KIP 130: Expose states of active tasks to KafkaStreams public API

Status

Current state: *Accepted*

Discussion thread: *here*

JIRA: *here*

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

Motivation

In Kafka 0.10.1.0, toString() methods have been added to the public API of the KafkaStreams class to print useful information about the representation of the topology DAG.

If this method can be used to debug topologies during development we cannot used it to monitor a KafkaStreams application in an production environment.

Currently there is no way to have the details about the states of either active or standby tasks, neither its partition assignments.

To improve Streams' debuggability we propose to expose the states of tasks throught the public API KafkaStreams.

The most close API to this is StreamsMetadata, however it aggregates the tasks across all threads and only present the aggregated set of the assigned partitions.

This KIP add a new method to allow access to runtime information (i.e threads/tasks details of the local stream instance).

Also, exposing both active and standby tasks is important as this can be used to debug partition assigments when num.standby.replicas != 0.

The task-level information could be polled in a programmatic way for monitoring purposes.

For instance, this will allow applications to expose a REST API to get the global state of a kstreams topology. In addition, this could encourage the community to develop some KafkaStreams UI tooling.

Public Interfaces

This KIP will add the method Set<ThreadMetadata> KafkaStreams#localThreadsMetadata(). This method will return a set ThreadMetadata representing the current threads running into the local stream instance.

Below are the new public classes. Those classes will be declared as inner classes into KafkaStreams :

ThreadState

```java
/**
 * Represents the state of a thread-thread running within a {@link KafkaStreams} application.
 */
public class ThreadMetadata {

    private final String threadName;

    private final String threadState;

    private final Set<TaskMetadata> activeTasks;

    private final Set<TaskMetadata> standbyTasks;

    public ThreadMetadata(String threadName, String threadState, Set<TaskMetadata> activeTasks,
Set<TaskMetadata> standbyTasks) {
        this.threadName = threadName;
        this.threadState = threadState;
        this.activeTasks = activeTasks;
        this.standbyTasks = standbyTasks;
    }

    public String threadState() {
        return threadState;
    }

    public String threadName() {
        return threadName;
    }

    public Set<TaskMetadata> activeTasks() {
        return activeTasks;
    }

    public Set<TaskMetadata> standbyTasks() {
        return standbyTasks;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        ThreadMetadata that = (ThreadMetadata) o;

        if (!threadName.equals(that.threadName)) return false;
        if (!threadState.equals(that.threadState)) return false;
        if (!activeTasks.equals(that.activeTasks)) return false;
        return  standbyTasks.equals(that.standbyTasks);

    }

    @Override
    public int hashCode() {
        int result = threadName.hashCode();
        result = 31 * result + threadState.hashCode();
        result = 31 * result + activeTasks.hashCode();
        result = 31 * result + standbyTasks.hashCode();
        return result;
    }

    @Override
    public String toString() {
        return "ThreadMetadata{" +
                "threadName=" + threadName +
                ", threadState=" + threadState +
                ", activeTasks=" + activeTasks +
                ", standbyTasks=" + standbyTasks +
                '}';
    }
}
```

TaskState

```
/**
 * Represents the state of a single (task) running within a {@link KafkaStreams} application.
 */
public class TaskMetadata {

    private final String taskId;

    private final Set<TopicPartition> assignedPartitions;

    public TaskMetadata(String taskId, Set<TopicPartition> assignedPartitions) {
        this.taskId = taskId;
        this.assignedPartitions = assignedPartitions;
    }

    public String taskId() {
        return taskId;
    }

    public Set<TopicPartition> assignedPartitions() {
        return assignedPartitions;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        TaskMetadata that = (TaskMetadata) o;

        if (!taskId.equals(that.taskId)) return false;
        return  assignedPartitions.equals(that.assignedPartitions);

    }

    @Override
    public int hashCode() {
        int result = taskId.hashCode();
        result = 31 * result + assignedPartitions.hashCode();
        return result;
    }

    @Override
    public String toString() {
        return "TaskMetadata{" +
                "taskId=" + taskId +
                ", assignedPartitions=" + assignedPartitions +
                '}';
    }
}
```

Proposed Changes

This new feature require to add a new method to KafkaStreams to expose thread/tasks details.

```
/**
 * Returns information about the local stream threads running in a {@link KafkaStreams} application.
 *
 * @return the set of {@link ThreadMetadata}.
 */
public Set<ThreadMetadata> localThreadsMetadata() {
    validateIsRunning();
    Set<ThreadMetadata> threadMetadata = new HashSet<>();
    for (int i = 0; i < threads.length; i++)
        threadMetadata.add(threads[i].threadMetadata());
    return threadMetadata;
}
```

In addition, the current toString() method should be deprecated as it would result to return inconsistent information with the new API.

A straightforward first pass is GitHub PR 2612

Compatibility, Deprecation, and Migration Plan

No compatibility issues foreseen.

Rejected Alternatives

1. Add a new method threadStates to public API of StreamsMetadata to expose current states of running threads and tasks. This alternative was rejected because the method would return null for remote application.