This is a Tutorial on using apache spark to access twitter live streaming services for real time data analytics.

1) You need to have experience in programming a little bit and Apache Spark, and also you need to have experience already in Scala, but if you don't you can just download the code from GitHub and run the code just to see how it works

2) Login into twitter and access this URL to get your credentials for twitter streaming analysis. This is the link: https://apps.twitter.com/

3) Install the sbt package. Sbt is a building tool for scala, and java. I have the sbt 0.13 tools installed on my hortonworks sandbox. I am guessing you have the hortonworks sandbox also, so you can go ahead and follow my instructions on how to install it. For me, I have it in a directory that I created, and I really just do a sudo mv whenever I am creating a new project.
    a) In your terminal, do: mkdir sbt.
        i) cd sbt
    b) wget https://github.com/sbt/sbt/releases/download/v0.13.15/sbt-0.13.15.tgz
    c) tar xvf sbt-0.13.15.tgz

    Whenever you need to build a project, you can just move the sbt file into your project and build it.
4) We need to install Cassandra, I chose the Cassandra database, because it is built for scale architecture, that is capable of handling large amounts of data. You can read about Cassandra on datasax website, but I won't keep you waiting, lets go through the process of installing Cassandra. I like to use emacs when editing textfile so if you don't have emacs, do a simple, yum install emacs, to install it.
    a) emacs /etc/yum.repos.d/datastax.repo

    b) Add this text into the datastax.repo file
    ```
    [datastax]
    name = DataStax Repo for Apache Cassandra
    baseurl = http://rpm.datastax.com/community
    enabled = 1
    gpgcheck = 0
    ```

    Save the package and exit the text editor.

    c) sudo yum install dsc30

    d) Some people follow this steps to get it running,
    systemctl start Cassandra
    Systemctl status cassandra
    Systemctl enable Cassandra
    cqlsh

    That didn't work for me, so I did

Cassandra
cqlsh.

After a week of doing this, this stopped, working because of some versions dependency, so now I do,
Cassandra,
cqlsh 127.0.0.1 9042 --cqlversion="3.4.0"
It says my clash CQL spec version is 3.4.0

5) Create a new folder that you want to start the project in.

   a) Create a build.sbt file and copy this into it
   This is what my build.sbt file looks like, and yours will look similar to this. I will also be posting it on github, so you can pull this from github easily, no worries.
   if you are using a library that I do not have, please go to https://mvnrepository.com/artifact/org.apache.spark and add the librarydependencies into the build.sbt file, if you don't, you will definitely encounter terrible, terrible bugs, like I did, when I first started out.

```
lazy val root = (project in file("."))
 .settings(
  name := "apache-spark-ng",
  organization := "com.example",
  scalaVersion := "2.11.8",
  autoScalaLibrary := false,
  version := "0.1.0-SNAPSHOT"
 )

mergeStrategy in assembly <<= (mergeStrategy in assembly) { (old) =>
  {
  case PathList("META-INF", xs @ _*) => MergeStrategy.discard
  case x => MergeStrategy.first
  }
}

libraryDependencies += "org.apache.bahir" % "spark-streaming-twitter_2.11" % "2.1.0"
//libraryDependencies += "org.apache.spark" % "spark-streaming-twitter_2.11" % "1.5.2"
libraryDependencies += "org.apache.spark" % "spark-streaming_2.11" % "2.0.2"
libraryDependencies += "org.apache.spark" % "spark-core_2.11" % "2.0.2"
libraryDependencies += "org.apache.spark" % "spark-streaming_2.11" % "2.1.0"
libraryDependencies += "org.apache.spark" % "spark-tags_2.11" % "2.0.2"
libraryDependencies += "org.apache.spark" % "spark-sql_2.11" % "2.0.2"
libraryDependencies += "org.postgresql" % "postgresql" % "9.3-1102-jdbc41"
libraryDependencies += "org.apache.commons" % "commons-dbcp2" % "2.0.1"
libraryDependencies += "com.datastax.spark" % "spark-cassandra-connector_2.11" % "2.0.0-M3"
//libraryDependencies += "com.datastax.spark" % "spark-cassandra-connector-demos_2.11" % "1.0.6"
libraryDependencies += "com.datastax.spark" % "spark-cassandra-connector-embedded_2.11" % "2.0.1"
libraryDependencies += "com.datastax.cassandra" % "cassandra-driver-core" % "3.2.0"
```

   b) Create a directory called project
   i) create a folder called assembly.sbt
   ii) it should look like this

```
resolvers += Resolver.url("sbt-plugin-releases-scalasbt", url("http://repo.scala-sbt.org/scalasbt/sbt-plugin-releases/"))

addSbtPlugin("com.eed3si9n" % "sbt-assembly" % "0.12.0")
```

c) still in the project folder right? Create a file called build.properties
   i) type in “sbt.version=0.13.15”
   ii) we are done with the project folder, get out of it, and back into your project main folder

d) remember the sbt file we downloaded and unpacked? mv the sbt folder into your folder.

6) Now the coding part.

a) create a file called Tutorial. Scala.

i) Import the Library

```
import org.apache.spark._
import org.apache.spark.SparkContext._
import org.apache.spark.streaming._
import org.apache.spark.streaming.twitter._
import org.apache.spark.streaming.StreamingContext._
import TutorialHelper._
import org.apache.spark.sql._
import org.apache.spark.sql.SQLContext
import org.apache.spark.SparkConf
import java.sql.Connection
import java.sql.Statement
import org.apache.commons.dbcp2._
import com.datastax.spark.connector.cql.CassandraConnector
import com.datastax.spark.connector.util.Logging
import com.datastax.spark.connector.embedded._
import com.datastax.spark.connector._
import com.datastax.spark.connector.streaming._
```

ii) Okay I have a confession to make, This is my first time trying out Scala, but I decided to do this in Scala anyways, since spark was built in Scala.

b) create a singleton object called Tutorial, with an object called main. I came from a java background, so everything is a little bit different, but almost seems the same.
in Java we would do something like this:

```
public static void main(String[] args){
    ...
}
```

but in scala guess what?

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

    //The sparkconf stores configurationg paramaters that the spark driver passes to spark context
    val conf = new SparkConf().setMaster("local[2]").setAppName("Tutorial").set("spark.cassandra.connection.host", "127.0.0.1")
```

d) in the next line, I declared a variable called conf. The conf variable holds the parameter configurations for our applications. On my application, I am running on two threads, also set the App name which I called Tutorial, and I also set my Cassandra database connection. We actually do not have to use a database, so if you don't want it, you are free to take the database parameters configurations out.

e) Next, I created a variable and called it sc, which is the sparkcontext, and passed the spark configuration to the spark context. I will advise reading https://spark.apache.org/docs/latest/configuration.html to understand what all of these does. These are spark properties needed to configure spark

```
//creating a spark context. A spark context allows your spark driver to access the cluster through a resource manager
val sc = new SparkContext(conf)
```

f) I believe if you want to use the spark-sql, you will have to use something like this.

```
//creating a sqlcontext by passing it to an existing spark context
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
```

I am using Cassandra because I also wanted to learn about it, so feel free to pick your poison.

g) Next, set the configuration parameters for Cassandra, and pass it to a spark configuration.

```
  /** Creates the keyspace and table in Cassandra. */
CassandraConnector(conf).withSessionDo { session =>
session.execute(s"DROP KEYSPACE IF EXISTS demo")
session.execute(s"CREATE KEYSPACE IF NOT EXISTS demo WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor': 1}")
session.execute(s"CREATE TABLE IF NOT EXISTS demo.wordcount (word TEXT PRIMARY KEY, count COUNTER)")
session.execute(s"TRUNCATE demo.wordcount")
}
```

If you understand mysql, you should understand what I am doing here, if you don't, please check the datastax documentation. It helps.
or check out this link
http://gettingstartedwithcassandra.blogspot.com/2011/06/cassandra-keyspaces-what-are-they.html

Having used mysql while programming in Php, I would say keyspace is equivalent to the name of a database, we also create table in Cassandra, and they are called Tables. The replication factor here refers to the number of nodes that act as copies to each row of data. When you run your application, it displays this

```
17/06/10 07:32:16 INFO TableWriter: Wrote 0 rows to demo.wordcount in 0.001 s.
17/06/10 07:32:16 INFO TableWriter: Wrote 1 rows to demo.wordcount in 0.011 s.
-------------------------------------------
Time: 1497079936000 ms
```

Here for each row of data written to the database, the number of nodes that the replication factor refers to is 1. Read the link very well to get a better understanding of how Cassandra works.

h) Next I declared variables to store the twitter api informations. So get your keys from twitter Api, and assign them to each variable.

i) I declared a variable, called a streamingContext method. Streaming context is the entry point for spark streaming context, it takes in the different sparkconf and SC parameters configurations that we created earlier. It has a batch interval of four seconds. You can change it to whatever you want, and just check to see how long they take. You can also read more about it on the Apache spark website.

```
al ssc = new StreamingContext(sc, Seconds(4))
val tweets = TwitterUtils.createStream(ssc, None, Array("Boko", "Haram", "BokoHaram", "Boko Haram"))
```

Also created a variable called tweets, and we assign it to a Twitter Library called TwitterUtils. Twitter Util creates an input stream that returns tweets received from Twitter with the authentication we provided. You can read more about it on Apache Spark Website. Let me explain the keywords in the Array lol. So I decided to filter it. Boko Haram is a terrorists organization terrorizing my country right now, so I am filtering it by that, and I will like to do analysis based on that. Semantic Analysis. Twitter does have a sentiment Analysis libraries, but yeah, I do not think it has been trained to do something like this, so I am working on mine, and I will post this in the coming weeks once I am done with this. It is a lot of learning curve, and I am just not there yet.

j) Saved it Cassandra

```
.saveToCassandra( "demo",  "wordcount", SomeColumns("word", "count"))
```

1) Things to improve on
   a) Operate on each element in the rdd with foreach.
      i) count how many word in each elements.
      ii) save it into the database.
      iii) And whatever computation you want to do for each elements.
      iv) you would do something like:

```
tweets.foreach{ word => {
                        "whatever you want to do"
                        }}
```

      v) you could get each word that starts with @, and save it in another column as usernames. People being mentioned or retweeted. I did this in my other example, which I haven't written the documentation yet, and that is also in python.
      vi) if you want to do something like count the occurrence of each word in a row, you can do something like this. Ignore the first line, since we already have an existing rdd. I don't know if I am 100% correct, but we can only use a transformation or action on a rdd.

```
val myLines =  sc.textFile("hdfs://sandbox.hortonworks.com/tmp/Tutorial/Helloworld.txt").flatMap(_.split(" "))
val wordCounts = myLines.map((_, 1)).reduceByKey(_ + _)
val count = wordCounts.collect()
print(count)
```

Once you are done and ready to run your spark job, Follow these steps

a) Move out from the twitter directory to root
b) cd /usr/hdp/current/spark2-client
c) change spark version to version 2. Export SPARK_MAJOR_VERSION=2
d) go back into the twitter directory,
e) run sbt assembly, and you jar file is ready for launching. The jar file is created according to my build.Sbt file structure.
f) start Cassandra
g) spark-submit --master local[8] target/scala-2.11/apache-spark-ng-assembly-0.1.0-SNAPSHOT.jar Tutorial.scala

There you have it, a live spark streaming, which analysis can be done, semantic analysis, and so forth. My next tutorial that I am working on will be going into the Cassandra database, getting each tweets, and building a live semantic analysis on each tweets. I started already on this, and I am actually building it using PYSPARK. So yes, get ready for a little bit of python. I am still in the process of studying how sentiment analysis works. and I am currently reading Bing Liu
Book on “Sentiment Analysis and opinion mining.” Let me share some screenshots from my next work.

```
class userInterface():
        def __init__(self):
            value = []
            self.value = value

        def getContext(self):
            df = sqlContext.read\
                .format("org.apache.spark.sql.cassandra")\
                .options(table="wordcount", keyspace= "demo")\
                .load()
            df.select("word")
            return df

        def splitTweets(self):
            df = u.getContext()
            df.createOrReplaceTempView("tweets")
            results = sqlContext.sql("SELECT word FROM tweets")
            resultsDF = results.rdd.map(lambda p: p.word)
```

Thanks to https://zeppelin.apache.org/docs/0.6.1/quickstart/tutorial.html for the configureTwitterCredentials method in the TutorialHelper singleton object file.