KIP-598: Augment TopologyDescription with store and source / sink serde information

Status

Current state: *"Under Discussion"*

Discussion thread: TBD

JIRA:

key	summary	type	created	updated	due	assignee	reporter	priority	status	resolution

JQL and issue key arguments for this macro require at least one Jira application link to be configured

Released: 2.6 (target)

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

Motivation

Today we have multiple ways to infer and inherit serde along the topology, and only fall back to the configured serde when inference does not apply. More specifically, the serde overriding precedence is the following:

1. Serde specified explicitly via the control objects, such as Consumed, Produced, Grouped, etc.
2. Serde inferred from the parent / child operators which are specified explicitly. For example, for a topology like **builder.stream(.., Consumed.of(..)).groupByKey().reduce()** where the source stream's key-value serdes are specified via `Consumed`, then the reduced table's key and value serdes can inherit from the source stream since their types are not changed.
3. Serde specified via the global config ("default.key.serde" and "default.value.serde").

Since this serde overriding logic is executed implicitly, it is hard for users to infer which serdes are actually going to be used. And if a user mistakenly sets a serde or simply default to the configured serde which mismatches, they will not get informed until they start processing and get a runtime ClassCastException.

So I'd propose we augment the topology description with serde information on place that would execute serde, i.e. source / sink topics and state store operators.

Public Interfaces

The generated String from TopologyDescription#toString would be augmented with the serde information in the form of:

```
keySerde: [SerdeClassType], valueSerde: [SerdeClassType]
```

To illustrate with a concrete example, suppose we have the following application code:

```
    builder.stream("input", Consumed.with(STRING_SERDE, STRING_SERDE))
           .groupBy(..., Grouped.with(STRING_SERDE, STRING_SERDE))
           .windowedBy(...)
           .count(Materialized.as("counts"))
           .suppress(...withName("myname"))
           .toStream()
           .map(...)
           .to("output", Produced.with(STRING_SERDE, Serdes.Long()));
```

The current topology-description would be the following:

```
       Topologies:
          Sub-topology: 0
           Source: KSTREAM-SOURCE-0000000000 (topics: [input])
             --> KSTREAM-KEY-SELECT-0000000001
           Processor: KSTREAM-KEY-SELECT-0000000001 (stores: [])
             --> counts-repartition-filter
             <-- KSTREAM-SOURCE-0000000000
           Processor: counts-repartition-filter (stores: [])
             --> counts-repartition-sink
             <-- KSTREAM-KEY-SELECT-0000000001
           Sink: counts-repartition-sink (topic: counts-repartition)
             <-- counts-repartition-filter

         Sub-topology: 1
           Source: counts-repartition-source (topics: [counts-repartition])
             --> KSTREAM-AGGREGATE-0000000002
           Processor: KSTREAM-AGGREGATE-0000000002 (stores: [counts])
             --> myname
             <-- counts-repartition-source
           Processor: myname (stores: [myname-store])
             --> KTABLE-TOSTREAM-0000000006
             <-- KSTREAM-AGGREGATE-0000000002
           Processor: KTABLE-TOSTREAM-0000000006 (stores: [])
             --> KSTREAM-MAP-0000000007
             <-- myname
           Processor: KSTREAM-MAP-0000000007 (stores: [])
             --> KSTREAM-SINK-0000000008
             <-- KTABLE-TOSTREAM-0000000006
           Sink: KSTREAM-SINK-0000000008 (topic: output-suppressed)
             <-- KSTREAM-MAP-0000000007
```

With this proposal, the augmented topology-description would be the following:

```
        Topologies:
           Sub-topology: 0
            Source: KSTREAM-SOURCE-0000000000 (topics: [input], keySerde: StringDeserializer, valueSerde:
StringDeserializer)
              --> KSTREAM-KEY-SELECT-0000000001
            Processor: KSTREAM-KEY-SELECT-0000000001 (stores: [])
              --> counts-repartition-filter
              <-- KSTREAM-SOURCE-0000000000
            Processor: counts-repartition-filter (stores: [])
              --> counts-repartition-sink
              <-- KSTREAM-KEY-SELECT-0000000001
            Sink: counts-repartition-sink (topic: counts-repartition, keySerde: StringSerializer, valueSerde:
StringSerializer)
              <-- counts-repartition-filter

          Sub-topology: 1
            Source: counts-repartition-source (topics: [counts-repartition], keySerde: StringDeserializer,
valueSerde: StringDeserializer)
              --> KSTREAM-AGGREGATE-0000000002
            Processor: KSTREAM-AGGREGATE-0000000002 (stores: [(counts, serdes: [StringSerde, LongSerde])])
              --> myname
              <-- counts-repartition-source
            Processor: myname (stores: [(myname-store, serdes: [SessionWindowedSerde, FullChangeSerde])])
              --> KTABLE-TOSTREAM-0000000006
              <-- KSTREAM-AGGREGATE-0000000002
            Processor: KTABLE-TOSTREAM-0000000006 (stores: [])
              --> KSTREAM-MAP-0000000007
              <-- myname
            Processor: KSTREAM-MAP-0000000007 (stores: [])
              --> KSTREAM-SINK-0000000008
              <-- KTABLE-TOSTREAM-0000000006
            Sink: KSTREAM-SINK-0000000008 (topic: output-suppressed, keySerde: StringSerializer, valueSerde:
LongSerializer)
              <-- KSTREAM-MAP-0000000007
```

In order to support that, I'd propose the make the following API augments on the TopologyDescription and its corresponding children classes:

```
    interface Processor extends Node {
        /**
         * The names of all connected stores.
         * @return set of store names
         */
        @Deprecated
        Set<String> stores();

        /**
         * The set of all connected stores.
         * @return set of stores
         */
        Set<Store> storeSet();                    <---- NEW FUNC
    }

    /**
     * A state store of a topology                    <---- NEW CLASS
     */
    interface Store {
        /**
         * Name of the stat store
         */
        String name();

        /**
         * Name of the corresponding changelog topic of this store.
         * @return name of the changelog topic; null if the store is not logging enabled
         */
        String changelogTopic();
```

```
    /**
     * Names of serde classes that are associated with the store
     */
    List<String> serdeNames();
}

interface Source extends Node {
    ....

    /**
     * Names of key serde class used for this source node
     */
    String keySerdeName();                    <---- NEW FUNC

    /**
     * Names of value serde class used for this source node
     */
    String valueSerdeName();                    <---- NEW FUNC
}

interface Sink extends Node {
    ....

    /**
     * Names of key serde class used for this source node
     */
    String keySerdeName();                    <---- NEW FUNC

    /**
     * Names of value serde class used for this source node
     */
    String valueSerdeName();                    <---- NEW FUNC
}

interface Subtopology {
    /**
     * Internally assigned unique ID.
     * @return the ID of the sub-topology
     */
    int id();

    /**
     * All nodes of this sub-topology.
     * @return set of all nodes within the sub-topology
     */
    Set<Node> nodes();

    /**
     * All source nodes of this sub-topology.
     * @return set of all source nodes within the sub-topology
     */
    Set<Source> sourceNodes();

    /**
     * All sink nodes of this sub-topology.
     * @return set of all sink nodes within the sub-topology
     */
    Set<Sink> sinkNodes();

    /**
     * All state stores of this sub-topology.
     * @return set of all state stores within the sub-topology
     */
    Set<Store> stores();                    <---- NEW FUNC
}
```

And with the augmented programing interface, we can also allow users to loop over all source / sink nodes and all state stores of a sub-topology so that we can expose all topics (sink, source, intermediate and changelog) as:

```
for (Subtopology subTopology: topology.describe().subtopologies()) {
    for (Source source: subTopology.sourceNodes()) { /* get source and intermediate topics */ }
    for (Sink sink: subTopology.sinkNodes()) { /* get sink and intermediate topics */ }
    for (Store store: subTopology.stores()) { /* get changelog topics */ }
}
```

The reason we did not expose APIs for topic names directly is that for source nodes, it is possible to have Pattern and for sink nodes, it is possible to have topic-extractors, and hence it's better to let users leveraging on the lower-level APIs to construct the topic names programmatically themselves.

Proposed Changes

In order to fall back to global config values, we will need to leverage on the newly added `StreamsBuilder#build(Properties)`; if the old `StreamsBuilder#build()` is called, then serde information would not be exposed via the description (i.e. they will be null) since it is not yet "determined". Also if the TopologyDescription is from the topology built from `StreamsBuilder#build()`, then its `toString` function would not be augmented as well.

Note that the augmented topology description only contains the serde class name, but it does not necessarily include the inner class name.

Compatibility, Deprecation, and Migration Plan

If there are any applications that depends on parsing the string value for, e.g. visualizing the topology description, then their code needs to be updated accordingly. I think this is okay to break such compatibility without introducing a deprecation phase of it since we are leveraging on the newly added `build()` function.

Rejected Alternatives

None.