Day 4 – Lab2:

IoT Spark Stream Solution

NOTE: Please complete the `iot-kafka` lab before trying this lab, as this lab reuses the data producer from that one.

This example reuses the `gps-pump`, and therefore, the sample data, provided in the `iot-kafka` lab.
We're going to do the same as that lab, count the number of times a vehicle parked in a certain position, except we'll use Spark streaming to do it, being fed from a Kafka topic.

Lab

There is only one project in this lab.

- `gps-monitor-spark`: consumes simulated messages and produces vehicles that have parked.

The idea is that we'll start the `gps-monitor-spark`, then start producing messages from the `gps-pump`.
gps-monitor-spark
Using your favorite editor, open the file `iot-kafka/gps-monitor-spark/src/main/java/app/GpsMonitor.java`.
First, we instantiate a `JavaStreamingContext` given a `SparkConf` & a `Duration`. Then, we use `KafakUtils`, from Spark's Kafka streaming connector, to create a stream of lines coming in from the configured input topic. The Kafka properties come in via resource `streams.properties`.
Next, we establish the `Topology` of our stream by invoking methods on the stream that represent the processing logic that we want to perform.

- `map` transforms the incoming messages from `ConsumerRecord`s containing a tab-separated values string into arrays of strings.
- `filter` takes care to ensure that the data we get is in the expected format *and* represents a "parked" record by ensuring that the vehicle's speed is less than `1.0`. Any poorly formatted or other records are discarded.
- `mapToPair` prepares us to count things up by converting the incoming "parked" record into a tuple of vehicle id plus geohash and an initial count of `1`.

- `reduceByKey` uses the tuples produced in the prior step to count each unique combination of vehicle id & geohash, yielding our desired values.
- `print` simply prints those counts.

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

Do it!

Build the consumer application

Open a new terminal in the `iot-spark/gps-monitor-spark` directory and build it:
```
$ docker run -it --rm -v "$(cd "$PWD/../../../.."; pwd)":/course-root -w /course-root/labs/06-Streaming/iot-spark/gps-monitor-spark -v "$HOME/.m2/repository":/root/.m2/repository maven:3-jdk-11 ./mvnw clean package
```

Run everything

Now, it's time to fire up Kafka & Spark, the `gps-monitor-spark` and the `gps-pump`.
First, ensure any prior Kafka clusters from any previous labs are stopped.

Open another new terminal in the lab's root directory, `06-Streaming`, and start the Kafka & Spark cluster:
```
$ docker-compose -f spark-streaming.yaml up
```

Once the log output from the above commands stops being written, open yet another terminal in the lab's root directory and create our topics then listen to the output topic using a console consumer:

```
$ docker-compose -f spark-streaming.yaml exec kafka bash
I have no name!@2ec21727cbdc:/$ kafka-topics.sh --boostrap-server kafka:9092 --create --topic gps-locations
```

Return to the terminal in which you built the `gps-monitor-spark` project, and submit the Spark job:
```
$ docker-compose -f spark-streaming.yaml exec spark-master bash
I have no name!@2ec21727cbdc:/$ spark-submit --master spark://spark-master:7077 --class app.GpsMonitor /lab-root/iot-spark/gps-monitor-spark/target/gps-monitor-spark-1.0.0-SNAPSHOT.jar
```

After successful submission, that terminal will now periodically produce empty output until we start pumping data into the Kafka topic with the `gps-pump`.
Next, return to the terminal in which you built the `iot-kafka/gps-pump` project, and start it:

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

You should see activity in the two project terminals. Return your attention to the terminal running the Spark job. You should see output similar to the following:

```
-------------------------------------------
Time: 1645132250000 ms
-------------------------------------------
(120@9v6mjwp,1)
(111@9v6mjwp,1)
(111@9v6mjy2,1)
(107@9v6mjy9,11)
(111@9v6mjwn,2)
(111@9v6mjy0,1)
(120@9v6mjy8,2)
(88@9ufw1jp,5)
(120@9v6mjwn,1)
(120@9v6mjy9,15)
...
```

This output is showing you the count of a given vehicle in each location.

Congratulations, you've completed this lab!