# 1 Implementation

In this section we will describe our project work. We will start by presenting the working environment . Then we move to talk about each part of our project , presenting screenshots and giving the examples of each Implementation.

## 1 Work environnement

This part presents the Hardware and Software environment available for this project.

### 1.1 Hardware environnement

In our project we developed our application using a computer with the following characteristics :

• Process Intel Core i5 4310M .

• Memory RAM: 16 GB .

• Space Discs: 500 GB.

## 2 Software environnement

We’ve used for our project the following Software :

• VMware Workstation : Our application is deployed under VMware Workstation virtualization platform as a set of decoupled service. Each service per container.

• Centos 7 : Our application operating system is Centos 7 carring Docker as demon.

• Docker : is our main virtualization tool and our containers manager.

• Intellij idea : is our application Back End IDE , we used it for Scala and java (spring) developement.

• Visual Studio Code : is our application Front End IDE , we used it for angular 2 and webpack (TypeScript developement).

## 3 Realized work

### 3.1 Environnement preparation

Our application require some containers preparation. The following list contain all needed containers.

• Kafka container : used for collecting data and stream them through a message queue. This container does not existe in dockerhub repository so we should Dockerize it(Dockerize kafka means create a container that contain kafka)

• Zookeeper container : used to save brokers address and messages offsets . this container exists in docker hub.

• Scala container : used to build the Hadoop container.This container does not existe in dockerhub repository so we should Dockerize it

• Hadoop container : used to build the spark container and provide HDFS persistance and YARN (ressource manager) for the spark jobs. This container does not existe with the Hadoop 2.7 and centos 7 version, so we should Dockerize it

• Spark container : used to to consume the data from kafka , process it then feed it to the serving layer (persisting results in cassandra).

• Cassandra container : used as a serving layer . this container exists in docker hub , we will be using the 2.2 cassandra version.

• Zeppelin container : used as a notebook to interpret Cassandra and spark then display the results in a dashboard.

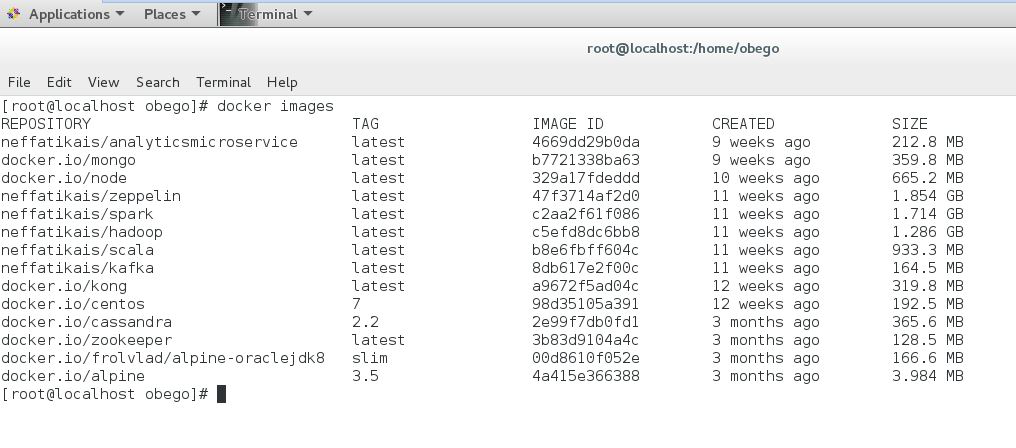
• Spring boot containers : we need some containers that carry each one a microservice. This containers just need to have a JVM.

• Mongodb containers : each service is linked to a mongodb container. this containers exists in dockerhub.

• FrontEnd container : container for the angular dashboard.

#### 3.1.1 Dockernize Tools

In this section we present the process of dockernizing our containers .All installation setps are written in Dockerfiles and each container has an OS kernel. We will be using alpine and centos 7. The next figure show all our docker images:

 [filiales et clients de Sofrecom.]Docker images

**Dockernize Kafka**

To dockernize this container, we will start by putting a base OS version wich is alpine linux . The next step is the Jdk installation. After fixing all paths we move to install kafka.

 [filiales et clients de Sofrecom.]Kafka container

**Dockernize Scala**

To dockernize this container, we will start by putting a base OS version wich is centos 7 . The next step is the Jdk installation. After fixing all paths we move to install Scala.

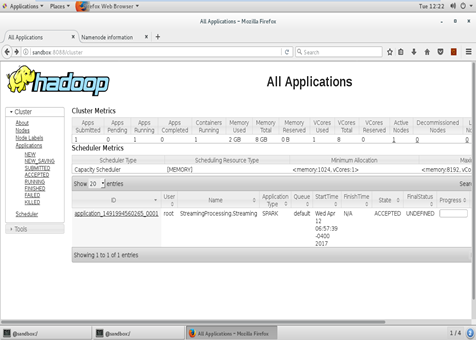
 [filiales et clients de Sofrecom.]Scala container

**Dockernize Hadoop**

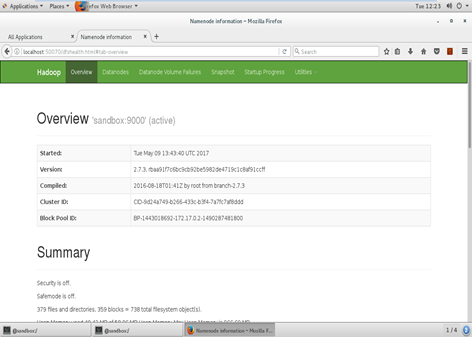
To dockernize this container, we will start from the previous Scala container. The next step is the Hadoop installation which takes too much time due to the big configuration amount.

 [filiales et clients de Sofrecom.]Hadoop Container

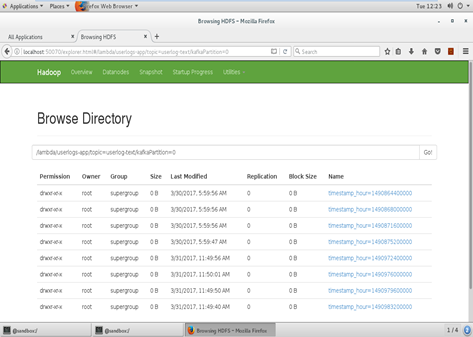
After dockernizing Hadoop 2.7 we are able now to see some Doshboard. Hadoop YARN ResourceManager HTTP UI :

 [filiales et clients de Sofrecom.]Hadoop YARN ResourceManager HTTP UI

Hadoop HDFS NameNode HTTP UI :

 [filiales et clients de Sofrecom.]Hadoop HDFS NameNode HTTP UI

We can Check Files existance through the HDFS NameNode UI :

 [filiales et clients de Sofrecom.]Browser files HDFS

**Dockernize Spark**

This container is the main container. We built the last two ones for this step. This container is based on the haddop one . Spark uses HDFS to persist data and uses YARN as a ResourceManager.This container require much more then the last one in terms of configuration. We chose to built spark on yarn to benefit the real sparks power.

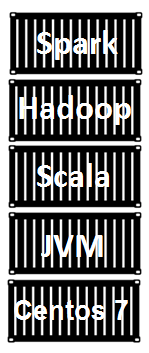
So spark present in totaly 3 deployment mode :

Standalone deployment: With the standalone deployment one we can statically allocate resources on all or a subset of machines in a Hadoop cluster and run Spark side by side with Hadoop MR. The user can then run arbitrary Spark jobs on his HDFS data. Its simplicity makes this the deployment of choice for many Hadoop 1.x users.

Hadoop Yarn deployment: Hadoop users who have already deployed or are planning to deploy Hadoop Yarn can simply run Spark on YARN without any pre- installation or administrative access required. This allows users to easily integrate Spark in their Hadoop stack and take advantage of the full power of Spark, as well as of other components running on top of Spark.

Spark In MapReduce (SIMR): For the Hadoop users that are not running YARN yet, another option, in addition to the standalone deployment, is to use SIMR to launch Spark jobs inside MapReduce. With SIMR, users can start experimenting with Spark and use its shell within a couple of minutes after downloading it! This tremendously lowers the barrier of deployment, and lets virtually everyone play with Spark.

*Notice: Spark is an in-Memory data processing framwork so when trying to configure it make sure you put the right RM consumption.*

 [filiales et clients de Sofrecom.]Spark container .

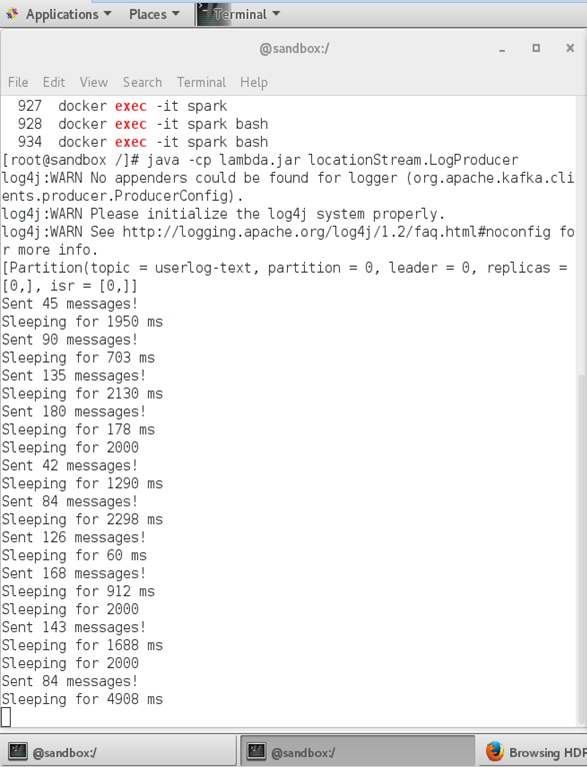
**Dockernize Zeppelin**

Zeppelin is a notebook that allow as to interpret spark or cassandra . Zeppelin container is built on a an alpine linux operating system.

 [filiales et clients de Sofrecom.]Zeppelin container.

### 3.2 Log Producer

The log producer is a Scala program that act as a producer for our kafka OSS. We ve used this log producer to perform a real data stream flow.

 [filiales et clients de Sofrecom.]Scala application log producer.

### 3.3 Spark Jobs

Applying the lambda architecture require running at least two jobs, in our case we ve created three spark jobs using Scala :

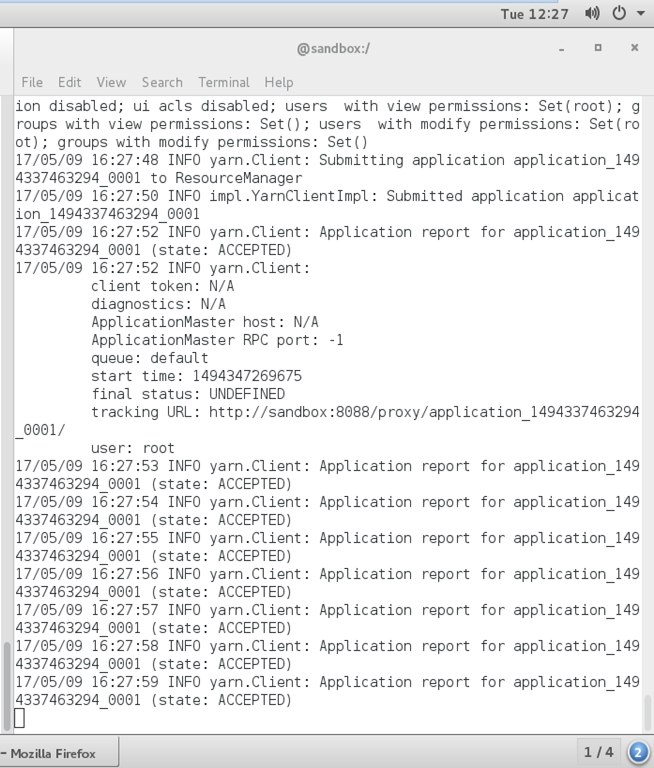
• BatchSaveTOHDFS: This job persists realtime data to HDFS. It consumes data flow from Kafka and persist it to HDFS continuously.

• StreamingProcessing: This job is the realtime engine, it consumes data from Kafka and process it and gives results in realtime.

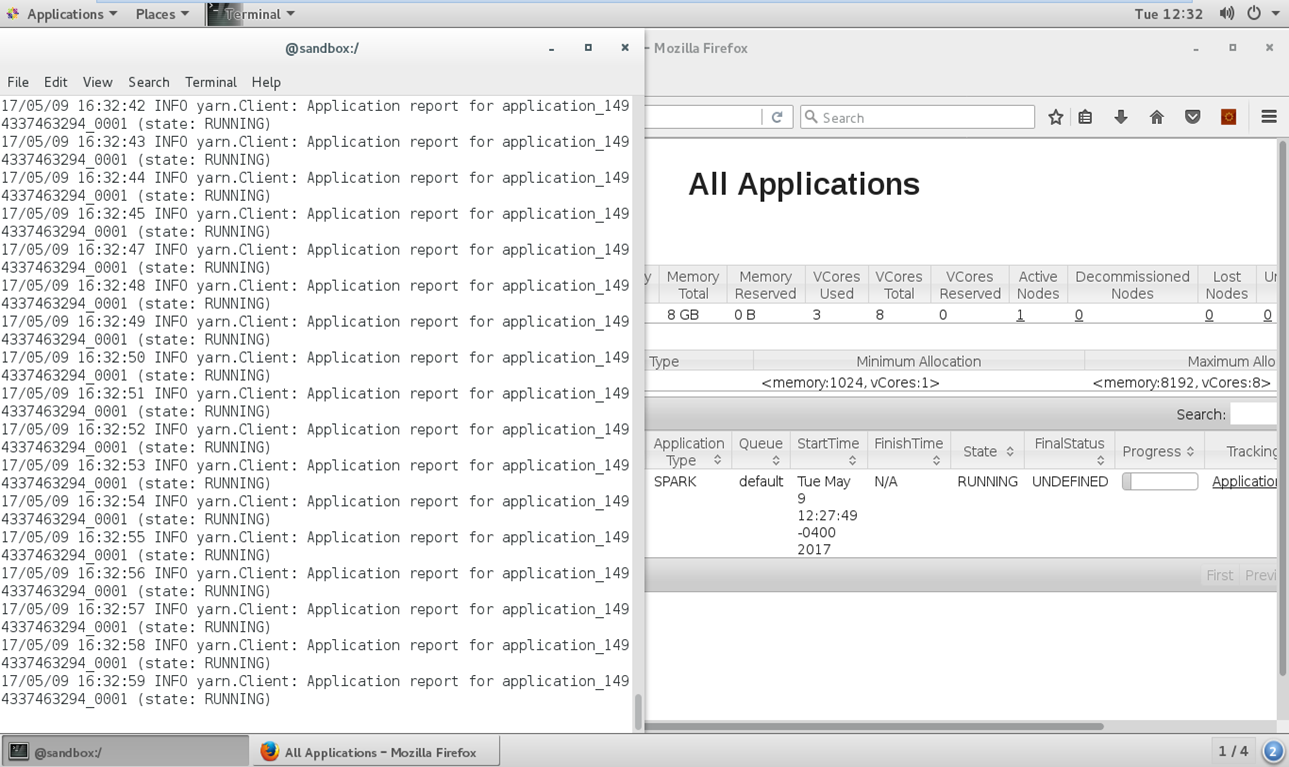
• BatchProcessing: This job is the batch engine, it processes the data already persisted in HDFS and gives results each 8 hours.

#### 3.3.1 BatchProcessing

The next figure show the BatchProcessing job in it’s accepted status:

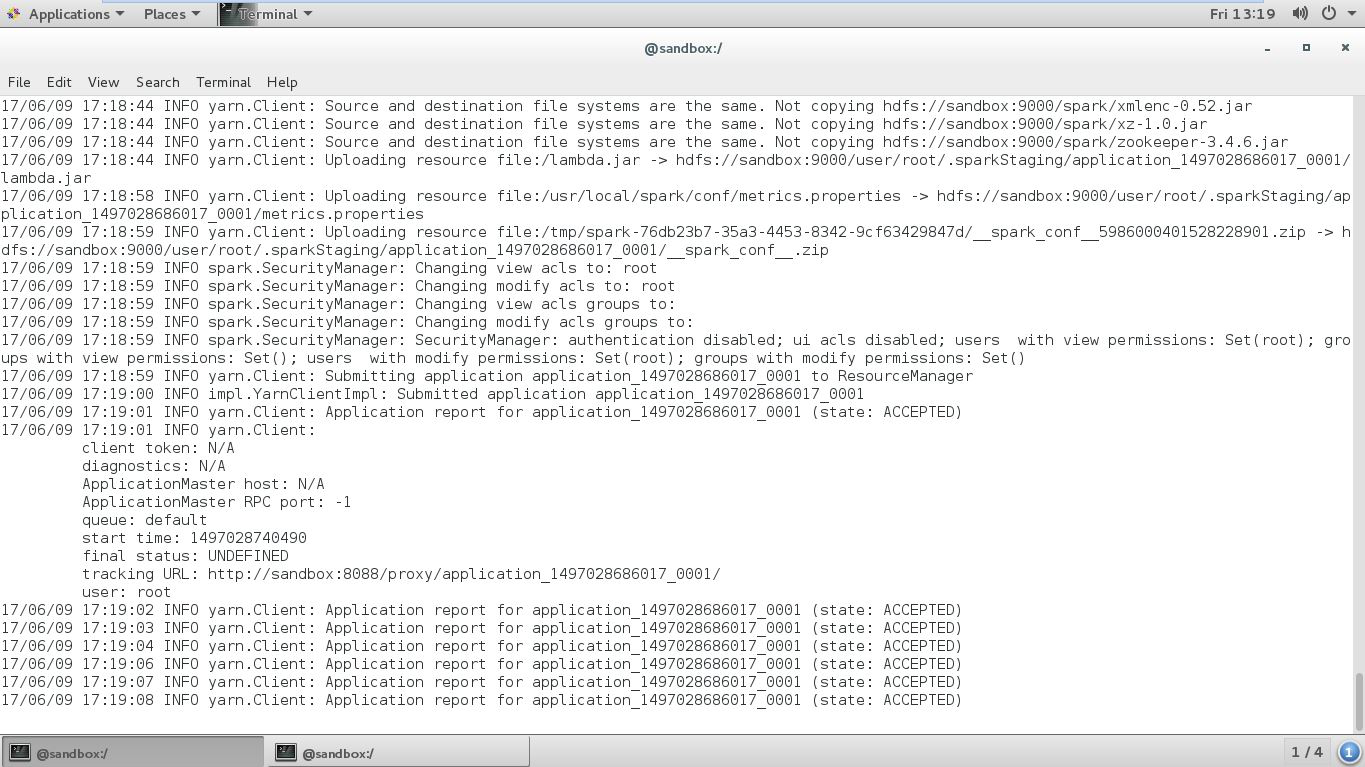
 [filiales et clients de Sofrecom.]BatchProcessing spark job in accepted status.

The next figure show the BatchProcessing job in it’s running status:

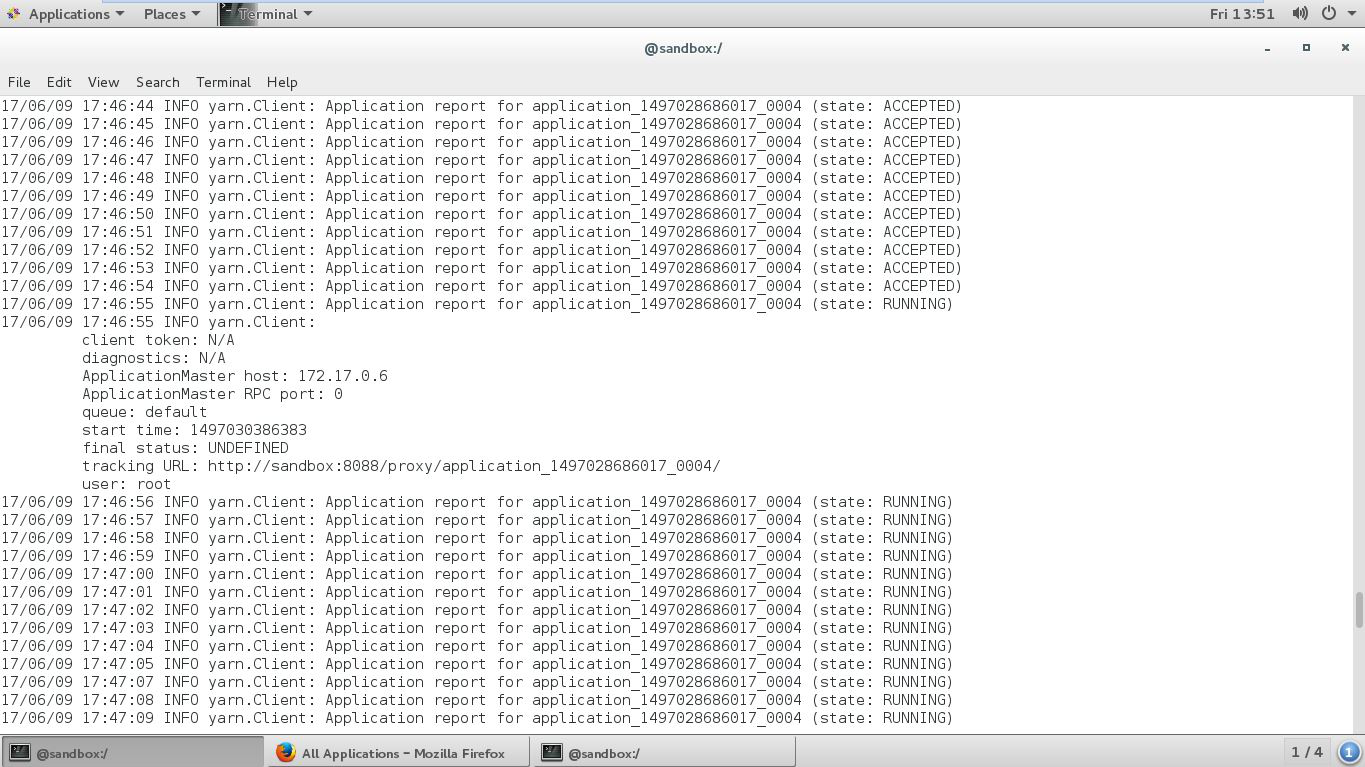
 [filiales et clients de Sofrecom.]BatchProcessing spark job in running status.

#### 3.3.2 StreamingProcessing

The next figure show the StreamingProcessing job in it’s accepted status:

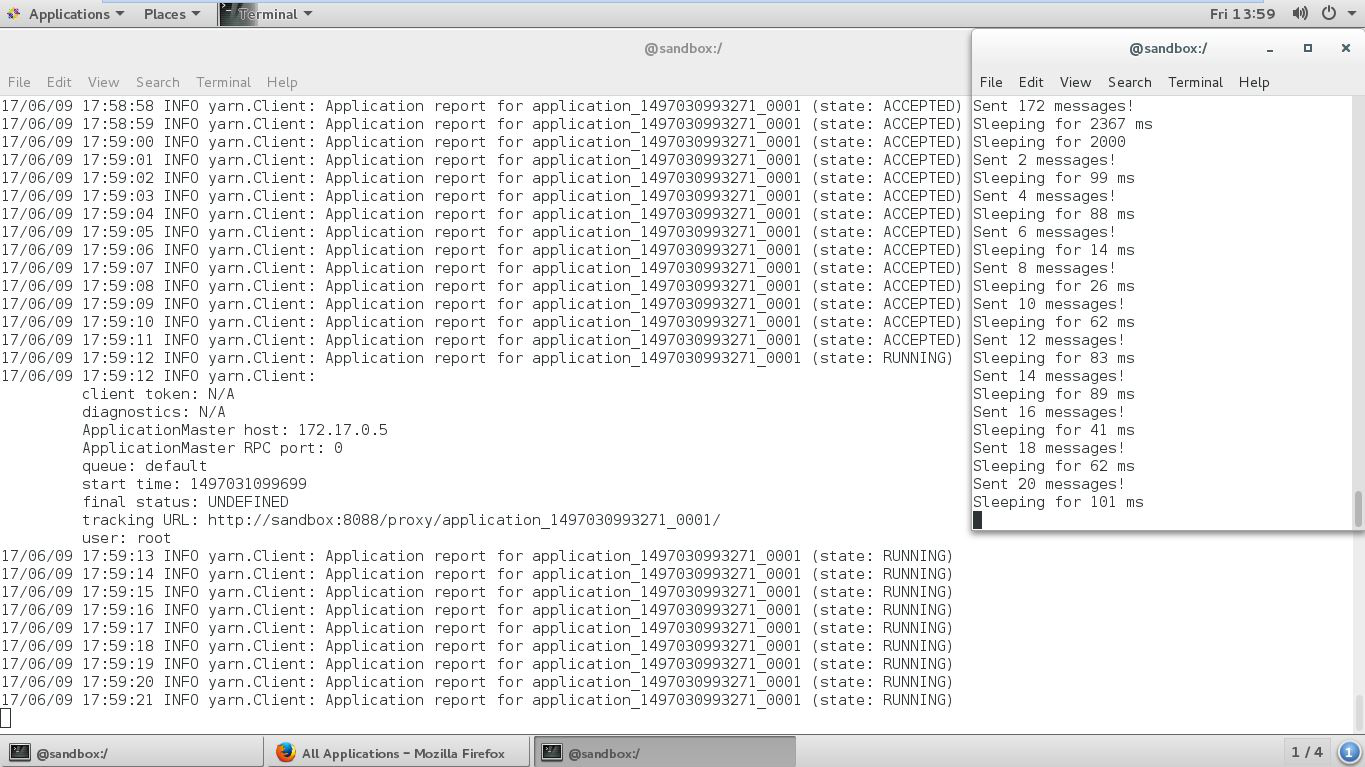
 [filiales et clients de Sofrecom.]StreamingProcessing spark job in accepted status.

The next figure show the BatchProcessing job in it’s running status:

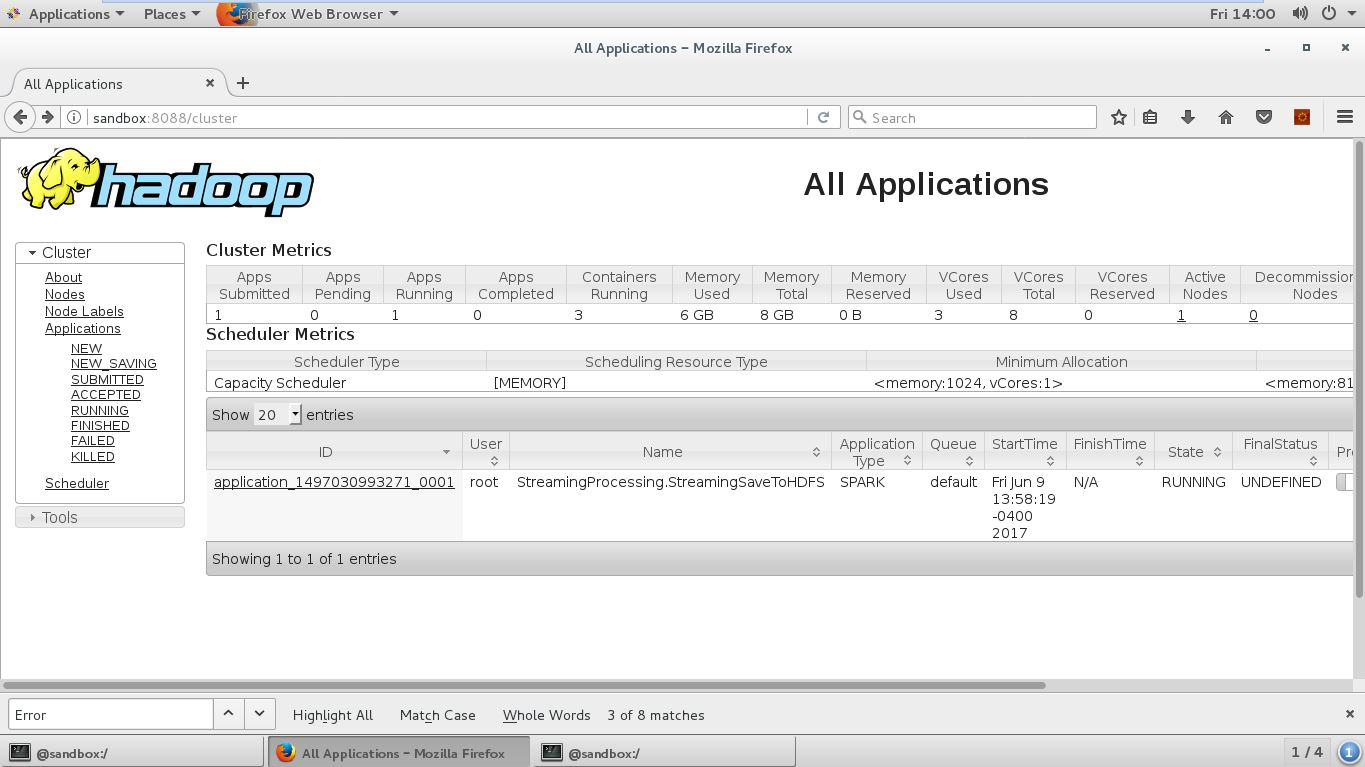
 [filiales et clients de Sofrecom.]StreamingProcessing spark job in running status.

#### 3.3.3 BatchSaveTOHDFS

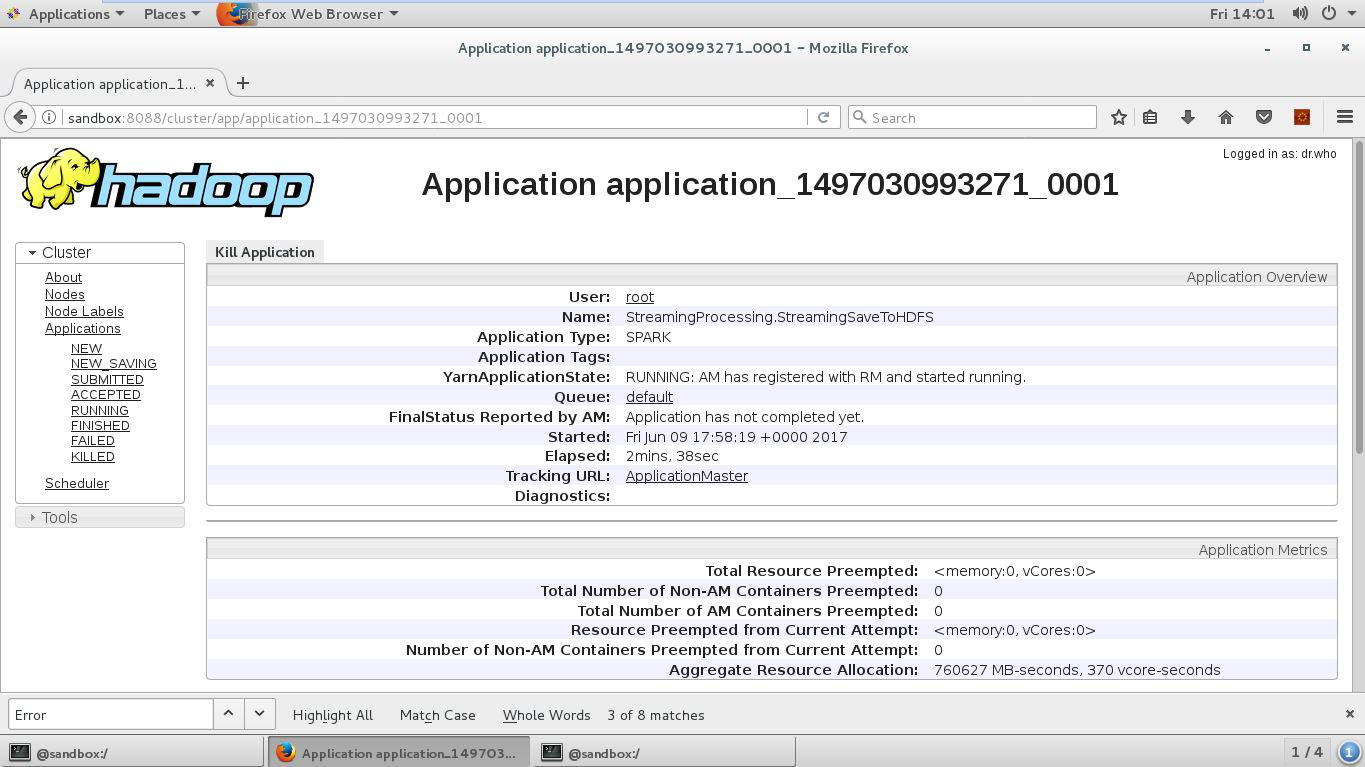
The next figure show the BatchSaveToHDFS job in it’s accepted and running status:

 [filiales et clients de Sofrecom.]BatchSaveToHDFS spark job in accepted status.

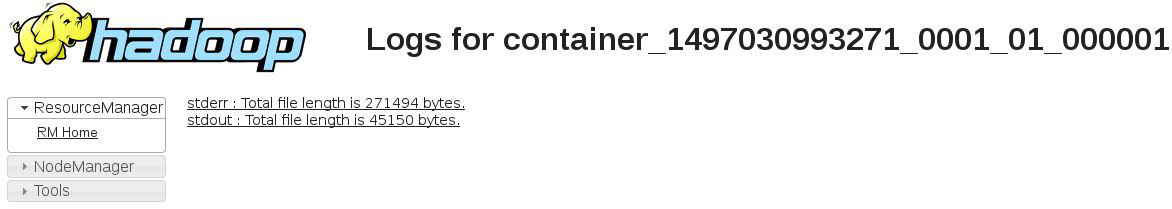
The next figure show the job Hadoop Yarn UI status:

 [filiales et clients de Sofrecom.]Running job status Hadoop Yarn UI .

The next figure show the Running application Master:

 [filiales et clients de Sofrecom.]Application Master dashboard .

The next figure show the running job log:

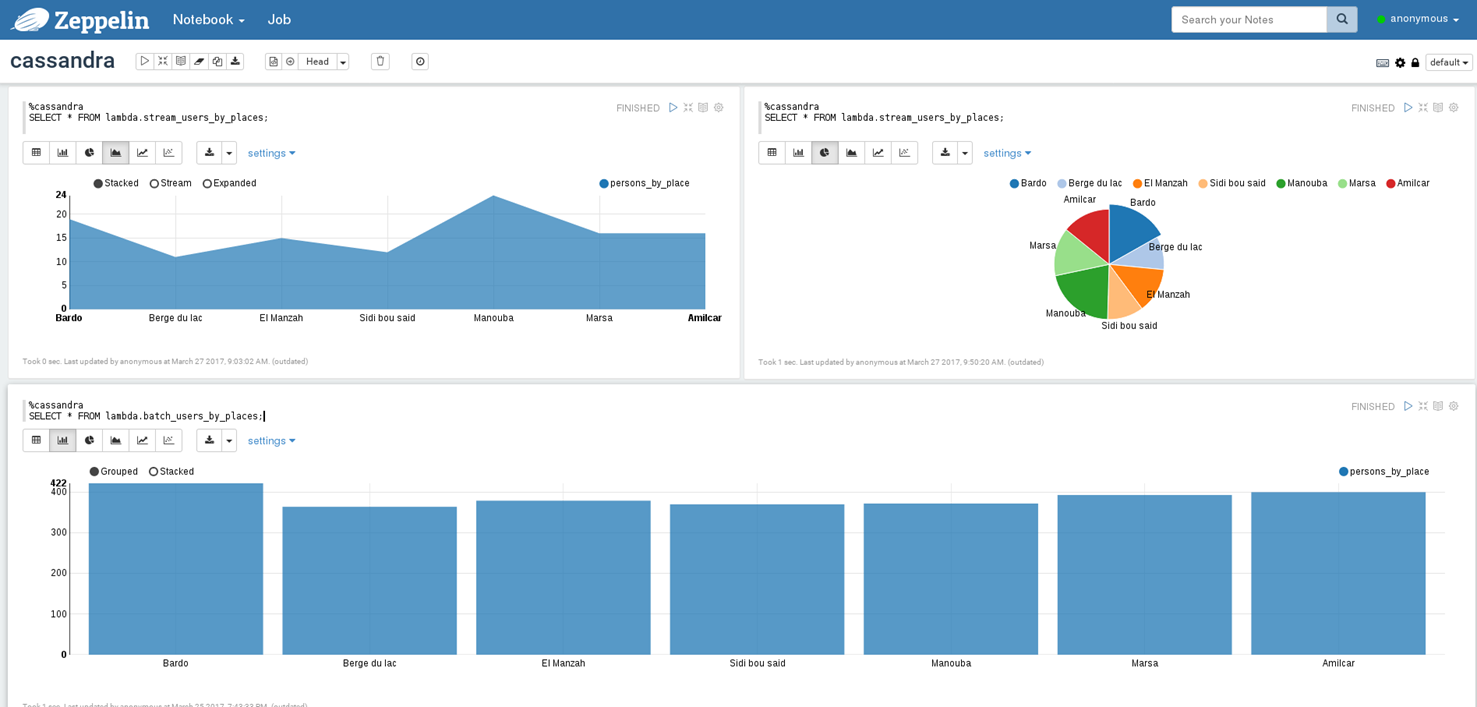
 [filiales et clients de Sofrecom.]Job log .

The next figure show the running job output:

 [filiales et clients de Sofrecom.]Job log Resualts .

### 3.4 Show Resualts on Apache Zeppelin

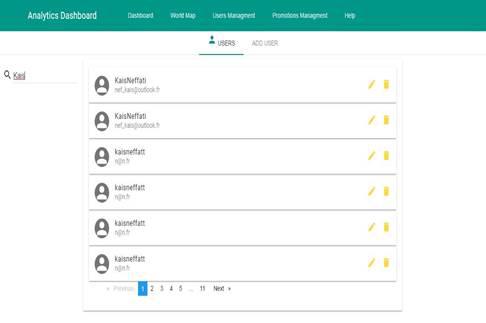
The next figure show Analytics results displayed on Zeppelin dashboard :

 [filiales et clients de Sofrecom.]Zeppelin dashboard.

### 3.5 Account Managment

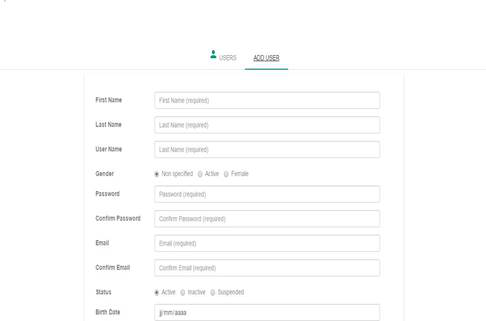
#### 3.5.1 Accounts

The next figure show the list Accounts :

 [filiales et clients de Sofrecom.]Account list.

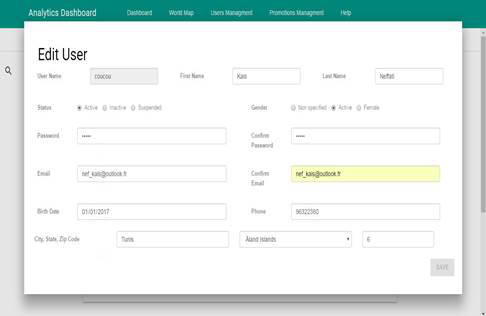
#### 3.5.2 Account add

The next figure show the form add account :

 [filiales et clients de Sofrecom.]Add account.

#### 3.5.3 Account Update

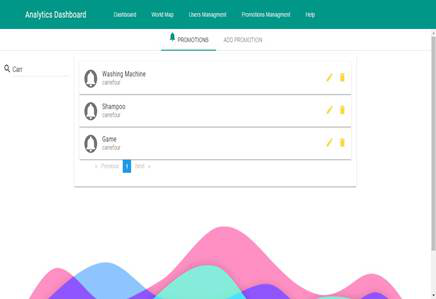
The next figure show the form update account :

 [filiales et clients de Sofrecom.]Update account.

### 3.6 Promotion Managment

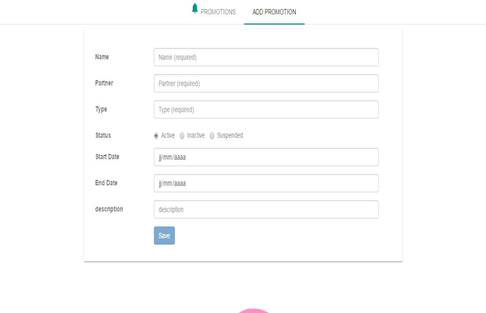
#### 3.6.1 Promotions

The next figure show the list Promotions :

 [filiales et clients de Sofrecom.]Promotions list.

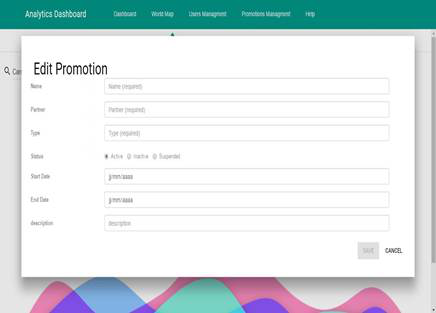
#### 3.6.2 Promotion add

The next figure show the form add Promotion :

 [filiales et clients de Sofrecom.]Add promotions.

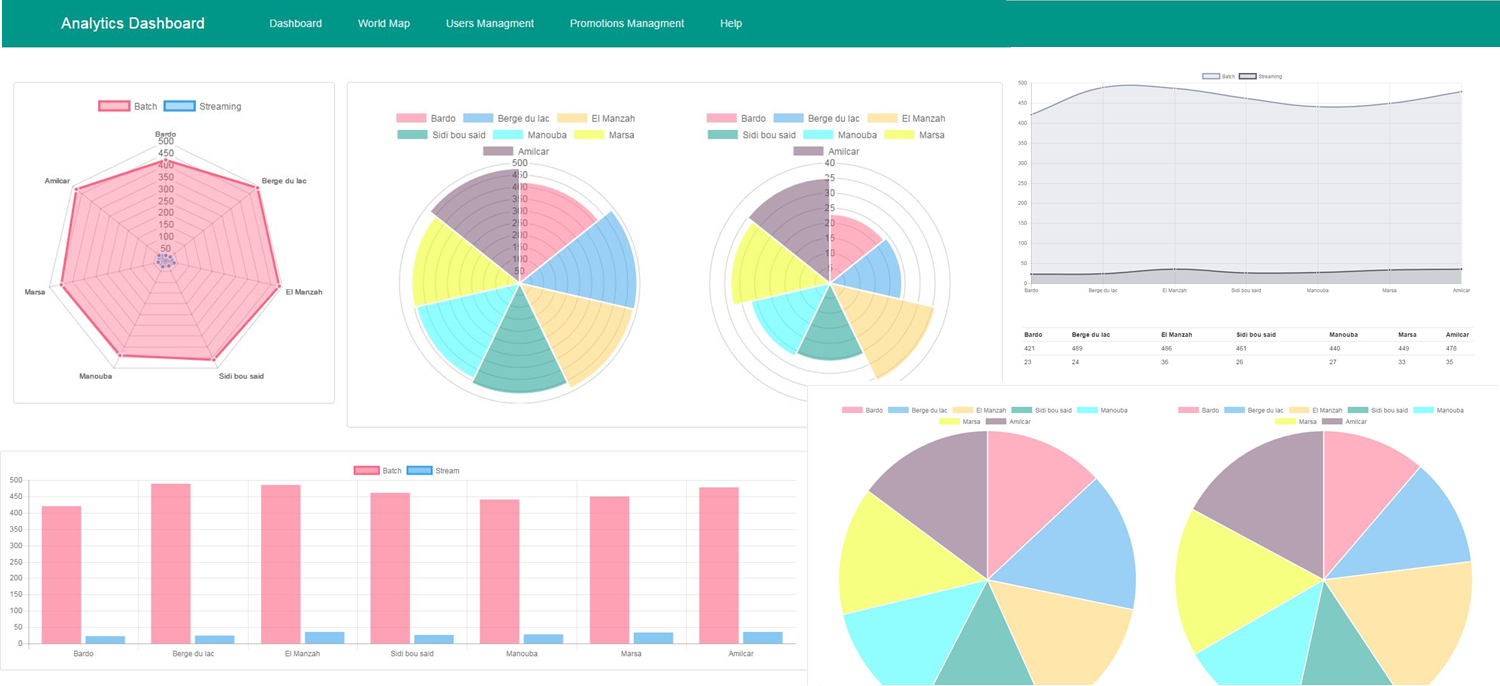
#### 3.6.3 Promotion Update

The next figure show the form update Promotion :

 [filiales et clients de Sofrecom.]Update promotions.

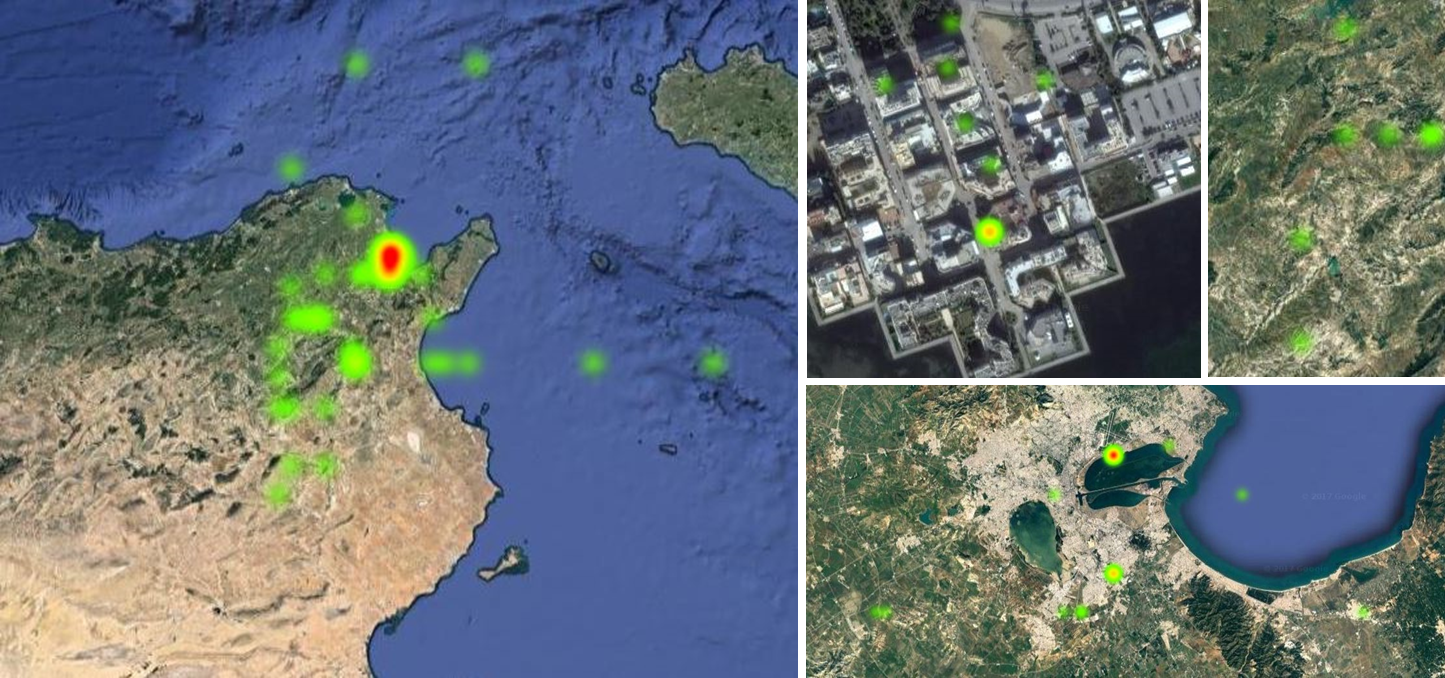
### 3.7 Display Analytics

The next figures show the Data Analytics Doshboard , this dashboard is update each time new data comes :

 [filiales et clients de Sofrecom.]Display analytics dashboard.

### 3.8 Heat Map users

The next figure show the Data Analytics Doshboard , this dashboard is update each time new data comes :

 [filiales et clients de Sofrecom.]Heat map users dashboard.

### 3.9 Summary

In this chapter, we presented our project implementation. First, we have prepared the work environment which takes too much time configuring, adapting versions and dealing with system engineering issues. Second, we ve moved to link each container to the other one. Then we started making some jobs like the log producer, the BatchSaveToHDFS, the StreamingProcessing and the Batch Processing. Third, we ve implemented the microservices architecture that allows account (users) or promotions management. Finally, we have our results (Analytics, Cruds and Heat Map) displayed on a single dashboard.

### 3.10 General conclusion

This project was designed to handle scaling in term of features and treatment. Our project is still incomplete. There is a lot of features that we would like to involve. Some of them are:

• A gateway api, that handle authentication using Spring boot and zuul.

• A container manager like docker Swarm or Kubernities that takes care of high availability and scalability.

• So much functional features like recommendation engine using Spark mllib.

• A mobile application that takes the log producer place.

• ...

Finally, We have been focused on creating a well designed system that can handle all of the requirements presented in the second chapter. The technical and the functional branches are now implemented and move to gather in a straight line.