KIP-247: Add public test utils for Kafka Streams

Status

Current state: *Accepted* [VOTE] KIP-247: Add public test utils for Kafka Streams

Discussion thread: [DISCUSS] KIP-247: Add public test utils for Kafka Streams

JIRA: Unable to render Jira issues macro, execution error.

Released: target version 1.1.0

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

Motivation

Kafka Streams topologies can be quite complex and it is important for users to have the ability to test their code. This is especially important if the Processor API is used. At the moment, we only have some helper classes in our internal unit test package. Even if this package could be used, there is no guarantee about API stability and users also pull in a large test artifact even if they are only interested in a few test helper classes. Thus, we should provide a public test artifact that supports people to test their code.

Public Interfaces

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

public class TopologyTestDriver {
    public TopologyTestDriver(Topology topology, Properties config); // initialized WallClockMockTimeMs with
System.currentTimeMillis()

    public TopologyTestDriver(Topology topology, Properties config, long initialWallClockTimeMs);

    // event-time is automatically advanced based on the provided input records, and thus event-time
punctuation are triggered automatically
    // wall-clock time is mocked and not advanced automatically; user can advance wall-clock time manually, and
thus, trigger wall-clock time punctuations manually
    public void pipeInput(ConsumerRecord<byte[], byte[]> record); // can trigger event-time punctuation
    public void pipeInput(List<ConsumerRecord<byte[], byte[]>> records); // can trigger event-time punctuation
    public void advanceWallClockTime(long advanceMs); // can trigger wall-clock-time punctuation

    // methods for result verification

    public ProducerRecord<byte[], byte[]> readOutput(String topic);
    public <K, V> ProducerRecord<K, V> readOutput(String topic, Deserializer<K> keyDeserializer,
Deserializer<V> valueDeserializer);

    public Map<String, StateStore> getAllStateStores()
    public StateStore getStateStore(String name);
    public <K, V> KeyValueStore<K, V> getKeyValueStore(String name);
    public <K, V> WindowStore<K, V> getWindowStore(String name);
    public <K, V> SessionStore<K, V> getSessionStore(String name);

    public void close();
}
```

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

public class ConsumerRecordFactory<K, V> {

    // default
    public ConsumerRecordFactory(String defaultTopicName, Serializer<K> keySerializer, Serializer<V>
valueSerializer); // initialized startTimestampMs with System.currentTimeMillis() and autoAdvanceMs with zero
    public ConsumerRecordFactory(String defaultTopicName, Serializer<K> keySerializer, Serializer<V>
valueSerializer, long startTimestampMs);
    public ConsumerRecordFactory(String defaultTopicName, Serializer<K> keySerializer, Serializer<V>
valueSerializer, long startTimestampMs, long autoAdvanceMs);

    // no default topic name; requires to specify topic name in #create(...)
    public ConsumerRecordFactory(Serializer<K> keySerializer, Serializer<V> valueSerializer);
    public ConsumerRecordFactory(Serializer<K> keySerializer, Serializer<V> valueSerializer, long
startTimestampMs);
    public ConsumerRecordFactory(Serializer<K> keySerializer, Serializer<V> valueSerializer, long
startTimestampMs, long autoAdvanceMs);

    public void advanceTimeMs(long advanceMs);

    // create single records with default topic name
    public ConsumerRecord<byte[], byte[]> create(K key, V value, long timestampMs);
    public ConsumerRecord<byte[], byte[]> create(K key, V value);
    public ConsumerRecord<byte[], byte[]> create(V value, long timestampMs);
    public ConsumerRecord<byte[], byte[]> create(V value);


    // create list of records with default topic name
    public List<ConsumerRecord<byte[], byte[]>> create(List<KeyValue<K, V>> keyValues);
    public List<ConsumerRecord<byte[], byte[]>> create(List<KeyValue<K, V>> keyValues, long startTimestamp,
long advanceMs);
    public List<ConsumerRecord<byte[], byte[]>> create(List<KeyValue<K, V>> keyValues, long startTimestamp);


    // overwrite default topic name
    public ConsumerRecord<byte[], byte[]> create(String topicName, K key, V value, long timestampMs);
    public ConsumerRecord<byte[], byte[]> create(String topicName, K key, V value);
    public ConsumerRecord<byte[], byte[]> create(String topicName, V value, long timestampMs);
    public ConsumerRecord<byte[], byte[]> create(String topicName, V value);

    // those methods allow to access regular as well as global stores
    public List<ConsumerRecord<byte[], byte[]>> create(String topicName, List<KeyValue<K, V>> keyValues);
    public List<ConsumerRecord<byte[], byte[]>> create(String topicName, List<KeyValue<K, V>> keyValues, long
startTimestamp, long advanceMs);
    public List<ConsumerRecord<byte[], byte[]>> create(String topicName, List<KeyValue<K, V>> keyValues, long
startTimestamp);

}
```

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

public class OutputVerifier {

    public static <K,V> void compareValue(ProducerRecord<K, V> record, V expectedValue) throws AssertionError;
    public static <K,V> void compareValue(ProducerRecord<K, V> record, ProducerRecord<K, V> expectedRecord)
throws AssertionError;

    public static <K,V> void compareKeyValue(ProducerRecord<K, V> record, K expectedKey, V expectedValue)
throws AssertionError;
    public static <K,V> void compareKeyValue(ProducerRecord<K, V> record, ProducerRecord<K, V> expectedRecord)
throws AssertionError;

    public static <K,V> void compareValueTimestamp(ProducerRecord<K, V> record, V expectedValue, long
expectedTimestamp) throws AssertionError;
    public static <K,V> void compareValueTimestamp(ProducerRecord<K, V> record, ProducerRecord<K, V>
expectedRecord) throws AssertionError;

    public static <K,V> void compareKeyValueTimestamp(ProducerRecord<K, V> record, K expectedKey, V
expectedValue, long expectedTimestamp) throws AssertionError;
    public static <K,V> void compareKeyValueTimestamp(ProducerRecord<K, V> record, ProducerRecord<K, V>
expectedRecord) throws AssertionError;

}
```

Proposed Changes

We are adding the above described test helper classes in a **new artifact** `streams-test-utils` such that people can easily include it as a dependency to their build. The main test class is the test driver while the others a auxiliary classes to generate test data etc.

For time base operations it's important to allow fine grained controll over time. Therefore, we provide a mock time class that can be ingested into the driver and can be used to generte input data. To avoid testing boiler plate code, we provide a rich amount of overload methods and a `ConsumerRecordFactory` that simplifies to generate `ConsumeRecords<byte[],byte[]>` instead of using the verbose `ConsumerRecord` constructor.

In the initla release, we mark all new classes with annotation `@Evolving`.

Compatibility, Deprecation, and Migration Plan

We are only adding new classes. There are no compatiblity issues.

Test Plan

We need to test all added classes with unit tests. Integration or system test are not required.

Rejected Alternatives

None.