Kafka Streams provides a rich feature set for transforming your data. In this lesson, we will focus on stateless transformations. We will discuss the difference between stateful and stateless transformations, and we will demonstrate how to use several of the stateless transformations that are available as part of the Streams API.

Relevant Documentation

- Kafka Streams Developer Guide — Stateless Transformations

Lesson Reference

1. Clone the starter project, if you haven't already done so in a previous lesson.

```
cd ~/
git clone https://github.com/linuxacademy/content-ccdak-kafka-streams.git
cd content-ccdak-kafka-streams
```

2. Edit the `StatelessTransformationsMain` class.

```
vi src/main/java/com/linuxacademy/ccdak/streams/StatelessTransformationsMain.java
```

3. Implement a Streams application that performs a variety of stateless transformations.

```
package com.linuxacademy.ccdak.streams;

import java.util.LinkedList;
import java.util.List;
import java.util.Properties;
import java.util.concurrent.CountDownLatch;
import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.KeyValue;
import org.apache.kafka.streams.StreamsBuilder;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.Topology;
import org.apache.kafka.streams.kstream.KStream;

/*import java.util.Properties;
import java.util.concurrent.CountDownLatch;*/

public class StatelessTransformationsMain {<pre><code>public static void main(String[] args) {
    // Set up the configuration.
    final Properties props = new Properties();
    props.put(StreamsConfig.APPLICATION_ID_CONFIG, "stateless-transformations-example");
    props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
    props.put(StreamsConfig.CACHE_MAX_BYTES_BUFFERING_CONFIG, 0);
    // Since the input topic uses Strings for both key and value, set the default Serdes to String.
    props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
    props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());

    // Get the source stream.
    final StreamsBuilder builder = new StreamsBuilder();
    final KStream&lt;String, String&gt; source = builder.stream("stateless-transformations-input-topic");

    // Split the stream into two streams, one containing all records where the key begins with "a", and t
    KStream&lt;String, String&gt;[] branches = source
        .branch((key, value) -&gt; key.startsWith("a"), (key, value) -&gt; true);
    KStream&lt;String, String&gt; aKeysStream = branches[0];
    KStream&lt;String, String&gt; othersStream = branches[1];

    // Remove any records from the "a" stream where the value does not also start with "a".
    aKeysStream = aKeysStream.filter((key, value) -&gt; value.startsWith("a"));
```

```
    // For the "a" stream, convert each record into two records, one with an uppercased value and one wit
    aKeysStream = aKeysStream.flatMap((key, value) -&gt; {
        List&lt;KeyValue&lt;String, String&gt;&gt; result = new LinkedList&lt;&gt;();
        result.add(KeyValue.pair(key, value.toUpperCase()));
        result.add(KeyValue.pair(key, value.toLowerCase()));
        return result;
    });

    // For the "a" stream, modify all records by uppercasing the key.
    aKeysStream = aKeysStream.map((key, value) -&gt; KeyValue.pair(key.toUpperCase(), value));

    //Merge the two streams back together.
    KStream&lt;String, String&gt; mergedStream = aKeysStream.merge(othersStream);

    //Print each record to the console.
    mergedStream.peek((key, value) -&gt; System.out.println("key=" + key + ", value=" + value));

    //Output the transformed data to a topic.
    mergedStream.to("stateless-transformations-output-topic");

    final Topology topology = builder.build();
    final KafkaStreams streams = new KafkaStreams(topology, props);
    // Print the topology to the console.
    System.out.println(topology.describe());
    final CountDownLatch latch = new CountDownLatch(1);

    // Attach a shutdown handler to catch control-c and terminate the application gracefully.
    Runtime.getRuntime().addShutdownHook(new Thread("streams-wordcount-shutdown-hook") {
        @Override
        public void run() {
            streams.close();
            latch.countDown();
        }
    });

    try {
        streams.start();
        latch.await();
    } catch (final Throwable e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
    System.exit(0);
}</code></pre>}
```

4. In a separate session, starts a `kafka-console-producer` to produce data to the input topic.

```
kafka-console-producer --broker-list localhost:9092 --topic stateless-transformations-input-topic --prope
```

5. Publish an initial record to automatically create the topic.

```
a:a
```

6. In the previous session, run your code.

```
./gradlew runStatelessTransformations
```

7. In another session, start a `kafka-console-consumer` to view records being published to the output topic, then publish some records and examine how your Streams application modifies them.

```
kafka-console-consumer --bootstrap-server localhost:9092 --topic stateless-transformations-output-topic -
```