Projet Scala

Réalisé par :

- Bouchaara Oumayma.
- Idhika Oumaima.
- Tazzi Karim.

Encadré par :

- Ziyati El Houssaine.

Introduction

Spark streaming avec Scala est un projet réalisé dans le cadre de l'élément de module « Langage Scala » .On est arrivé à travers ce projet à mettre en pratique l'ensemble des outils et techniques présentés tout au long des séances .Dans ce sens, nous profitons ,tout d'abord, de l'occasion pour adresser nos grands respects a notre cher professeur « El Houssaine Ziyati »qui a toujours fait preuve de sa fidélité envers sa profession et son partage incessant envers ses étudiants .nous le remercions aussi pour le choix du projet. Un projet qui nous a permis de nous donner une image assez concrète de l'ensemble des notions déjà vues en classe mais aussi de nous lancer dans des recherches approfondies pour pouvoir donner naissance à un projet bien fait. Nous étions tout au début bien conscients que ce n'est pas trop facile de gérer de tels projets tous seuls, mais l'ensemble des conseils et des astuces procurés par notre cher professeur nous a bien aplati la tâche. Dans ce qui suit, on va présenter le projet, mettre l'accent sur les outils utilisés soit en termes de langages ou en termes de logiciels et expliquer le fonctionnement du code déployé.

Objectif du Projet

Le projet consiste à envoyer des données plutôt des logs à travers Spark streaming vers ElasticSearch et essayer de les visualiser en temps réel tout en utilisant Kibana. Nous avons essayé dans un premier temps d'utiliser un fichier plat et de visualiser les données, après nous nous sommes lancées dans le cote de l'envoi des données en temps réel en nous basant sur Spark streaming. Tout au long de ce projet, on s'est basé sur un ensemble d'outils qu'on va mettre en exergue dans ce qui suit.la figure suivante explique de manière simplifiée le projet de Spark

Les Outils

- ✓ **Langage Scala :** 

Scala est un langage de programmation multi-paradigme conçu à l'École polytechnique fédérale de Lausanne (EPFL) pour exprimer les modèles de programmation courants dans une forme concise et élégante. Scala intègre les paradigmes de programmation orientée objet et de programmation fonctionnelle, avec un typage statique. Il concilie ainsi ces deux paradigmes habituellement opposés et offre au développeur la possibilité de choisir le paradigme le plus approprié à son problème.

- ✓ **Apache Spark :** 

est un Framework open source de calcul distribué. Il s'agit d'un ensemble d'outils et de composants logiciels structurés selon une architecture définie.
Apache Spark est un moteur d'analyse unifié et ultra-rapide pour le traitement de données à grande échelle. Il permet d'effectuer des analyses de grande ampleur par le biais de machines de Clusters. Il est essentiellement dédié au Big Data et Machine Learning.

- ✓ **Kibana :** 

est une interface utilisateur gratuite et ouverte qui vous permet de visualiser vos données Elasticsearch et de naviguer dans la Suite Elastic.

- ✓ **Elasticsearch :** 

Elasticsearch est un moteur de recherche et d'analyse RESTful distribué, conçu pour répondre à une multitude de cas d'utilisation. Et leur liste ne cesse de s'enrichir. Véritable clé de voûte de la Suite Elastic, il centralise le stockage de vos données et assure une recherche ultra-rapide, une très grande pertinence et des analyses aussi puissantes que scalables.

- ✓ **Intellij IDA:** 

également appelé « IntelliJ », « IDEA » ou « IDJ » est un environnement de développement intégré (en anglais Integrated Development Environment - IDE) destiné au développement de logiciels informatiques reposant sur la technologie Java. Il est développé par JetBrains (anciennement « IntelliJ ») et disponible en deux versions, l'une communautaire, open source, sous licence Apache 2 et l'autre propriétaire, protégée par une licence commerciale. Tous deux supportent les langages de programmation Java, Kotlin, Groovy et Scala.

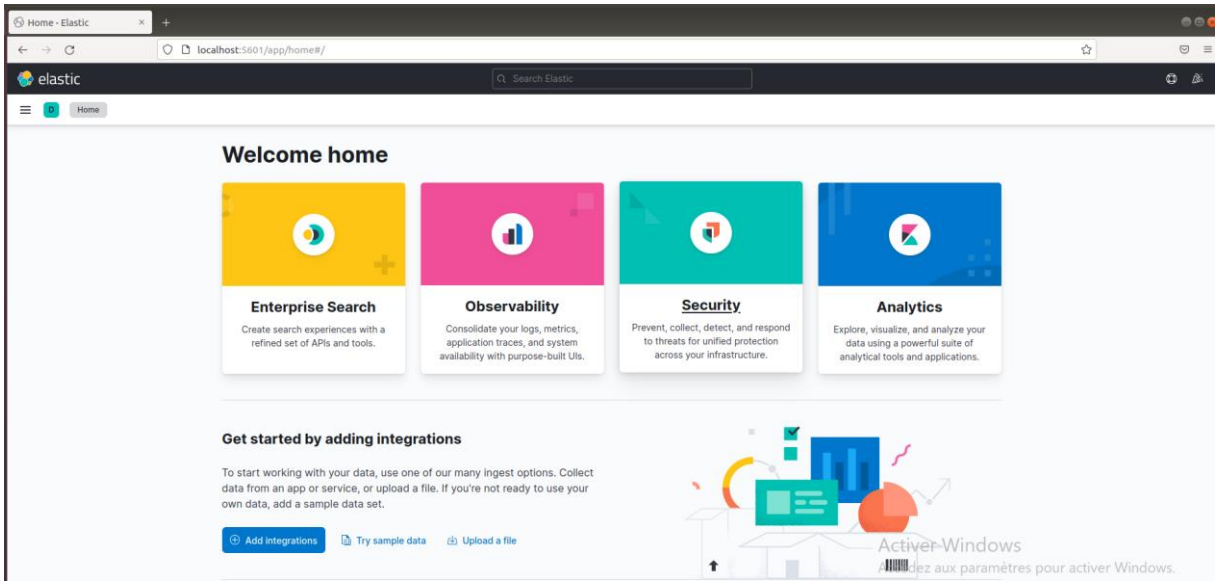
Fonctionnement

Pour mettre en œuvre ce projet, on est passe par un ensemble d'étapes comme le montre les lignes qui suivent.

Installation de Kibana Et ElasticSearch

Nous nous sommes tout d'abord lancés dans l'installation de Kibana et ElasticSearch qui nous ont permis de stocker les données et de les visualiser. Pour ce faire, nous nous sommes basés sur la documentation suivante https://phoenixnap.com/kb/how-to-install-elk-stack-on-ubuntu .

L'interface trouvée finalement est la suivante :



Utilisation d'un fichier plat

Nous avons préféré ne pas commencer directement par le Spark streaming et donc nous avons commencé tout d'abord par un fichier plat .Ceci nous a bien aidé a comprendre le fonctionnement et nous a facilité la tâche.

Nous avons exécuté le fichier log-generator.py pour un bout de moment ,chose qui a permis de remplir le fichier link.txt par quelques lignes ayant une structure bien précise. Nous avons récupéré les sites dans un fichier csv et les avons envoyé a ElasticSearch et puis la visualisation à l'aide de Kibana.

En termes de code, nous avons exécuté le fichier log-generator.py suivant :

```
#!/usr/bin/python
import time
import random

def buildURL():
    # build random URL
    url_list=['www.quicloud.com', 'www.cloudera.com', 'www.infiniteskills.com', 'flume.apache.org','hadoop.apache.org','console.amazon.com','www.globalknowledge.com
    return random.choice(url_list)

def buildPath():
    # build random Path
    path_list=['/index.html', '/contact.html', '/contact/submit.html', '/about.html','/products.html','/services.html','/support.html']
    return random.choice(path_list)

def buildHTTP():
    # build random HTTP code
    a=200
    b=599
    return random.randrange(a,b,1)
```

```python
def getHTTP():
    # build random HTTP status
    HTTP_status = [ 100, 101, 102, 200, 201, 202, 203, 204, 205, 206, 207, 226, 300, 301, 302, 303, 304, 305, 307, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409,
    return random.choice(HTTP_status)

def buildIP():
    # build random IP
    b1 = random.randrange(0, 255, 1)
    b2 = random.randrange(0, 255, 1)
    b3 = random.randrange(0, 255, 1)
    b4 = random.randrange(0, 255, 1)
    return str(b1) + '.' + str(b2) + '.' + str(b3) + '.' + str(b4)

log_File = open('link.txt', 'w')

count = 0
# infinite daemon
```

```python
while True:
  if(count > 1000):
    count = 0
    time.sleep(5); #sleep 5 seconds between writes
  else:
    #Http = buildHTTP()
    Http = getHTTP()
    count = count + 1
    Http = str(Http)
    Url = buildURL()
    Path = buildPath()
    Ip = str(buildIP())

    line = "HTTP " + Http + " " + Url + " " + Path + " " + Ip + "\n"
    #print(line)
    log_File.write(line)

log_File.close()
```

Ce code nous permet de remplir le fichier link.txt comme suit :

```
HTTP 205 www.globalknowledge.com /contact.html 234.23.118.93
HTTP 507 console.amazon.com /contact/submit.html 75.105.199.214
HTTP 407 www.cloudera.com /contact/submit.html 68.222.217.77
HTTP 410 www.gigaom.com /about.html 109.50.210.143
HTTP 204 flume.apache.org /support.html 206.182.165.50
HTTP 503 flume.apache.org /services.html 124.222.117.211
HTTP 500 www.globalknowledge.com /contact.html 159.48.27.230
HTTP 201 console.amazon.com /contact.html 47.50.61.67
HTTP 204 www.quicloud.com /about.html 146.85.117.171
HTTP 415 www.cloudera.com /contact.html 141.92.91.218
HTTP 100 hadoop.apache.org /contact/submit.html 144.102.170.17
HTTP 202 flume.apache.org /contact/submit.html 253.85.174.232
HTTP 304 www.globalknowledge.com /support.html 214.74.224.101
HTTP 422 hadoop.apache.org /support.html 171.83.36.231
HTTP 414 www.gigaom.com /products.html 67.181.24.16
HTTP 206 flume.apache.org /contact.html 111.102.171.128
HTTP 414 www.globalknowledge.com /services.html 221.127.214.28
HTTP 404 www.globalknowledge.com /contact/submit.html 247.97.46.105
HTTP 405 www.globalknowledge.com /index.html 113.87.12.66
HTTP 301 www.cloudera.com /services.html 197.95.97.91
```

On a essayé après de lire ce fichier et de récupérer les noms des sites dans la variable links :

```scala
def main(args: Array[String]) {

  // Set the log level to only print errors
  Logger.getLogger( name = "org").setLevel(Level.ERROR)

  // Create a SparkContext using every core of the local machine
  val sc = new SparkContext( master = "local[*]",  appName = "scalaproject")


  val lines = sc.textFile( path = "link.txt")


  val links = lines.map(x => x.split( regex = " ")(2))
```

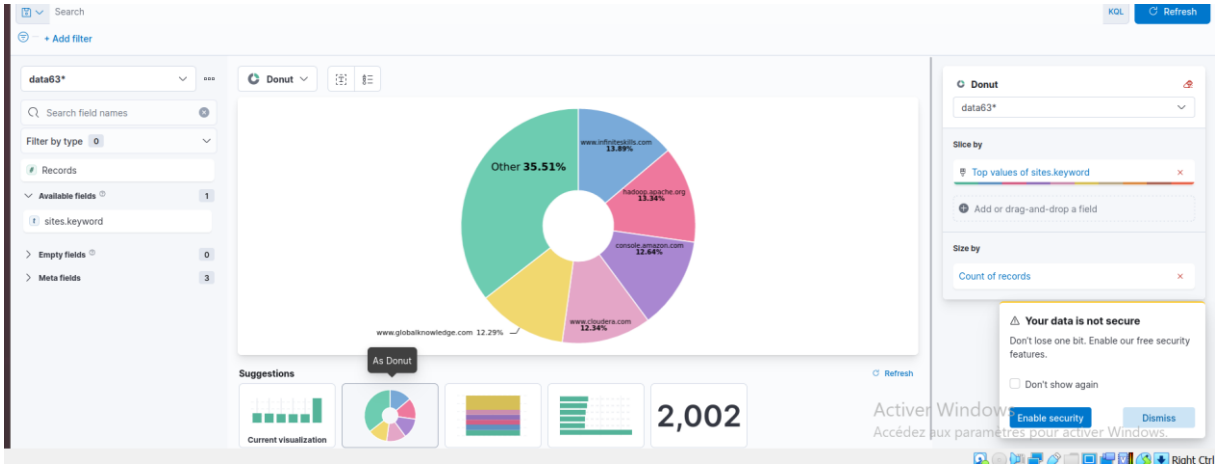
On a transformé les données stockées dans la variable links en un dataframe et on l'a écrit dans un fichier csv :

```scala
import spark.implicits._


val df = links.toDF( colNames = "sites")
df.coalesce( numPartitions = 1).write.option("header",true).format( source = "csv").save( path = "../datas/data63.csv")
```

On lit ce fichier csv et on l'envoie a ElasticSearch :

```scala
val lines1=spark.read.option("header","true").csv( path = "../datas/data63.csv");
lines1.write
  .format( source = "org.elasticsearch.spark.sql")
  .option("es.port","9200")
  .option("es.nodes","localhost")
  .mode( saveMode = "append")
  .save( path = "data63/_doc")
```

On visualise les données à travers Kibana :



Spark streaming

Maintenant on passe à l'objectif principal du projet, celui du Spark streaming. Donc, on a lancé l'exécution du fichier log-generator.py, ce qui permet de remplir le fichier link .txt de manière infinie, et on a essayé d'écrire ces données dans un fichier csv et de l'envoyer en temps réel vers elasticsearch pour les visualiser à l'aide de Kibana.

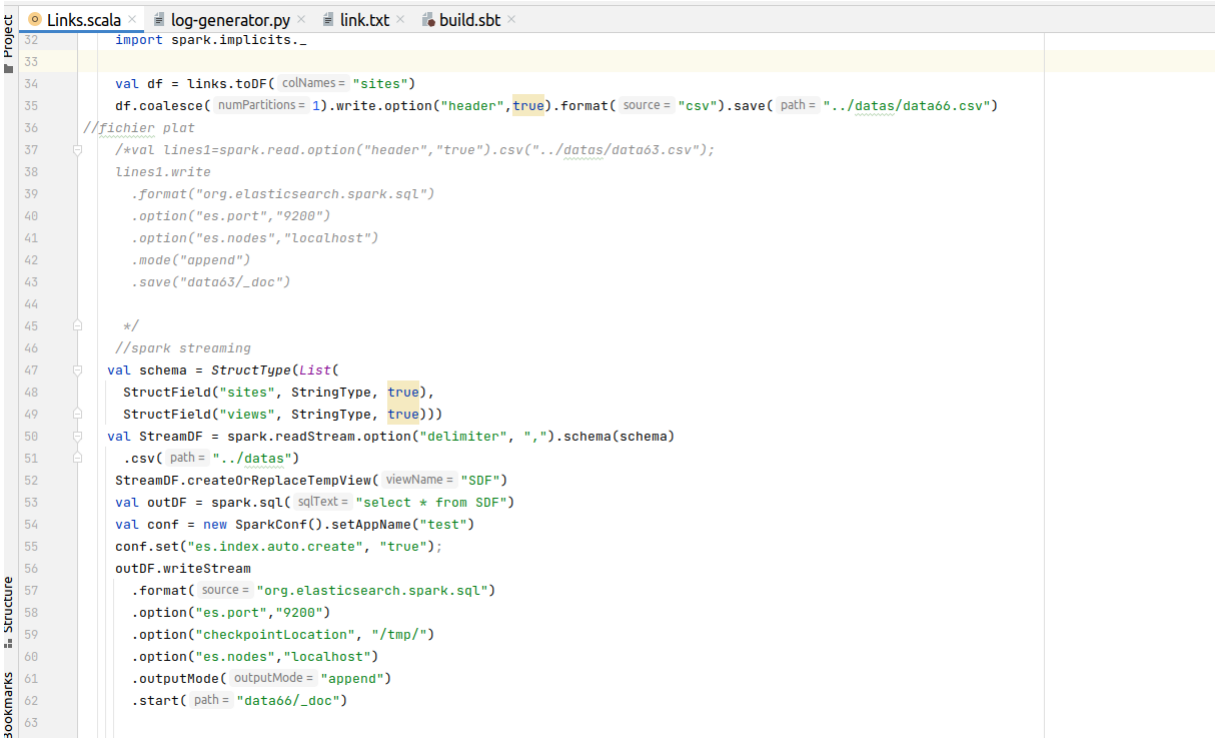
En termes de code, de même, on commence tout d'abord par exécuter le fichier log-generator.py de manière infinie, de récupérer les données mais cette fois-ci avec Spark streaming comme suit :

```scala
val schema = StructType(List(
  StructField("sites", StringType, true),
  StructField("views", StringType, true)))
val StreamDF = spark.readStream.option("delimiter", ",").schema(schema)
  .csv( path = "../datas")
StreamDF.createOrReplaceTempView( viewName = "SDF")
val outDF = spark.sql( sqlText = "select * from SDF")
val conf = new SparkConf().setAppName("test")
conf.set("es.index.auto.create", "true");
outDF.writeStream
  .format( source = "org.elasticsearch.spark.sql")
  .option("es.port","9200")
  .option("checkpointLocation", "/tmp/")
  .option("es.nodes","localhost")
  .outputMode( outputMode = "append")
  .start( path = "data66/_doc")
```

Le code entier qui résume la partie de l'utilisation d'un fichier plat et de celle du Spark streaming est le suivant :

Links.scala | log-generator.py | link.txt | build.sbt

```scala
import org.apache.spark._
import org.apache.spark.SparkContext._
import org.apache.log4j._
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types.{StringType, StructField, StructType}
import org.apache.spark.SparkConf
object Links{

  /** Our main function where the action happens */
  def main(args: Array[String]) {

    // Set the log level to only print errors
    Logger.getLogger( name = "org").setLevel(Level.ERROR)

    // Create a SparkContext using every core of the local machine
    val sc = new SparkContext( master = "local[*]",  appName = "scalaproject")


    val lines = sc.textFile( path = "link.txt")


    val links = lines.map(x => x.split( regex = " ")(2))


    val spark = SparkSession
      .builder
      .appName( name = "SparkSQL")
      .master( master = "local[*]")
      .getOrCreate()

```



Les dépendances

Les dépendances utilisées sont rassemblées dans le fichier build.sbt comme suit :

Conclusion

.

C'est un projet qui a fait appel à un ensemble d'outils. Il nous a bien permis d'assimiler beaucoup de notions comme le passage vers ElasticSearch ,la visualisation à travers Kibana, le Spark streaming. On s'est mis face à maintes reprises à un ensemble d'erreurs et on s'est trouvé aussi dans de vraies confusions mais à travers ces « blocages » qu'on est arrivé à apprendre et à tirer du bon profit a partir de ce projet qui est bel et bien un projet très intéressant

Bibliographie

- ✓ https://phoenixnap.com/kb/how-to-install-elk-stack-on-ubuntu
- ✓ https://spark.apache.org/docs/latest/streaming-programming-guide.html
- ✓ https://www.elastic.co/guide/en/elasticsearch/hadoop/current/spark.html
- ✓ https://www.youtube.com/watch?v=E_hpyO-0NoM
- ✓ https://www.elastic.co/fr/kibana/

Fin