Day 4 – Lab1:

IoT Kafka Stream Solution

In this example, we process real-world vehicle IoT data. Our data is in a tab-separated values file `iot-kafka/gps-pump/src/main/resources/data.tsv`. This file contains rows of vehicle sensor data representing car movements.

Data Schema

The data fields are:

- Col 1: Device ID (unique for each vehicle)
- Col 2: Instant of the observation (in ISO-8601 format)
- Col 3: Speed of the vehicle
- Col 4: The compass direction of the vehicle
- Col 5: The longitude of the GPS coordinates
- Col 6: The latitude of the GPS coordinates

Accuracy of GPS Coordinates

For GPS coordinates, about 100 meters accuracy is roughly coordinates rounded to 3 decimal places.

See GIS Accuracy for a detailed explanation.

We'll use a trick using geohashing. Geohashing is an open-source algorithm that splits the world into tiles. You can read about geohashes here

Goal

The goal is to count the number of times a car is observed parked at the same location (that is, the same geohash) with accuracy of about 100 meters. We'll use Kafka Streams to do so.

Lab

There are two projects in the lab.

- `gps-pump`: produces simulated messages from a tab-separated values file.
- `gps-monitor`: consumes simulated messages and produces vehicles that have parked.

The idea is that we'll start the `gps-monitor` and a console consumer of the final output topic, then start producing messages from the `gps-pump`.

`gps-pump`

In your favorite editor, open file `iot-kafka/gps-pump/src/main/java/app/GpsDeviceSimulator.java`. Notice that it takes a Kafka producer and the data file name, which must be on the application's classpath.

All it does is read the data file from beginning to end, streaming each literal, tab-delimited line as a Kafka message on the `gps-locations` topic, then restarts from the top each time we hit the end of the file.

The file `iot-kafka/gps-pump/src/main/java/app/GpsDeviceSimulatorApp.java` simply contains the bootstrap logic to configure dependencies then instantiate and start the `GpsDeviceSimulator`.

`gps-monitor`

Now open file `iot-kafka/gps-monitor/src/main/java/app/GpsMonitor.java`. Notice that it uses a `StreamsBuilder` to create the stream processing topology that implements our business logic.

1. The first step in our topology is to split the tab-delimited line into an array for downstream processing, using `map`.
2. Next, we use `filter` to keep only those messages that contain valid data & have a speed value of less than one, meaning "parked".
3. Then we use `map` to transform the "parked" record into a string form of a vehicle & location using the `toString()` method of `LocationKey`.
4. Now, we group by vehicle & location, and
5. `count` them up, giving us the running count of the number of times a vehicle was parked at a location.
6. Finally, we convert the counts from `Long` to `String` to abide by our serde's requirements,
7. Convert the `KTable` of counts back to a `KStream`, and
8. Produce those records on the configured output topic.

We then build our `Topology`, and get a `KafkaStreams` reference to `start`, then run until the user hits enter.

Do it!

Build both applications

Open a terminal in the `gps-pump` directory and issue the following command:

```
$ docker run -it --rm -v "$(cd "$PWD/../../../.."; pwd)":/course-root -w /course-root/labs/06-Streaming/iot-kafka/gps-pump -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.

Similarly, open another terminal in the `gps-monitor` directory and build it:

```
$ docker run -it --rm -v "$(cd "$PWD/../../../.."; pwd)":/course-root -w /course-root/labs/06-Streaming/iot-kafka/gps-monitor -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.

Run everything

Now, it's time to fire up Kafka, a console consumer of the final output topic, the `gps-monitor` and the `gps-pump`.

Open another new terminal in the lab's root directory, `06-Streaming`, and start the Kafka cluster:

```
$ docker-compose -f kafka-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 kafka-streaming.yaml exec kafka bash
I have no name!@2ec21727cbdc:/$ for it in gps-locations frequent-parking; do kafka-topics.sh --boostrap-server kafka:9092 --create --topic $it; done
I have no name!@2ec21727cbdc:/$ kafka-console-consumer.sh --boostrap-server kafka:9092 --topic frequent-parking --property print.key=true
```

That terminal will now be quiet until our `gps-monitor` produces some messages on the `frequent-parking` topic.

Return to the terminal in which you built the `gps-monitor` project, and fire it up:

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

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.

Next, return to the terminal in which you built the `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
```

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.

You should see activity in the two project terminals. After some time, you'll see activity in the console consumer terminal similar to the following:

```
88@9uftzw3      6
88@9uftzqs      24
88@9ufw1jp      30
120@9v6mjy8     12
111@9v6mjwn     12
120@9v6mjwn     6
111@9v6mjwp     6
120@9v6mjwp     6
111@9v6mjy2     6
111@9v6mjy0     6
120@9v6mjy3     24
...
```

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

Congratulations, you've completed this lab!