

# Compte rendu

de Antoine Malaval

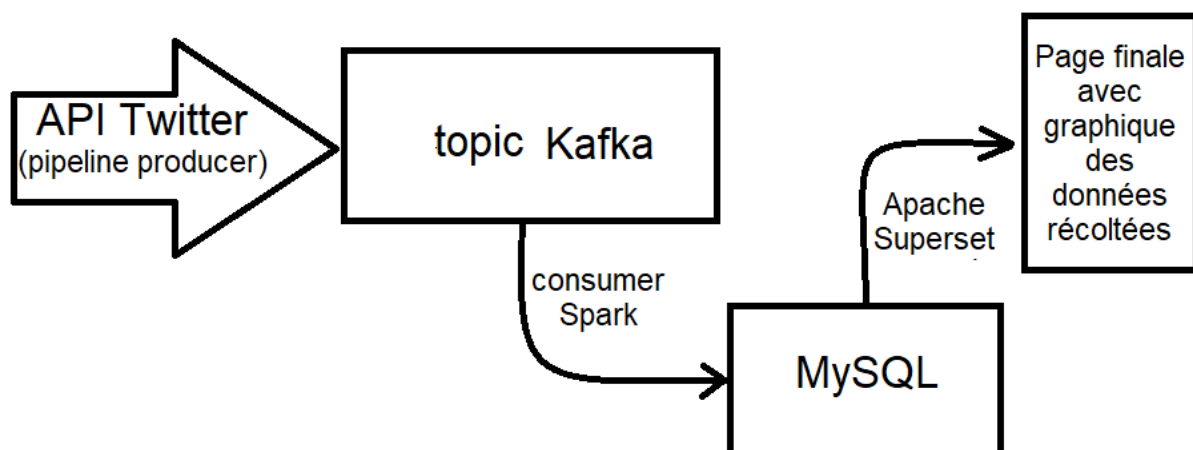
L'objectif du projet est de créer un pipeline streaming dans le but de créer des graphes pour afficher les données recueillies.

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 X
C: > Users > antoi > OneDrive > Documents > esi5e-pipeline-streaming-twitter > producer-streaming-twitter.py
12 producer = KafkaProducer(bootstrap_servers='192.168.33.13:9092')
13
14
15 class Listener(Stream):
16
17     tweetsProceeded = []
18     limit = 100
19
20     def on_data(self, raw_data):
21         self.process_data(raw_data)
22         return True
23
24     def process_data(self, raw_data):
25         if len(self.tweetsProceeded) == self.limit:
26             print("disconnecting...")
27             self.disconnect()
28         else:
29             message = json.loads(raw_data)
30             if message['user']['location'] is not None:
31                 print(message)
32                 producer.send('tweets', raw_data)
33                 self.tweetsProceeded.append(raw_data)
34
35     def on_error(self, status_code):
36         print(status_code)
37         if status_code == 420:
38             # returning false in on_data disconnects the stream
39             return False
40
41
42 # start the stream
43 if __name__ == "__main__":
44     auth = tweepy.OAuthHandler(API_KEY, API_KEY_SECRET)
45     auth.set_access_token(ACCESS_TOKEN, ACCESS_TOKEN_SECRET)
46
47     api = tweepy.API(auth)
48
49     listener = Listener(API_KEY, API_KEY_SECRET,
50                         ACCESS_TOKEN, ACCESS_TOKEN_SECRET)
51
52     keywords = ["*"]
53
54     listener.filter(track=keywords)
55
```

```

vagrant@vagrant:~/vagrant$ python3 producer-streaming-twitter.py
{"created_at": "Wed May 04 16:26:30 +0000 2004", "id": 1521689530361637904, "id_str": "1521689530361637904", "text": "Glennane Smith From the first testament +",
"display_text_range": [125, 41], "source": "via href='http://twitter.com/download/iphone' rel='nofollow' Twitter for iPhone", "truncated": false, "in_reply_to_status_id": 1521689530361637904, "in_reply_to_status_id_str": "1521689530361637904", "in_reply_to_user_id": 32652003, "in_reply_to_user_id_str": "32652003",
"status": 1521689530361637904, "user": {"id": 32652003, "id_str": "32652003", "name": "Doc Tash Denmark", "screen_name": "N0rmark", "location": "BT, 3.435
212, -111.943974", "url": "http://culturedcrows.com", "description": "60b, ATC, CPW [aka/her] #edacore #47 #socialjustice and #city warrior #mother wife and
#activist! X/02805 Belarus New Jersey California", "translator": false, "verified": false, "followers_count": 698, "friends
count": 811, "listed_count": 11, "favorites_count": 21957, "statuses_count": 5994, "created_at": "Tue Mar 03 18:04:40 +0000 2009", "utc_offset": None, "time
zone": None, "geo_enabled": true, "lang": None, "contributors_enabled": false, "is_translator": false, "profile_background_color": "#808080", "profile_background_i
mage_url": "http://abs.twimg.com/images/themes/theme1/bg.gif", "profile_background_image_url_https": "https://abs.twimg.com/images/themes/theme1/bg.gif", "profile
background_image_https": false, "profile_link_color": "#FF0000", "profile_sidebar_border_color": "#000000", "profile_sidebar_fill_color": "#FF7F00", "profile_text_color":
"#000000", "profile_use_background_image": true, "profile_images_url": "http://pbs.twimg.com/profile_images/1478436621128794625/3U0W0004_normal.jpg", "profile_
image_url_https": "http://pbs.twimg.com/profile_images/1478436621128794625/3U0W0004_normal.jpg", "default_profile_image": false, "following":
None, "follow_request_sent": None, "notifications": None, "withheld_in_countries": [], "geo": None, "coordinates": None, "place": None, "contributors":
None, "is_quote_status": false, "quote_count": 0, "reply_count": 0, "retweet_count": 0, "favorite_count": 0, "entities": {"hashtags": [], "urls": [], "user_mentions":
[]}, "screen_name": "Glennane Smith", "name": "Glennane", "id": 19152377, "id_str": "19152377", "indices": [0, 141], "symbols": [], "favorited": false, "retweeted":
false, "filter_level": "low", "lang": "en", "timestamp": "1651689711627")

```

[illegible]

```
[{"created_at": "Wed May 04 16:28:31 +0000 2022", "id": "1521809520261627004", "in_reply_to_status_id": "1521886520261627004", "text": "@SuzanneSmith From the first testment = \"\", \"display_text_range\": [15, 41], \"source\": \"@\" href=\"http://v.v/twitter.com/downloads/iphone\" rel=\"nofollow\" @#0x0e1d9ef7 for iPhone@#0c3c4ae083; \"truncated\": false, \"in_reply_to_status_id\": \"1521871374669817216\", \"in_reply_to_user_id_str\": \"1521871374669817216\", \"in_reply_to_user_id\": \"22652933\", \"in_reply_to_user_id_str\": \"22652933\", \"in_reply_to_screen_name\": \"@SuzanneSmith\", \"user\": {\"id\": \"22652933\", \"id_str\": \"22652933\", \"name\": \"Doc Tash Bernick\", \"screen_name\": \"@DRumark\", \"location\": \"\", \"url\": \"http://www.culturedermos.com\", \"description\": \"F&B, ABG, CPWA (she/her) Educator PAT #SocialJustice and #Equity Warrior \u2714\u2721\u2714\ufe0f #Mother #Wife and #ActiveCoul \u2714\u2714 #cisfic \u2714\u2714 #dcuff \u2714\u2714 #2000s \u2714\u2714 #ufef #Delarus \u2714\u2714 #alveaf New Jersey \u2714\u2714 #allveaf California\", \"translator_type\": \"none\", \"protected\": false, \"verified\": false, \"followers_count\": 698, \"friends_count\": 131, \"listed_count\": 11, \"favorites_count\": 12167, \"statuses_count\": 6164, \"created_at\": \"Tu e Mar 03 18:04:04 +0000 2005\", \"utc_offset\": null, \"time_zone\": null, \"geo_enabled\": true, \"lang\": null, \"contributors_enabled\": false, \"is_translator\": false, \"profile_background_color\": \"#E0E0E0\", \"profile_background_image_url\": \"http://v.v/assets/twing.com/images/themes/theme2/bg.gif\", \"profile_picture_background_image_url\": \"https://v.v/pbs.twimg.com/profile_images/1478436621129794625/3JdWNM04_normal.jpg\", \"profile_text_color\": \"#333333\", \"profile_use_background_image\": true, \"profile_image_url\": \"http://v.v/pbs.twimg.com/profile_images/1478436621129794625/3JdWNM04_normal.jpg\", \"profile_banner_url\": \"https://v.v/pbs.twimg.com/profile_banners/1478436621129794625/157464253\", \"default_profile\": false, \"default_profile_image\": false, \"following\": null, \"follow_request_sent\": null, \"notifications\": null, \"withheld_in_countries\": [], \"geo\": null, \"coordinates\": null, \"place\": null, \"contributors\": null, \"is_quote_status\": false, \"quote_count\": 0, \"reply_count\": 0, \"retweet_count\": 0, \"favorite_count\": 0, \"entities\": {\"hashtags\": [], \"urls\": [], \"user_mentions\": [{\"screen_name\": \"Suzanne Smith\", \"name\": \"Suzanne\", \"id\": \"19152377\", \"id_str\": \"19152377\", \"indices\": [0, 14]}]}, \"favorited\": false, \"retweeted\": false, \"filter_level\": \"low\", \"lang\": \"en\", \"timestamp_ms\": \"1651881711627\"}
```

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

```
1 import string
2 from pyspark.sql import SparkSession, Row
3 from pyspark.streaming import StreamingContext
4 from pyspark.streaming.kafka import KafkaUtils
5 from pyspark.sql.types import FloatType, StringType
6 from pyspark.sql.functions import to_date
7 import json
8 import re
9
10 emoji_pattern = re.compile("[
11     u'\U0001F600-\U0001F64F' # emoticons
12     u'\U0001F300-\U0001F5FF' # symbols & pictographs
13     u'\U0001F680-\U0001F6FF' # transport & map symbols
14     u'\U0001F1E0-\U0001F1FF' # flags (iOS)
15     u'\U00002702-\U000027B0'
16     u'\U000024C2-\U0001F251'
17     u'\U0001F926-\U0001F937'
18     u'\U00010000-\U00010fff'
19     u'\u280d'
20     u'\u2640-\u2642'
21     u'\u2600-\u2B55'
22     u'\u23cf'
23     u'\u23e9'
24     u'\u231a'
25     u'\u3030'
26     u'\ufe0f'
27     "]+", flags=re.UNICODE)
28
29 def getHashtags(rdd_collected):
30     list_hashtags = []
31     for element in rdd_collected:
32         print(element)
33         for hashtag in element:
34             if hashtag["text"] != None:
35                 list_hashtags.append(hashtag["text"])
36     return list_hashtags
37
38 def stripTextAndRemoveEmoji(text):
39     stringWithoutEmoji = emoji_pattern.sub(r'', text)
40     nameIsBlank = re.search('\s+$', stringWithoutEmoji)
41     if not stringWithoutEmoji or nameIsBlank:
42         return "No name"
43     return stringWithoutEmoji.strip()
44
45
46
47 def process(time, rdd):
48     print("----- ts -----" % str(time))
49     if not rdd.isEmpty():
50         print(rdd)
51         locationDF = rdd.map(lambda tweet: Row(username=stripTextAndRemoveEmoji(tweet['user']['name']),
52         sb_friends=tweet['user']['friends_count'])).toDF()
53
54         locationDF = locationDF.withColumn("username", locationDF["username"].cast(StringType())) \
55             .withColumn("sb_friends", locationDF["sb_friends"].cast(IntegerType()))
56
57         locationDF.printSchema()
58
59         locationDF.write.format('jdbc').options(
60             url='jdbc:mysql://192.168.33.10/data',
61             dbtable='users',
62             user='admin',
63             password='admin').mode('append').save()
64
65         rdd_collected = rdd.map(lambda tweet: tweet["entities"]["hashtags"]).collect()
66
67         if len(rdd_collected) >= 1:
68             hashtags_list = getHashtags(rdd_collected)
69             rdd_hashtags = sc.parallelize(hashtags_list)
70             if rdd_hashtags.isEmpty() == False:
71                 hashtagsDF = rdd_hashtags.map(lambda hashtag: Row(hashtag=hashtag)).toDF()
72                 hashtagsDF = hashtagsDF.withColumn("hashtag", hashtagsDF["hashtag"].cast(StringType()))
73                 hashtagsDF.printSchema()
74                 hashtagsDF.write.format('jdbc').options(
75                     url='jdbc:mysql://192.168.33.10/data',
76                     dbtable='hashtags',
77                     user='admin',
78                     password='admin').mode('append').save()
79
80
81
82 spark = SparkSession.builder \
83     .master("local[2]") \
84     .appName("data") \
85     .getOrCreate()
86
87
88
89 sc = spark.sparkContext
90 sc.setLogLevel("ERROR")
91 ssc = StreamingContext(sc, 10)
92
93 directKafkaStream = KafkaUtils.createDirectStream(ssc, ["tweets"], {"metadata.broker.list": "192.168.33.13:9092"})
94 rdd = directKafkaStream.map(lambda tweet: json.loads(tweet[1]))
95 rdd.foreachRDD(process)
96
97
98
99 ssc.start()
100 ssc.awaitTermination()
```

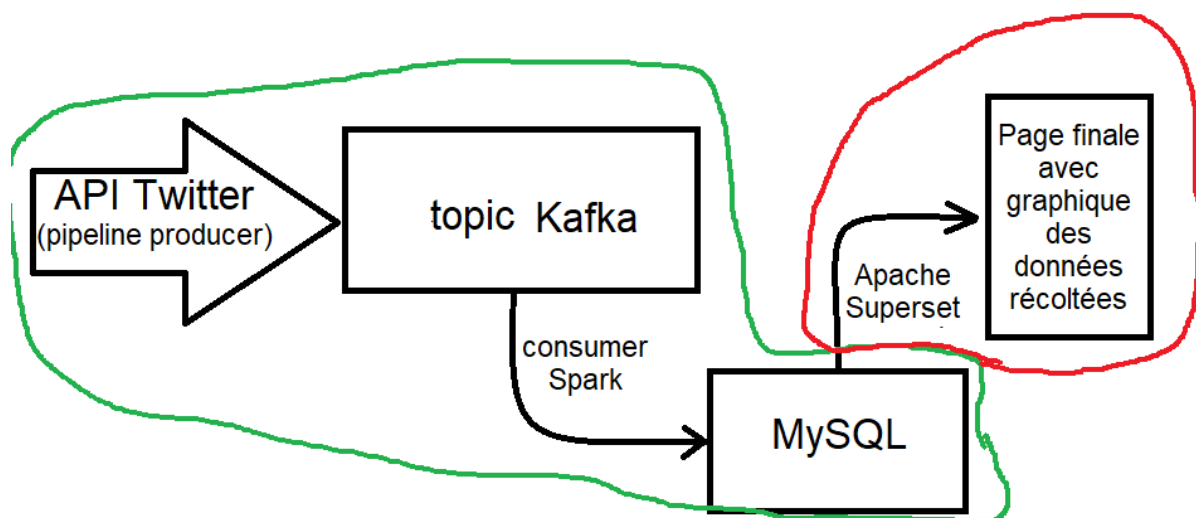
Après avoir installé le wget spark-streaming-kafka et le déplacer dans le bon répertoire (à savoir les jars de spark), nous récupérons les données sur le consumer spark avec la commande spark-submit :

```
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 :

Edit database

MySQL

MySQL\_data

BASIC

ADVANCED

HOST \*

127.0.0.1

PORT \*

3306

DATABASE NAME \*

data

Copy the name of the database you are trying to connect to.

USERNAME \*

admin

PASSWORD

\*\*\*\*\*

DISPLAY NAME \*

MySQL\_data

Pick a nickname for this database to display as in Superset.

ADDITIONAL PARAMETERS

e.g. param1=value1&param2=value2

Add additional custom parameters

☐ SSL

CLOSE

FINISH

Nous “convertissons” la base de données en dataset :

Add dataset

DATABASE

mysql MySQL\_data

SCHEMA

data

SEE TABLE SCHEMA

hashtags

CANCEL

ADD

The screenshot shows the Tableau interface for a data visualization. On the left, the 'data.hashtags' data source is selected. The 'Columns' shelf contains the calculated field 'COUNT(\*)'. The 'Visualization' type is set to 'Pie Chart'. The main view displays a pie chart with many segments, each representing a hashtag and its count. The largest segments are labeled 'GPG' and 'BTS'. The top right of the interface shows '221 rows' and a time filter set to '00:00:00.00'.

[illegible]