

RAPPORT PROJET FINAL

Data Mining, Warehouse, Viz, BigData

Enseignant lecteur : Mohamed Souidi

Étudiants : VAYNE Mathis – BOUCHET Thomas

Année d'étude : 2ème année ITS (2020 - 2023)

Date rendu final : 05/01/2022

Lien GitHub du projet : <https://github.com/ThomasToto/Projet-final-Big-Data>

I] Présentation projet

Lors de ce projet final le but est de créer une infrastructure virtuelle permettant la récupération de données via une source de données, d'un transfert de ces données en streaming afin de les stocker et de les représenter.

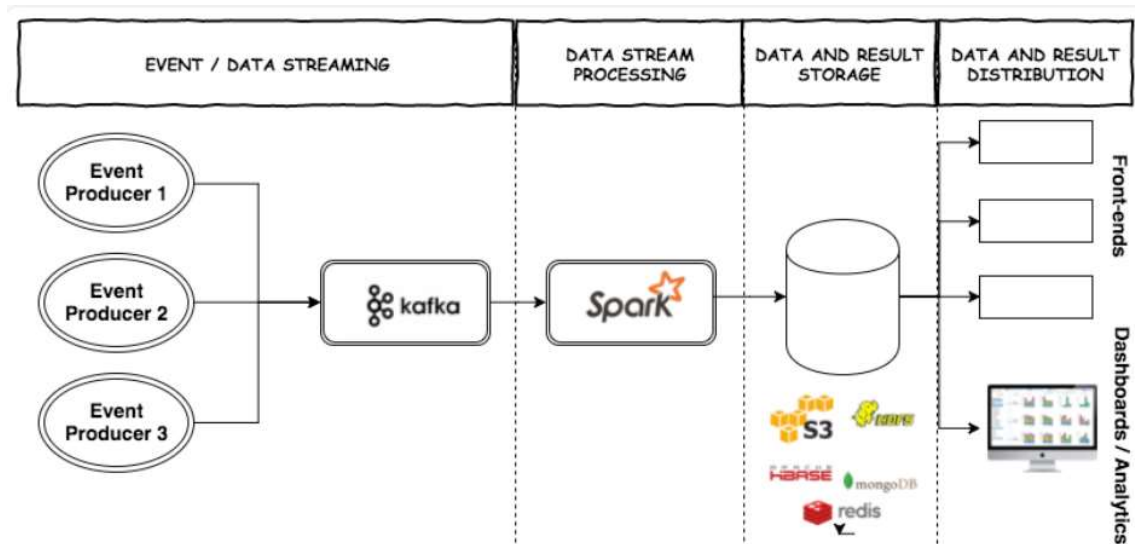


Illustration : Schéma illustrant l'architecture générale du projet

II] Description des outils

Derrière cette infrastructure se cache plusieurs outils que nous avons utilisé durant le projet. Tout d'abord l'utilisation du langage de programmation Python est omniprésent dans tous les scripts du projet.

Dans un premier temps dans le but de récupérer des données massives et en temps réel nous utilisons Tweepy. Tweepy est un wrapper Python pour l'API Twitter. Il accède aux API Twitter REST (y compris Search) et Stream.

Dans un deuxième temps nous utilisons l'outil Kafka. Kafka est une plateforme distribuée de diffusion de données en continu, capable de publier, stocker, traiter et souscrire à des flux d'enregistrement en temps réel. Elle est conçue pour gérer des flux de données provenant de plusieurs sources et les fournir à plusieurs utilisateurs.

Dans un troisième temps nous utilisons Spark. Spark est un logiciel open-source de calcul distribué disposant de la plus grande communauté de contributeurs en Big Data. L'avantage de Spark c'est qu'il est rapide, il est capable d'être 100 fois plus rapide que Hadoop pour le traitement de données à grande échelle grâce à une base de données en mémoire optimisée.

Nous utilisons pour le stockage de données l'outil MySQL. MySQL est un système de gestion de base de données. Son rôle est de stocker les données, sous forme de tables, et de permettre la manipulation de ces données à travers le langage de requête SQL.

Enfin, pour visualiser toutes ces données de façon intuitive nous utilisons Apache Superset. Apache Superset est un logiciel open source de Datavisualisation plutôt orienté sur les données massives.

III] Recommendations :

Durant ce projet nous avons utilisé plusieurs outils et nous pouvons vous en recommander quelques un. Tout d'abord l'outil Tweepy est très intéressant car il permet d'utiliser assez facilement l'API Twitter via Python. Nous n'avons eu aucun problème dessus donc nous ne pouvons que recommander.

Enfin, l'outil Superset recommandé par notre enseignant était pour nous inconnu mais il a été très simple d'utilisation, la création des graphiques est presque « magique » tellement elle est simple, ce qui est très appréciable.

IV] Résultat final

Le résultat final est celui qu'on espérait, c'est à dire que nous avons réussi à finaliser le projet en le rendant totalement fonctionnel.

Pour ce projet nous avons décidé d'adopter le cas d'utilisation suivant : Les pays les plus actifs sur Twitter.

Pour expliquer le résultat final de toute l'infrastructure nous allons décomposer l'explication par machine virtuelle.

Machine virtuelle n°1 (192.168.33.10) :

Pour le bon fonctionnement de la machine virtuelle nous avons alloué à la machine virtuelle une adresse IP (192.168.33.10).

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  config.vm.box = "bento/ubuntu-18.04"
  # Configure provisioning
  config.vm.provision "shell", path: "bootstrap.sh"
  # Configure network ports
  config.vm.network "private_network", ip: "192.168.33.10"

  config.vm.provider "virtualbox" do |vb|
    vb.cpus = 2
    vb.memory = 1024
    vb.gui = false
    vb.name = "kafka"
  end
end
```

Illustration : Script VagrantFile

Enfin, nous exécutons le script bootstrap.sh pour mettre à jour le système, installer Java, installer python3 et installer Kafka.

```
#!/bin/bash

# Functions

DATE() {
    date '+%Y-%m-%d %H:%M:%S'
}

# Let's go
# Update the System
echo "[$(DATE)] [Info] [System] Updating the system..."
apt update &> /dev/null
# Install Java
if [ $(dpkg-query -W -f='${Status}' openjdk-8-jdk 2>/dev/null | grep -c "ok installed") -eq 0 ];
then
    echo "[$(DATE)] [Info] [Java] Installing Java..."
    add-apt-repository -y ppa:openjdk-r/ppa &> /dev/null
    apt update &> /dev/null
    apt -y install openjdk-8-jdk &> /dev/null
fi
# Setup python3
echo "[$(DATE)] [Info] [Python] Setup python3..."
rm -rf /usr/bin/python
ln -s /usr/bin/python3 /usr/bin/python

# Install kafka
echo "[$(DATE)] [Info] [Kafka] Installing kafka..."
wget https://dlcdn.apache.org/kafka/3.0.0/kafka_2.13-3.0.0.tgz &> /dev/null
tar -xzf kafka_2.13-3.0.0.tgz
mv kafka_2.13-3.0.0 /usr/local/kafka
```

Illustration : Script bootstrap.sh

La première machine virtuelle contient les scripts producer.py, consumer.py et keys.py.

Nous allons dans un premier voir le script producer.py. Ce script utilise la librairie Tweepy permettant de récupérer les tweets. Pour mettre en place le cas d'utilisation expliqué précédemment nous avons récupéré tous les tweets avec la localisation activée. Une fois les tweets triés nous avons récupéré les pays de ces tweets afin de compter le nombre occurrence des pays.

Pour cela nous avons vérifié si dans le format JSON du tweet récupéré contenant l'attribut *place*.

```

root@vagrant:/usr/local/kafka# python consumer.py
{'created_at': 'Wed Dec 22 00:18:36 +0000 2021', 'id': 1473447846791856132, 'id_str': '1473447846791856132', 'text': 'RT
@JonyMah45: Check out my Gig on Fiverr: I will do creative unique logo design https://t.co/4US4VctZaV \n#logo #design #
Mufc #HJPatel #blo...', 'source': '<a href="https://digitalinspiration.com" rel="nofollow">AI News BOT</a>', 'truncated':
False, 'in_reply_to_status_id': None, 'in_reply_to_status_id_str': None, 'in_reply_to_user_id': None, 'in_reply_to_user_
id_str': None, 'in_reply_to_screen_name': None, 'user': {'id': 1134668128489656320, 'id_str': '1134668128489656320', 'na
me': 'Shy BOT', 'screen_name': 'ShyBOT7', 'location': None, 'url': None, 'description': 'Gathering #AI & #TensorFlow rel
ated tweets. I belong to @rebeccapark', 'translator_type': 'none', 'protected': False, 'verified': False, 'followers_cou
nt': 543, 'friends_count': 1, 'listed_count': 26, 'favourites_count': 9, 'statuses_count': 106075, 'created_at': 'Sat Ju
n 01 03:48:57 +0000 2019', 'utc_offset': None, 'time_zone': None, 'geo_enabled': False, 'lang': None, 'contributors_enab
led': False, 'is_translator': False, 'profile_background_color': 'F5F8FA', 'profile_background_image_url': '', 'profile_
background_image_url_https': '', 'profile_background_tile': False, 'profile_link_color': '1DA1F2', 'profile_sidebar_bord
er_color': 'C0DEED', 'profile_sidebar_fill_color': 'DDEEFF', 'profile_text_color': '333333', 'profile_use_background_ima
ge': True, 'profile_image_url': 'http://pbs.twimg.com/profile_images/1134668251844136960/VXLCIeXE_normal.jpg', 'profile_
image_url_https': 'https://pbs.twimg.com/profile_images/1134668251844136960/VXLCIeXE_normal.jpg', 'default_profile': Tru
e, 'default_profile_image': False, 'following': None, 'follow_request_sent': None, 'notifications': None, 'withheld_in_c
ountries': [], 'geo': None, 'coordinates': None, 'place': None, 'contributors': None, 'retweeted_status': {'created_at'
: 'Tue Dec 21 23:55:49 +0000 2021', 'id': 1473442113652084740, 'id_str': '1473442113652084740', 'text': 'Check out my Gi
g on Fiverr: I will do creative unique logo design https://t.co/4US4VctZaV \n#logo #design #Mufc_ https://t.co/aJlb5lUWh
d', 'source': '<a href="https://mobile.twitter.com" rel="nofollow">Twitter Web App</a>', 'truncated': True, 'in_reply_to
_status_id': None, 'in_reply_to_status_id_str': None, 'in_reply_to_user_id': None, 'in_reply_to_user_id_str': None, 'in_
reply_to_screen_name': None, 'user': {'id': 1437676399184146435, 'id_str': '1437676399184146435', 'name': 'Jony Mahmud',
'screen_name': 'JonyMah45', 'location': 'Dhaka, Bangladesh', 'url': 'https://www.fiverr.com/share/eEbkKg', 'description
': 'Graphic/Logo designer \nLove to work, like to use my creativity, want to make my clients happy. jonymahmud047@gmail.
com', 'translator_type': 'none', 'protected': False, 'verified': False, 'followers_count': 460, 'friends_count': 1459, 'l
isted_count': 0, 'favourites_count': 1084, 'statuses_count': 2114, 'created_at': 'Tue Sep 14 07:16:11 +0000 2021', 'utc
_offset': None, 'time_zone': None, 'geo_enabled': False, 'lang': None, 'contributors_enabled': False, 'is_translator': F
alse, 'profile_background_color': 'F5F8FA', 'profile_background_image_url': '', 'profile_background_image_url_https': ''
, 'profile_background_tile': False, 'profile_link_color': '1DA1F2', 'profile_sidebar_border_color': 'C0DEED', 'profile_s
idebar_fill_color': 'DDEEFF', 'profile_text_color': '333333', 'profile_use_background_image': True, 'profile_image_url':

```

Illustration : Exemple de tweets reçu avant filtrage

```

root@vagrant:/usr/local/kafka# python consumer.py
United States
$United Kingdom
$Brasil
$United States
$United Kingdom
$United States
$United States
$Polska
$India

```

Illustration : Exemple de tweets reçu après filtrage

Une fois le triage effectué les messages sont envoyés dans le topic *quickstart-events* et seront reçus par le script *consumer.py*.

Pour mettre en place cela nous avons réalisé les commandes suivantes :

Pour lancer l'environnement Kafka :

Lancer le service ZooKeeper :

bin/zookeeper-server-start.sh config/zookeeper.properties

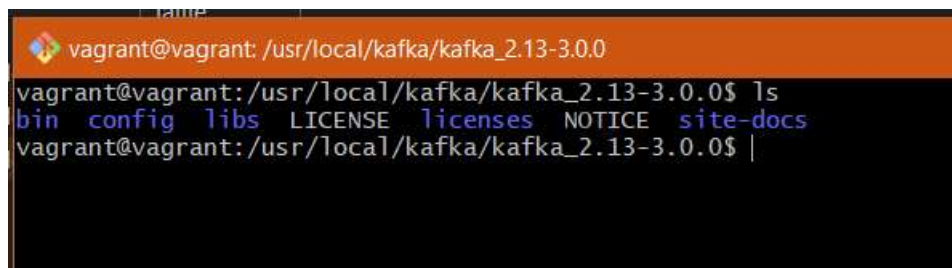
Lancer le broker service Kafka :

bin/kafka-server-start.sh config/server.properties

Et enfin créer un topic :

bin/kafka-topics.sh --create --partitions 1 --replication-factor 1 --topic quickstart-events --bootstrap-server localhost:9092

L'environnement crée ressemble à cela :

A terminal window with an orange title bar showing the path 'vagrant@vagrant: /usr/local/kafka/kafka_2.13-3.0.0'. The terminal text shows the user running 'ls' in the directory '/usr/local/kafka/kafka_2.13-3.0.0', resulting in a listing of files: 'bin', 'config', 'libs', 'LICENSE', 'licenses', 'NOTICE', and 'site-docs'.

```
vagrant@vagrant: /usr/local/kafka/kafka_2.13-3.0.0  
vagrant@vagrant:/usr/local/kafka/kafka_2.13-3.0.0$ ls  
bin  config  libs  LICENSE  licenses  NOTICE  site-docs  
vagrant@vagrant:/usr/local/kafka/kafka_2.13-3.0.0$ |
```

Illustration : Environnement Kafka

```
root@vagrant: /usr/local/kafka
GNU nano 2.9.3 producer.py

# -*- coding: utf-8 -*-
"""
Created on Sun Dec 19 15:09:04 2021

@author: Thomas
"""

import tweepy
import json
from kafka import KafkaProducer

from keys import API_KEY, API_SECRET_KEY, ACCESS_TOKEN, ACCESS_TOKEN_SECRET

producer = KafkaProducer(bootstrap_servers='localhost:9092')

# Création d'un StreamListener
class MyStreamListener(tweepy.StreamListener):

    def on_data(self, raw_data):
        self.process_data(raw_data)
        return True

    def process_data(self, raw_data):
        data = json.loads(raw_data)
        try:
            if 'place' in data and data['place']:
                producer.send("quickstart-events", str.encode(raw_data))
                return True
        except BaseException as e:
            print("Error on_data: %s" % str(e))
            return True

    def on_error(self, status_code):
        print(status_code)

# Création d'un stream
class MyStream():

    def __init__(self, auth, listener):
        self.stream = tweepy.Stream(auth=auth, listener=listener)

    def start(self):
        self.stream.sample(stall_warnings=None)
```

Illustration : Partie n°1 du script producer.py


```

if __name__ == "__main__":
    listener = MyStreamListener()

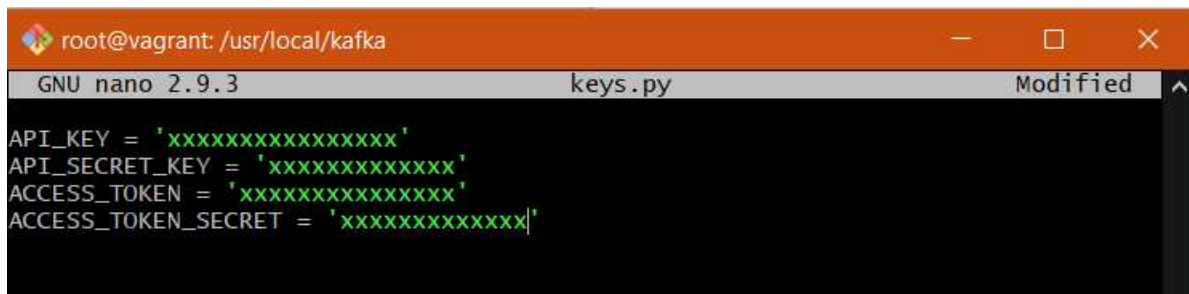
    auth = tweepy.OAuthHandler(API_KEY, API_SECRET_KEY)
    auth.set_access_token(ACCESS_TOKEN, ACCESS_TOKEN_SECRET)

    stream = MyStream(auth, listener)
    stream.start()

```

Illustration : Partie n°2 du script producer.py

A noter que le script *producer.py* utilise le script *keys.py* qui contient les clés publiques et secrètes ainsi que les tokens publics et secrets.
Ces informations étant privées vous trouverez une reconstitution du fichier mais ce n'est pas celui que nous avons utilisé.



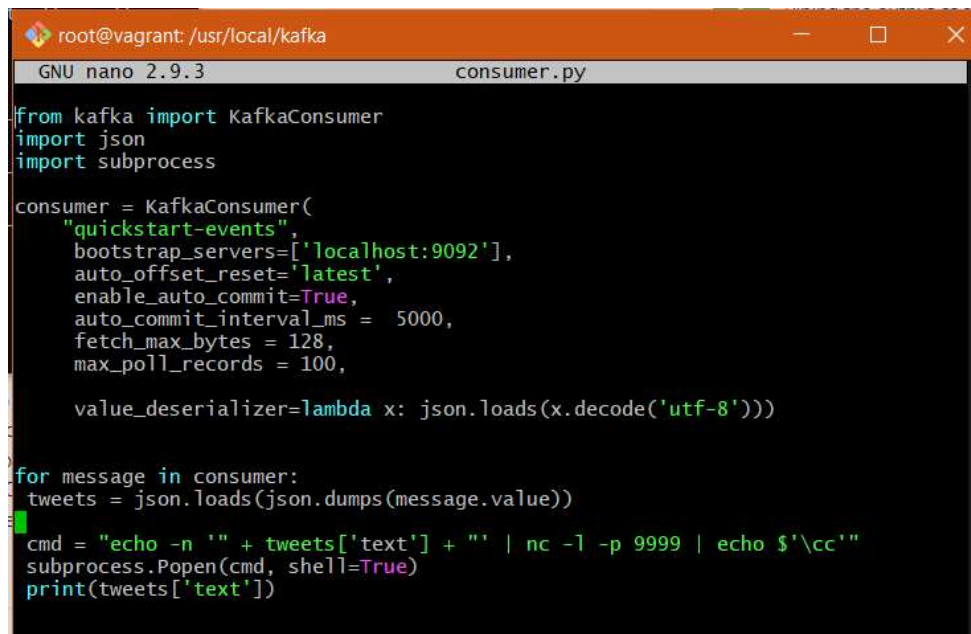
```

root@vagrant: /usr/local/kafka
GNU nano 2.9.3 keys.py Modified
API_KEY = 'xxxxxxxxxxxxxxxx'
API_SECRET_KEY = 'xxxxxxxxxxxxxxxx'
ACCESS_TOKEN = 'xxxxxxxxxxxxxxxx'
ACCESS_TOKEN_SECRET = 'xxxxxxxxxxxxxxxx'

```

Illustration : Exemple du fichier keys.py

Les noms de pays sont ensuite récupérés par le script *consumer.py* afin de les envoyer via une pipeline vers la machine virtuelle Spark (192.168.33.11).



```

root@vagrant: /usr/local/kafka
GNU nano 2.9.3 consumer.py
from kafka import KafkaConsumer
import json
import subprocess

consumer = KafkaConsumer(
    "quickstart-events",
    bootstrap_servers=['localhost:9092'],
    auto_offset_reset='latest',
    enable_auto_commit=True,
    auto_commit_interval_ms = 5000,
    fetch_max_bytes = 128,
    max_poll_records = 100,

    value_deserializer=lambda x: json.loads(x.decode('utf-8')))

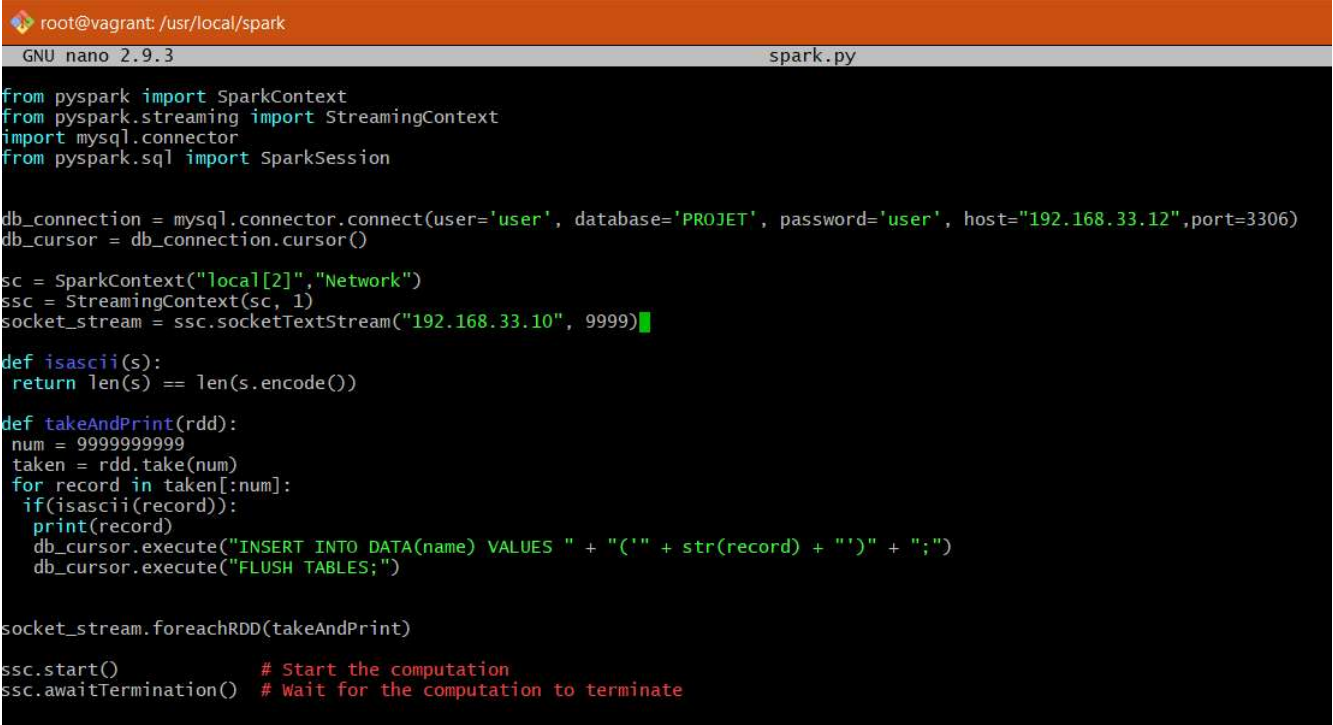
for message in consumer:
    tweets = json.loads(json.dumps(message.value))
    cmd = "echo -n '" + tweets['text'] + "' | nc -l -p 9999 | echo $\cc"
    subprocess.Popen(cmd, shell=True)
    print(tweets['text'])

```

Illustration : Script consumer.py

Machine virtuelle n°2 (192.168.33.11) :

La deuxième machine virtuelle contient le script *spark.py*.
Ce fichier récupère les informations envoyées par le script *consumer.py*.



```
root@vagrant: /usr/local/spark
GNU nano 2.9.3 spark.py

from pyspark import SparkContext
from pyspark.streaming import StreamingContext
import mysql.connector
from pyspark.sql import SparkSession

db_connection = mysql.connector.connect(user='user', database='PROJET', password='user', host="192.168.33.12",port=3306)
db_cursor = db_connection.cursor()

sc = SparkContext("local[2]", "Network")
ssc = StreamingContext(sc, 1)
socket_stream = ssc.socketTextStream("192.168.33.10", 9999)

def isascii(s):
    return len(s) == len(s.encode())

def takeAndPrint(rdd):
    num = 999999999
    taken = rdd.take(num)
    for record in taken[:num]:
        if(isascii(record)):
            print(record)
            db_cursor.execute("INSERT INTO DATA(name) VALUES " + "(" + str(record) + ")" + ";")
            db_cursor.execute("FLUSH TABLES;")

socket_stream.foreachRDD(takeAndPrint)

ssc.start() # Start the computation
ssc.awaitTermination() # Wait for the computation to terminate
```

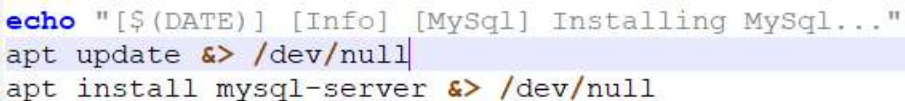
Illustration : Script spark.py

Le script crée une connexion avec la base de données située sur la machine virtuelle n°3 (192.168.33.12). Puis récupère les données de la pipeline pour les isoler individuellement et trier afin de les enregistrer dans la base de données.

Machine virtuelle n°3 (192.168.33.12) :

La machine virtuelle n°3 contient la base de données MySQL stockant toutes les données ainsi que le serveur Superset permettant la visualisation finale des données.

Le fichier VagrantFile appelle le script bootstrap.sh contenant l'installation de MySQL .



```
echo "[$(DATE)] [Info] [MySQL] Installing MySQL..."
apt update &> /dev/null
apt install mysql-server &> /dev/null
```

Illustration : Extrait du fichier bootstrap.sh

Nous avons dans un premier temps créé la base de données MySQL nommé *PROJET* puis une table *DATA* contenant un attribut *name*.

```
mysql> SHOW TABLES;
+-----+
| Tables_in_PROJET |
+-----+
| DATA             |
+-----+
1 row in set (0.00 sec)
```

Illustration : Structure de la base de données

```
mysql> SELECT * FROM DATA;
+-----+
| name |
+-----+
| United States |
| United Kingdom |
| United States |
| United States |
| United Kingdom |
| United States |
| Polska |
| Brasil |
| India |
| Argentina |
| United States |
| Italy |
| India |
| Brasil |
| Argentina |
+-----+
```

Illustration : Contenu de la base de données

Par la suite nous avons installé Superset.

L'installation de Superset c'est faite avec les commandes suivantes :

Les dépendances :

sudo apt update

sudo apt upgrade

sudo apt-get install build-essential libssl-dev libffi-dev python3-dev python3-pip libsasl2-dev libldap2-dev

L'environnement virtuel :

sudo apt-get install python3-venv

Créer et activer l'environnement :

python3 -m venv superset-env

source superset-env/bin/activate

Installation superset :

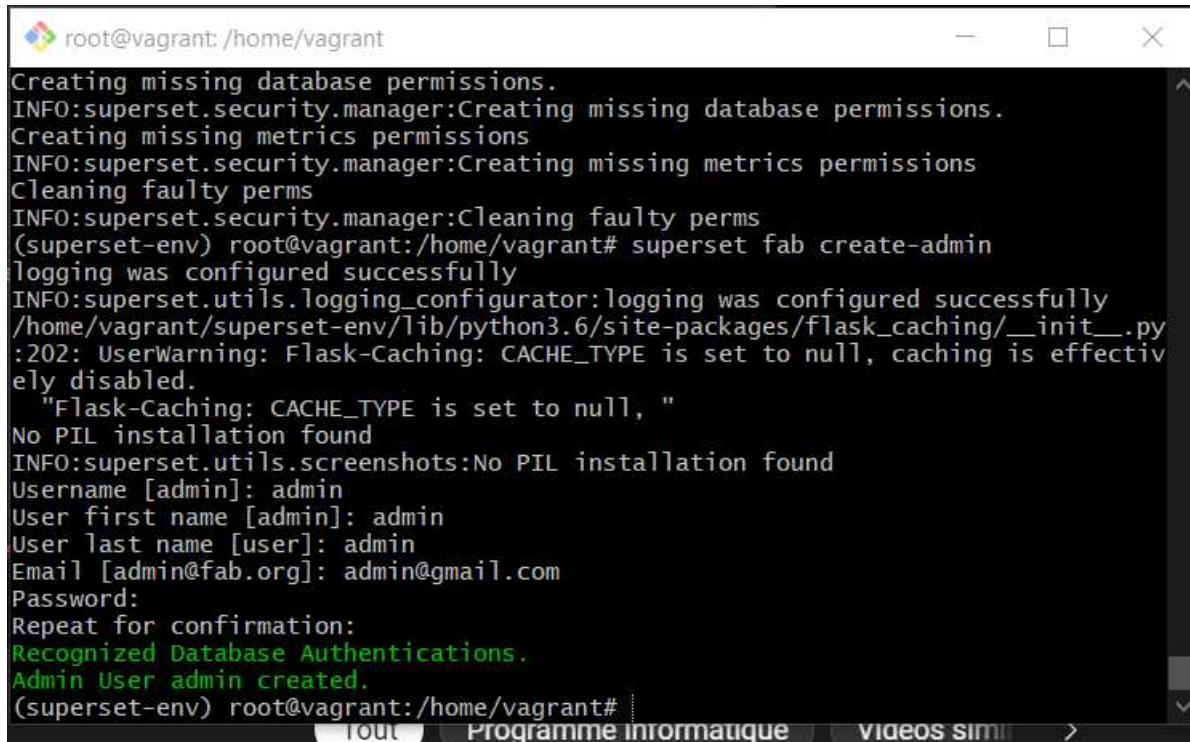
pip install apache-superset

Initialisation de la base de données :

superset db upgrade

Création d'un rôle par défaut et des permissions :
superset init

Création d'un utilisateur admin :
superset fab create-admin

A terminal window titled 'root@vagrant: /home/vagrant' showing the execution of Superset commands. The output includes messages about creating database and metrics permissions, cleaning faulty permissions, and successfully configuring logging. It then prompts for the creation of an admin user, showing fields for username (admin), first name (admin), last name (admin), email (admin@gmail.com), and password. The process concludes with 'Recognized Database Authentications.' and 'Admin User admin created.'

```
root@vagrant: /home/vagrant
Creating missing database permissions.
INFO:superset.security.manager:Creating missing database permissions.
Creating missing metrics permissions
INFO:superset.security.manager:Creating missing metrics permissions
Cleaning faulty perms
INFO:superset.security.manager:Cleaning faulty perms
(superset-env) root@vagrant:/home/vagrant# superset fab create-admin
logging was configured successfully
INFO:superset.utils.logging_configurator:logging was configured successfully
/home/vagrant/superset-env/lib/python3.6/site-packages/flask_caching/__init__.py:202: UserWarning: Flask-Caching: CACHE_TYPE is set to null, caching is effectively disabled.
  "Flask-Caching: CACHE_TYPE is set to null, "
No PIL installation found
INFO:superset.utils.screenshots:No PIL installation found
Username [admin]: admin
User first name [admin]: admin
User last name [user]: admin
Email [admin@fab.org]: admin@gmail.com
Password:
Repeat for confirmation:
Recognized Database Authentications.
Admin User admin created.
(superset-env) root@vagrant:/home/vagrant#
```

Illustration : Création de l'utilisateur admin

Lancement du serveur :
superset run -p 8080 -h 0.0.0.0

Enfin, la partie sûrement la plus intéressante, la visualisation des données.
Pour cela nous avons configuré Superset pour qu'il puisse avoir accès à la base de données et donc aux données.

Puis nous avons créé 2 graphiques afin d'illustrer nos données.

Pays utilisant le plus twitter DRAFT ☆

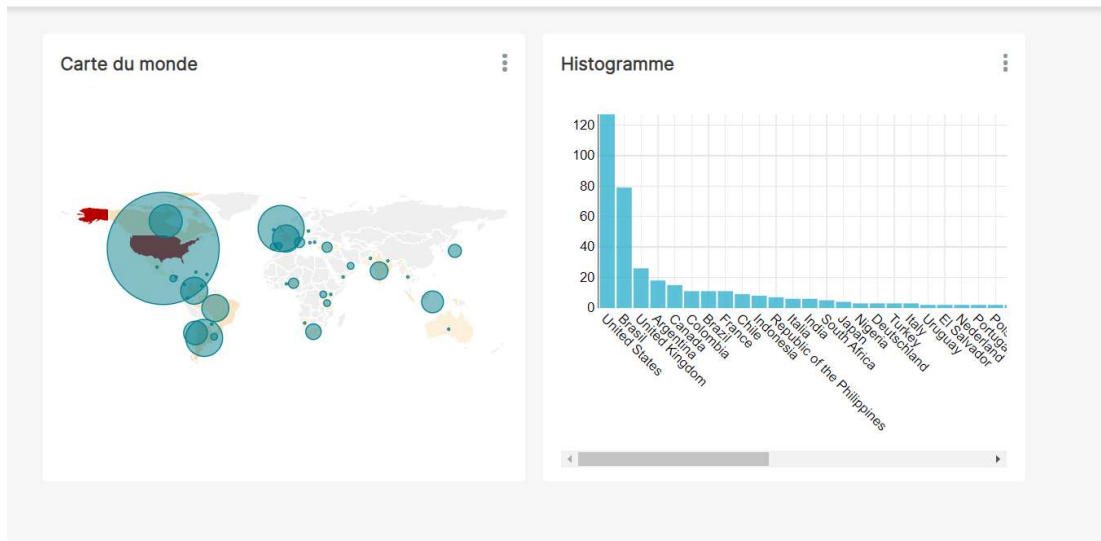


Illustration : Illustration du dashboard contenant les 2 graphiques

Le premier graphique que nous avons fait un histogramme permettant de voir rapidement les pays les plus actifs que Tweeter.

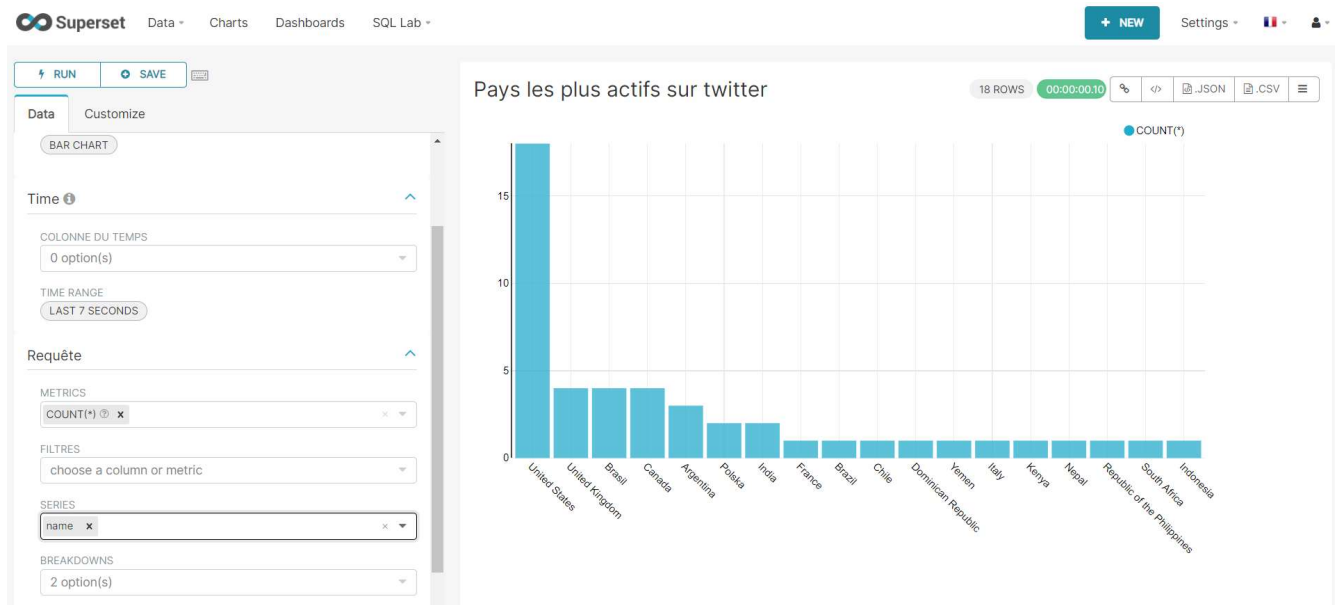


Illustration : Histogramme montrant les pays les plus actifs sur Twitter

Et le deuxième graphique est une carte du monde avec le nombre de tweets par pays. Cette représentation permet d'avoir des informations géographiques supplémentaires.

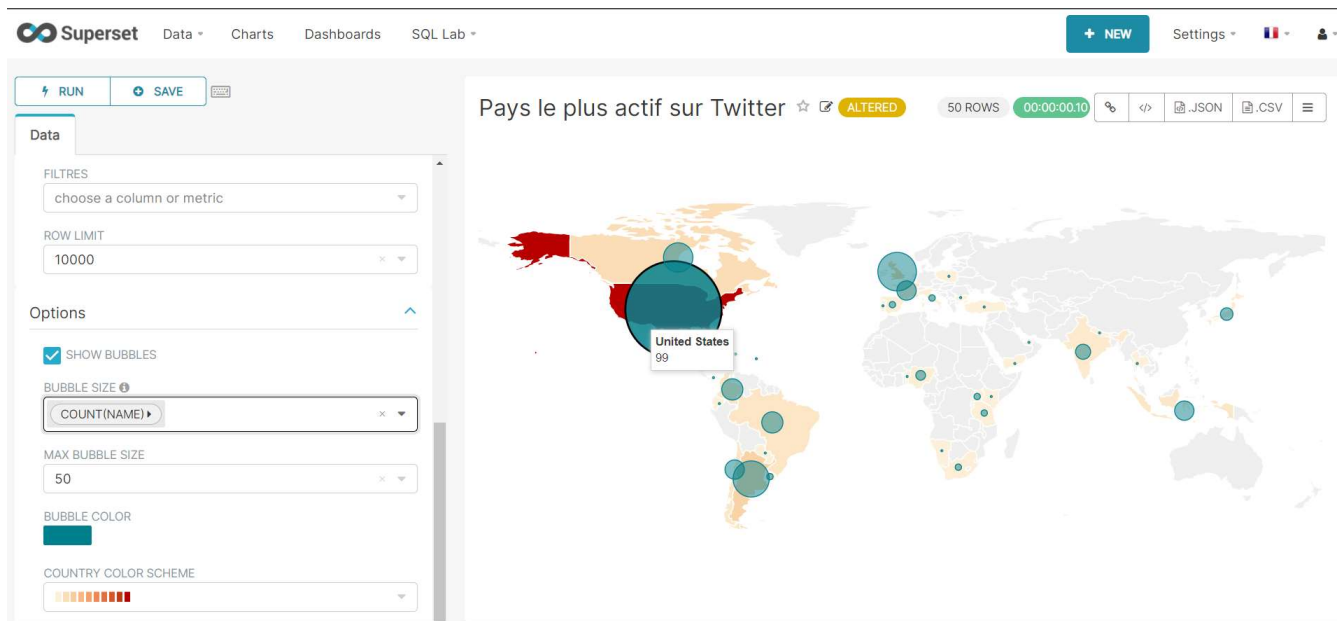


Illustration : Carte du monde montrant les pays les plus actifs sur Twitter

Pour les deux graphiques Superset effectue un COUNT de l'attribut *name* permettant de calculer le nombre d'occurrence de chaque pays dans la base de données.

Pour résumer on se retrouve avec ceci :

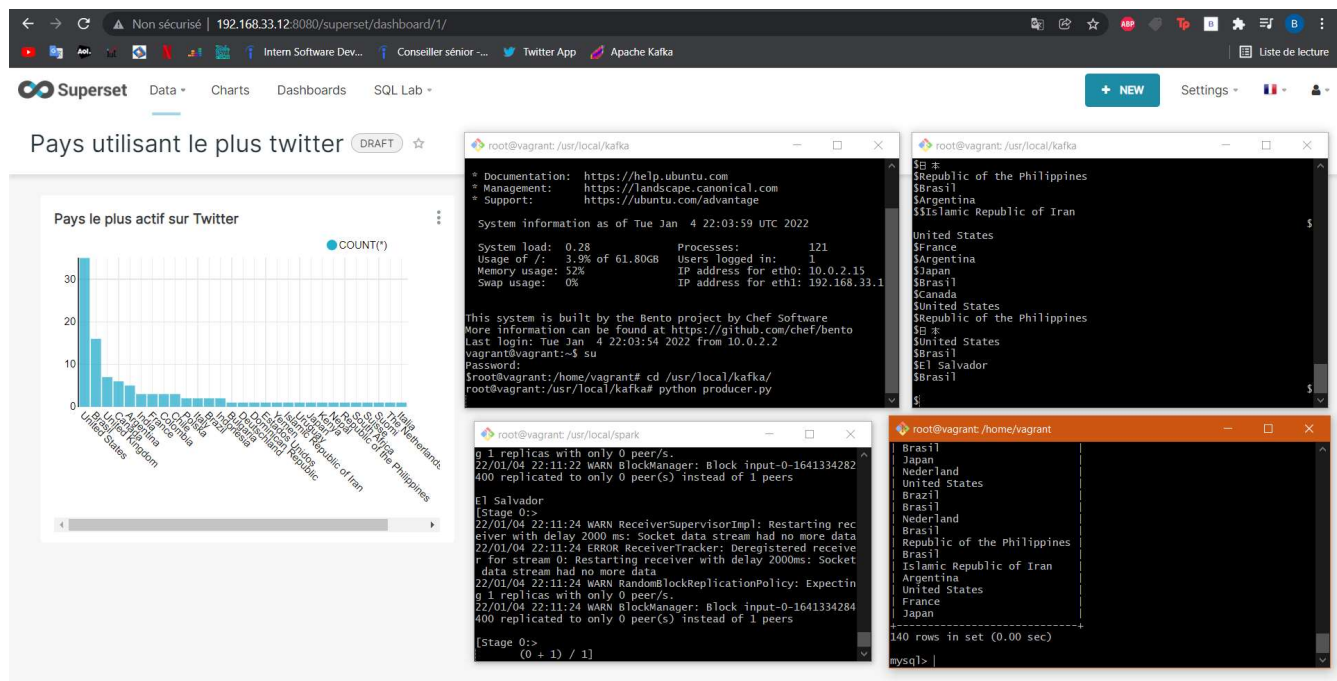


Illustration : Résumé du résultat final

On se retrouve avec 2 terminaux sur la machine virtuelle exécutant respectivement les scripts *producer.py* et *consumer.py*. Un autre terminal exécutant le script *spark.py*. Un autre terminal pour voir en temps réel la base de données se remplir de données. Et enfin, les graphiques affichant en temps réel les données stockées dans la base de données.

V| Difficultés rencontrées

Lors de ce projet plusieurs points nous ont bloqué.

Droits et permissions du compte développeur Twitter

Premièrement lors de l'utilisation de Tweepy. Dans le cas de notre projet un compte développeur Twitter avec le minimum de droits et permissions ne suffisait pas. Nous avons donc dû effectuer une demande auprès de Twitter.

La demande consistait à décrire en anglais nos intentions avec l'API twitter, dans quelle entreprise ont travaillé, quelle était la finalité des données, est ce qu'ont été rattaché à un état etc.

Twitter a par la suite accepté notre demande et a élevé les droits et permissions de notre compte. Celui-ci nous a parfaitement convenu par la suite.

Isolation individuelle des données après le stream :

Afin d'enregistrer toutes nos données dans la BD MySQL nous étions contraint de récupérer un par un les données envoyées par le stream.


```
sc = SparkContext("local[2]", "Network")
ssc = StreamingContext(sc, 1)
socket_stream = ssc.socketTextStream("192.168.33.10", 9999)
```

Illustration : Extrait du script spark.py

L'extrait ci-dessus permet d'expliquer la situation. Pour effectuer le transfert entre Kafka et Spark nous avons créé une pipeline. La variable *socket_stream* contenait alors un objet *DStream* contenant en temps réel les données transitant dans la pipeline.

Néanmoins, nous avons pris beaucoup de temps à comprendre comment récupérer concrètement et individuellement les données.

Il nous a donc fallu consulter la documentation d'Apache Spark. Nous avons remarqué qu'une fonction permettait l'affichage des données. La fonction *pprint*.

```
def pprint(self, num=10):
    """
    Print the first num elements of each RDD generated in this DStream.

    Parameters
    -----
    num : int, optional
        the number of elements from the first will be printed.
    """
    def takeAndPrint(time, rdd):
        taken = rdd.take(num + 1)
        print("-----")
        print("Time: %s" % time)
        print("-----")
        for record in taken[:num]:
            print(record)
        if len(taken) > num:
            print("...")
        print("")

    self.foreachRDD(takeAndPrint)
```

Illustration : Fonction pprint, extrait de la documentation Apache Spark

(Lien : <https://spark.apache.org/docs/latest/api/python/modules/pyspark/streaming/dstream.html#DStream.pprint>)

Nous avons aussi remarqué qu'une fonction *foreachRDD* existait. Celle-ci permettait l'exécution d'une fonction sur chaque RDD.

```
def foreachRDD(self, func):
    """
    Apply a function to each RDD in this DStream.
    """
    if func.__code__.co_argcount == 1:
        old_func = func
        func = lambda t, rdd: old_func(rdd)
    jfunc = TransformFunction(self._sc, func, self._jrdd_deserializer)
    api = self._ssc._jvm.PythonDStream
    api.callForeachRDD(self._jdstream, jfunc)
```

[docs]

Illustration : Fonction foreachRDD, extrait de la documentation Apache Spark
 (Lien : <https://spark.apache.org/docs/latest/api/python/modules/pyspark/streaming/dstream.html#DStream.foreachRDD>)

Nous avons donc adapté notre code afin de répondre à nos besoins. Premièrement nous avons redéfini la fonction *takeAndPrint* située initialement dans la fonction *pprint*. Celle-ci prend toutes les données envoyées aux différents intervalles de temps et les traite individuellement afin de les intégrer dans une requête SQL. Puis nous appelons cette fonction pour chaque RDD via la fonction *foreachRDD* que nous n'avons pas modifié.

```
def takeAndPrint(rdd):
    num = 999999999
    taken = rdd.take(num)
    for record in taken[:num]:
        if(isascii(record)):
            print(record)
            db_cursor.execute("INSERT INTO DATA(name) VALUES " + "(" + str(record) + ")" + ";")
            db_cursor.execute("FLUSH TABLES;")

socket_stream.foreachRDD(takeAndPrint)
```

Illustration : Extrait du script producer.py

Gestion des données non ASCII

Twitter étant un réseau social utilisé par énormément d'utilisateurs il arrivait que certains localisations récupérées n'étaient pas dans le format ASCII.

```
KeyboardInterrupt
root@vagrant:/usr/local/kafka# python consumer.py
United States
$United Kingdom
$Brasil
$United States
$United Kingdom
$United States
$United States
$Polksa
$India
$Argentina
$日本
$United States
$Italy
$日本
$India
$M xico
$Brasil
$Argentina
$United States
$United States
$$United States
```

Illustration : Extrait du r sultat du script consumer.py

On remarque bien sur cette capture d cran que certaines donn es n taient pas dans le format ASCII. Or cela posait probl me pour la suite lors de l'enregistrement en base de donn es via MySQL. Nous avons donc d  effectuer un triage des donn es pour ne s lectionner que celle dans le format ASCII.

Cette t che est effectu e par la fonction suivante :

```
def isascii(s):
    return len(s) == len(s.encode())
```

Illustration : Fonction isascii, extrait du script consumer.py

Connexion MySQL / Superset :

Un autre difficult  est apparue lors de la liaison entre la base de donn es MySQL et l'outil Apache Superset. Il faut savoir que pour cr er la liaison entre une base de donn es et Superset il est n cessaire d'entrer une SQLAlchemy URL.

Add Database

Database *	Database
SQLAlchemy URI *	SQLAlchemy URI Refer to the SQLAlchemy docs for more information on how to structure your URI. TEST CONNECTION

Illustration : Capture d cran de Superset

Sachant que nous utilisions MySQL le format de l'URL devait être le suivant :

```
mysql://username:password@ip_adress:port/db_name
```

Or malgré avec ce format ceci ne fonctionnait pas. En fait, le problème était le suivant. Notre base de données et notre serveur Superset était situé sur deux machines virtuelles différentes. Or notre base de données n'était pas au début configurée pour autoriser l'accès à d'autres adresses IP que localhost.

Dans un soucis de faciliter nous avons résolu le problème en installant Apache Superset sur la même VM que celle où était installé MySQL.

VI] Nos impressions et l'apprentissage

Nous avons beaucoup aimé ce projet car nous avions aucune connaissance sur le sujet. Nous avons jamais eu l'occasion d'effectuer un projet sur ce sujet. Néanmoins, nous avons des connaissances en informatique. Mathis a obtenu un DUT Réseaux et télécoms avant de rejoindre l'école et Thomas un DUT Informatique donc nous sommes partis sur de bonnes bases.

Malgré aucune connaissance sur le sujet nous avons effectué énormément de recherches sur Internet afin de se documenter de comprendre le fonctionnement de chaque outil. Nous avons aimé le format proposé pour ce projet, c'est à dire que l'infrastructure générale et le nom des outils avaient été expliqué en cours mais le reste du travail était à faire en autonomie. Ceci nous a permis de découvrir énormément de choses et de nous améliorer dans le domaine du Big Data.

Malgré plusieurs difficultés la satisfaction du projet finalisé et fonctionnel est ce que l'on a le plus apprécié.