

**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. You 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). You don’t need 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

You need to develop the following applications of Figure 1: Passenger Trips, Routes, Kafka Streams. These are stand-alone applications. You 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 you to determine. Use clean solutions that serialize complex data structures as JSON strings. This includes the use of standard Kafka Serializer/De-serializer constructs.

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

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

You should include means in the application to enable a fast verification of the results being stored.

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

To automate the build of the project, you should use Maven.

**What needs to be done and demonstrated to work:**

1. Add route suppliers to the database

2. List route suppliers from the database

3. Add routes to the database

4. Get the passengers per route (use information from trips topic only)

5. Get the available seats per route (Use information from routes topic only)

6. Get The Occupancy percentage per trip

7. Get total passengers (from trips topic only)

8. Get total seating available for all routes (from the routes topic only)

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

17. Serializer/Deserializer (CustomSerializer or JSON serialized to a Java object. e.g. KStream<String, Route>)

18. Configure Kafka with multiple brokers to enable Fault-Tolerance

19. Source connections to read from the database

20. Sink connections to send results to the database

21. Enabling simple verification of results