

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

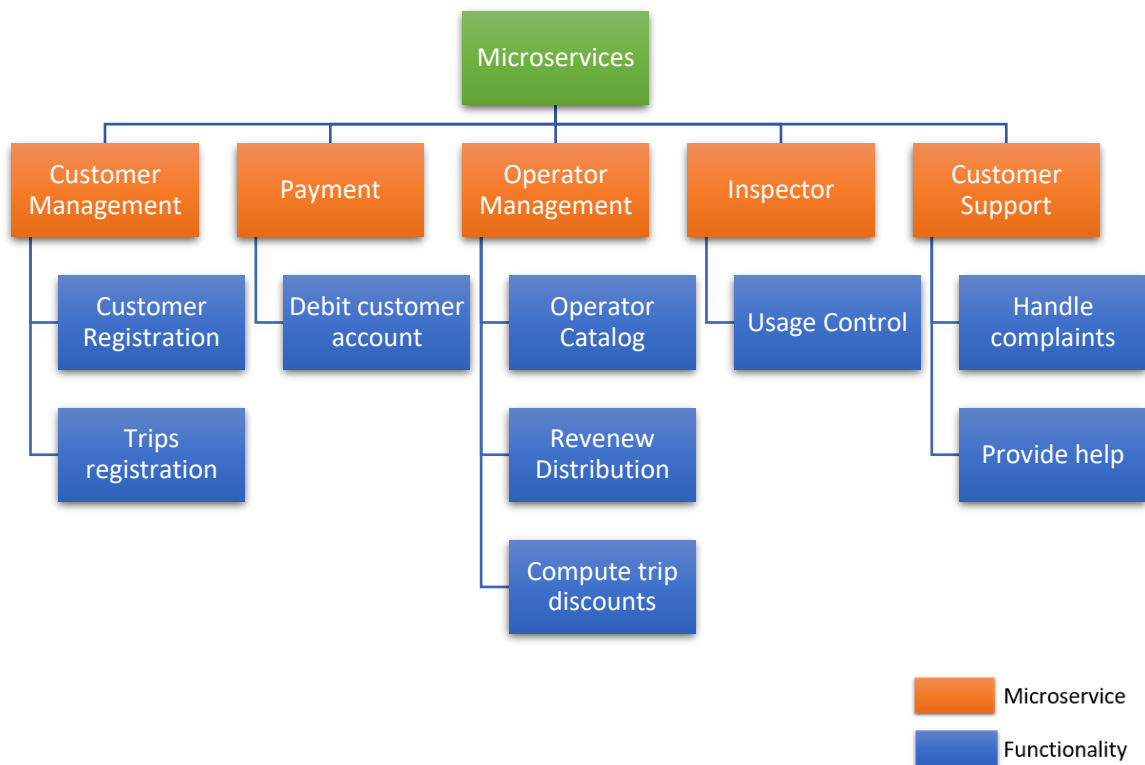


Fig. 1 – Microservices and functionalities

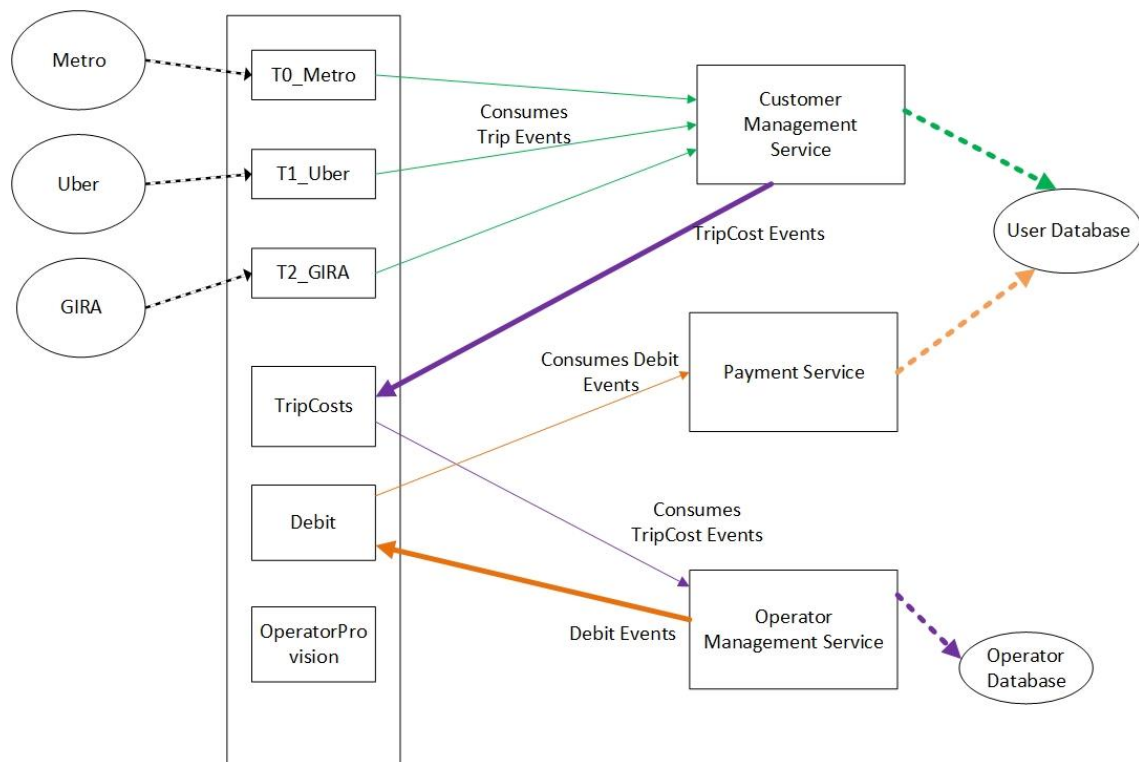


Fig. 2 – Taxation Event flows diagram

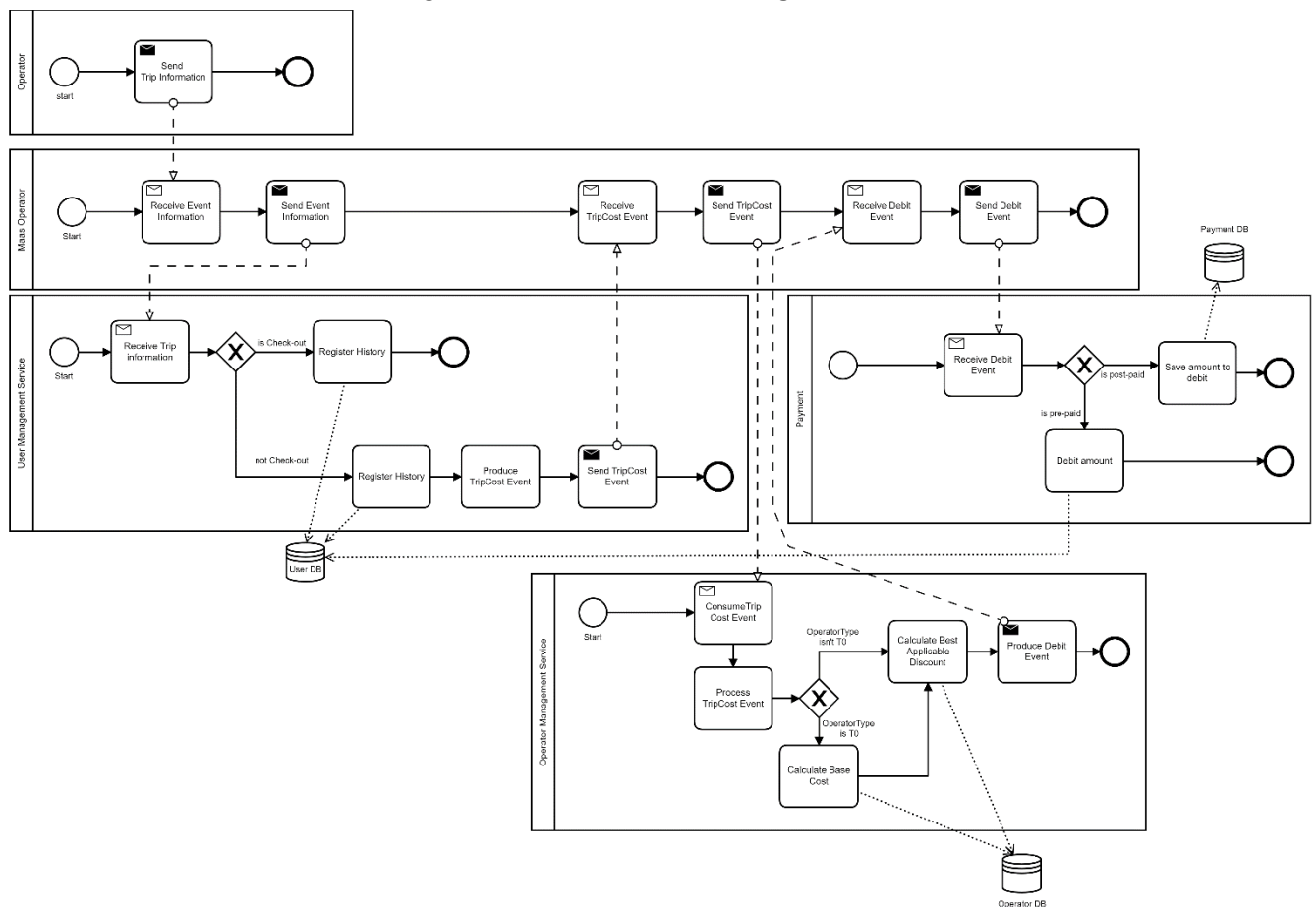


Fig. 3 – Taxation Process diagram

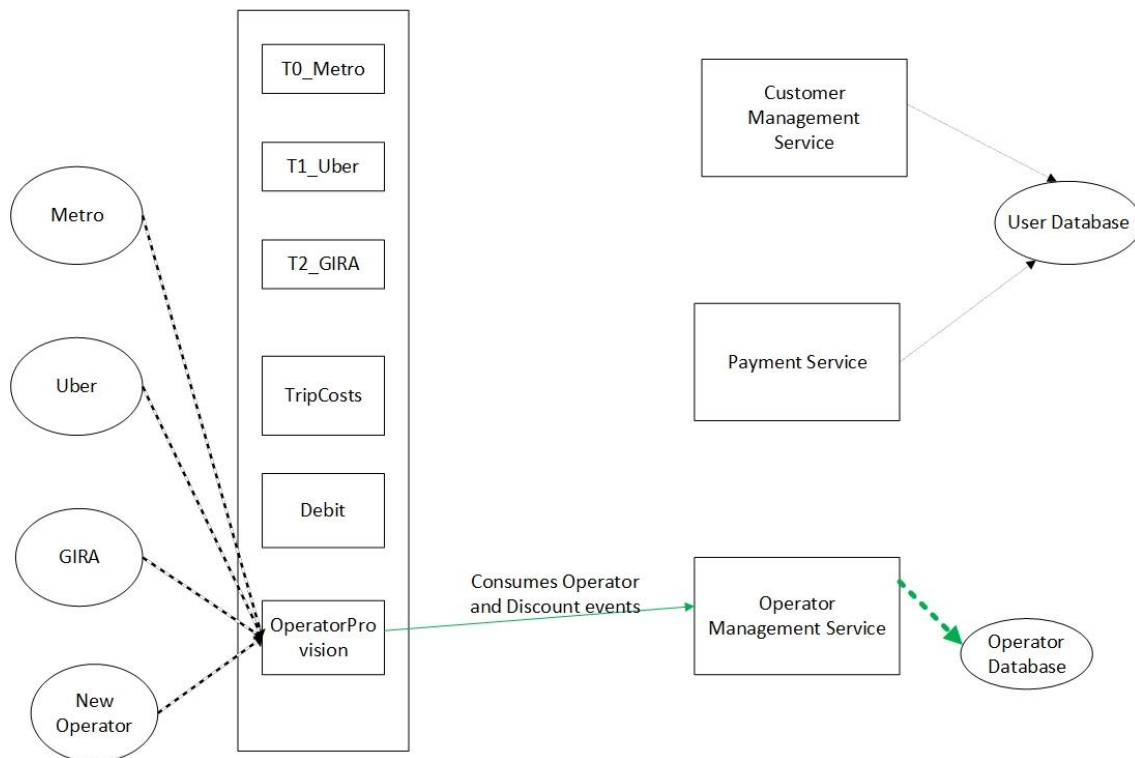


Fig. 4 – Operator and Discount Event flows diagram

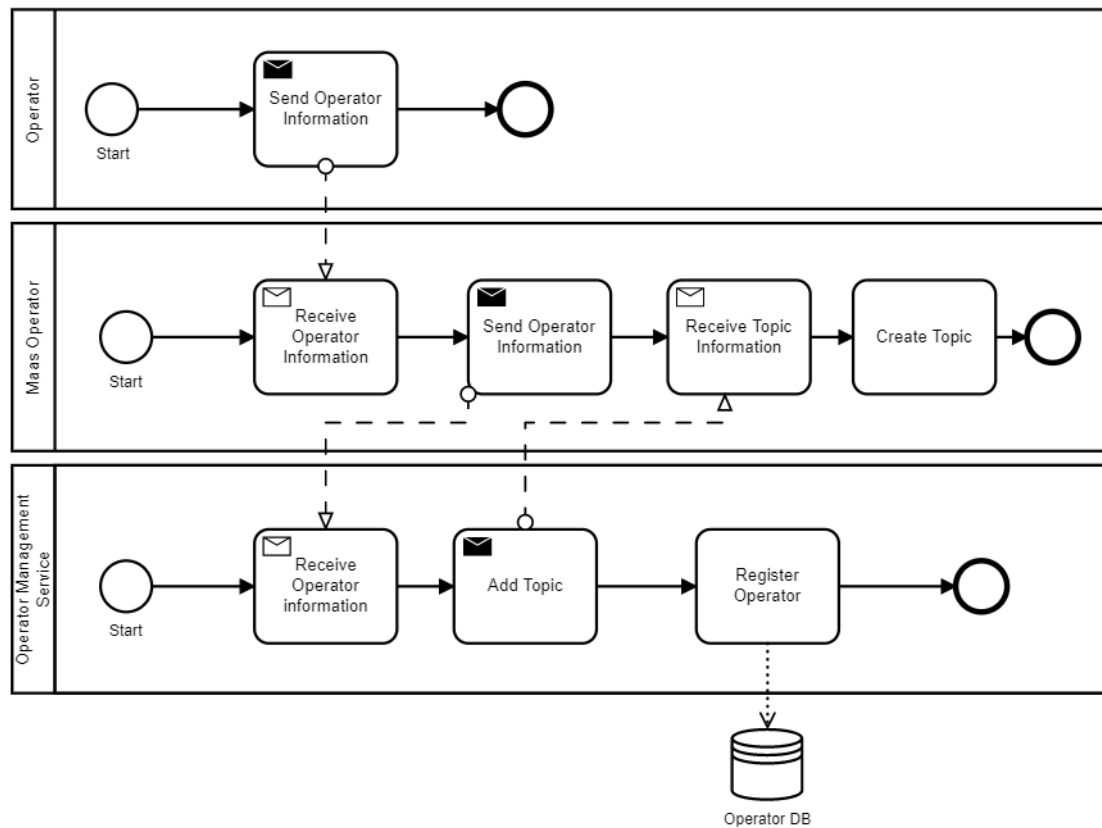


Fig. 5 – Operator Registration Process diagram

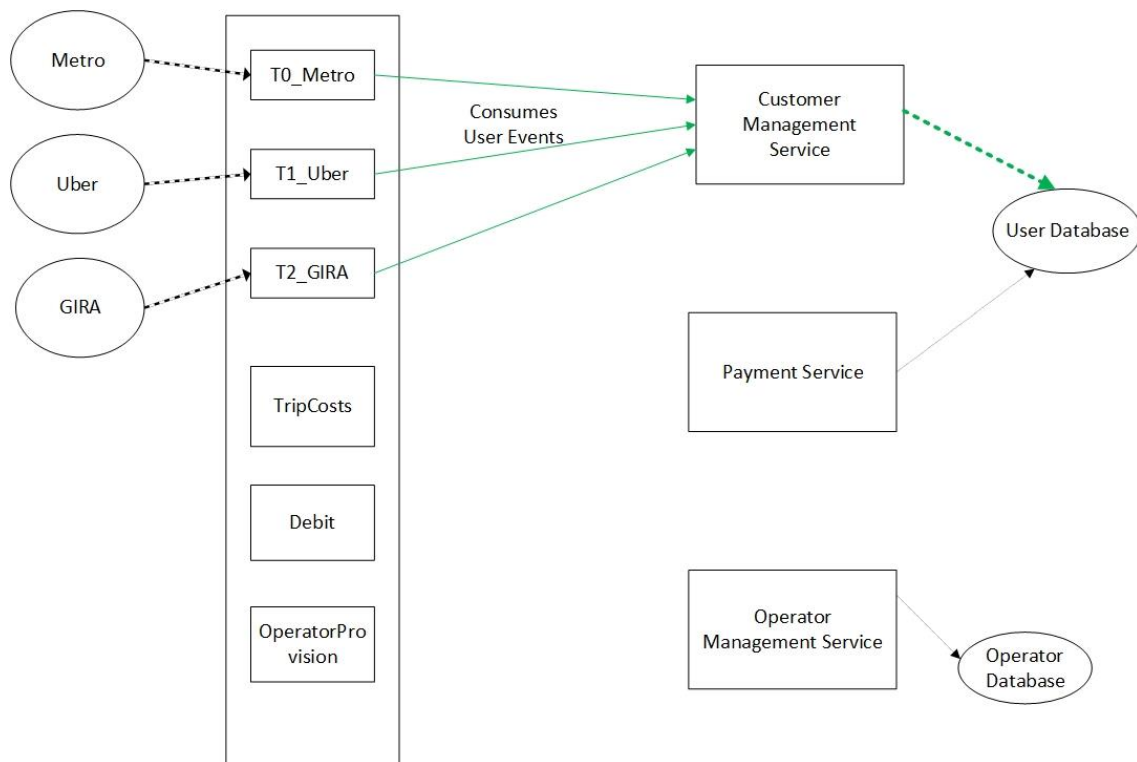


Fig. 6 – User Event flows diagram

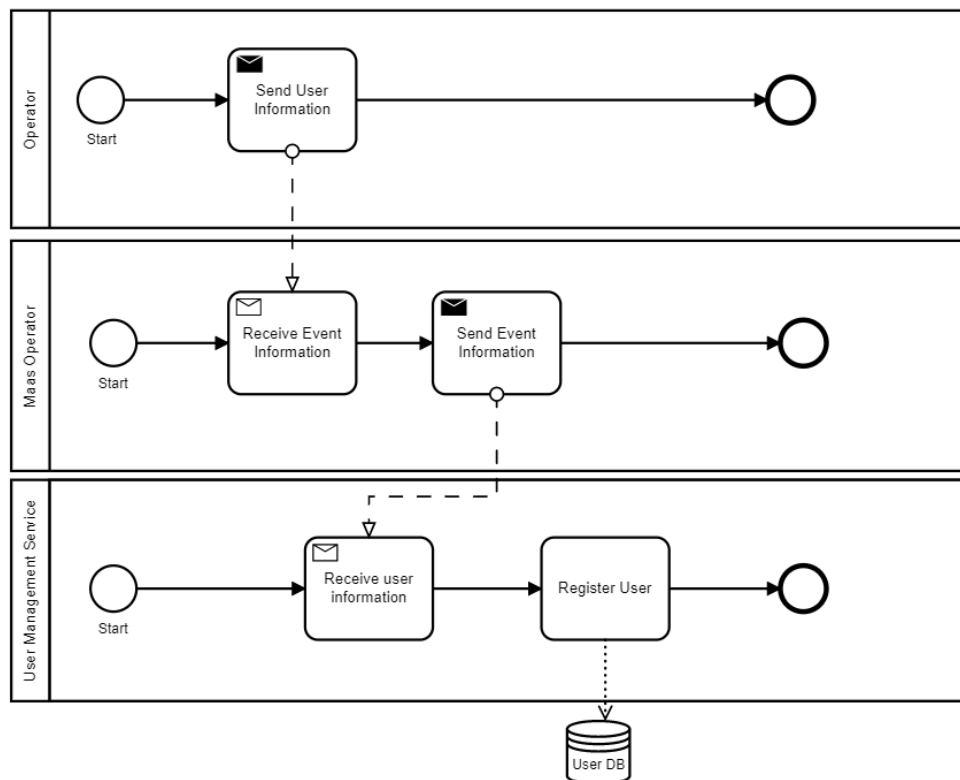


Fig. 7 – User Registration Process diagram

2. Chosen microservices

Our project plans the usage of 5 microservices to provide all the needed functionalities:

- **Customer Management Service:** this service aggregates all the functionalities that have to do with the users like consuming trip events, provide information about the trips to Operator Management Service, creation of users and maintain user history about the trips
- **Payment Service:** this service has the responsibility of debiting from the user accounts considering all the business rules
- **Operator Management Service:** this service aggregates all the functionalities that have to do with the operators and discounts. It deals with the creation of all them and consumes the events produced by the User Management Service, processes them by applying the available discounts and produces a debit event for the Payment Service to consume
- **Inspector:** this service has the responsibility of checking if there is any abnormality in the system like for example a user making a checkout without having checked in
- **Customer Support:** this service handles complaints and provides help to the user when there is some problem

We have decided to implement the User Management Service and the Operator Management Service because we consider them the core of the Maas Operator. These services comprise the functionalities that deal with customer accounts, operators, discounts, and the processing of trip events.

3. 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
- **operator** that contains the name of the operator
- **info** that contains information about a trip. Inside this field there are fields with the token of the customer, station, and a timestamp.

Examples:

Check-in in metro

JSON

```
{
  "event": {
    "eventType": "t0-check-in",
    "operator": "Metro",
    "info": {
      "Token": "t15345",
      "Station": "Odivelas",
      "Timestamp": "2020-02-29 18:23:41.278"
    }
  }
}
```

XML

```
<event>
  <eventType>t0-check-in</eventType>
  <info>
    <Station>Odivelas</Station>
    <Timestamp>2020-02-29 18:23:41.278</Timestamp>
    <Token>t15345</Token>
  </info>
  <operator>Metro</operator>
</event>
```

Check-out in metro

JSON

```
{
  "event": {
    "eventType": "t0-check-out",
    "operator": "Metro",
    "info": {
      "Token": "t5431",
      "Station": "Alameda",
      "Timestamp": "2020-02-29 18:23:47.718"
    }
  }
}
```

XML

```
<event>
  <eventType>t0-check-out</eventType>
  <info>
    <Station>Alameda</Station>
```

```
<Timestamp>2020-02-29 18:23:47.718</Timestamp>
<Token>t5431</Token>
</info>
<operator>Metro</operator>
</event>
```

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

These events contain as main fields:

- **eventType** that says “t1”
- **operator** that contains the name of the operator
- **info** that contains information about a trip. Inside this there are fields with the token of the customer, price, and a timestamp.

Example:

JSON

```
{
  "event": {
    "eventType": "t1",
    "operator": "Uber",
    "info": {
      "Token": "t243",
      "Price": "20.63",
      "Timestamp": "2020-02-29 19:45:58.638"
    }
  }
}
```

XML

```
<event>
  <eventType>t1</eventType>
  <info>
    <Price>20.63</Price>
    <Timestamp>2020-02-29 19:45:58.638</Timestamp>
    <Token>t2</Token>
  </info>
  <operator>Uber</operator>
</event>
```

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

These events contain as main fields:

- **eventType** that says “t2”
- **operator** that contains the name of the operator

- **info** that contains information about a trip. Inside this field there are fields with the token of the customer, time spent with the vehicle, price of the ride and a timestamp.

Example:

JSON

```
{
  "event": {
    "eventType": "t2",
    "operator": "Gira",
    "info": {
      "Token": "t1",
      "Time": "3600",
      "Price": "12.60",
      "Timestamp": "2020-02-29 20: 57: 10.294"
    }
  }
}
```

XML

```
<event>
  <eventType>t2</eventType>
  <info>
    <Price>12.60</Price>
    <Time>3600</Time>
    <Timestamp>2020-02-29 20: 57: 10.294</Timestamp>
    <Token>t1</Token>
  </info>
  <operator>Gira</operator>
</event>
```

- **New User Events**

These events contain as main fields:

- **eventType** that says “new-user”
- **user** that contains information about the user. This field contains information about a user: id, email, plan type, first name, last name, and balance
- The plan type is used by the Payment Service to define how he should charge the customer:
 - **pre-paid** – costumers that load their account with a certain amount and are charged per each trip
 - **post-paid** – costumers who accumulate a debt and at the end of the month are debited that amount
 - **generalPass** – costumers who pay a monthly subscription to have a pass that can be used for every operator

- **passTN** – costumers who pay a monthly subscription to have a pass that can be used for every operator of type TN
- **combined_TX_TY** - costumers who pay a monthly subscription to have a pass that can be used for every operator of type TX and TY

Example:

JSON

```
{
  "event": {
    "eventType": "new-user",
    "info": {
      "id": "69c594cfdeeaedd220",
      "email": "user@gmail.com",
      "planType": "pre-paid",
      "firstName": "Paulo",
      "lastName": "Neves",
      "balance": "500"
    }
  }
}
```

XML

```
<event>
  <eventType>new-user</eventType>
  <info>
    <balance>500</balance>
    <email>user@gmail.com</email>
    <firstName>Paulo</firstName>
    <id>69c594cfdeeaedd220</id>
    <lastName>Neves</lastName>
    <planType>pre-paid</planType>
  </info>
</event>
```

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** that says “trip-cost”
- **info** that contains information about a trip and the user that made that trip. The info field contains: cost of the trip (null if operator type is t0), token of the user, plan type, operator name and timestamp.

Example:

JSON

```
{
  "event": {
    "eventType": "trip-cost",
    "info": {
      "cost": "23",
      "token": "69c594cfdeeaedd220",
      "planType": "pre-paid",
      "operatorName": "Uber",
      "timeStamp": "2020-02-29 20:57:10.294"
    }
  }
}
```

XML

```
<event>
  <eventType>trip-cost</eventType>
  <info>
    <cost>23</cost>
    <operatorName>Uber</operatorName>
    <planType>pre-paid</planType>
    <timeStamp>2020-02-29 20:57:10.294</timeStamp>
    <token>69c594cfdeeaedd220</token>
  </info>
</event>
```

Operator Management Service:

This service has as input TripCost Event, Operator events and Discount 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)

- Operator Events

These events contain as main fields:

- **eventType** that says “new-operator”
- **operator** that contains the name of the operator
- **info** that contains information about the operator. The info field contains the type of the operator and the base cost of their service (null if type of the operator is t1 or t2)

Example:

JSON

```
{
  "event": {
    "eventType": "new-operator",
    "operator": "Carris",
    "info": {
      "operatorType": "t0",
      "baseCost": "2.25"
    }
  }
}
```

XML

```
<event>
  <eventType>new-operator</eventType>
  <info>
    <baseCost>2.25</baseCost>
    <operatorType>t0</operatorType>
  </info>
  <operator>Carris</operator>
</event>
```

- **Discount Events**

These events contain as main fields:

- **eventType** that says “new-discount”
- **operator** the name of the operators that benefit from this discount
- **info** that contains information about the discount. The info field contains the name of the discount, discountId, value of the discount, period where the discount is applicable and a field containing which plan types does this discount apply to.

Example:

JSON

```
{
  "event": {
    "eventType": "new-discount",
    "operator": [
      {
        "operator": "Gira"
      }
    ],
    "info": {
```

```
    "name": "Dia do Ambiente",
    "discountId": "Gira-1-12",
    "value": "20",
    "beginAt": "2020-06-05 00:00:0.000",
    "endAt": "2020-06-05 23:59:59.999",
    "appliesToPlanType": [
      {
        "plan": "generalPass"
      },
      {
        "plan": "pre-paid"
      }
    ]
  }
}
```

XML

```
<event>
  <eventType>new-discount</eventType>
  <info>
    <appliesToPlanType>
      <element>
        <plan>generalPass</plan>
      </element>
      <element>
        <plan>pre-paid</plan>
      </element>
    </appliesToPlanType>
    <beginAt>2020-06-05 00:00:0.000</beginAt>
    <discountId>Gira-1-12</discountId>
    <endAt>2020-06-05 23:59:59.999</endAt>
    <name>Dia do Ambiente</name>
    <value>20</value>
  </info>
  <operator>
    <element>
      <operator>Gira</operator>
    </element>
  </operator>
</event>
```

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** that says “debit”
- **info** that contains information about the debit. The info field contains information about the amount to debit, plan type and the token of the user.

Example:

JSON

```
{
  "event": {
    "eventType": "debit",
    "info": {
      "token": "69c594cfdeeaedd220",
      "planType": "pre-paid",
      "amount": "20"
    }
  }
}
```

XML

```
<event>
  <eventType>debit</eventType>
  <info>
    <amount>20</amount>
    <planType>pre-paid</planType>
    <token>69c594cfdeeaedd220</token>
  </info>
</event>
```

4. 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 will not 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 three new topics: TripCosts, Debit and OperatorProvision

```
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
```

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

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.

The OperatorProvision topic will be used for the Operator, Service and Discount events, the operators will produce these types of events to update the Service Catalogue.

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.

The operator topics created in the first sprint remained with the same configurations.

5. Functional integration of the two microservices with the previous Kafka topics

We've decided to use an AWS Lambda function to implement the User Management Service and javax.jws.WebService library to implement Operator Management Service.

User Management Service pseudo code:

function handleRequest:

startService <- beginEvent

bd_connect <- connectToDatabase

consumer <- prepareConsumer

while(true):

consumerRecords <- consumer.poll()

for each record in consumerRecords **do**

message <- record.value

extractedEvent <- parse(message)

if extractedEvent != null and bd_connect == ok **do**

processEvent(extractedEvent)

else

throw exception

endif

endfor

consumer commit offset

endwhile

end

function processEvent:

```
eventType <- extractedEvent.type  
eventInfo <- extractedEvent.info
```

switch(eventType):

```
  case "t0-check-in":  
    insertT0InfoInDB <- eventInfo  
    break  
  case "t0-check-out":  
    insertT0InfoInDB <- eventInfo  
    break  
  case "t1":  
    insertT1InfoInDB <- eventInfo  
    break  
  case "t2":  
    insertT2InfoInDB <- eventInfo  
    break  
  case "new-user":  
    insertUserInDB <- eventInfo  
    break  
  default:  
    log  
    break
```

endswitch

end

Note: in the insertInfo functions a tripCost event will be sent after inserting data in the database

Operator Management Service pseudo code:

function startService:

```
bd_connect <- connectToDatabase
```

```
consumer <- prepareConsumer
```

```
producer <- prepareProducer
```

while(true):

```
  consumerRecords <- consumer.poll()
```

```
  for each record in consumerRecords do
```

```
    message <- record.value
```

```
    extractedEvent <- parse(message)
```

```
    if extractedEvent != null and bd_connect == ok do
```

```
      processEvent(extractedEvent)
```

```
    else
```

```
      throw exception
```

```
    endif
```

```
    endfor
    consumer commit offset
endwhile
end

function processEvent:
    eventType <- extractedEvent.type
    eventInfo <- extractedEvent.info

    switch(eventType):
        case "trip-cost":
            processTripCost <- eventInfo
            break
        case "new-operator":
            insertNewOperatorInDB <- eventInfo
            break
        case "new-discount":
            insertNewDiscountInDB <- eventInfo
            break
        default:
            log
            break
    endswitch
end

function processTripCost:
    if tripCost event cost == "null" do
        baseCost <- getOperatorBaseCost
    else
        baseCost <- tripCost event cost
    endif

    discount <- searchBestApplicableDiscount
    debitamount <- basecost * discount
    produceDebitEvent <- debitInfo
end
```

6. Implementation of the two microservices

UserDB Database configurations:

- Engine Type: MySQL
- Version: 5.7.22
- Template: Free Tier
- DB instance identifier: userdb
- DB instance size: db.t2.micro
- Storage type: General Purpose (SSD)
- Allocated storage: 20 GiB
- Enable storage autoscaling: true

- Maximum storage threshold: 1000 GiB
- Virtual Private Cloud (VPC): Default VPC
- Subnet group: default-vpc-8af6c4f0
- Publicly accessible: Yes
- VPC Security Groups: default and launch-kafka
- Availability zone: No preference
- Database port: 3306
- Database authentication options: Password Authentication

OperatorDB Database configurations:

- Engine Type: MySQL
- Version: 5.7.22
- Template: Free Tier
- DB instance identifier: OperatorDB
- DB instance size: db.t2.micro
- Storage type: General Purpose (SSD)
- Allocated storage: 20 GiB
- Enable storage autoscaling: true
- Maximum storage threshold: 1000 GiB
- Virtual Private Cloud (VPC): Default VPC
- Subnet group: default-vpc-8af6c4f0
- Publicly accessible: Yes
- VPC Security Groups: default and launch-kafka
- Availability zone: No preference
- Database port: 3306
- Database authentication options: Password Authentication

We also added an inbound rule in launch-kafka security group on port 3306 for the database connections and another on port 9997 for OperatorManagementService.

Relational Model of customerManagementDB

Info:

FK: Foreign key

__: Primary key

- userInfo(token, email, firstName, lastName, planType)
- userBalance(token, balance)
 - token: FK(userInfo)

We've decided to separate the balance from the userInfo table because the balance is going to be accessed by Payment Service and the rest of the tables are going to be accessed by User Management Service, this way we can provide some isolation.

- history(tripID, time_stamp, token, operatorName)
 - token: FK(userInfo)
- T0_History(tripID, time_stamp, station, isCheckIn)
 - tripID, time_stamp: FK(history)

- T1_History(tripID, time_stamp, price)
 - tripID, time_stamp: FK(history)
- T2_History(tripID, time_stamp, time, price)
 - tripID, time_stamp: FK(history)

Relational Model of operatorManagementDB

- operator(operatorName, operatorType, price)
- discount(discountId, discountName, value, beginAt, endAt)
 - operatorId: FK(service)
- planType(plan)
- discount_planType(discountId, plan)
 - plan: FK(planType)
- operator_discount(operatorName, discountId)
 - operatorName: FK(operator)
 - discountId: FK(discount)

7. Functional testing

- **We started by creating two producers, one for Metro trip events and another for GIRA trip events:**

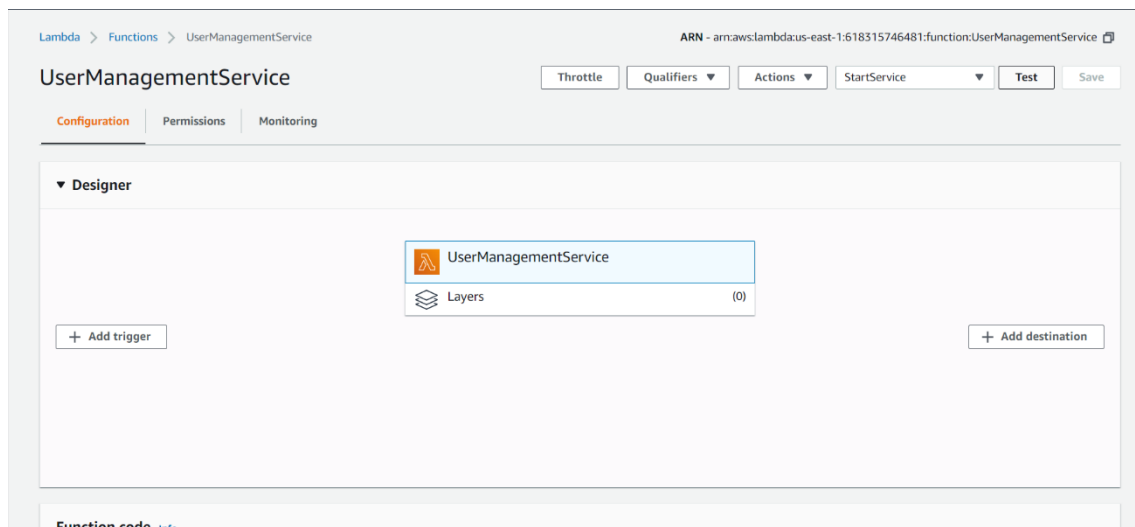
```
java -jar ProducerProvider2-0.0.1-SNAPSHOT.jar --provider-name Metro --broker-list
34.228.247.65:9093,34.228.247.65:9094,34.228.247.65:9095 --topic T0_Metro --token-list
jjgdgjs --throughput 200 --typeMessage JSON
```

```
java -jar ProducerProvider2-0.0.1-SNAPSHOT.jar --provider-name GIRA --broker-list
34.228.247.65:9093,34.228.247.65:9094,34.228.247.65:9095 --topic T2_GIRA --token-list
jjgdgjs --throughput 200 --typeMessage JSON
```

- **Then we started the UserManagementService with StartService test event:**

StartService event:

```
{
  "action": "begin"
}
```



- Then we started the `OperatorManagementService` with the command:
`mvn exec:java -D"exec.mainClass"="WebService.OperatorManagementServicePublisher"`

```
[ec2-user@ip-172-31-47-40 OperatorWebservice]$ mvn exec:java -D"exec.mainClass"="WebService.OperatorManagementServicePublisher"
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building OperatorManagementService 0.0.1-SNAPSHOT
[INFO]
[INFO] -----
[INFO] --- exec-maven-plugin:1.6.0:java (default-cli) @ OperatorManagementService ---
```

We also used an event that invoked the `startService` method to start this service:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:web="http://WebService/">
<soapenv:Header/>
<soapenv:Body>
<web:startService>
</web:startService>
</soapenv:Body>
</soapenv:Envelope>
```

- We also created a consumer for the `Debit` topic to represent the `PaymentService` with the command:

```
/usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server <Public_DNS>:9093,
<Public_DNS>:9094, <Public_DNS>:9095 --topic Debit
```

- At this point the system is completely running:
 - producers were sending events

IE – MEIC-A – 2019/20, 2ºsem
Sprint 2 – Grupo 3 – 86394 e 87671
Instituto Superior Técnico

```
waiting...2020-04-23 03:14:05.242
Fire-and-forget stopped.
This is the message to send = {"event":{"eventType":"t2", "operator":"GIRA", "info":{"Token": "jjdgdjs", "Time": "535", "Price": "2.2204995", "Timestamp": "2020-04-23 03:14:05.545" }}}
Sending new message to Kafka... with key=1587608045550
Sent...
waiting...2020-04-23 03:14:05.559
Fire-and-forget stopped.
This is the message to send = {"event":{"eventType":"t2", "operator":"GIRA", "info":{"Token": "jjdgdjs", "Time": "617", "Price": "28.591806", "Timestamp": "2020-04-23 03:14:05.862" }}}
Sending new message to Kafka... with key=1587608045865
Sent...
waiting...2020-04-23 03:14:05.871
Fire-and-forget stopped.
This is the message to send = {"event":{"eventType":"t2", "operator":"GIRA", "info":{"Token": "jjdgdjs", "Time": "485", "Price": "42.24518", "Timestamp": "2020-04-23 03:14:06.181" }}}
Sending new message to Kafka... with key=1587608046181
Sent...
waiting...2020-04-23 03:14:06.184
Fire-and-forget stopped.
This is the message to send = {"event":{"eventType":"t2", "operator":"GIRA", "info":{"Token": "jjdgdjs", "Time": "5", "Price": "50.72834", "Timestamp": "2020-04-23 03:14:06.486" }}}
Sending new message to Kafka... with key=1587608046491
Sent...
waiting...2020-04-23 03:14:06.5
Fire-and-forget stopped.
This is the message to send = {"event":{"eventType":"t2", "operator":"GIRA", "info":{"Token": "jjdgdjs", "Time": "887", "Price": "20.168304", "Timestamp": "2020-04-23 03:14:07.471" }}}
Sending new message to Kafka... with key=1587608046819
Sent...
waiting...2020-04-23 03:14:06.842
Fire-and-forget stopped.
This is the message to send = {"event":{"eventType":"t2", "operator":"GIRA", "info":{"Token": "jjdgdjs", "Time": "868", "Price": "95.716736", "Timestamp": "2020-04-23 03:14:07.15" }}}
Sending new message to Kafka... with key=1587608047154
Sent...
waiting...2020-04-23 03:14:07.167
Fire-and-forget stopped.
This is the message to send = {"event":{"eventType":"t2", "operator":"GIRA", "info":{"Token": "jjdgdjs", "Time": "356", "Price": "33.865147", "Timestamp": "2020-04-23 03:14:07.478" }}}
Sending new message to Kafka... with key=1587608047478
Sent...
waiting...2020-04-23 03:14:07.488
Fire-and-forget stopped.
This is the message to send = {"event":{"eventType":"t2", "operator":"GIRA", "info":{"Token": "jjdgdjs", "Time": "879", "Price": "30.616892", "Timestamp": "2020-04-23 03:14:07.794" }}}
Sending new message to Kafka... with key=1587608047795
Sent...
waiting...2020-04-23 03:14:07.799
```

```
waiting...2020-04-23 03:13:40.418
Fire-and-forget stopped.
This is the message to send = {"event":{"eventType":"t0-check-in", "operator":"Metro", "info":{"Token": "jjdgdjs", "Station": "overviolent", "Timestamp": "2020-04-23 03:13:40.719" }}}
Sending new message to Kafka... with key=1587608020719
Sent...
waiting...2020-04-23 03:13:40.72
Fire-and-forget stopped.
This is the message to send = {"event":{"eventType":"t0-check-out", "operator":"Metro", "info":{"Token": "jjdgdjs", "Station": "semitruth", "Timestamp": "2020-04-23 03:13:41.022" }}}
Sending new message to Kafka... with key=1587608021022
Sent...
waiting...2020-04-23 03:13:41.023
Fire-and-forget stopped.
This is the message to send = {"event":{"eventType":"t0-check-in", "operator":"Metro", "info":{"Token": "jjdgdjs", "Station": "dehort", "Timestamp": "2020-04-23 03:13:41.324" }}}
Sending new message to Kafka... with key=1587608021325
Sent...
waiting...2020-04-23 03:13:41.326
Fire-and-forget stopped.
This is the message to send = {"event":{"eventType":"t0-check-out", "operator":"Metro", "info":{"Token": "jjdgdjs", "Station": "subsystems", "Timestamp": "2020-04-23 03:13:41.627" }}}
Sending new message to Kafka... with key=1587608021628
Sent...
waiting...2020-04-23 03:13:41.629
Fire-and-forget stopped.
This is the message to send = {"event":{"eventType":"t0-check-in", "operator":"Metro", "info":{"Token": "jjdgdjs", "Station": "confiscates", "Timestamp": "2020-04-23 03:13:41.93" }}}
Sending new message to Kafka... with key=1587608021931
Sent...
waiting...2020-04-23 03:13:41.932
Fire-and-forget stopped.
This is the message to send = {"event":{"eventType":"t0-check-out", "operator":"Metro", "info":{"Token": "jjdgdjs", "Station": "orangite", "Timestamp": "2020-04-23 03:13:42.234" }}}
Sending new message to Kafka... with key=1587608022234
Sent...
waiting...2020-04-23 03:13:42.235
Fire-and-forget stopped.
This is the message to send = {"event":{"eventType":"t0-check-in", "operator":"Metro", "info":{"Token": "jjdgdjs", "Station": "cheeked", "Timestamp": "2020-04-23 03:13:42.536" }}}
Sending new message to Kafka... with key=1587608022537
Sent...
waiting...2020-04-23 03:13:42.538
Fire-and-forget stopped.
This is the message to send = {"event":{"eventType":"t0-check-out", "operator":"Metro", "info":{"Token": "jjdgdjs", "Station": "subsystems", "Timestamp": "2020-04-23 03:13:42.841" }}}
Sending new message to Kafka... with key=1587608022841
Sent...
waiting...2020-04-23 03:13:42.843
```

- UserManagementService was consuming events:

02:10:05	handleRequest: topic = T0_Metro, partition = 1, offset = 8652, customer = 1587607803323, message = {"event":{"eventType":"t0-check-in", "operator":"Metro", "info":{"Token": "jjdgdjs", "Station": "overviolent", "Timestamp": "2020-04-23 03:13:40.719" }}}}
02:10:05	parseEvent(operator):Metro
02:10:05	parseEvent(eventType):t0-check-in
02:10:05	parseEvent(info):{"Station":"stupefied", "Token":"jjdgdjs", "Timestamp":"2020-04-23 03:10:03.323"}

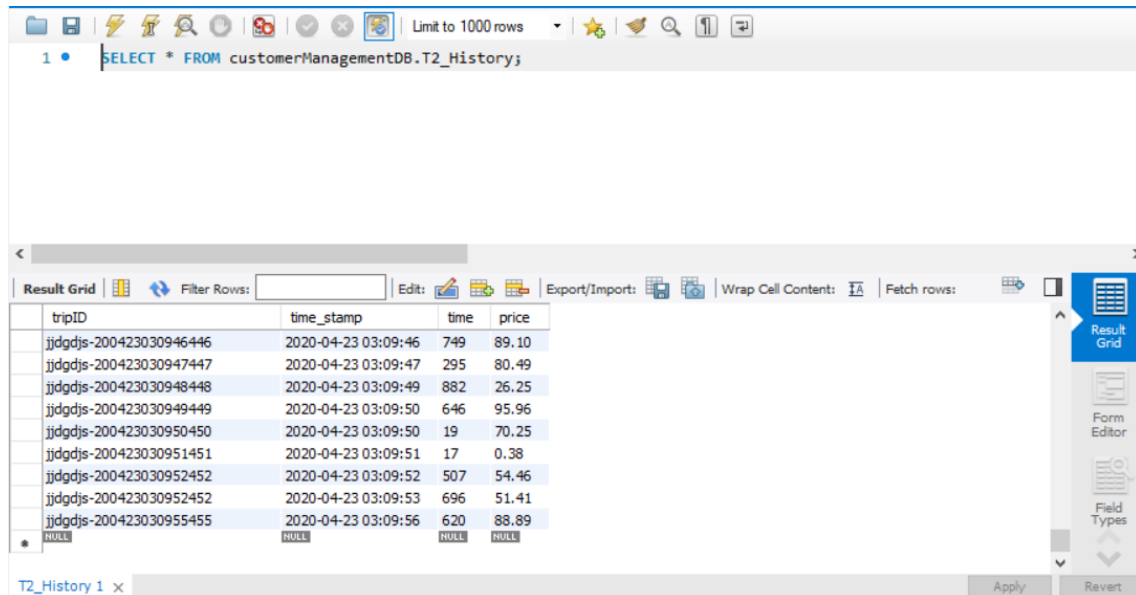
- OperatorManagementService was consuming the Trip events produced by UserManagementService:

```
produceDebitEvent : Sent -> {"event":{"eventType":"debit", "info":{"token": " jjgdjjs", "planType": "post-paid", "amount": "2.25" }}}
startService: records = 0
startService: records = 0
startService: records = 0
startService: records = 0
startService: records = 0
startService: records = 1
startService: topic = TripCosts, partition = 1, offset = 3112, customer = TripCostsKey, message = {"event":{"eventType":"trip-cost", "info":{"cost": "null", "token": " jjgdjjs", "planType": "post-paid", "operatorName": "Metro", "timestamp": "2020-04-23 03:14:56.679" }}}
processEvent(eventType):trip-cost
processEvent(info):{"timestamp":"2020-04-23 03:14:56.679","cost":"null","planType":"post-paid","operatorName":"Metro","token":" jjgdjjs"}
processTripCostEvent: basecost is null -> null
Got operator cost ->2.25
produceDebitEvent : Sent -> {"event":{"eventType":"debit", "info":{"token": " jjgdjjs", "planType": "post-paid", "amount": "2.25" }}}
startService: records = 0
startService: records = 0
startService: records = 0
startService: records = 1
startService: topic = TripCosts, partition = 1, offset = 3113, customer = TripCostsKey, message = {"event":{"eventType":"trip-cost", "info":{"cost": "92.340775", "token": " jjgdjjs", "planType": "post-paid", "operatorName": "GIRA", "timestamp": "2020-04-23 03:14:57.035" }}}
processEvent(eventType):trip-cost
processEvent(info):{"timestamp":"2020-04-23 03:14:57.035","cost":"92.340775","planType":"post-paid","operatorName":"GIRA","token":" jjgdjjs"}
processTripCostEvent: basecost is -> 92.340775
produceDebitEvent : Sent -> {"event":{"eventType":"debit", "info":{"token": " jjgdjjs", "planType": "post-paid", "amount": "92.340775" }}}

```

- OperatorManagementService was also creating entries in the history tables

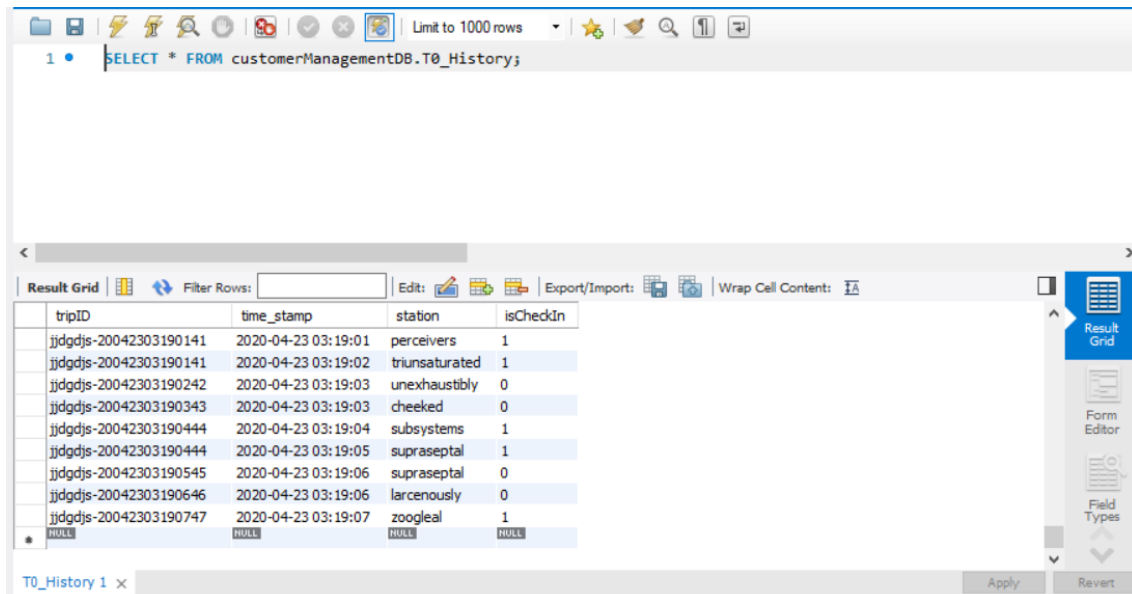
T2 specific History table



The screenshot shows a database management interface. At the top, a SQL query is entered: `SELECT * FROM customerManagementDB.T2_History;`. Below the query, a table grid displays the results. The table has four columns: `tripID`, `time_stamp`, `time`, and `price`. There are 11 rows of data, all with `tripID` values starting with `jjgdjjs-2004230309` and `time_stamp` values from `2020-04-23 03:09:46` to `2020-04-23 03:09:56`. The `time` and `price` columns contain numerical values. The last row has `NULL` values for `time` and `price`. The interface includes a toolbar with various icons and a sidebar with options like `Result Grid`, `Form Editor`, and `Field Types`.

tripID	time_stamp	time	price
jjgdjjs-200423030946446	2020-04-23 03:09:46	749	89.10
jjgdjjs-200423030947447	2020-04-23 03:09:47	295	80.49
jjgdjjs-200423030948448	2020-04-23 03:09:49	882	26.25
jjgdjjs-200423030949449	2020-04-23 03:09:50	646	95.96
jjgdjjs-200423030950450	2020-04-23 03:09:50	19	70.25
jjgdjjs-200423030951451	2020-04-23 03:09:51	17	0.38
jjgdjjs-200423030952452	2020-04-23 03:09:52	507	54.46
jjgdjjs-200423030952452	2020-04-23 03:09:53	696	51.41
jjgdjjs-200423030955455	2020-04-23 03:09:56	620	88.89
NULL	NULL	NULL	NULL

T0 specific History table



The screenshot shows a database query tool interface. At the top, a SQL query is entered: `SELECT * FROM customerManagementDB.T0_History;`. Below the query, a "Result Grid" displays the data. The grid has four columns: `tripID`, `time_stamp`, `station`, and `isCheckIn`. The data rows show various trip IDs, timestamps from 2020-04-23, and station names like `perceivers`, `triunsaturated`, `unexhaustibly`, `cheeked`, `subsystems`, `supraseptal`, `supraseptal`, `larcenously`, and `zoogaleal`. The `isCheckIn` column contains values 1, 1, 0, 0, 1, 1, 0, 0, and 1. A "NULL" row is also present at the bottom of the data grid. The interface includes a toolbar with icons for filtering, editing, and exporting, and a sidebar with options for the Result Grid, Form Editor, and Field Types.

tripID	time_stamp	station	isCheckIn
jjdgdjs-20042303190141	2020-04-23 03:19:01	perceivers	1
jjdgdjs-20042303190141	2020-04-23 03:19:02	triunsaturated	1
jjdgdjs-20042303190242	2020-04-23 03:19:03	unexhaustibly	0
jjdgdjs-20042303190343	2020-04-23 03:19:03	cheeked	0
jjdgdjs-20042303190444	2020-04-23 03:19:04	subsystems	1
jjdgdjs-20042303190444	2020-04-23 03:19:05	supraseptal	1
jjdgdjs-20042303190545	2020-04-23 03:19:06	supraseptal	0
jjdgdjs-20042303190646	2020-04-23 03:19:06	larcenously	0
jjdgdjs-20042303190747	2020-04-23 03:19:07	zoogaleal	1
NULL	NULL	NULL	NULL

- Finally, the consumer of the Debit Topic was consuming debit events produced by the OperatorManagementService

```
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "2.25" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "2.25" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "99.250916" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "89.62095" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "2.25" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "2.25" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "2.243358" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "86.66959" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "34.038265" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "2.25" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "2.2204995" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "42.24518" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "95.716736" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "2.25" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "14.057863" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "2.25" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "5.1779804" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "2.25" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "2.25" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "50.71078" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "61.239964" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "52.157562" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "2.25" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "59.071922" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "36.416973" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "4.723483" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "86.16592" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "2.25" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "40.818798" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "98.260345" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "2.25" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "79.90732" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "31.39118" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "2.25" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "11.476177" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "2.25" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "2.25" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "13.879097" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "9.296203" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "52.171505" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "2.25" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "16.61591" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "71.3151" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "61.945515" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "7.457906" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "2.25" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "85.449684" } } }
{ "event": { "eventType": "debit", "info": { "token": "jjdgdjs", "planType": "post-paid", "amount": "87.43312" } } }
```

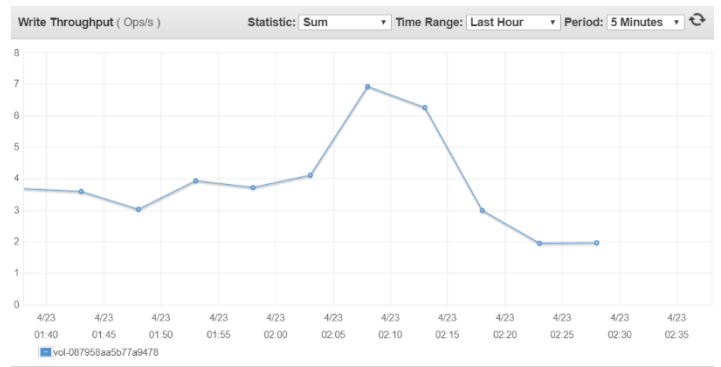
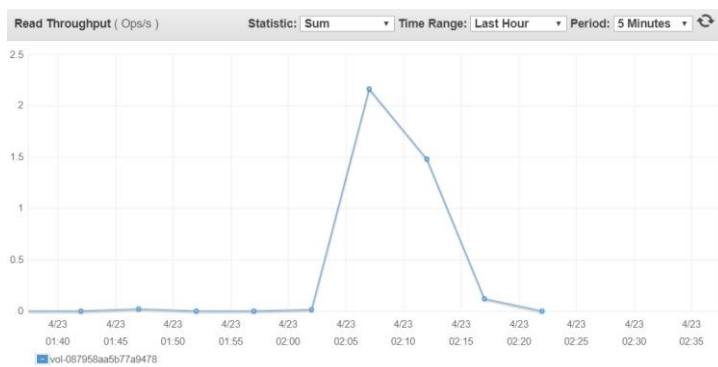
- In addition, we stopped one Kafka broker at 03:15 but the consuming and producing of events did not stop:

```
startService: topic = TripCosts, partition = 1, offset = 3215, customer = TripCostsKey, message = {"event":{"eventType":"trip-cost", "info":{"cost": "null", "token": " jjdgdjs", "planType":  
"post-paid", "operatorName": "Metro", "timeStamp": "2020-04-23 03:16:17.115" }}}  
processEvent(eventType):trip-cost  
processEvent(info):{"timeStamp":"2020-04-23 03:16:17.115","cost":"null","planType":"post-paid","operatorName":"Metro","token":" jjdgdjs"}  
processTripCostEvent: basecost is null -> null  
Got operator cost ->2.25  
produceDebitEvent : Sent -> {"event":{"eventType":"debit", "info":{"token": " jjdgdjs", "planType": "post-paid", "amount": "2.25" }}}  
startService: records = 0  
startService: records = 0  
startService: records = 0  
startService: records = 0  
startService: records = 1  
startService: topic = TripCosts, partition = 1, offset = 3216, customer = TripCostsKey, message = {"event":{"eventType":"trip-cost", "info":{"cost": "72.53035", "token": " jjdgdjs", "planTyp  
e": "post-paid", "operatorName": "GIRA", "timeStamp": "2020-04-23 03:16:17.549" }}}  
processEvent(eventType):trip-cost  
processEvent(info):{"timeStamp":"2020-04-23 03:16:17.549","cost":"72.53035","planType":"post-paid","operatorName":"GIRA","token":" jjdgdjs"}  
processTripCostEvent: basecost is -> 72.53035  
produceDebitEvent : Sent -> {"event":{"eventType":"debit", "info":{"token": " jjdgdjs", "planType": "post-paid", "amount": "72.53035" }}}  
startService: records = 0  
startService: records = 0
```

Analytics

Note: the graphs were made using a different timezone (-1 hour)

- Trip events started being sent at 03:05. We can notice that read throughput and write throughput start rising around that time. Read Latency starts increasing and maintains an exponential increase until the end of the test. Write Latency starts to drop but some minutes after starts rising.
- We stopped the Kafka Broker at 03:15. We can notice a drop in the write throughput and write latency around this time, but the system continued working with a smaller read throughput as well.



IE – MEIC-A – 2019/20, 2ºsem
Sprint 2 – Grupo 3 – 86394 e 87671
Instituto Superior Técnico

