KIP-813: Shareable State Stores

This KIP dives deeper into the idea of Shareable State Stores, the ability to use data within a state store across multiple applications without duplicating it on topic level.

Status

Current state: Accepted

Discussion thread: https://lists.apache.org/thread/n0rbdpbwn9p92xd6mn5m73tlmbqp1627

JIRA: 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

The use of state stores is crucial in developing streaming applications. They allow us to have a fault-tolerant way of storing and retrieving state within our processors as well as exposing that data outside of our topology. A compacted topic(the changelog) is being used to make the fault-tolerant characteristic possible. This changelog is considered internal to the statestore and is set as part of the internal logic within a topology. There is no way to pass the compacted topic for a statestore to be used.

With the introduction of tiered storage support, it becomes possible for compacted topics to grow up to the point where it becomes unfeasible to copy the data around. Therefore having a single compacted topic serving multiple state stores starts to make sense.

Public Interfaces

org/apache/kafka/streams/Topology will be extended with an addReadOnlyStateStore method allowing the changelog topic to be passed in.

Proposed Changes

Extend the topology API with the following method:

```
addReadOnlyStateStore(final StoreBuilder<?> storeBuilder,
                      final String sourceName,
                      final TimestampExtractor timestampExtractor,
                      final Deserializer<KIn> keyDeserializer,
                      final Deserializer<VIn> valueDeserializer,
                      final String topic,
                      final String processorName,
                      final ProcessorSupplier<KIn, VIn, Void, Void> stateUpdateSupplier)
```

The method signature is aligned with the one for adding global state stores. Similarly, there is an overloaded method which doesn't take the TimestampExtractor:

```
addReadOnlyStateStore(final StoreBuilder<?> storeBuilder,
                      final String sourceName,
                      final Deserializer<KIn> keyDeserializer,
                      final Deserializer<VIn> valueDeserializer,
                      final String topic,
                      final String processorName,
                      final ProcessorSupplier<KIn, VIn, Void, Void> stateUpdateSupplier)
```

The first method would be a wrapper around the following calls:

- Create a source for the topic and KV deserializers being passed in.
- Create a processor based on the ProcessorSupplier.
- Create a state store based on the passed store builder.

Logging on the passed-in StoreBuilder will be disabled to prevent changes to the compacted topic.

Compatibility, Deprecation, and Migration Plan

The proposal is an addition to existing functionality and should not have any impact to existing users or the way they have been using Kafka Streams.

Rejected Alternatives

Consider two applications; A and B. A writes state s to its state store. This state store is backed by a changelog topic, one that's internal to A. If B would like to lookup s, there are two possible approaches to take.

Copy approach

Do similar processing A is doing to interpret the same events A does and as a result ending up with a copy of the state. This would also require the state store in B to have its own internal changelog topic effectively creating a copy of all state on a new topic.

While this approach can work if there is a limited amount of data on the changelog topic, it becomes problematic when the amount of data grows. With the introduction of tiered storage support, it became more likely for state to grow beyond obvious numbers (into the TB, even possibly PB)