There is a lot you can do with aggregations and joins, but Kafka also provides the ability to perform aggregations and joins within the context of specific time buckets known as windows. In this lesson, we will discuss what windowing is, and we will demonstrate the use of windowing in the context of a basic aggregation.

Relevant Documentation

- Kafka Streams Developer Guide — Windowing

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 `WindowingMain` class.

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

3. Implement a Streams application that demonstrates windowing.

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

import java.time.Duration;
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.StreamsBuilder;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.Topology;
import org.apache.kafka.streams.kstream.KGroupedStream;
import org.apache.kafka.streams.kstream.KStream;
import org.apache.kafka.streams.kstream.KTable;
import org.apache.kafka.streams.kstream.Produced;
import org.apache.kafka.streams.kstream.TimeWindowedKStream;
import org.apache.kafka.streams.kstream.TimeWindows;
import org.apache.kafka.streams.kstream.Windowed;
import org.apache.kafka.streams.kstream.WindowedSerdes;

public class WindowingMain {<pre><code>public static void main(String[] args) {
    // Set up the configuration.
    final Properties props = new Properties();
    props.put(StreamsConfig.APPLICATION_ID_CONFIG, "windowing-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();
    KStream&lt;String, String&gt; source = builder.stream("windowing-input-topic");

    KGroupedStream&lt;String, String&gt; groupedStream = source.groupByKey();

    // Apply windowing to the stream with tumbling time windows of 10 seconds.
    TimeWindowedKStream&lt;String, String&gt; windowedStream = groupedStream.windowedBy(TimeWindows.of(Du

    // Combine the values of all records with the same key into a string separated by spaces, using 10-se
    KTable&lt;Windowed&lt;String&gt;, String&gt; reducedTable = windowedStream.reduce((aggValue, newValue
```

```
    reducedTable.toStream().to("windowing-output-topic", Produced.with(WindowedSerdes.timeWindowedSerdeFr

    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-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. Open a separate session. Start a `kafka-console-producer` to produce data to the input topic.

```
kafka-console-producer --broker-list localhost:9092 --topic windowing-input-topic --property parse.key=tr
```

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

```
a:a
```

6. In the previous session, run your code.

```
./gradlew runWindowing
```

7. Open an additional session. Start a `kafka-console-consumer` to view records being published to the output topic, then publish some records to the input topic and examine how your Streams application modifies them.

```
kafka-console-consumer --bootstrap-server localhost:9092 --topic windowing-output-topic --property print.
```