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

Instituto Superior Técnico – MEIC-A

*Project Proposal*

1. **Definition of the mobility operators and respective messages**

Mobility as a Service is an analogy of the usual Software as a Service model popularized by the Cloud for the software industry. The idea is the same, people will use the transport network they see fitting better their needs, be it the Public Transport Operators like Metro or Buses, Taxis or the new players such as Uber or Cabify and all other innovative alternatives for personal transportation like rental bike, scooters, motorcycles, etc.

The innovation is the seamless use of all of them without the usual difficulties of different ticketing and payments systems, and the negative incentive for such use due to incompatibilities between cards, apps, tariffs, monthly subscriptions, etc.

The main idea of Maas is that one can take any transportation system and in the background his usage is being registered and one will pay for mobility according to the schema that best suits his needs.

As part of the proposal, we aim to integrate three different transportation operators, that provide different means of transportation. We chose the different operators: **Metro, Uber and Gira**.

The innovation is the seamless use of all of them without the usual difficulties of different ticketing and payments systems, and the negative incentive for such use due to incompatibilities between cards, apps, tariffs, monthly subscriptions, etc.

Examples of messages received in a Kafka broker (sent in JSON format):

{“Metro”: {“CheckIn”: {“Token”: “t1”, “Station”: “Odivelas”, “Timestamp”: “2020-02-29 18:23:41.278”}}}

{“Metro”: {“CheckOut”: {“Token”: “t1”, “Station”: “Alameda”, “Timestamp”: “2020-02-29 18:23:47.718”}}}

{“Uber”: {“Usage”: {“Token”: “t2”, “Price”: “70.51901”, “Timestamp”: “2020-02-29 19:45:58.638”}}}

{“Gira”: {“Usage”: {“Token”: “t1.4”, “Distance”: “13.553344”, “Timestamp”: “2020-02-29 20:57:10.294”}}}

{“Discounts”: {“Value”: {“Token”: “t3”, “Discount Value”:”1”}}}

1. **Definition of the event queueing integration: Topics, Partitions**

Each Operator has a Topic, and each Topic has 3 partitions.

The partitions in the log allow the log to scale beyond a size that will fit on a single server. Each individual partition must fit on the servers that host it, but a topic may have many partitions so it can handle an arbitrary amount of data and they act as the unit of parallelism.

**Metro**: Topic t0

**Uber**: Topic t1

**Gira**: Topic t2

And we created another Topic for **Discounts.**

1. **Definition of the fault tolerance requirements for Kafka**

Since there are 3 Operators, there will be 3 **brokers**.

The **replication factor** controls how many servers will replicate each message that is written. We set the Replication Factor to 3, so up to 2 servers can fail before we lose access to your data.

There is one Zookeeper for each Topic.

The Kafka cluster durably persists all published records using a configurable retention period. The **retention period** is set for 48 hours, so for the two days after the record is published, it is available for consumption, after which it will be discarded to free up space.

We need to have 2 **Consumer Groups**, one for Revenue Service and other for Users Service, since they must read messages from every operator. If all the consumer instances have the same consumer group, then the records will effectively be load balanced over the consumer instances.

1. **Kafka installation**

We followed these steps from the tutorials for practical classes:

P2. B. Creating and launching an AWS EC2 instance

P2. C. Access the AWS EC2 instance using PuTTY

P2. E. Access the AWS EC2 instance using FileZilla

P2. F. Install Kafka in the AWS EC2 instance

Creating 3 brokers:

**cp /usr/local/kafka/config/server.properties /usr/local/kafka/config/server-1.properties**

**cp /usr/local/kafka/config/server.properties /usr/local/kafka/config/server-2.properties**

**cp /usr/local/kafka/config/server.properties /usr/local/kafka/config/server-3.properties**

|  |  |  |
| --- | --- | --- |
| Broker-1:  config/server-1.properties:  **broker.id=0**  **listeners=PLAINTEXT://*<YourIP\_or\_DNS>*:9093**  **offsets.topic.replication.factor=3**  **transaction.state.log.replication.factor=3**  **transaction.state.log.min.isr=3**  **log.dir=/tmp/kafka-logs-0**  **log.retention.hours = 48** | Broker-2:  config/server-2.properties:  **broker.id=0**  **listeners=PLAINTEXT://*<Public DNS>*:9094**  **offsets.topic.replication.factor=3**  **transaction.state.log.replication.factor=3**  **transaction.state.log.min.isr=3**  **log.dir=/tmp/kafka-logs-1**  **log.retention.hours = 48** | Broker-3:  config/server-3.properties:  **broker.id=0**  **listeners=PLAINTEXT://*<Public DNS>*:9095**  **offsets.topic.replication.factor=3**  **transaction.state.log.replication.factor=3**  **transaction.state.log.min.isr=3**  **log.dir=/tmp/kafka-logs-2**  **log.retention.hours = 48** |

We added these commands to .bash\_profile:

**sudo /usr/local/kafka/bin/kafka-server-start.sh -daemon /usr/local/kafka/config/server-1.properties**

**sudo /usr/local/kafka/bin/kafka-server-start.sh -daemon /usr/local/kafka/config/server-2.properties**

**sudo /usr/local/kafka/bin/kafka-server-start.sh -daemon /usr/local/kafka/config/server-3.properties**

Opened the in-bound ports 9093, 9094 and 9095 in the AWS EC2 console.

Created the 4 Topics:

**sudo /usr/local/kafka/bin/kafka-topics.sh --create --zookeeper *<Public DNS>*:2181 -replication-factor 3 --partitions 3 --topic T0\_METRO**

**sudo /usr/local/kafka/bin/kafka-topics.sh --create --zookeeper *<Public DNS>*:2181 -replication-factor 3 --partitions 3 --topic T1\_UBER**

**sudo /usr/local/kafka/bin/kafka-topics.sh --create --zookeeper *<Public DNS>*:2181 -replication-factor 3 --partitions 3 --topic T2\_GIRA**

**sudo /usr/local/kafka/bin/kafka-topics.sh --create --zookeeper *<Public DNS>*:2181 -replication-factor 3 --partitions 3 --topic Discounts**

And verified**: ps -ef |grep java |grep server**

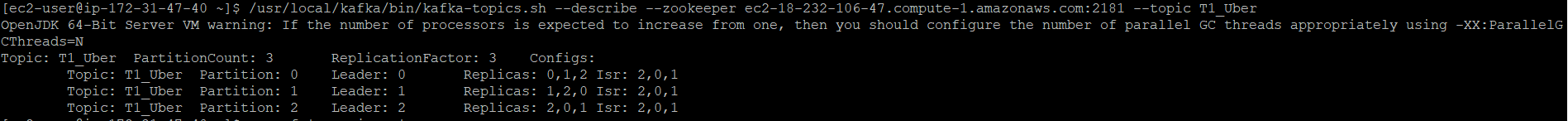
**sudo /usr/local/kafka/bin/kafka-topics.sh --list --zookeeper *<Public DNS>*:2181**

**sudo /usr/local/kafka/bin/kafka-topics.sh --describe --topic Topic\_Name --zookeeper *<Public DNS>*:2181**

1. **Kafka parametrization**
2. **Test of the integration using applications for event generation**

We decided to make a failure test for our Maas Operator using the T1\_Uber topic.

* We started by checking that all the 3 brokers were well configured and who were the leaders of the partitions of this topic:



* Then we created a producer with the command:

java -jar MaaSMessageTaxiGenerator.jar --broker-list <Public\_DNS>:9093, <Public\_DNS>:9094, <Public\_DNS>:9095 --topic T1\_Uber --token-list jkdjdjs --throughput 2000 --typeMessage JSON

(We decided to use a big value for the throughput to test if our Maas Operator could handle a big load)

* Then we created a consumer with the command:

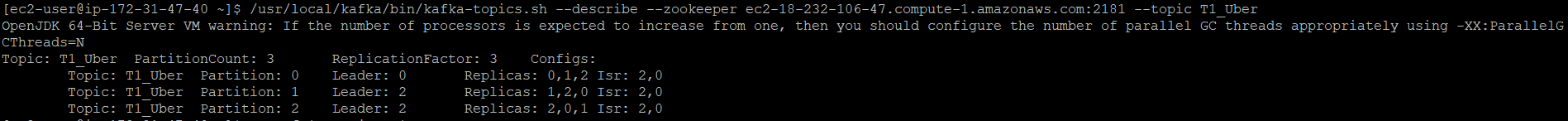
/usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server <Public\_DNS>:9093, <Public\_DNS>:9094, <Public\_DNS>:9095 --topic T1\_Uber --group g1

* After some time, we ran this command to check the PID of the brokers:

ps -ef |grep java |grep server

* Then we stopped broker 2:

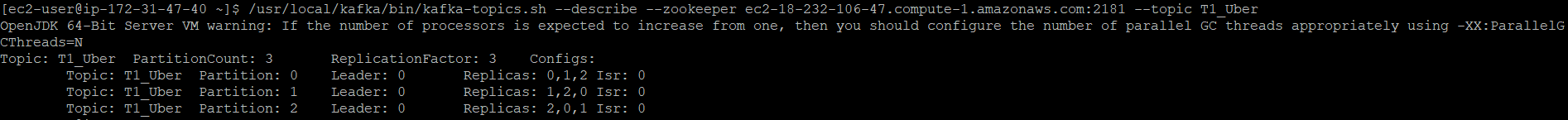
sudo kill <Broker\_2\_PID>

* And checked the change in the leaders:

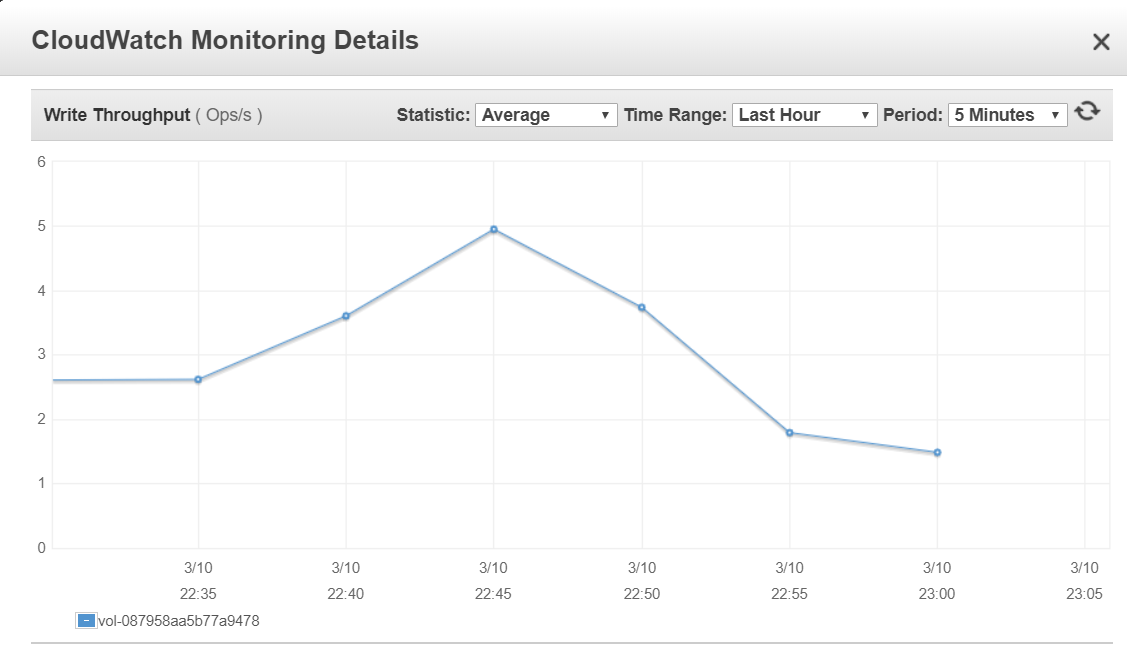
The consumer continued to receive the messages

* After some time, we stopped broker 3:

sudo kill <Broker\_3\_PID>

* And checked the change in the leaders:

**Write throughput**



1.4866666

2.62

3.60666

1.79333

3.74

4.95333333

22:35 – Messages started being sent

22:45 – Broker 2 stopped

22:50 – Broker 3 stopped

23:00 – We stopped the producer

By analyzing this graph, we can notice a drop of the throughput when the first broker went down (4.95333 to 3,74 ≈ 25%). When the second broker went down there was a drop as well (3.74 to 1.4866 ≈ 60%). Although there were drops in the throughput the consumer continued to receive all the messages so we can conclude that our Maas Operator can tolerate two faults maintaining the correct functioning of the system.