UNIVERSIDADE DE COIMBRA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
Departamento de Engenharia Informática

3º Projeto / Project #3

Integração de Sistemas/
Enterprise Application Integration

2024/25 – 1st Semester
MEI, MSI

Deadline: 2024-12-20

Nota: A fraude denota uma grave falta de ética e constitui um comportamento não admissível num estudante do ensino superior e futuro profissional. Qualquer tentativa de fraude pode levar à reprovação na disciplina tanto do facilitador como do prevaricador.

MUITO IMPORTANTE: o código entregue pelos alunos vai ser submetido a um sistema de deteção de fraudes.

VERY IMPORTANT: the code delivered by students will be submitted to a fraud detection system.

Message Oriented Middleware (MOM) and Kafka Streams

Objectives

- Learn how to create simple asynchronous and message-oriented applications.
- Learn to use Kafka Streams.

Resources

Apache Kafka Introduction: https://kafka.apache.org

Kafka Streams: https://kafka.apache.org/documentation/streams/

Complete Kafka Producer with Java:
https://www.conduktor.io/kafka/complete-kafka-producer-with-java/
Kafka Producer base documentation:
https://docs.confluent.io/platform/current/clients/producer.html

Complete Kafka Consumer with Java:
https://www.conduktor.io/kafka/complete-kafka-consumer-with-java/
Kafka Consumer base documentation:
https://docs.confluent.io/platform/current/clients/consumer.html

Look for the Kafka and Kafka Streams materials available on UC Student.

Make sure you understand the following concepts:

- Producer
- Consumer (and Consumer Groups)
- Topic
- Partition and partition offset
- Broker
- Zookeper (KRaft now)

Kafka Training (doesn't count for evaluation)

1. You should start by installing Kafka (or find out where it is installed if you are using a container).

2. You may look at the reading material suggested before and follow that material to have a basic example running with a producer and consumer.

3. What happens to the messages that arrive at the topic before the subscriber makes the subscription? How do you configure the subscriber to read historical data, i.e., all messages from the beginning?

4. How do you configure different partitions inside a topic?

5. What is the point of Consumer Groups? How do they work?

6. Now, read the Kafka Streams tutorial and run the example that counts the words that go through a stream.

7. Refer to the following blog message for this and the following exercises:
   https://eai-course.blogspot.com/2018/11/playing-with-kafka-streams.html

   Use Kafka Streams to count the events that happened for a given key. Display the result as 'k->v' in the final stream.

8. How do you change the type of data of the keys and values that you send?

9. How do you switch the key and value of a stream?

10. Now sum all the values that showed up for each key.

11. Now, control the time considered in the stream to be 1 minute.

12. How could you send complex data instead of primitive types?

13. Imagine that you have two streams, *e.g.*, debit and credit. How do you create a third stream with the current balance?

14. Use the Kafka Connect application to periodically fetch data from a database table (source). You can find help for this operation in the following blog message: https://eai-course.blogspot.com/2019/11/how-to-configure-kafka-connectors.html

15. Now do the inverse operation: send from a topic to a database table (sink).

16. Data in Kafka Topics are streams of bytes. This means that data needs to be serialized and de-serialized into and from appropriate formats for consumption. Kafka supports serialization and de-serialization (serdes) of primitive types (String, Long, etc), custom serializers and common serialization formats, namely, AVRO, Protobuf & JSON. Using the below guides try and serialize data to and from Kafka topics with an Avro, Protobuf or JSON formatter WITHOUT Schema Registry.
https://docs.confluent.io/platform/current/streams/developer-guide/datatypes.html

17. Kafka supports several delivery semantics: “At least once”, “At most once” and “Exactly one”. There are a series of pre-requisites and caveats, especially for the “exactly once” semantics (e.g. idempotent producer, isolation level of the consumer), namely the way Kafka deals with the consumer offsets. Try and implement a highly constrained scenario (one consumer, one producer) where “exactly once” semantics can occur.
https://docs.confluent.io/kafka/design/delivery-semantics.html

Project Description (for evaluation)

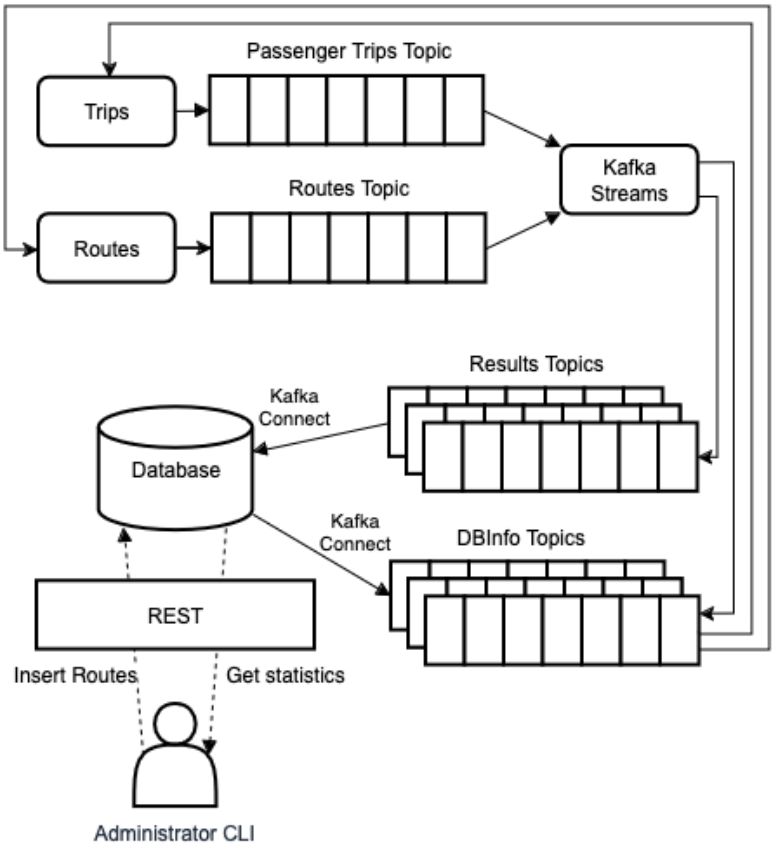

Figure 1 - Overview of the Urban Mobility company

Refer to Figure 1, summarizing an Urban Mobility company heavily relying on Kafka. The company is built around the following Kafka topics:

- **DBInfo** topics, where one or more source connectors write a list of Route suppliers and a list of Route passenger capacities from the database.
- **Routes** topic, where a process simulates routes being provided by suppliers. This process continuously generates Route capacities listed in the DBInfo topic. Each route includes its passenger capacity, the origin and destination, the transport type, e.g., Bus, Taxi, Train, Metro, Scooter, and the Operator name/identifier that provides the route. Students are expected to create Routes by randomly creating capacity (e.g 10, 200, 1), destination and origin (or route name/Id), the type and a route Operator.
- **Trips** topic, where a process simulates customers and continually writes trip data. Each trip has a Route/name id reference, a destination and origin, a Passenger name/identifier, and a transport type (e.g., Bus, Taxi, Train, Metro, Scooter). Students need not to be concerned about whether the route capacity is

full, if the route origin and/or destination match or if the transport type matches the existing entries the routes topics

- One or more applications using Kafka streams will listen to these topics and compute a few metrics, as enumerated in the requirements, including occupancies, passenger averages, total trips, etc. This or these applications will then write their computations to another topic, the **Results** topics, from where a Kafka connector will write the results back to the database.

A server-side application will read data from the database and expose the data via REST. This could be accessed, *e.g.*, by a command line application to let the administrator write and read data through the REST services. No authentication is necessary.

Components to Develop

Students need to develop the following applications of Figure 1: Passenger Trips, Routes, Kafka Streams. These are stand-alone applications. Students must also configure Kafka connectors to automatically extract and write data from/to the database via the Results and DBInfo topic. They may interact directly with the database, *i.e.*, they do not need to implement the Administrator CLI and the REST server.

The precise definition of the communication format is left for the students to determine. Clean solutions that serialize complex data structures as JSON strings are encouraged. This includes the use of standard Kafka Serializer/De-serializer constructs.

To simplify the configuration work, the professors will provide a folder with YAML and Docker files with the configuration of multiple services. Students must later configure the `lib` directory mentioned in the YAML file and optionally the `config` directory as well.

Requirements

Students should use Kafka Streams to implement the following requirements and write the results to the database:

1. Add route suppliers to the database. To simplify route suppliers cannot be deleted and optionally not changed.
2. List route suppliers from the database.
3. Add routes to the database. Again, these cannot be deleted but may be changed if students wish.
4. Get the passengers per route
5. Get the available seats per route
6. Get The Occupancy percentage per route
7. Get total passengers (from trips)
8. Get total seating available for all routes
9. Get The Occupancy percentage total (for all routes)
10. Get the average number of passengers per transport type
11. Get the transport type with the highest number of served passengers (only one if there is a tie)
12. Get the routes with the least occupancy per transport type
13. Get the most used transport type in the last hour (use a tumbling time window)

14. Get the least occupied transport type in the last hour (use a tumbling time window)
15. Get the name of the route operator with the most occupancy. Include the value of occupancy
16. Get the name of the passenger with the most trips

Solutions should maximize the computations that students do with Kafka streams and should not depend on complex database queries to extract results (for example, students should use Kafka Stream to compute the average amount spent, instead of computing them on the database).

Students should include means in their application to enable a fast verification of the results they are storing.

Students should also leverage Apache Kafka fault-tolerance mechanisms, *i.e.*, configure a multi-broker setup to prevent data losses due to a potential broker crash. Therefore, students should be able to stop one of the brokers without losing information or halting the entire system.

Final Delivery

- The project should be made in groups of 2 students. We do expect all the members of the group to be fully aware of all the parts of the code that is submitted. Work together!
- To automate the build of the project, students should use Maven.
- Students should submit the project part that is for evaluation in a zip file with the **source** of a complete Maven project, using Inforestudante. Do not forget to associate your work colleague during the submission process.
- Grading is performed based on individual student defenses.
- No report is necessary.

Good Luck!