SE424: Distributed Systems	**Recitation 3d**
Semester 2 5786	**22 March 2026**
Lecturer: Michael J. May	**Kinneret College**

Kafka Streams

In this recitation, we'll use Kafka Streams, a streaming library that enhances Kafka with support for unending and seemingly infinite sources of data. Streams interfaces are commonly used for consuming sensor data, real world measurements, or web data that never stops.

The steps below are based on the Kafka Streams QuickStart. You can find it online at: `https://kafka.apache.org/42/streams/quickstart/`.

1 Word Count Demo

We're going to use a Kafka demo class that comes with the standard Kafka distribution. It's called WordCountDemo and can be found at the following link: `https://github.com/apache/kafka/blob/4.2/streams/examples/src/main/java/org/apache/kafka/streams/examples/wordcount/WordCountDemo.java`

The gist of the code is as follows:

```
public static final String INPUT_TOPIC = "streams-plaintext-input";
public static final String OUTPUT_TOPIC = "streams-wordcount-output";

// Serializers/deserializers (serde) for String and Long types
final Serde<String> stringSerde = Serdes.String();
final Serde<Long> longSerde = Serdes.Long();

// Construct a 'KStream' from the input topic "streams-plaintext-input", where message
// values represent lines of text (for the sake of this example, we ignore whatever may be
// stored in the message keys).
KStream<String, String> textLines = builder.stream(
      "streams-plaintext-input",
      Consumed.with(stringSerde, stringSerde)
    );

KTable<String, Long> wordCounts = textLines
    // Split each text line, by whitespace, into words.
    .flatMapValues(value -> Arrays.asList(value.toLowerCase().split("\W+")))

    // Group the text words as message keys
    .groupBy((key, value) -> value)

    // Count the occurrences of each word (message key).
    .count();

// Store the running counts as a changelog stream to the output topic.
wordCounts.toStream().to("streams-wordcount-output",
    Produced.with(Serdes.String(), Serdes.Long()));
```

Listing 1: WordCountDemo gist

We're going to use to the demo code in a Kafka environment and setup a producer and consumer around it. Word count is a very popular "Hello world" application, so you'll see it in many sample streaming and distributed algorithms processing examples. The main novelty here is that it uses Kafka's backend for processing which is highly fault tolerant and distributable. Streams are also built to handle unbounded data, so it outputs results every so often rather than waiting for the end.

The following steps will build a setup as shown in Figure 1. We'll write the pieces one step at a time.

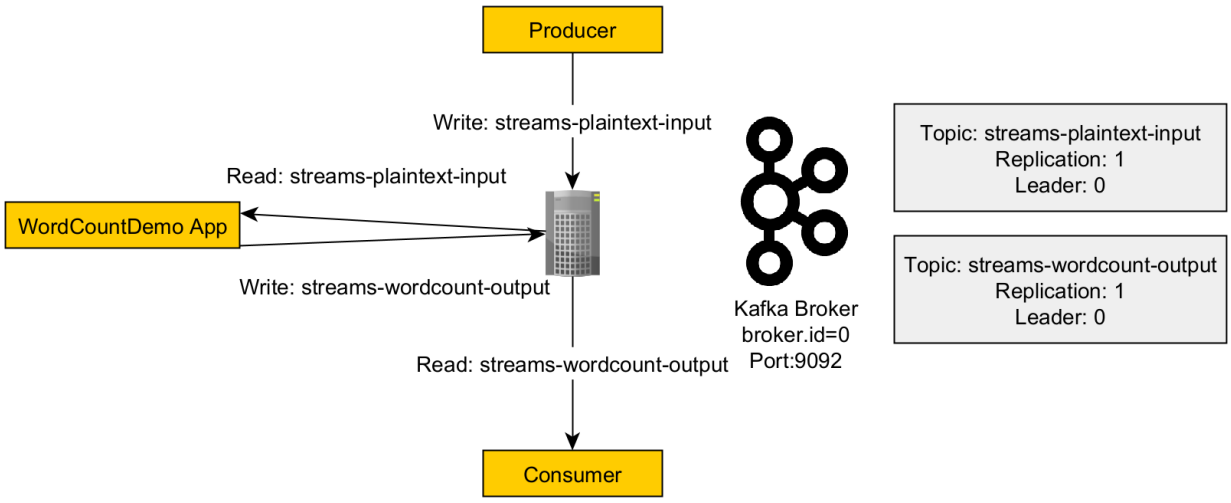

Figure 1: Kafka Streams WordCount topology

1.1 Installing Kafka

The first thing to do is download Kafka if you haven't already. You can download it from: `https://www.apache.org/dyn/closer.lua/kafka/4.2.0/kafka_2.13-4.2.0.tgz?action=download`

After you downloaded, it unzip the archive. You must run Kafka under Linux, so make sure you're in a Linux installation or VM.

Note: If you downloaded the unzipped file via Windows and then copied the directory to Linux (WSL) you'll likely need to run the following command from within the downloaded directory to ensure that all shell scripts for Kafka are runnable. The command must be run from within the kafka directory (that's the part before the $ in the line below). The actual command starts from the word "chmod":

```
~/kafka_2.13-4.2.0/bin$ chmod 700 *.sh
```

2 Starting the Kafka Broker

2.1 Cluster ID

Once you have Kafka unzipped and ready, the next step is to choose a cluster ID for the Kafka Broker. The cluster ID is used to organize the brokers into an identifiable group and is used for their coordination. We'll use the Kafka storage script and a random number generator to create a new cluster ID. Run the following command from within the Kafka directory:

```
KAFKA_CLUSTER_ID="$(bin/kafka-storage.sh random-uuid)"
```

2.2 Log Directories

Then we need to set up the log directories for the brokers. Kafka's logs are essential to its behavior since brokers use them to keep the message history. We'll use the Kafka storage script again to set up the log storage directory:

```
bin/kafka-storage.sh format --standalone -t $KAFKA_CLUSTER_ID -c config/server.properties
```

If you open up the configuration file config/server.properties, you can see where the Kafka logs will be stored:

```
############################# Log Basics #############################

# A comma separated list of directories under which to store log files
log.dirs=/tmp/kraft-combined-logs
```

2.3 Starting the Broker

Now let's use the following command to start the Kafka broker:

```
bin/kafka-server-start.sh config/server.properties
```

The default server properties now uses the KRaft consensus protocol, so we don't need to set up a Zookeeper instance.

3 Preparing Kafka Topics

The next step is to create the topics that will be used for inputting words into the WordCount program and receiving the output as shown in Figure 1.

Note: Since you started a Kafka broker in the previous section on the command line, you'll need to open a **new command line interface** to continue.

3.1 Input Topic

We'll call the input topic `streams-plaintext-input`. The name must be precise since it's hard coded into the WordCountDemo class above. We use the following script to open a new topic with that name:

```
bin/kafka-topics.sh --create --bootstrap-server localhost:9092
    --replication-factor 1   --partitions 1
    --topic streams-plaintext-input
```

Make sure the command all appears on a single line - the line breaks above are just to make it easier to read. The tool should output the following response:

```
Created topic "streams-plaintext-input".
```

3.2 Output Topic

Now let's create the output topic `streams-wordcount-output` using a similar command (again all on a single line):

```
bin/kafka-topics.sh --create
    --bootstrap-server localhost:9092
    --replication-factor 1
    --partitions 1
    --topic streams-wordcount-output
    --config cleanup.policy=compact
```

The tool should output a success response:

```
Created topic "streams-wordcount-output".
```

3.3 Review the Topics

We can check that the topics were created successfully using the **describe** command from the Kafka topics script. The command queries the Kafka broker that listens on localhost with port 9092 (the one we started before).

```
bin/kafka-topics.sh --bootstrap-server localhost:9092 --describe --exclude-internal
```

The output from the command should be something like this:

```
Topic:streams-wordcount-output  PartitionCount:1        ReplicationFactor:1     Configs:
    cleanup.policy=compact,segment.bytes=1073741824
  Topic: streams-wordcount-output  Partition: 0  Leader: 0  Replicas: 0  Isr: 0
Topic:streams-plaintext-input   PartitionCount:1        ReplicationFactor:1     Configs:
    segment.bytes=1073741824
  Topic: streams-plaintext-input  Partition: 0  Leader: 0  Replicas: 0  Isr: 0
```

You can see from the description output that:

PartitionCount There is only one partition per topic, meaning that all data is kept in a single server.

Configs There are some configuration values per topic such as the number of bytes in a segment (1GB) and the use of the compact cleanup policy for old log information (as opposed to delete).

Partition, Leader, Replicas, Isr All values are 0 since there is only one broker here.

4 Staring the Wordcount Application

Now we'll start the Wordcount application using the Kafka run class script:

```
bin/kafka-run-class.sh org.apache.kafka.streams.examples.wordcount.WordCountDemo
```

As shown in Figure 1, the application reads from the **streams-plaintext-input** topic and outputs to the **streams-wordcount-output** topic after performing the word counting. The application writes its output to Kafka streams, so there isn't any STDOUT output to see.

5 Starting the Producer and Consumer

Now let's start the **producer** that will send data to the WordCount via Kafka and the **consumer** that will read the output from WorCount via Kafka. We'll use the provided console producer and console consumer for simplicity. In a real world situation, you'd build applications that send and receive from Kafka streams.

5.1 Starting the Producer

Open a new console and enter the following command to start the console producer:

```
bin/kafka-console-producer.sh
    --bootstrap-server localhost:9092
    --topic streams-plaintext-input
```

5.2 Starting the Consumer

Open a new console and enter the following command to start the console consumer:

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092
    --topic streams-wordcount-output
    --from-beginning
    --formatter-property print.key=true
    --formatter-property print.value=true
    --formatter-property key.deserializer=org.apache.kafka.common.serialization.
        StringDeserializer
    --formatter-property value.deserializer=org.apache.kafka.common.serialization.
        LongDeserializer
```

The output on the topic consists of **key-value pairs** where the key is the word and the value is the count for each word so far.

The consumer receives many parameters. Here's what they mean:

from beginning Ensures that the consumer gets all messages from the topic, even ones that were sent before it turned on

print.key true The consumer will print the keys (the words)

print.value true The consumer will print the values (the counts)

key deserializer The keys will be deserialized as Strings

value deserializer The values will be deserialized as Long integers

6 Test Data

Now let's put some test data into the topic to watch how it works. We'll put some words into the console producer and watch the output from the console consumer.

6.1 Console Producer Input

Run the console producer command if you haven't already. If you left the console producer open from before, you can use it.

```
bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic streams-plaintext-
    input
```

Enter in some text. Some recommended text follows. Press enter after each line.

```
I think that I shall never see

A graph more lovely than a tree.
```

6.2 Console Consumer Output

Run the console consumer command if you haven't already. If you left the console consumer open from before, you don't need to do anything.

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092
    --topic streams-wordcount-output
    --from-beginning
    --formatter-property print.key=true
    --formatter-property print.value=true
    --formatter-property key.deserializer=org.apache.kafka.common.serialization.
        StringDeserializer
    --formatter-property value.deserializer=org.apache.kafka.common.serialization.
        LongDeserializer
```

You should see something like the following:

```
i       1
think   1
that    1
i       2
shall   1
never   1
see     1
        1
a       1
graph   1
more    1
lovely  1
than    1
a       2
tree    1
```

Notice how the count updates as we go along - "i" goes $1 \rightarrow 2$ and "a" goes $1 \rightarrow 2$. Due to the streaming nature of the input, the algorithm produces output every so often as values change.

6.3 Providing a File

We can also provide an entire file to the word count application using the console producer. Use the following command with the sample file provided on Moodle.

```
bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic streams-plaintext-
    input < frankenstein.txt
```

Notice the word count results at the consumer after the large file has been fed in.