining stream
     * threads are adapted so that the sum of the cache sizes over all stream threads equals the total
     * cache size specified in configuration {@code cache.max.bytes.buffering}.
     *
     * @return name of the removed stream thread or empty if a stream thread could not be removed because
     *         no stream threads are alive
     */
    public Optional<String> removeStreamThread();

    /**
     * Removes one stream thread out of the running stream threads from this Kafka Streams client.
     *
     * The removed stream thread is gracefully shut down. This method does not specify which stream
     * thread is shut down.
     *
     * Since the number of stream threads decreases, the sizes of the caches in the remaining stream
     * threads are adapted so that the sum of the cache sizes over all stream threads equals the total
     * cache size specified in configuration {@code cache.max.bytes.buffering}.
     *
     * If the given timeout is exceeded, the method will throw a {@code TimeoutException}.
     *
     * @param timeout
     * @return name of the removed stream thread or empty if a stream thread could not be removed because
     *         no stream threads are alive
     */
    public Optional<String> removeStreamThread(final Duration timeout);
}
```

## Proposed Changes

We propose to add the above methods to the `KafkaStreams` class. The behavior of those methods is described in this section alongside other behavioral changes we propose.

Currently, when a Kafka Streams client is started via `KafkaStreams#start()`, it starts as many stream threads as specified in configuration `num.stream.threads`.

When `KafkaStreams#addStreamThread()` is called, a new full-fledged stream thread will be started in addition to the stream threads started by `KafkaStreams#start()`. The new stream thread will use the same configuration as the existing stream threads. The sum of the cache sizes over the stream thread of a Kafka Streams client specified in configuration `cache.max.bytes.buffering` will be redistributed over the new stream thread and the existing ones, i.e., the cache of each stream thread will be resized after the next rebalance. Starting a new stream thread will trigger a rebalance. Once the new stream thread has been assigned tasks, it will start to execute them as any other pre-existing stream thread. A new stream thread can only be added if the Kafka Streams client is in state `RUNNING` or `REBALANCING`. Method `KafkaStreams#addStreamThread()` will block until a new stream thread could be started and it will return the name of the started stream thread. If no stream thread could be started due to the state of the Kafka Streams client it will return earlier with an empty optional.

The name of the new stream thread will follow the same structure of the names of the existing stream threads, i.e., `[clientId] + "-StreamThread-" + [thread index]`. The thread index is either the next index or the thread index of a stream thread that previously died or that was previously removed with `KafkaStreams#removeStreamThread()` from the client. For example, if a client has three stream threads named `clientA-StreamThread-1`, `clientA-StreamThread-2`, and `clientA-StreamThread-3`, a new stream thread added with `KafkaStreams#addStreamThread()` will be named `clientA-StreamThread-4`. If stream thread `clientA-StreamThread-2` dies or is removed with `KafkaStreams#removeStreamThread()` from the client, the next stream thread that is added with `KafkaStreams#addStreamThread()` will be called `clientA-StreamThread-2`. If a stream thread calls `KafkaStreams#addStreamThread()` in its uncaught exception handler, the new stream thread cannot have the name of the dying stream thread since the dying stream thread has not been dead yet from a JVM point of view when `KafkaStreams#addStreamThread()` is called.

When `KafkaStreams#removeStreamThread()` is called, a running stream thread in the Kafka Streams client is shut down. It is not specified which stream thread is shut down. The chosen stream thread will stop executing tasks and close all its resources. The sum of the cache sizes over the stream thread of a Kafka Streams client specified in configuration `cache.max.bytes.buffering` will be redistributed over the remaining stream threads, i.e., the cache of each remaining stream thread will be resized after the next rebalance. Shutting down a stream thread will trigger a rebalance (also if static membership is configured). If the last running stream thread is shut down with `KafkaStreams#removeStreamThread()`, the Kafka Streams client will stay in state `RUNNING`. If a new stream thread is added via `KafkaStreams#addStreamThread()`, the client will transit to state `REBALANCING` and then `RUNNING` when it will restart processing input records. Method `KafkaStreams#removeStreamThread()` will block until the shut down of the stream thread completed and it will return the name of the shut down stream thread. If no stream thread could be removed because no alive stream threads exist for the Kafka Streams client, it will return earlier with an empty optional. If `KafkaStreams#removeStreamThread(final Duration)` exceeds the given timeout, it will throw a `TimeoutException`.

Stream threads that are in state `DEAD` will be removed from the set of stream threads of a Kafka Streams client to avoid unbounded increase of the number of stream threads kept in a client. Dead stream threads will be removed independently from whether they were started during the start of the Kafka Streams client or through a call to `KafkaStreams#addStreamThread()`. `KafkaStreams#localThreadsMetadata()` will not return metadata of stream threads that are in state `DEAD`. As currently, the Kafka Streams client will transit to `ERROR` if the last alive stream thread dies exceptionally.

To monitor the number of stream threads that died exceptionally, i.e., failed, in the course of time, we propose to add the following client-level metric:

type: stream-metrics  
client-id: [client-id]  
name: failed-stream-threads

Metric `failed-stream-threads` records the total number of stream threads that failed so far for a given Kafka Streams client.

The number of stream threads is not persisted across restarts. That means that a client will always start as many stream threads as specified in configuration `num.stream.threads` during start-up. Even though `KafkaStreams#addStreamThread()` and `KafkaStreams#removeStreamThread()` have been called since the last start of the client.

## Examples of Adding a Stream Thread in an Uncaught Exception Handler

The following example uncaught exception handler starts a stream thread when another stream thread is killed due to a `ProcessorStateException`:

```
kafkaStreams.setUncaughtExceptionHandler((thread, exception) -> {
    if (exception instanceof ProcessorStateException) {
        log.error(String.format("Thread %s died due to the following exception:", thread.getName()),
            exception);
        final Optional<String> nameOfAddedStreamThread = Optional.empty();
        do {
            nameOfAddedStreamThread = kafkaStreams.addStreamThread();
        } while (!nameOfAddedStreamThread.isPresent() && kafkaStreams.isRunningOrRebalancing())
        log.debug("New stream thread named {} was added", nameOfAddedStreamThread.get())
    } else {
        log.error("The following uncaught exception was not handled: ", exception)
    }
});
```

## Compatibility, Deprecation, and Migration Plan

The proposal is backward-compatible because it only adds new methods and does not change any existing methods. The only proposed change that slightly changes the current behavior is to not return metadata of stream threads in state `DEAD` in the result of `KafkaStreams#localThreadsMetadata()`. We regard this change as minor and not relevant to operational continuity.

No methods need to be deprecated and no migration plan is required.

## Rejected Alternatives

- Report stream threads in state `DEAD` in calls to `KafkaStreams#localThreadsMetadata()` until the next call to `KafkaStreams#addStreamThread()` or `KafkaStreams#removeStreamThread()`. This behavior was regarded as too unusual and with little value.