Exercise 13

Stream processing with Kafka and Siddhi

Prior Knowledge
Unix Command Line Shell
JSON

Learning Objectives
Understanding Kafka
Siddhi SQL complex event processing

Software Requirements

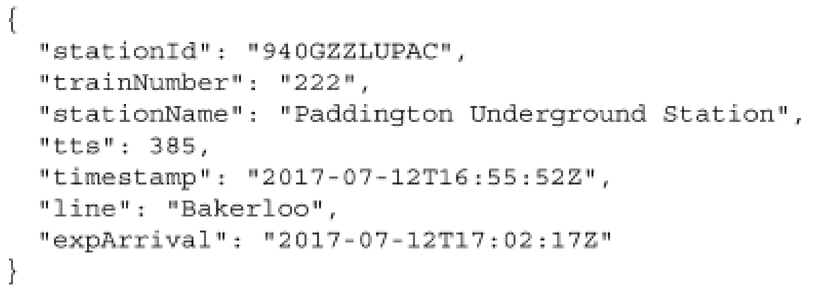
- Kafkacat
- Nano text editor or other text editor

1. I have started a system running on the Internet that is accessing TfL's prediction API. This is publishing data onto a Kafka topic.

2. You can look at those entries using a tool called **kafkacat**

```
kafkacat -b kafka.freo.me -t tfl -o end
```

3. You should see a lot of JSON scrolling past. Each line looks like this when pretty printed:



```
{
  "stationId": "940GZZLUPAC",
  "trainNumber": "222",
  "stationName": "Paddington Underground Station",
  "tts": 385,
  "timestamp": "2017-07-12T16:55:52Z",
  "line": "Bakerloo",
  "expArrival": "2017-07-12T17:02:17Z"
}
```

4. The fields are mainly self-explanatory. The only non-self-explanatory field is **tts** which is the time to the next station. In general this should be equal to (expArrival - timestamp) and in the event above we can see that this is true.

5. Each train will have multiple entries, predicting its time of arrival at several stations ahead of its current position. Just to make it more complex, the trainnumbers are not unique (only unique to a line).

6. There are two files you need to proceed further. The first is a program I have provided that runs Siddhi queries against this event stream. The second is some SiddhiQL that will be run.

7. Make a new directory:
```
mkdir ~/stream
cd ~/stream
```

8. Download the two files:
```
wget https://freo.me/sk-jar -O sk.jar
wget https://freo.me/plan-siddhi -O plan.siddhi
```

9. If you want to look at the code, you can browse it here: https://github.com/pzfreo/siddhi-kafka/tree/master/sk/src/main/java/me/freo/sk

10. There are instructions below for building the code, if you want to change or edit it.

11. Take a look at plan.siddhi

    It is split into separate phases. The first phase is dead simple… it just outputs what is input.

12. The documentation for this language is available here: https://siddhi.io/

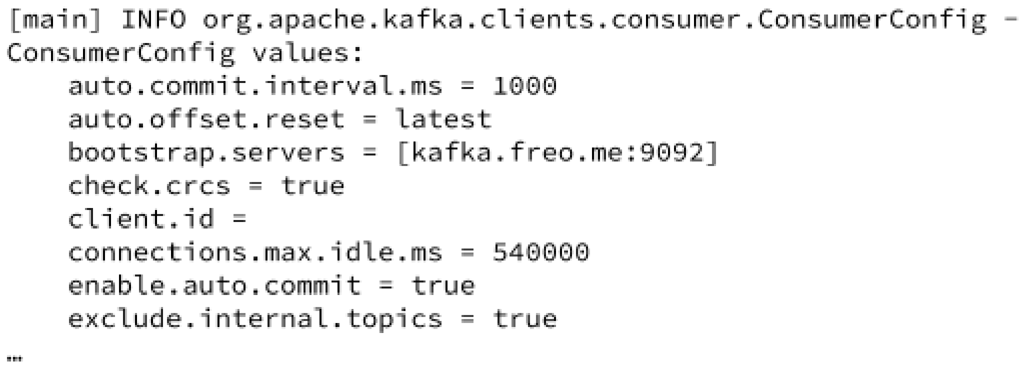
13. Before modifying the plan, it is worth understanding what you can and can't change.

There are a couple of points where the code is 'hard-coded' to this setup and plan. Firstly, it is specifically looking to the broker (*kafka.freo.me*) and topic (*tfl*). Secondly, it is expecting a specific input stream called *tflstream*. Thirdly, it will take whatever output you send to a stream called *outstream* and print it as JSON to the console.

All of these could be improved easily, but they don't affect our learning objectives.

14. Firstly, lets run the current plan, which simply copies the input stream to the output stream.

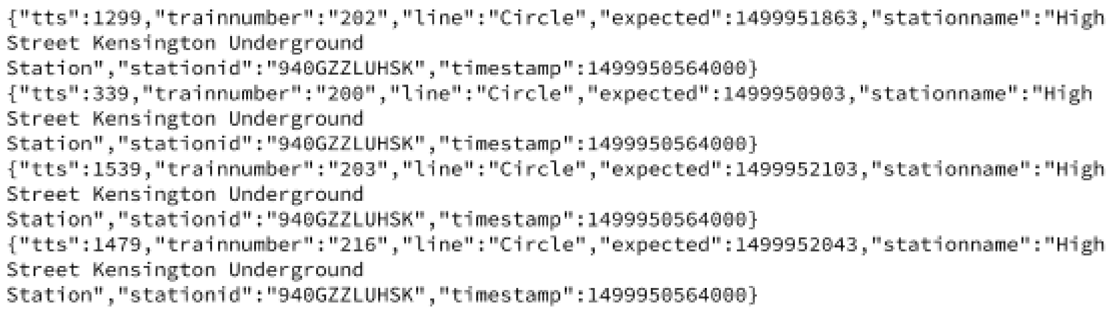
```
java -jar sk.jar plan.siddhi
```



```
[main] INFO org.apache.kafka.clients.consumer.ConsumerConfig -
ConsumerConfig values:
    auto.commit.interval.ms = 1000
    auto.offset.reset = latest
    bootstrap.servers = [kafka.freo.me:9092]
    check.crcs = true
    client.id =
    connections.max.idle.ms = 540000
    enable.auto.commit = true
    exclude.internal.topics = true
…
```

You may not see it because of scrolling, but the first output is a bunch of Kafka logging:

15. You should now see lots of log like:



```
{"tts":1299,"trainnumber":"202","line":"Circle","expected":1499951863,"stationname":"High
Street Kensington Underground
Station","stationid":"940GZZLUHSK","timestamp":1499950564000}
{"tts":339,"trainnumber":"200","line":"Circle","expected":1499950903,"stationname":"High
Street Kensington Underground
Station","stationid":"940GZZLUHSK","timestamp":1499950564000}
{"tts":1539,"trainnumber":"203","line":"Circle","expected":1499952103,"stationname":"High
Street Kensington Underground
Station","stationid":"940GZZLUHSK","timestamp":1499950564000}
{"tts":1479,"trainnumber":"216","line":"Circle","expected":1499952043,"stationname":"High
Street Kensington Underground
Station","stationid":"940GZZLUHSK","timestamp":1499950564000}
```

16. When you are happy with the output, hit Ctrl-C

17. Now comment the first query and uncomment the second. Re-run the code.

18. This will now show all the trains expected in the next 10 seconds

19. Re-comment this and uncomment the next phase (PHASE 2). This next query is a bit cleverer. It groups the trains by trainNumber and identifies the lowest expected time for that train. It then restricts the output to that event only. Effectively, these events tell you for each train, which station it is due at next and when. This outputs every 20 seconds.

    In addition, this creates a new column called train, which joins the train and the line to give each train a unique identifier.

Rerun the code again.

20. Finally, comment phase 2 and uncomment phase 3. This now has two queries. The first query populates an intermediate stream (just the same as phase 2, but no longer named *outstream*).

The next query demonstrates three new things.

1) Internal streams and using the output of one query as the input to another query.
2) Partitioning. This lets us specify that we are only interested in what happens to a particular train. This is really important, because it cuts down the combinatorial problem for the engine. In addition, this is key for sharding and clustering.
3) Patterns and Sequences. We are now going to look at how the current event compares to previous events. See https://docs.wso2.com/display/DAS400/Patterns+and+Sequences

The query now looks for trains where the current expected arrival time is more than 30 seconds later than the previous expected arrival time.

Rerun the phase 3 code.

21. The code is actually not just finding late trains. It is also finding the cases where the train has changed destination station.

22. Fix the code so it only finds delayed trains.

23. Congrats you are done.

Some extensions:

24. Output how many trains are running at any given time.

25. Calculate the average time between stations.

26. Create an alert that fires if the average time goes up by more than 2 seconds compared to the previous average. Choose a period to measure this on.

27. Building the code (requires git and maven)

```
git clone https://github.com/pzfreo/siddhi-kafka.git
cd siddhi-kafka
cd sk
mvn clean package
```