KIP-300: Add Windowed KTable API in StreamsBuilder

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

We have an existing table() API in the StreamsBuilder which could materialize a Kafka topic into a local state store called KTable. Sometimes we have certain requirement to materialize a windowed topic (or changlog topic) created by another Stream application into local store, too. The current interface could only accept key-value store, which is not ideal. In this KIP, we would like to address this problem by creating new APIs to support the materialization of a windowed KTable, storing as either window store or session store.

The tricky part is that when building this API, in the source processor point of view, the windowed topic input should be (Windowed<K> key, V value). Note that this is different from a normal topic as the serdes required here should be windowed serdes. Let's clear the four different cases involved in the discussion:

1. ***Non-windowed topic materialized to key-value store.*** *This is the most common case and has already been covered by table() API.*
2. ***Non-windowed topic materialized to window store.*** *This is a fallacious requirement because we could easily use aggregate() API to generate a window store based on non-windowed topic.*
3. ***Windowed topic (KStream changelog) materialized to key-value store.*** *This is also a rare requirement to discuss, because the natural difference between key-value store and window store is that window store sets a retention of the data. By materializing windowed topic to key-value we lost the control on the TTL, which leads to wrong representation of the changelog data.*
4. ***Windowed topic (KStream changelog) materialized to window store.*** *This is a missing requirement which needs to be addressed by our new API. Currently it's very hard to share a changelog between stream applications, and it could be really useful to share the same state store across applications by this API.*

Public Interfaces

The current KTable APIs are defined in the StreamsBuilder class:

StreamsBuilder.java

```
public synchronized <K, V> KTable<K, V> table(final String topic, final Consumed<K, V> consumed, final
Materialized<K, V, KeyValueStore<Bytes, byte[]>> materialized)
public synchronized <K, V> KTable<K, V> table(final String topic);
public synchronized <K, V> KTable<K, V> table(final String topic, final Consumed<K, V> consumed);
public synchronized <K, V> KTable<K, V> table(final String topic, final Materialized<K, V, KeyValueStore<Bytes,
byte[]>> materialized);
```

Through `Materialized` struct, we could pass in a KeyValueStore<Bytes, byte[]> struct as the local state store. In fact, underlying KTable class by default stores data in a key-value store backed up by RocksDB. We want to also support window store which is a very natural requirement if we are materializing a windowed topic with windowed key.

Proposed Changes

We would like to add 8 new APIs to support window store and session store as underlying storage option for windowed topic.

StreamsBuilder.java

```
// New APIs: window store materialization to time windowed KTable
public synchronized <K, V> KTable<Windowed<K>, V> windowedTable(final String topic, final Duration windowSize,
final Consumed<K, V> consumed, final Materialized<K, V, WindowStore<Bytes, byte[]>> materialized);
public synchronized <K, V> KTable<Windowed<K>, V> windowedTable(final String topic, final Duration windowSize);
public synchronized <K, V> KTable<Windowed<K>, V> windowedTable(final String topic, final Duration windowSize,
final Consumed<K, V> consumed);
public synchronized <K, V> KTable<Windowed<K>, V> windowedTable(final String topic, final Duration windowSize,
final Materialized<K, V, WindowStore<Bytes, byte[]>> materialized);

// New APIs: session store materialization to session windowed KTable
public synchronized <K, V> KTable<Windowed<K>, V> sessionedTable(final String topic, final Consumed<K, V>
consumed, final Materialized<K, V, SessionStore<Bytes, byte[]>> materialized);
public synchronized <K, V> KTable<Windowed<K>, V> sessionedTable(final String topic);
public synchronized <K, V> KTable<Windowed<K>, V> sessionedTable(final String topic, final Consumed<K, V>
consumed);
public synchronized <K, V> KTable<Windowed<K>, V> sessionedTable(final String topic, final Materialized<K, V,
SessionStore<Bytes, byte[]>> materialized);
```

As the new API suggests, we are tailing from a windowed changelog topic to materialize the data as a KTable of type <Windowed<K>, V> for processing. Internally, the Consumed struct will be converted to <Windowed<K>, V> to correctly deserialize the changelog records. For Materialized the serde type will still be <K, V> because the window store needs raw key serdes and automatically wrapped with windowed key serde (Checkout WindowedKeySchema. toStoreKeyBinary). These details however, are hided from end user. After KIP-393 we have built the constructor which could wrap around a general key serde to make it a window serde, so stream user doesn't need to worry about the type casting, providing raw key serdes should be suffice.

A `windowSize` duration is required to properly initialize the time windowed serde. This is because the underlying storage for windowed records are only storing window start timestamp for space efficiency. When using the new time windowed API, it is required to explicitly pass in positive window size for initialization. User must be aware of the windowed topic window size in order to properly deserialize the topic. For session window serde, `windowSize` config is not needed, because we don't know the individual window size beforehand, so each record will store both start and end time.

Compatibility, Deprecation, and Migration Plan

This KIP will not change the existing table() API, which should be backward compatible.

Rejected Alternatives

We start by changing the store type on the table API to support window store:

StreamsBuilder.java

```
public synchronized <K, V> KTable<K, V> table(final String topic, final Materialized<K, V, WindowStore<Bytes,
byte[]>> materialized);
```

However, this straightfoward solution hits 2 problems:

1. The store type could not be changed due to Java "method has same erasure" error
2. Even if we name the API to windowedTable, it is still not ideal because we saw certain KTable return type in other classes such as in KGroupedStream:

KGroupedStream.java

```
<W extends Window> KTable<Windowed<K>, Long> count(final Windows<W> windows, final String queryableStoreName);
```

So we could see that if we return KTable<K, V> in the above table API for window store, we are introducing inconsistent API to the outside user. By defining the output as KTable<Windowed<K>, V> the user could be clear that we are using window store in the underlying implementation.