Compte rendu

de Antoine Malaval

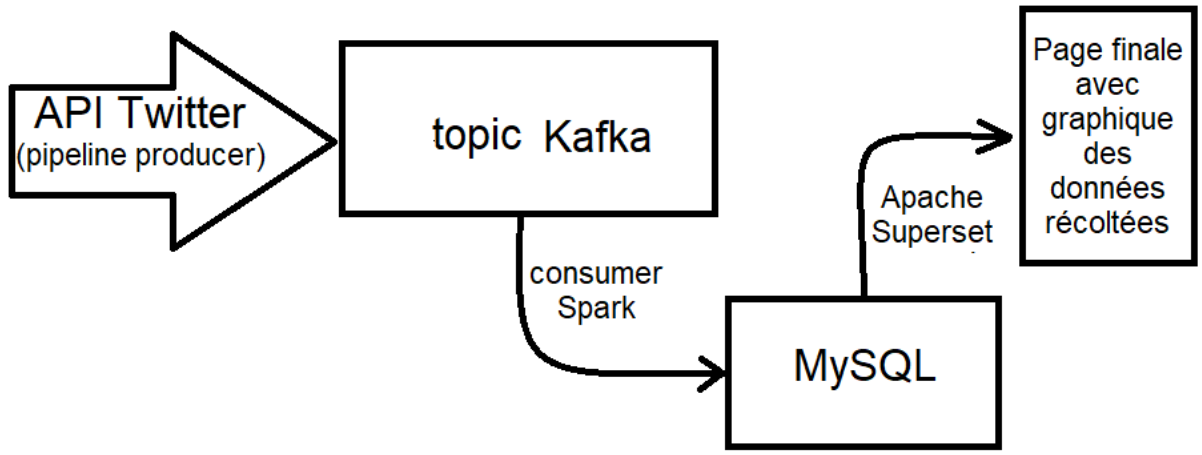
L'objectif du projet est de créer un pipeline streaming dans le but de créer des graphes pour afficher les données recueillies.

Pour se faire, nous avons utilisé 4 technologies :

- Kafka
- Spark
- une base de données MySQL
- Apache superset

Plus techniquement, le projet a pour objectif de recueillir des données via l'API de Twitter, et de les envoyer sur le topic kafka que l'on crée. Par la suite un consumer spark se connecte à kafka afin de traiter les données reçues, les mettre en forme et les insérer dans la base de données MySQL. Pour finir, Apache Superset visualise les données en se connectant à notre base MySQL pour afficher les graphiques.

résumé de l'utilisation des différents outils :



Dans un premier temps, nous devons installer des machines virtuelles grâce à Vagrant sur VirtualBox :

- une machine virtuelle pour python (script pour produire la donnée de l'API)
- une machine virtuelle pour Kafka
- une machine virtuelle pour Spark
- une machine virtuelle pour la base de données MySQL et Apache Superset

Une fois les machines créées nous nous sommes connecté aux machines en utilisant vagrant ssh.

Sur la machine Kafka, nous avons créé le topic qui nous servira à connecter le *producer* et le *consumer* puis envoyer des données de test.

```
vagrant@vagrant:~/kafka/bin$ ./kafka-topics.sh --create --topic tweets-topic --bootstrap-server 192.168.33.13:9092
Created topic tweets-topic.
```

J’ai choisi de me focaliser sur le nombre de followers des utilisateurs et les hashtags.

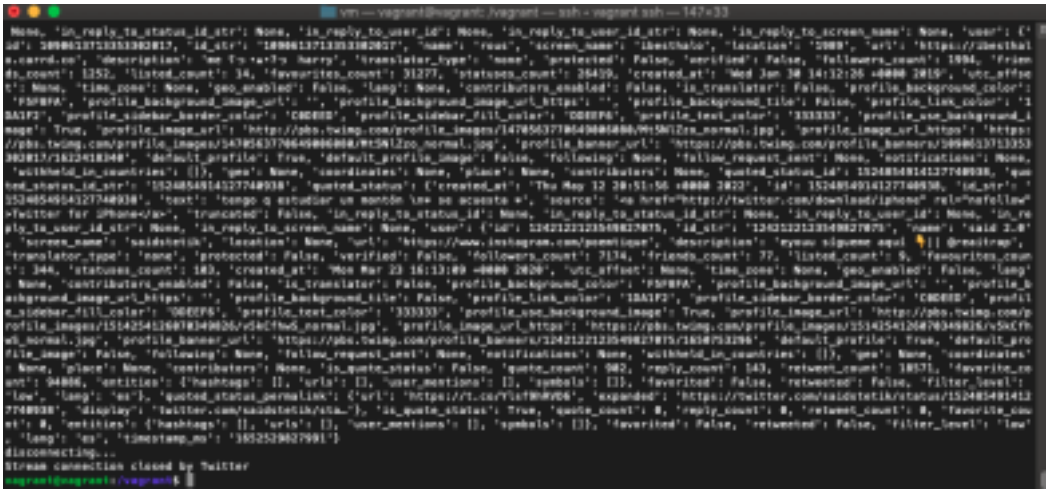
producer-streaming-twitter.py

C: > Users > antoi > OneDrive > Documents > esi5e-pipeline-streaming-twitter > producer-streaming-twitter.py

```python
producer = KafkaProducer(bootstrap_servers='192.168.33.13:9092')


class Listener(Stream):

    tweetsProceeded = []
    limit = 100

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

    def process_data(self, raw_data):
        if len(self.tweetsProceeded) == self.limit:
            print("disconnecting...")
            self.disconnect()
        else:
            message = json.loads(raw_data)
            if message['user']['location'] is not None:
                print(message)
                producer.send('tweets', raw_data)
                self.tweetsProceeded.append(raw_data)

    def on_error(self, status_code):
        print(status_code)
        if status_code == 420:
            # returning false in on_data disconnects the stream
            return False


# start the stream
if __name__ == "__main__":
    auth = tweepy.OAuthHandler(API_KEY, API_KEY_SECRET)
    auth.set_access_token(ACCESS_TOKEN, ACCESS_TOKEN_SECRET)

    api = tweepy.API(auth)

    listener = Listener(API_KEY, API_KEY_SECRET,
                        ACCESS_TOKEN, ACCESS_TOKEN_SECRET)

    keywords = ["*"]

    listener.filter(track=keywords)

```

Comme on peut le voir en lançant le script ci-dessus, les tweets sont affiché sur la console

une fois que le nombre de message à été produit, le stream se stop :



En lançant un kafka-console-consumer nous voyons bien les tweets passer également :

Puis nous avons créé le job Spark qui va consommer les données du topic :

```
import string
from pyspark.sql import SparkSession, Row
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils
from pyspark.sql.types import FloatType, StringType
from pyspark.sql.functions import to_date
import json
import re

emoji_pattern = re.compile("["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
        u"\U00002702-\U000027B0"
        u"\U000024C2-\U0001F251"
        u"\U0001f926-\U0001f937"
        u'\U00010000-\U0010ffff'
        u"\u200d"
        u"\u2640-\u2642"
        u"\u2600-\u2855"
        u"\u23cf"
        u"\u23e9"
        u"\u231a"
        u"\u3030"
        u"\ufe0f"
                           "]+", flags=re.UNICODE)

def getHashtags(rdd_collected):
    list_hashtags = []
    for element in rdd_collected:
        print(element)
        for hashtag in element:
            if hashtag["text"] != None:
                list_hashtags.append(hashtag["text"])
    return list_hashtags
```

```
def stripTextAndRemoveEmoji(text):
    stringWithoutEmoji = emoji_pattern.sub(r'', text)
    nameIsBlank = re.search('^\s*$', stringWithoutEmoji)
    if not stringWithoutEmoji or nameIsBlank :
        return "No name"
    return stringWithoutEmoji.strip()
```

```
def process(time, rdd):
    print("========= %s =========" % str(time))
    if not rdd.isEmpty():
        print(rdd)
        locationDF = rdd.map(lambda tweet: Row(username=stripTextAndRemoveEmoji(tweet['user']['name']),
                           nb_friends=tweet['user']['friends_count'])).toDF()

        locationDF = locationDF.withColumn("username", locationDF["username"].cast(StringType())) \
                               .withColumn("nb_friends", locationDF["nb_friends"].cast(IntegerType()))

        locationDF.printSchema()

        locationDF.write.format('jdbc').options(
            url='jdbc:mysql://192.168.33.10/data',
            dbtable='users',
            user='admin',
            password='admin').mode('append').save()


        rdd_collected = rdd.map(lambda tweet: tweet["entities"]["hashtags"]).collect()

        if len(rdd_collected) >= 1:
            hashtags_list = getHashtags(rdd_collected)
            rdd_hashtags = sc.parallelize(hashtags_list)
            if rdd_hashtags.isEmpty() == False:
                hashtagsDF = rdd_hashtags.map(lambda hashtags: Row(hashtag=hashtags)).toDF()
                hashtagsDF = hashtagsDF.withColumn("hashtag", hashtagsDF["hashtag"].cast(StringType()))
                hashtagsDF.printSchema()
                hashtagsDF.write.format('jdbc').options(
                    url='jdbc:mysql://192.168.33.10/data',
                    dbtable='hashtags',
                    user='admin',
                    password='admin').mode('append').save()


spark = SparkSession.builder \
        .master("local[2]") \
        .appName("data") \
        .getOrCreate()
```

```
sc = spark.sparkContext
sc.setLogLevel("ERROR")
ssc = StreamingContext(sc, 10)

directKafkaStream = KafkaUtils.createDirectStream(ssc, ["tweets"], {"metadata.broker.list": "192.168.33.13:9092"})
rdd = directKafkaStream.map(lambda tweet: json.loads(tweet[1]))
rdd.foreachRDD(process)


ssc.start()
ssc.awaitTermination()
```

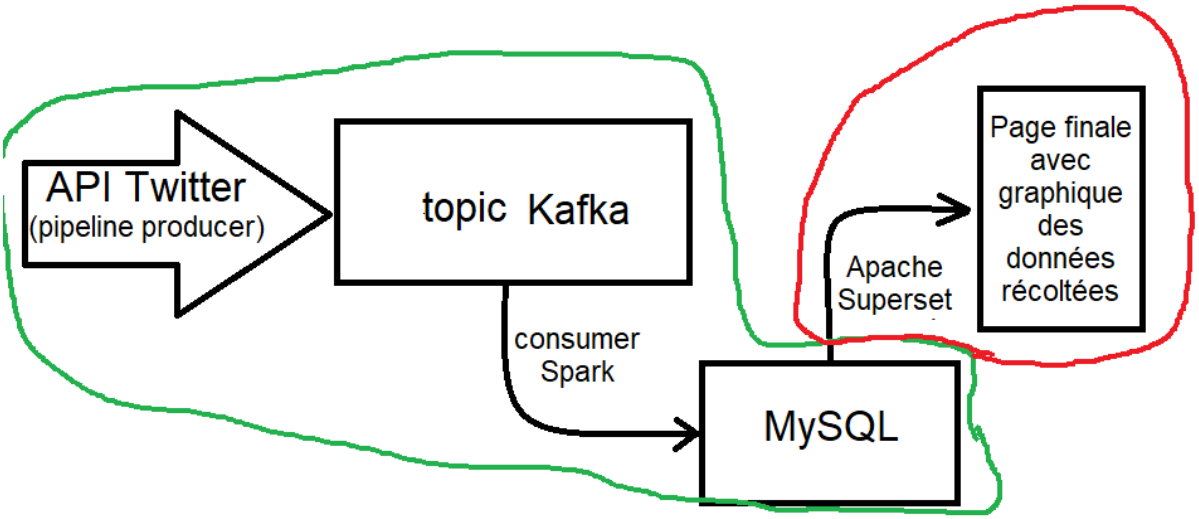
Après avoir installé le wget spark-streaming-kafka et le déplacer dans le bon répertoire (à savoir les jars de spark), nous récupérons les données sur le consumer spark avec la commande spark-submit :

```
vagrant@vagrant:~$ /usr/local/spark/bin/spark-submit spark-consumer-tweets.py
```

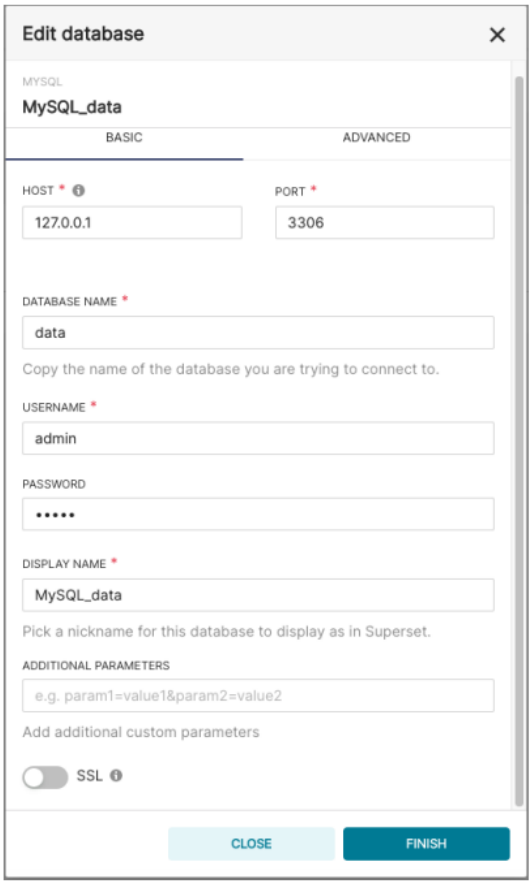
Pour vérifier que les données s'insèrent bien dans la base MySQL, je me suis connecté à MySQL et fait un Select pour les Users et les Hashtags.

select * from users;	select * from hashtags;
`1083 rows in set (0.01 sec)`	`401 rows in set (0.00 sec)`

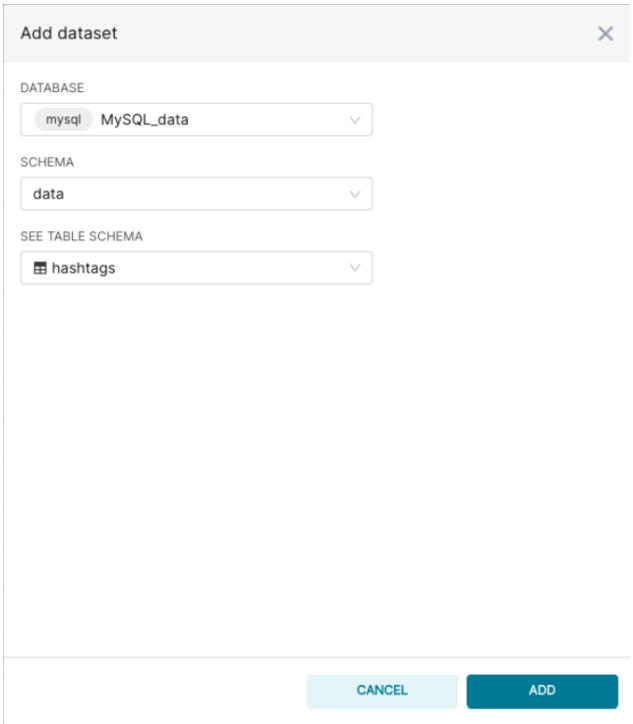
Pour reprendre mon précédent schéma, en vert ce que nous avons fait, en rouge ce qu'il nous reste à faire :



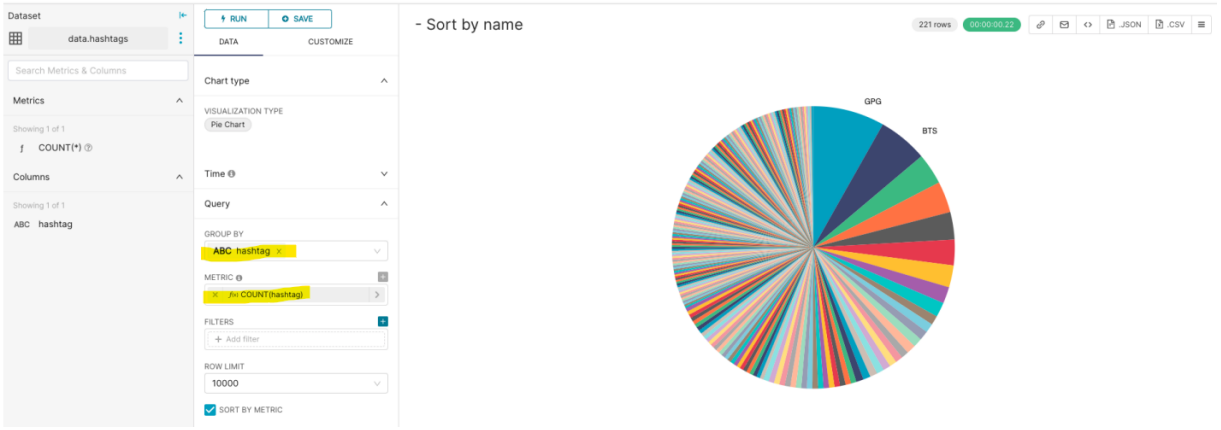
Nous devons donc connecter notre base de données sur Apache Superset afin d'effectuer des actions sur les données :



Nous "convertissons" la base de données en dataset :



J’ai ainsi créé un Chart en fonction du nombre des hashtag présents dans la base de données grâce au count() :



Pour finir, j’ai créé un autre graph pour afficher les utilisateurs ayant le plus d’amis sur les données récupérer :

- Users and friends

Show 200 entries

nb_friends	username
18553	allie.
9380	#TodosSomosUcrania #LaResistencia
7353	No name
6567	Fantasy * Fragment of Time *
6038	No name
5828	mikel ayestaran
5809	mika PROOF 0610
5375	duda de la roit THOR
5000	Michael Van Vaerenbergh
4977	@
4956	Abdulkareem
4947	Rain
4931	2015,10,18,
4910	may
4613	Mark W. Yusko #TwoPointOneQuadrillion
3702	@benedito_xr#RGSDV #MFSDV
3702	@benedito_xr#RGSDV #MFSDV
3702	@benedito_xr#RGSDV #MFSDV
3702	@benedito_xr#RGSDV #MFSDV
3702	@benedito_xr#RGSDV #MFSDV
3702	@benedito_xr#RGSDV #MFSDV
3702	@benedito_xr#RGSDV #MFSDV