# Day 3 – Lab1:

# Word Count with Kafka Streaming

In this lab, you'll use Kafka Streaming to count words in a stream of lines.

## The Kafka streaming app

### Examine the code

Using your favorite editor, open the file `wordcount-kafka-streaming/src/main/java/app/StreamExample.java`.

First, we get our Kafka properties, then instantiate a `StreamBuilder` to get a reference to a `KStream` via the builder's `stream` method, passing the topic name from which we'll consume messages. Then, we establish our stream's topology.

- `flatMapValues` takes the incoming stream of lines and returns a stream of words after splitting them on word boundaries.
- `filter` keeps only those items in the stream that are not blank, in case any of those managed to slip into the word stream.
- `map` transforms each word into a `KeyValue` instance using the word as the key *and* the value.
- `groupByKey` groups the `KeyValue` instances by word (the key).
- `count` creates a `KTable` containing the counts of each word.
- `toStream` converts the `KTable` to a `KStream` so that we can then produce values to the output topic.
- `map` converts each value in the stream to a `String` formatted with the word and its count separated by a colon.
- `to` streams the formatted word counts to the output topic.

NOTE: this topology isn't executed until the stream is started and lines are presented to the stream.

Now that we've created our stream processing topology, we instantiate a `KafkaStreams` instance, giving it our `Topology` and hook it up to Kafka via our Kafka client properties, then `start` the stream.

Lastly, for convenience, we wait for the user to hit enter to `close` the stream and exit.

### Build the streaming app

Now that we have coded our app, we need to build it. Fortunately, we can use `docker` for this so that we don't have to have `maven` and its prerequisites installed locally. Open a terminal in the lab's `wordcount-spark` directory and issue the following command:

```
$ docker run -it --rm -v "$(cd "$PWD/../.."; pwd)":/course-root -w "/course-
root/$(basename $(cd "$PWD/.."; pwd))/$(basename "$PWD")" -v
"$HOME/.m2/repository":/root/.m2/repository maven:3-jdk-11 ./mvnw clean package
```

On a windows machine, you have to replace the `$PWD` with the current directory and the `$HOME` with a directory where you have the `.m2` folder.
The command above will build and package our uber jar with the application and all of its dependencies.

## Run Kafka

In order to run our app, we first need to run Kafka. First, ensure that you've shut down any prior docker containers.

Next, open a new terminal in the lab's root directory & run the Docker Compose stack using the `kafka-streaming.yaml` configuration file:
`$ docker-compose -f kafka-streaming.yaml up`

You will see logs from all the containers that are launched as part of the solution. Once the terminal stops reflecting new output, the infrastructure is initialized.

## Start our streaming application

Now, let's start our streaming application connecting to Kafka running in our Docker environment:

```
$ docker run --network "$(cd .. && basename "$(pwd)" | tr '[:upper:]'
'[:lower:]')_default" --rm -it -v "$PWD:/pwd" -w /pwd openjdk:11 java -jar
target/wordcount-kafka-solution-*.jar
```

On a windows machine, you must replace the `$PWD` with the current directory and the `$HOME` with a directory where you have the `.m2` folder.
Now that our Kafka streaming application is running, it's time to feed it some input lines via the Kafka topic.

In yet another terminal, change into the lab's root directory again and this time, start a bash session in the *kafka* container:

```
$ docker-compose -f kafka-streaming.yaml exec kafka bash
```

At the subsequent prompt, create the topics & start a Kafka console consumer:

```
I have no name!@c07eea6aed61:/$ for it in input output; do kafka-topics.sh --create --bootstrap-server kafka:9092 --topic stream-$it; done
I have no name!@c07eea6aed61:/$ kafka-console-consumer.sh --bootstrap-server kafka:9092 --topic stream-output --from-beginning --property print.key=true
```

In another terminal, start a bash session and pump some lines into the kafka console producer:

```
$ docker-compose -f kafka-streaming.yaml exec kafka bash
I have no name!@c07eea6aed61:/$ cat /lab-root/ickle-pickle-tickle.txt | kafka-console-producer.sh --bootstrap-server kafka:9092 --topic stream-input
```

This should produce the output count in the terminal where we're running the console consumer:

```
went    1
for     1
ride    1
a       2
...
ickle   8
pickle  8
tickle  8
me      20
too     7
```

For fun, you can submit the full text of Leo Tolstoy's "War & Peace"!

```
I have no name!@c07eea6aed61:/$ cat /lab-root/war-and-peace.txt | kafka-console-producer.sh --bootstrap-server kafka:9092 --topic stream-input
```

In the lab's root directory, you can now bring down the cluster with the command

```
$ docker-compose -f kafka-streaming.yaml down
```

Congratulations, you've completed this lab!