# Day 3 – Lab1:

## **Spark Streaming with Kafka Example**

#### Introductions

In this example, we will write a Spark Streaming program that consumes messages from Kafka.

We will reuse the whole setup from the previous lab, so this lab is best done as a continuation of Kafka Streaming setup.

For Spark, we will use Scala. To build the program, you will need to download SBT, the Scala Build Tool. This is an easy install! Follow the installation instructions from the site <u>http://www.scala-sbt.org/</u> and install it for your machine.

## **Spark Streaming Program**

```
package com.scispike.kafka
import org.apache.kafka.common.serialization.{StringDeserializer, StringSerializer}
import org.apache.spark.SparkConf
import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe
import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent
import org.apache.spark.streaming.kafka010._
import org.apache.spark.streaming.{Duration, Seconds, StreamingContext}
import org.apache.log4j.{Level, Logger}
object SparkKafka {
 def main(args: Array[String]): Unit = {
   println("Spark Kafka Example - Word count from a Kafka stream")
   if (args.length < 3) {</pre>
     System.err.println(s"""
                           Usage: SparkKafka <brokers> <topics> <interval>
                           broker1, broker2
                             <topics> is a list of one or more kafka topics to
consume from
                             <interval> interval duration (ms)
       """.stripMargin)
     System.exit(1)
```

```
}
 // Show only errors in console
  val rootLogger = Logger.getRootLogger()
  rootLogger.setLevel(Level.ERROR)
  // Consume command line parameters
  val Array(brokers, topics, interval) = args
  // Create Spark configuration
  val sparkConf = new SparkConf().setAppName("SparkKafka")
 // Create streaming context, with batch duration in ms
  val ssc = new StreamingContext(sparkConf, Duration(interval.toLong))
  ssc.checkpoint("./output")
 // Create a set of topics from a string
 val topicsSet = topics.split(",").toSet
  // Define Kafka parameters
 val kafkaParams = Map[String, Object](
    "bootstrap.servers" -> brokers,
    "key.deserializer" -> classOf[StringDeserializer],
    "value.deserializer" -> classOf[StringDeserializer],
    "group.id" -> "use_a_separate_group_id_for_each_stream",
    "auto.offset.reset" -> "latest",
    "enable.auto.commit" -> (false: java.lang.Boolean))
 // Create a Kafka stream
 val stream = KafkaUtils.createDirectStream[String, String](
    ssc, PreferConsistent, Subscribe[String, String](topicsSet,kafkaParams))
  // Get messages - lines of text from Kafka
  val lines = stream.map(consumerRecord => consumerRecord.value)
 // Split lines into words
 val words = lines.flatMap(_.split(" "))
 // Map every word to a tuple
 val wordMap = words.map(word => (word, 1))
 // Count occurrences of each word
 val wordCount = wordMap.reduceByKey( + )
 //Print the word count
 wordCount.print()
 // Start stream processing
 ssc.start()
  ssc.awaitTermination()
}
```

}

### **Building and Packaging the Spark Program**

Go to the directory spark-kafka. You will see that it contains the file build.sbt. We will use it to compile and build a jar file that we can deploy to Spark. In that directory run the command in the terminal:

sbt assembly

The first time you run sbt, it may take a while, as SBT needs to download Scala in the right version and all needed libraries.

The result is that the jar file is created in the directory target/scala-2.11/spark-kafka.jar.

#### **Deploying the Program to Dockerized Spark**

We will move the file to the directory from where we will deploy to Spark:

```
mv target/scala-2.11/spark-kafka.jar ../docker/spark/
```

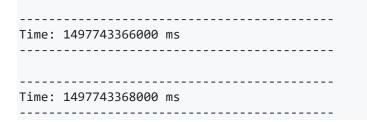
Go to the docker directory and run the deployment command.

If you don't have the docker cluster running already, you'll have to first run the dockercompose up command.

Run the command to submit the jar to Spark. The directory app in the image is mapped to directory spark on our machine, as set in the volumes parameter in the dockercompose.yml file. We are also passing command line parameters for the broker, topic, and interval:

```
docker-compose exec master spark-submit \
    --master spark://master:7077 \
    /app/spark-kafka.jar \
    kafka:9092 stream-input 2000
```

The program should be running and waiting for the input from the stream-input topic. In terminal, you should see the output of Spark Streaming:



#### Soon, we will see some word counts!

In some instances, we have observed crashes of the Spark program on deployment. If that happens, just re-run the previous command again. Sometimes, the dockerized Spark does not start until we attempt to deploy to it. You can observe the console of the docker compose and you will notice the logs for the start of the Spark master.

#### **Running the Spark Streaming with Kafka**

Run the producer as in the previous lab. Enter some text and observe the output.

In the producer terminal, run the producer and then enter a line of text:

```
$ docker-compose exec kafka /opt/kafka/bin/kafka-console-producer.sh --broker-list
kafka:9092 --topic stream-input
hello hello hi
```

In the Spark terminal, you will see the word counts for the interval:

```
Time: 1497746685000 ms
(hello,2)
(hi,1)
```

#### Done!

Congratulations - You've made it!