**Enterprise Integration (MEIC-A, 2019-20, 2º semestre)**

Instituto Superior Técnico – MEIC-A

*Sprint 2 Report*

1. **Definition of the microservices needed for the MaaS functionality**

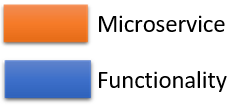


Fig. 1 – Microservices and functionalities

1. **Chosen microservices**

Customer Management and Operator management

1. **Microservices input and output**

Customer Management Service:

**Input**

This service takes as input NewUser events and trip events that are consumed from the operator topics.

* **Trip Events**

There are 3 types of trip events:

* **Type 0 operators** (Check in and check out method)

These events contain as main fields eventType to say if it is a check-in or a check-out and the name of the operator. Inside the second field there are fields with the token of the customer, station and a timestamp.

Examples:

Check-in in metro

{

    "event": {

        "eventType": "t0-check-in",

        "Metro": {

            "CheckIn": {

                "Token": "t1",

                "Station": "Odivelas",

                "Timestamp": "2020-02-29 18:23:41.278"

            }

        }

    }

}

Check-out in metro

{

    "event": {

        "eventType": "t0-check-out",

        "Metro": {

            "CheckOut": {

                "Token": "t1",

                "Station": "Alameda",

                "Timestamp": "2020-02-29 18:23:47.718"

            }

        }

    }

}

* **Type 1 operators** (Distance and time dependent method)

These events contain as main fields eventType to say this operator has type t1 and the name of the operator. Inside the second field there are fields with the token of the customer, price and a timestamp.

Example:

{

    "event": {

        "eventType": "t1",

        "Uber": {

            "Usage": {

                "Token": "t2",

                "Price": "70.51.901",

                "Timestamp": "2020-02-29 19:45:58.638"

            }

        }

    }

}

* **Type 2 operators** (Time dependent method)

These events contain as main fields eventType to say this operator has type t2 and the name of the operator. Inside the second field there are fields with the token of the customer, time spent, price and a timestamp.

Example:

{

    "event": {

        "eventType": "t2",

        "Gira": {

            "Usage": {

                "Token":"t1",

                "Time":"3600", -> just used for user history

                "Price": "12.60",

                "Timestamp":"2020-02-29 20: 57: 10.294"

            }

        }

    }

}

* **New User Events**

These events contain as main fields eventType and user. The user field contains information about a user: id, email, plan type, first name, last name, balance and a field that says if the user has a pass or not.

{

    "event":{

        "eventType": "new-user",

        "user": {

            "id": "69c594cfdeeaedd220",

            "email": "user@gmail.com",

            "planType": "pre-paid",

            "firstName": "Paulo",

            "lastName": "Neves",

            "balance": "500",

            "hasPass": "true"

        }

    }

}

<root>

  <eventType>new-user</eventType>

  <user>

    <id>69c594cfdeeaedd220</id>

    <email>user@gmail.com</email>

    <firstName>Paulo</firstName>

    <lastName>Neves</lastName>

    <planType>pre-paid</planType>

    <balance>500</balance>

    <hasPass>true</hasPass>

  </user>

</root>

**Output**

This service has as output a TripCost Event. This event has the goal provide information to the Operator Management Service compute the how much money goes to the operator and how much money should be debited from the user account.

* + **TripCost Events**

These events contain as main fields eventType and info. The info field contains information about the trip and the user: base cost of the trip, token of the user, info about whether a user has a pass or not, trip ID, type of operator, operator name and timestamp.

Example:

{

    "event": {

        "eventType": "trip-cost",

        "info": {

            "baseCost": "23",

            "token": "69c594cfdeeaedd220",

            "hasPass": "true",

            "tripId": "tod89430d",

            "operatorType": "t1",

            "operatorName": "Uber",

            "timeStamp": "2020-02-29 20:57:10.294"

        }

    }

}

Operator Management Service:

This service has as input TripCost Event, NewOperator events, NewService events and NewDiscount events. This event has the goal provide information to the Operator Management Service compute the how much money goes to the operator and how much money should be debited from the user account.

**Input**

* + **TripCost Events**

(Described before)

**Output**

This service has as output a Debit Event. This event has the goal to provide information to the Payment Service to debit money from the user account.

* **Debit Events**

These events contain as main fields eventType and info. The info field contains information about the amount to debit and the user ID.

Example:

{

    "event": {

        "eventType": "debit",

        "info": {

            "token": "69c594cfdeeaedd220",

            "amount": "20"

        }

    }

}

1. **Functional integration of the two microservices with the previous Kafka topics**

We started by deleting the topic that we created in the first sprint because we won’t need it anymore:

sudo /usr/local/kafka/bin/kafka-topics.sh --zookeeper <Public\_DNS>:2181, <Public\_DNS>:2182, <Public\_DNS>:2183 --delete --topic Discounts

Then we created two topics: TripCosts and Debit

sudo /usr/local/kafka/bin/kafka-topics.sh --create --zookeeper *localhost:*2181, localhost:2182, localhost:2183 -replication-factor 3 --partitions 3 --topic TripCosts

sudo /usr/local/kafka/bin/kafka-topics.sh --create --zookeeper *localhost:*2181, localhost:2182, localhost:2183 -replication-factor 3 --partitions 3 --topic Debit

The TripCosts topic will be used for the TripCosts events, the Customer Management Service will produce this type of events to this topic for them to be consumed by the Operator Management Service.

The Debit topic will be used for the Debit events, the Operator Management Service will produce this type of events to this topic for them to be consumed by the Payment Service.

We decided to create just one topic for each purpose for simplicity, but we added a replication factor of 3 to ensure some redundancy. The 3 partitions are just to allow the same degree of parallelism that the other topics have.

**(Create new topic for new services?)**

1. **Functional integration of the two microservices with the previous Kafka topics**
2. **Implementation of the two microservices**
3. **Functional testing**