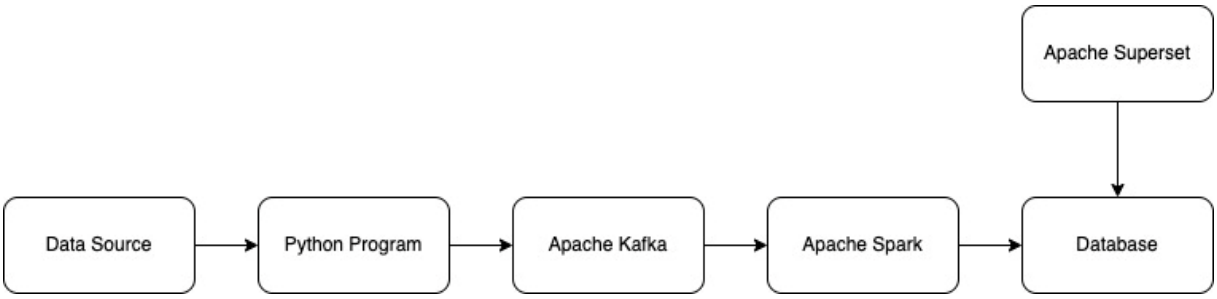
Aminata FADIGA
Nadeesha HAHARASINGHA
ITS2

Projet Data Mining

Le projet que nous devons réaliser consiste a mettre en œuvre la pipeline de traitement de data suivante :



Ainsi, nous avons choisi comme source de données un fichier csv contenant des données utilisateurs comme le nom, l'âge, la date de naissance ou encore la ville d'un utilisateur. Nous ouvrirons ce fichier puis l' enverrons à Kafka à travers un programme Python « Producer » qui nous permettra de les envoyer dans un bus Kafka.

Ensuite, dans un programme Spark, nous nous connecterons à Kafka afin de récupérer les données reçues dans le bus Kafka, ligne par ligne. Nous effectuerons ensuite plusieurs transformations sur les données et enfin, ces données modifiées finaux seront ensuite sauvegardées dans une table d' une base de données MySql puis affichées à l'aide de l'outil de visualisation Superset.

Afin de réaliser ce projet, nous avons suivi les étapes suivantes :

-Nous avons commencer par installer et initialiser trois machines virtuelles à l'aide de vagrant en utilisant les fichiers d'installation donnés et en utilisant la commande « vagrant up » :

- 1ère machine : Spark, on attribue l' adresse IP : 192.168.33.12
- 2ème machine : Kafka, on attribue l'adresse IP : 192.168.33.13. Pour démarrer Kafka, on lance les deux commandes suivantes dans deux sessions de terminaux différents :
    - `$ bin/zookeeper-server-start.sh config/zookeeper.properties`
    - `$ bin/kafka-server-start.sh config/server.properties`
    -

On y crée notre sujet « projet » avec la commande suivante :

- `$ bin/kafka-topics.sh --create --topic projet_datamining --bootstrap-server 192.168.33.13:9092`

Pour tester le fonctionnement directement dans un terminal, nous pouvons écrire dans le topic en tant que producer en lançant cette commande :

- `$ bin/kafka-console-producer.sh --topic projet_datamining --bootstrap-server 192.168.33.13:9092`

Et pour lire les données en tant que consumer, on lance cette commande :
- $ bin/kafka-console-consumer.sh --topic projet_datamining --from-beginning --bootstrap-server 192.168.33.13:9092

- 3ème machine : on y installe MySql ainsi que Superset et on attribue l'adresse IP : 192.168.33.14. Par ailleurs, on crée également notre base de données « projet » ainsi qu'une table « users » où nous enregistrerons nos données par la suite.

- Notre deuxième étape est de lier Spark et Kafka. Pour cela, on suit les commandes suivantes dans la machine virtuelle Spark:

On désactive la variable d'environnement PYSPARK_DRIVER_PYTHON :

```
root@vagrant:/home/vagrant# unset PYSPARK_DRIVER_PYTHON
```

On télécharge la librairie correspondante contenue dans le jar suivant :

```
root@vagrant:/home/vagrant# wget https://repo1.maven.org/maven2/org/apache/spark/spa
rk-streaming-kafka-0-8-assembly_2.11/2.4.7/spark-streaming-kafka-0-8-assembly_2.11-2
.4.7.jar
--2022-01-06 23:06:43--  https://repo1.maven.org/maven2/org/apache/spark/spark-strea
ming-kafka-0-8-assembly_2.11/2.4.7/spark-streaming-kafka-0-8-assembly_2.11-2.4.7.jar
Resolving repo1.maven.org (repo1.maven.org)... 151.101.120.209
Connecting to repo1.maven.org (repo1.maven.org)|151.101.120.209|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 11709363 (11M) [application/java-archive]
Saving to: 'spark-streaming-kafka-0-8-assembly_2.11-2.4.7.jar'

spark-streaming-kafk 100%[=====================>]  11.17M  22.4MB/s    in 0.5s

2022-01-06 23:06:44 (22.4 MB/s) - 'spark-streaming-kafka-0-8-assembly_2.11-2.4.7.jar
' saved [11709363/11709363]
```

Puis on le déplace dans le dossier local où se trouve les dépendances de spark :

```
root@vagrant:/home/vagrant# sudo mv spark-streaming-kafka-0-8-assembly_2.11-2.4.7.ja
r /usr/local/spark/jars/
```

- On suit les mêmes étapes pour la librairie permettant de lier mysql et spark.

➢ <u>Création du programme Pyhon :</u>

- Nous avons ensuite créer un programme Python « Producer » qui lit un fichier csv ligne par ligne et l'envoie dans le bus Kafka. Pour cela, nous nous sommes connecter à Kafka en indiquant l'adresse IP de la VM correspondante ainsi que le port et nous avons utiliser le code suivant :

Aminata FADIGA
Nadeesha HAHARASINGHA
ITS2

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Jan  7 01:15:44 2022

@author: nadee
"""


#Envoie un fichier json ligne ligne par ligne dans le bus Kafka
import json
from kafka import KafkaProducer

#On se connecte à la machine Kafka
producer = KafkaProducer(bootstrap_servers='192.168.33.13:9092', value_serializer=lambda v: json.dump

#On récupère chaque ligne du fichier json dans une liste
with open('data.csv', 'r') as f:
    listusers= f.readlines()


for i in listusers:
    user=i.strip()
    print(user)
    producer.send('projet_datamining', user)
```

On vérifie avec la commande Consumer pour lire les données citée ci-dessus :

```
vagrant@vagrant:~/kafka_2.13-3.0.0$ bin/kafka-console-consumer.sh --topic projet_datamining --from-beginning --bootst
rap-server 192.168.33.13:9092
"nom,age,genre,ville"
"Andrea,35,F,Paris"
"Alicia,22,F,Marseille"
"Julien,15,M,Nice"
"Laurent,37,M,Bordeaux"
"Sabrina,27,F,Paris"
"Kevin,15,M,Bordeaux"
"Lea,21,F,Nice"
"Michael,29,M,Paris"
"Andy,36,M,Grenoble"
"Justin,16,M,Marseille"
"Mich\u00c3\u00a8le,22,F,Paris"
"Andrea,16,F,Toulouse"
"Justine,17,F,Avignon"
"Mike,26,M,Paris"
"Nelson,30,M,Lyon"
"Joan,18,M, Paris"
"Michaela,24,F,Nantes"
"Anne,14,F,Montpellier"
"Justine,19,F,Strasbourg"
"Leo,39,M,Rennes"
```

- Création du programme Spark :

Nous avons ensuite créer le programme Consumer Spark qui consiste à se connecter à Kafka pour récupérer les données du fichier source csv qui ont été envoyés dans le bus Kafka puis à appliquer des transformations afin de trier et organiser les données reçues.
Pour cela, nous avons d'abord indiqué l'adresse IP de la machine virtuelle Kafka correspondante, ainsi que notre nom de topic « projet ». Pour chaque ligne de mots envoyé qui sont donc séparés des virgule car ils proviennent d'un fichier.csv, on transforme la chaine de caractères en liste de mots avec la commande : ligne.split(',')
.

Aminata FADIGA
Nadeesha HAHARASINGHA
ITS2

Enfin, on les sauvegarde dans la table « users » crée précédemment de la base de données MySql « projet » à l'aide du code suivant :

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.sql import SparkSession, Row
from pyspark.streaming.kafka import KafkaUtils


# Create a local StreamingContext with two working thread and batch interval of 1 second
sc = SparkContext("local[2]", "Kafka kWordCount")
sc.setLogLevel("ERROR")
ssc = StreamingContext(sc, 10)
ssc.checkpoint('.')


# Create a DStream that will connect to hostname:port, like localhost:9999
kds = KafkaUtils.createDirectStream(ssc, ["projet_datamining"], {"metadata.broker.list":
"192.168.33.13:9092"})

lines = kds.map(lambda x: x[1])

rdd = lines.map(lambda s: s.split(','))

# Print the first ten elements of each RDD generated in this DStream to the console
rdd.pprint()


def getSparkSessionInstance(sparkConf):
    if ("sparkSessionSingletonInstance" not in globals()):
        globals()["sparkSessionSingletonInstance"] = SparkSession \
            .builder \
            .config(conf=sparkConf) \
            .getOrCreate()
    return globals()["sparkSessionSingletonInstance"]

def process(time, rdd):
    print("========= %s =========" % str(time))
    # Get the singleton instance of SparkSession
    spark = getSparkSessionInstance(rdd.context.getConf())

    # Convert RDD[String] to RDD[Row] to DataFrame
    if not rdd.isEmpty():
        rowRdd = rdd.map(lambda w: Row(word=w))
        wordsDataFrame = spark.createDataFrame(rowRdd)

        # Creates a temporary view using the DataFrame
        wordsDataFrame.createOrReplaceTempView("words")
        # Do word count on table using SQL and print it
        wordCountsDataFrame = spark.sql("select word, count(*) as count from words group
by word")
        wordCountsDataFrame.write.format('jdbc').options(
            url='jdbc:mysql://192.168.33.10/spark',
            dbtable='wordcount',
            user='spark',
            password='password').mode('append').save()

#rdd.foreachRDD(process)


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

On teste ensuite dans Spark avec la commande suivante dans un terminal : /usr/local/spark/bin/spark-submit Consumer.py et on a le résultat suivant :

Aminata FADIGA
Nadeesha HAHARASINGHA
ITS2

```
-----------------------------------------
Time: 2022-01-07 02:48:50
-----------------------------------------


-----------------------------------------
Time: 2022-01-07 02:49:00
-----------------------------------------
['"nom', 'age', 'genre', 'ville"']
['"Andrea', '35', 'F', 'Paris"']
['"Alicia', '22', 'F', 'Marseille"']
['"Julien', '15', 'M', 'Nice"']
['"Laurent', '37', 'M', 'Bordeaux"']
['"Sabrina', '27', 'F', 'Paris"']
['"Kevin', '15', 'M', 'Bordeaux"']
['"Lea', '21', 'F', 'Nice"']
['"Michael', '29', 'M', 'Paris"']
['"Andy', '36', 'M', 'Grenoble"']
...
```

On teste maintenant que la base de données et la table est bien crée :

On teste maintenant l’affichage dans superset :