Testing Streams Code

Importing the Test Utilities

To test a Kafka Streams application, Apache Kafka® provides a test-utils artifact that can be added as regular dependency to your test code base.

Here is an example `pom.xml` snippet when using Maven:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-streams-test-utils</artifactId>
    <version>2.3.0-ccs</version>
    <scope>test</scope>
</dependency>
```

Testing a Streams Application

When you create a Streams application, you create a `Topology`, either using the StreamsBuilder DSL or using the low-level Processor API. Normally, you run the topology using the `KafkaStreams` class, which connects to your broker and begins processing when you cal `start()`. For testing, though, running a broker and making sure to clean up state between tests adds a lot of complexity and time.

Streams provides the `TopologyTestDriver` in the `kafka-streams-test-utils` package as a drop-in replacement for the `KafkaStreams` class. It has no external system dependencies, and it also processes input synchronously, so you can verify the results immediately after providing input. There are hooks for verifying data sent to output topics, and you can also query state stores maintained by your application under test.

To set it up:

```
// Processor API
Topology topology = new Topology();
topology.addSource("sourceProcessor", "input-topic");
topology.addProcessor("processor", ..., "sourceProcessor");
topology.addSink("sinkProcessor", "output-topic", "processor");
// or
// using DSL
StreamsBuilder builder = new StreamsBuilder();
builder.stream("input-topic").filter(...).to("output-topic");
Topology topology = builder.build();

// setup test driver
Properties config = new Properties();
config.put(StreamsConfig.APPLICATION_ID_CONFIG, "test");
config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "dummy:1234");
TopologyTestDriver testDriver = new TopologyTestDriver(topology, config);
```

The test driver accepts `ConsumerRecords` with key and value type `byte[]`. Because `byte[]` types can be problematic, you can use the `ConsumerRecordFactory` to generate records by providing regular Java types for key and values and the corresponding serializers.

```
ConsumerRecordFactory<String, Integer> factory = new ConsumerRecordFactory<>(
    "input-topic",
    new StringSerializer(),
    new IntegerSerializer()
);
testDriver.pipe(factory.create("key", 42L));
```

For result verification, you can specify corresponding deserializers when reading the output record from the driver so that you don't h with the default types (`byte[]`).

Expand Content

```
ProducerRecord<String, Integer> outputRecord = testDriver.readOutput(
    "output-topic",
    new StringDeserializer(),
    new LongDeserializer()
);
```

`ProducerRecord` contains all the record metadata in addition to the key and value, which can make it awkward to use with test frameworks' equality checks. Instead, consider using the `OutputVerifier` methods to compare just some parts of the records.

```
// throws AssertionError if key or value does not match
OutputVerifier.compareKeyValue(outputRecord, "key", 42L);
```

`TopologyTestDriver` supports punctuations, too. Event-time punctuations are triggered automatically based on the processed records' timestamps. Wall-clock-time punctuations can also be triggered by advancing the test driver's wall-clock time. The driver's wall-clock time must be advanced manually (this is for test stability).

```
testDriver.advanceWallClockTime(20L);
// supposing that a scheduled punctuator would emit this record...
ProducerRecord<String, Integer> outputRecord = testDriver.readOutput(
    "output-topic",
    new StringDeserializer(),
    new StringDeserializer()
);
OutputVerifier.compareKeyValue(outputRecord, "triggered-key", "triggered-value");
```

Additionally, state stores are accessible via the test driver before or after a test. Accessing stores before a test is useful to pre-populate a store with some initial values. After data was processed, expected updates to the store can be verified.

```
KeyValueStore store = testDriver.getKeyValueStore("store-name");
assertEquals("some value", store.get("some key"));
```

Always close the test driver at the end to make sure all resources are released properly.

```
testDriver.close();
```

Example

The following example demonstrates how to use the test driver and helper classes. The example creates a topology that computes the maximum value per key using a key-value-store. While processing, no output is generated, but only the store is updated. Output is only sent downstream based on event-time and wall-clock punctuations.

```
private TopologyTestDriver testDriver;
private KeyValueStore<String, Long> store;

private StringDeserializer stringDeserializer = new StringDeserializer();
private LongDeserializer longDeserializer = new LongDeserializer();
private ConsumerRecordFactory<String, Long> recordFactory = new ConsumerRecordFactory<>(new StringSerializer(), new LongSerializer());

@Before
public void setup() {
    Topology topology = new Topology();
    topology.addSource("sourceProcessor", "input-topic");
    topology.addProcessor("aggregator", new CustomMaxAggregatorSupplier(), "sourceProcessor");
    topology.addStateStore(
        Stores.keyValueStoreBuilder(
            Stores.inMemoryKeyValueStore("aggStore"),
            Serdes.String(),
            Serdes.Long()).withLoggingDisabled(), // need to disable logging to allow store pre-populating
        "aggregator");
    topology.addSink("sinkProcessor", "result-topic", "aggregator");

    // setup test driver
    Properties config = new Properties();
    config.setProperty(StreamsConfig.APPLICATION_ID_CONFIG, "maxAggregation");
    config.setProperty(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "dummy:1234");
    config.setProperty(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
    config.setProperty(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.Long().getClass().getName());
    testDriver = new TopologyTestDriver(topology, config);

    // pre-populate store
    store = testDriver.getKeyValueStore("aggStore");
    store.put("a", 21L);
}

@After
public void tearDown() {
    testDriver.close();
}

@Test
public void shouldFlushStoreForFirstInput() {
    testDriver.pipeInput(recordFactory.create("input-topic", "a", 1L, 9999L));
    OutputVerifier.compareKeyValue(testDriver.readOutput("result-topic", stringDeserializer, longDeserializer), "a", 2
1L);
    Assert.assertNull(testDriver.readOutput("result-topic", stringDeserializer, longDeserializer));
}

@Test
public void shouldNotUpdateStoreForSmallerValue() {
    testDriver.pipeInput(recordFactory.create("input-topic", "a", 1L, 9999L));
    Assert.assertThat(store.get("a"), equalTo(21L));
    OutputVerifier.compareKeyValue(testDriver.readOutput("result-topic", stringDeserializer, longDeserializer), "a", 2
1L);
    Assert.assertNull(testDriver.readOutput("result-topic", stringDeserializer, longDeserializer));
}

@Test
public void shouldUpdateStoreForLargerValue() {
    testDriver.pipeInput(recordFactory.create("input-topic", "a", 42L, 9999L));
    Assert.assertThat(store.get("a"), equalTo(42L));
    OutputVerifier.compareKeyValue(testDriver.readOutput("result-topic", stringDeserializer, longDeserializer), "a", 4
2L);
    Assert.assertNull(testDriver.readOutput("result-topic", stringDeserializer, longDeserializer));
}

@Test
public void shouldUpdateStoreForNewKey() {
    testDriver.pipeInput(recordFactory.create("input-topic", "b", 21L, 9999L));
    Assert.assertThat(store.get("b"), equalTo(21L));
    OutputVerifier.compareKeyValue(testDriver.readOutput("result-topic", stringDeserializer, longDeserializer), "a", 2
1L);
    OutputVerifier.compareKeyValue(testDriver.readOutput("result-topic", stringDeserializer, longDeserializer), "b", 2
1L);
    Assert.assertNull(testDriver.readOutput("result-topic", stringDeserializer, longDeserializer));
}

@Test
public void shouldPunctuateIfEvenTimeAdvances() {
    testDriver.pipeInput(recordFactory.create("input-topic", "a", 1L, 9999L));
    OutputVerifier.compareKeyValue(testDriver.readOutput("result-topic", stringDeserializer, longDeserializer), "a", 2
1L);

    testDriver.pipeInput(recordFactory.create("input-topic", "a", 1L, 9999L));
    Assert.assertNull(testDriver.readOutput("result-topic", stringDeserializer, longDeserializer));

    testDriver.pipeInput(recordFactory.create("input-topic", "a", 1L, 10000L));
    OutputVerifier.compareKeyValue(testDriver.readOutput("result-topic", stringDeserializer, longDeserializer), "a", 2
1L);
    Assert.assertNull(testDriver.readOutput("result-topic", stringDeserializer, longDeserializer));
}

@Test
public void shouldPunctuateIfWallClockTimeAdvances() {
    testDriver.advanceWallClockTime(60000);
    OutputVerifier.compareKeyValue(testDriver.readOutput("result-topic", stringDeserializer, longDeserializer), "a", 2
1L);
    Assert.assertNull(testDriver.readOutput("result-topic", stringDeserializer, longDeserializer));
}

public class CustomMaxAggregatorSupplier implements ProcessorSupplier<String, Long> {
    @Override
    public Processor<String, Long> get() {
        return new CustomMaxAggregator();
    }
}

public class CustomMaxAggregator implements Processor<String, Long> {
    ProcessorContext context;
    private KeyValueStore<String, Long> store;

    @SuppressWarnings("unchecked")
    @Override
    public void init(ProcessorContext context) {
        this.context = context;
        context.schedule(60000, PunctuationType.WALL_CLOCK_TIME, new Punctuator() {
            @Override
            public void punctuate(long timestamp) {
                flushStore();
            }
        });
        context.schedule(10000, PunctuationType.STREAM_TIME, new Punctuator() {
```

```
        context.schedule(10000, PunctuationType.STREAM_TIME, new Punctuator() {
            @Override
            public void punctuate(long timestamp) {
                flushStore();
            }
        });
        store = (KeyValueStore<String, Long>) context.getStateStore("aggStore");
    }

    @Override
    public void process(String key, Long value) {
        Long oldValue = store.get(key);
        if (oldValue == null || value > oldValue) {
            store.put(key, value);
        }
    }

    private void flushStore() {
        KeyValueIterator<String, Long> it = store.all();
        while (it.hasNext()) {
            KeyValue<String, Long> next = it.next();
            context.forward(next.key, next.value);
        }
    }

    @Override
    public void close() {}
}
```

Unit Testing for Processors

Using the Processor API, you can define custom `Processor`, `Transformer`, or `ValueTransformer` implementations.

Because these classes forward their results to the `ProcessorContext` rather than returning them, unit testing requires a mocked context capable of capturing forwarded data for inspection.

Streams provides `MockProcessorContext` in `kafka-streams-test-utils` for this purpose.

To begin with, instantiate your processor and initialize it with the mock context:

```
final Processor processorUnderTest = ...;
final MockProcessorContext context = new MockProcessorContext();
processorUnderTest.init(context);
```

If you need to pass configuration to your processor or set the default serdes, you can create the mock with config:

```
final Properties config = new Properties();
config.put(StreamsConfig.APPLICATION_ID_CONFIG, "unit-test");
config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "");
config.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
config.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.Long().getClass());
config.put("some.other.config", "some config value");
final MockProcessorContext context = new MockProcessorContext(config);
```

The mock captures any values that your processor forwards. You can make assertions on them:

```
processorUnderTest.process("key", "value");

final Iterator<CapturedForward> forwarded = context.forwarded().iterator();
assertEquals(forwarded.next().keyValue(), new KeyValue<>(..., ...));
assertFalse(forwarded.hasNext());

// you can reset forwards to clear the captured data. This may be helpful in constructing longer scenarios.
context.resetForwards();

assertEquals(context.forwarded().size(), 0);
```

If your processor forwards to specific child processors, you can query the context for captured data by child name:

```java
final List<CapturedForward> captures = context.forwarded("childProcessorName");
```

The mock also captures whether your processor has called `commit()` on the context:

```java
assertTrue(context.committed());

// commit captures can also be reset.
context.resetCommit();

assertFalse(context.committed());
```

In case your processor logic depends on the record metadata (topic, partition, offset, or timestamp), you can set them on the context, either all together or individually:

```java
context.setRecordMetadata("topicName", /*partition*/ 0, /*offset*/ 0L, /*timestamp*/ 0L);
context.setTopic("topicName");
context.setPartition(0);
context.setOffset(0L);
context.setTimestamp(0L);
```

Once these are set, the context continues returning the same values, until you set new ones.

In case your punctuator is stateful, the mock context allows you to register state stores. You are encouraged to use a simple in-memory store of the appropriate type (`KeyValue`, `Windowed`, or `Session`), since the mock context does *not* manage changelogs, state directories, etc.

```java
final KeyValueStore<String, Integer> store =
    Stores.keyValueStoreBuilder(
            Stores.inMemoryKeyValueStore("myStore"),
            Serdes.String(),
            Serdes.Integer()
        )
        .withLoggingDisabled() // Changelog is not supported by MockProcessorContext.
        .build();
store.init(context, store);
context.register(store, /*parameter unused in mock*/ null);
```

Processors can schedule punctuators to handle periodic tasks. The mock context does *not* automatically execute punctuators, but it does capture schedule calls so that you can unit test the punctuator scheduling behavior yourself:

```java
final MockProcessorContext.CapturedPunctuator capturedPunctuator = context.scheduledPunctuators().get(0);
final long interval = capturedPunctuator.getIntervalMs();
final PunctuationType type = capturedPunctuator.getType();
final boolean cancelled = capturedPunctuator.cancelled();
final Punctuator punctuator = capturedPunctuator.getPunctuator();
punctuator.punctuate(/*timestamp*/ 0L);
```

If you need to write tests involving automatic firing of scheduled punctuators, you should use the `TopologyTestDriver` on a simple topology containing your processor.

Please report any inaccuracies on this page or suggest an edit.

3 Votes

★★★★½

Last updated on Sep 10, 2019.