REAL TIME STREAMING

WITH Apache Spark

Twitter Sentiment Analysis

CSYE 7245: Big Data Systems and Intelligence Analytics
Prof. Krishnamurthy

Presented By
#AshwinDinoriya

Introduction

The main concept behind creating this project is to implement Apache Spark Streaming in combination with IBM Watson to perform sentiment analysis and track how a conversation is trending on Twitter.

Sentiment Analysis is imperative because using sentiment analysis on the largest focus group in the world (social media) you can gain a lot of value insights to help you make better decisions in your market strategy and competitiveness. Users would like to know which separate topics are talked about in the text, which of them are positive and which are negative. So I suppose there will be a trend towards greater use of NLP techniques (such as syntactic parsing, co-reference resolution, etc.), in addition to machine learning.

Objective

The goal of this project is to perform sentiment analysis on Twitter data for extracting sentiment insights to analyze Emotion attached to the tweet. The sentiment analysis can be useful for:

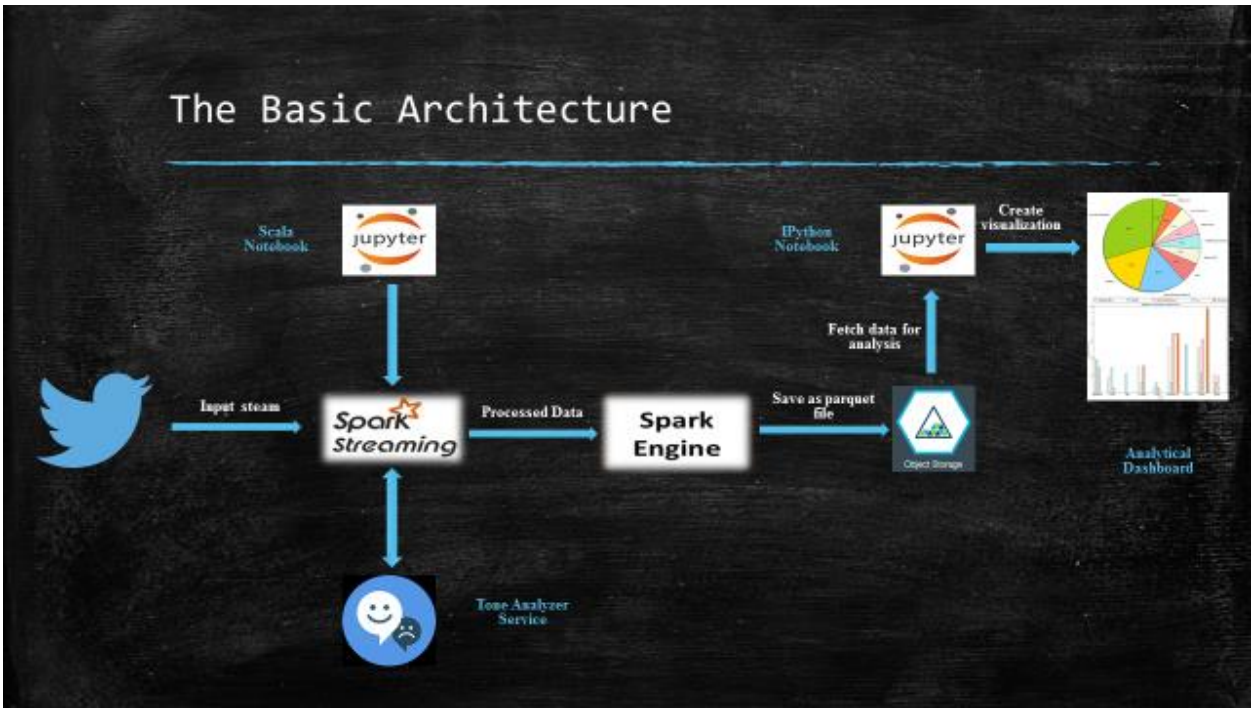
- Searching trending discussion on Twitter using hashtags to analyze opinion/emotions/moods of people by performing sentiment analysis.
- Real time analysis for most up to date insights (near real time actually)
- Detailed analysis of sentiments for better understanding, that's why Emotion based sentiment analysis

Following are some of the targets achieved through this project implementation:

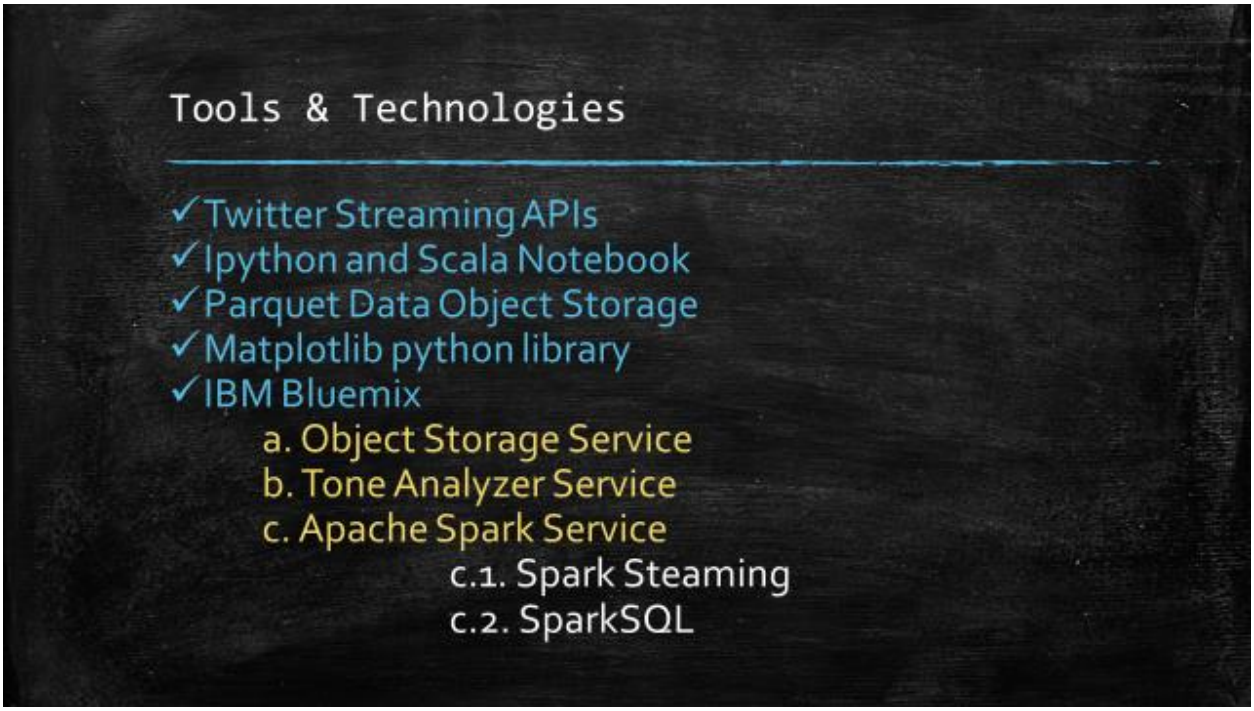
- Compute the distribution of tweets by sentiment scores
- Compute the top 10 hashtags contained in the tweets
- Visualize aggregated sentiment scores for the top 5 hashtags

Architecture

The following figure explains the basic architecture of this project. It includes source of data, processing unit, streaming technologies, analyzer services, storage service and creation of dashboard using tools and APIs.



Tools and Technologies



Twitter API: It is the source of the system. We are collecting real time tweets using API service provided by Twitter.

Spark Steaming: Breakdown the Streaming data into smaller pieces which are then sent to the Spark Engine

Tone Analyzer: It is accepting tweets as an input then using its own algorithm it is processing tweets and enriching it by providing sentiments involved in each tweet as output to Spark Streaming service.

Scala Notebook: With the help of Scala notebook, we are providing credentials to twitter and tone analyzer and performing simple analysis.

Spark Engine: It is actually helping us to create DataFrames from RDDs processed from Spark Streaming and making it ready to use for analysis.

Object Storage: This is a service provided by IBM Bluemix to store data easily using Open Stack swift implementation service. We have stored data here in parquet file format. Parquet is a columnar database.

IPython Notebook: This tool is used to create different analysis by converting parquet files into DataFrames and then DataFrames into RDDs and then creating visualizations.

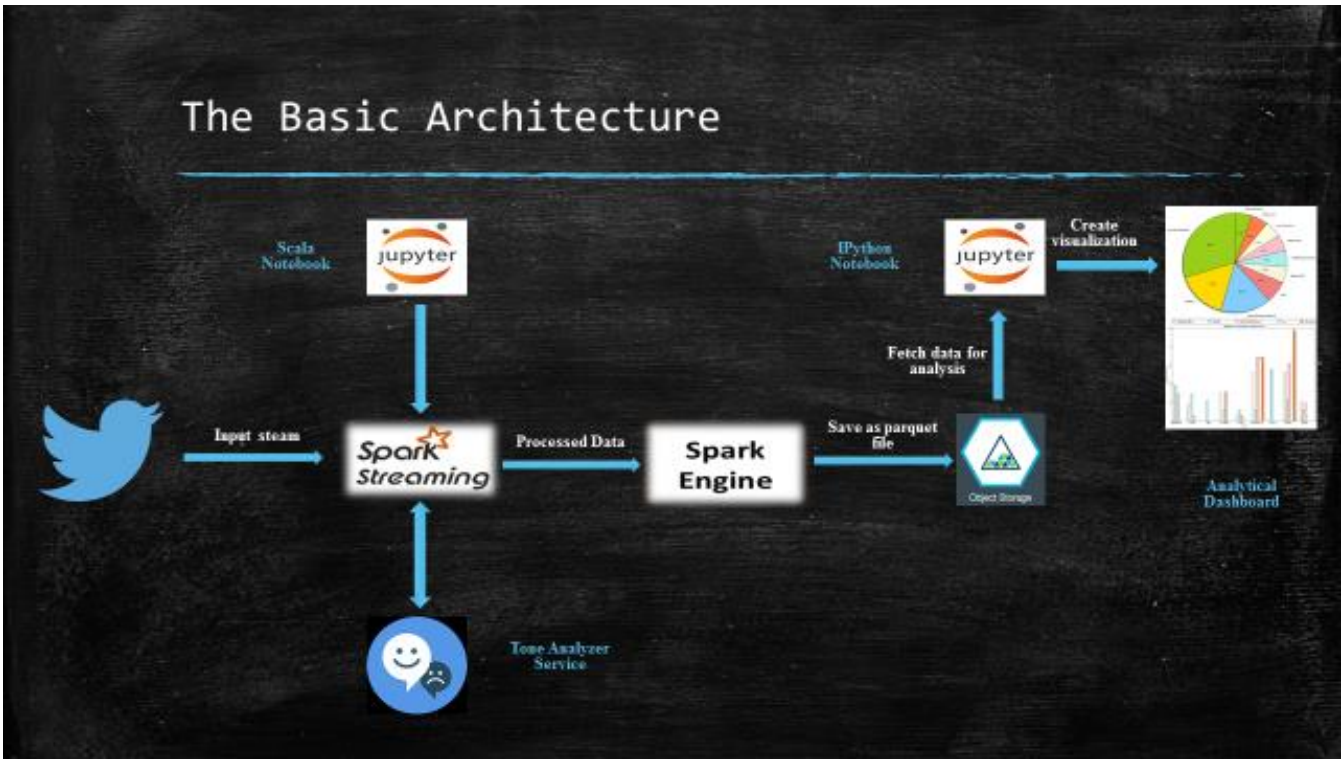
Analytical Dashboards: We have created dashboards to show our analysis report using ‘matplotlib’ library provided in Python.

Implementation

How it works

This project uses Spark Streaming to create a feed that captures live tweets from Twitter. We can optionally filter the tweets that contain the hashtag(s) of our choice. The tweet data is enriched in real time with various sentiment scores provided by the Watson Tone Analyzer service (available on Bluemix). This service provides insight into sentiment, or how the author feels. We then use Spark SQL to load the data into a DataFrame for further analysis. We can also save the data into a Cloudant database or a parquet file and use it later to track how you're trending over longer periods.

The following diagram shows the basic architecture of this app:



Build the application

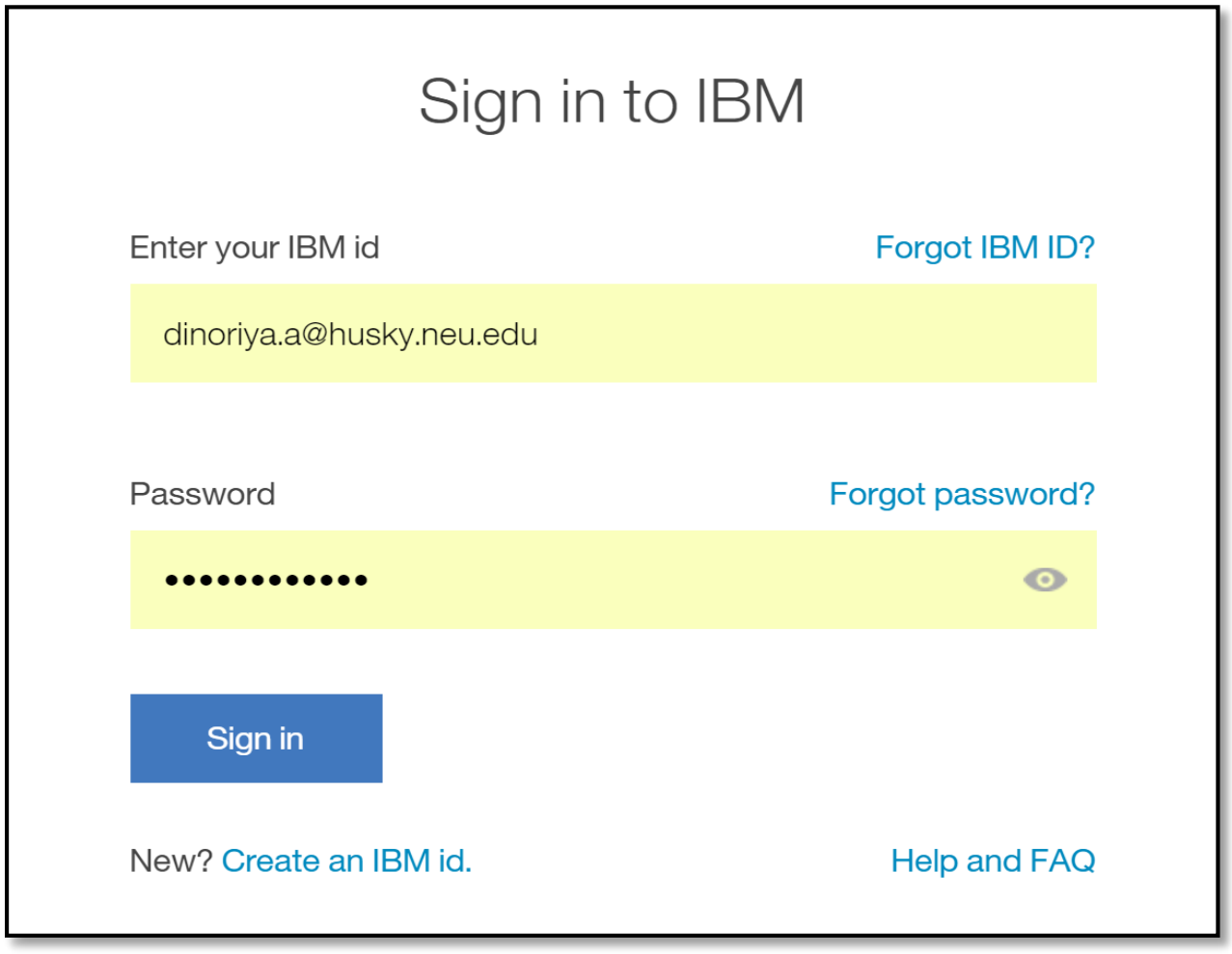
1. Clone the source code on local machine:
→ **git clone https://github.com/ibm-cds-labs/spark.samples.git**

2. Go to the sub-directory that contains the code for this application:
→ **cd streaming-twitter**

3. Compile and assemble the jar using the following command to create an uber jar (jar contains the code and all its dependencies):
→ **sbt assembly**

4. Post the jar on a publicly available url, by uploading the jar into a github repository

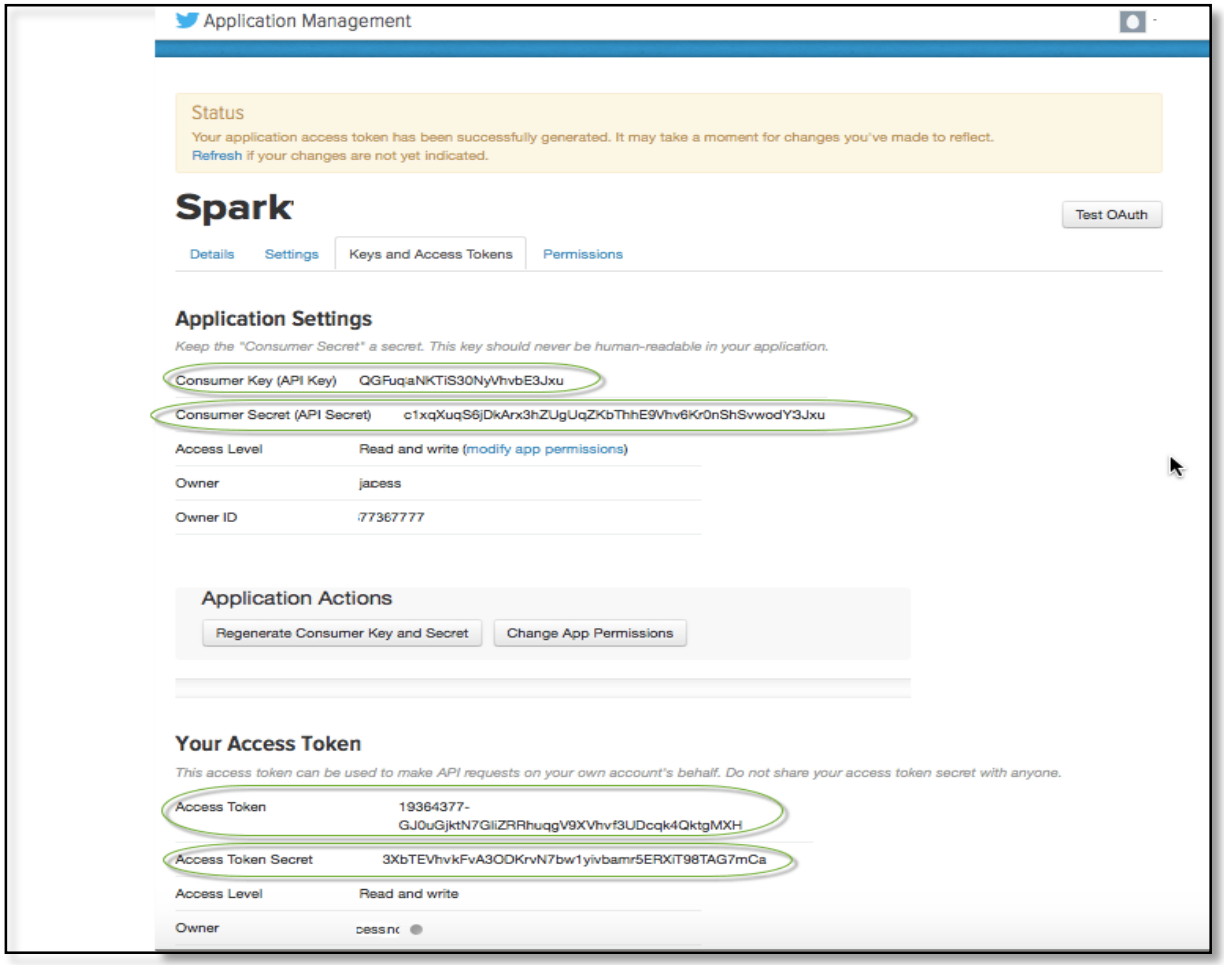
Create IBM Bluemix Account



Connect to Twitter

Create a new app on Twitter account and configure the OAuth credentials.

1. Go to https://apps.twitter.com/ . Sign in and click the **Create New App** button
2. Complete the required fields:
   - **Name** and **Description** can be anything you want.
   - **Website.** Enter any valid URL.
3. Below the developer agreement, turn on the **Yes, I agree** check box and click **Create your Twitter application**.
4. Click the **Keys and Access Tokens** tab.
5. Scroll to the bottom of the page and click the **Create My Access Tokens** button.
6. Copy **Consumer Key**, **Consumer Secret**, **Access Token**, **Access Token Secret**.

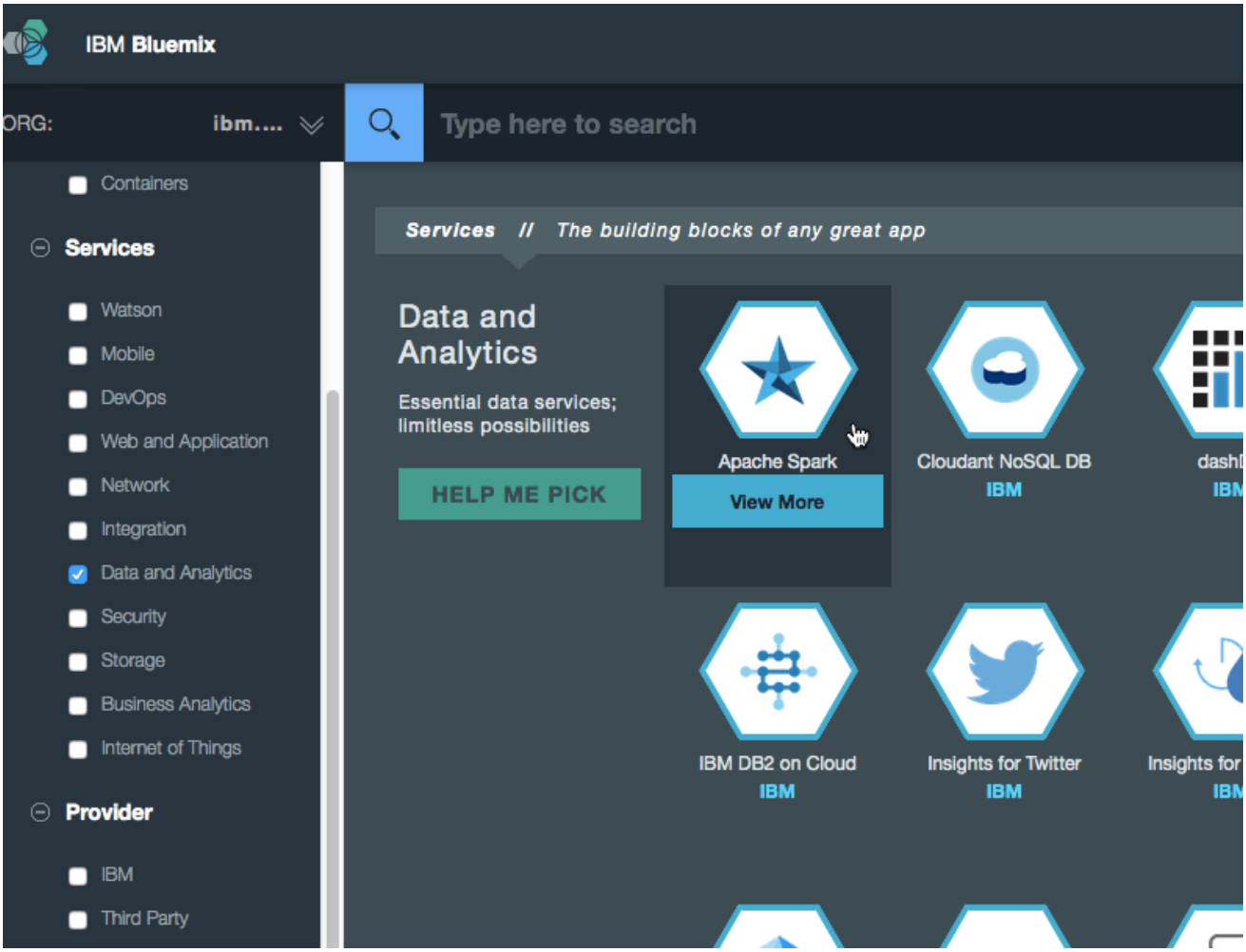


Initiate and run services on Bluemix

1. On Bluemix, initiate the IBM Analytics for Apache Spark service.

   a. In the top menu, click **Catalog**.

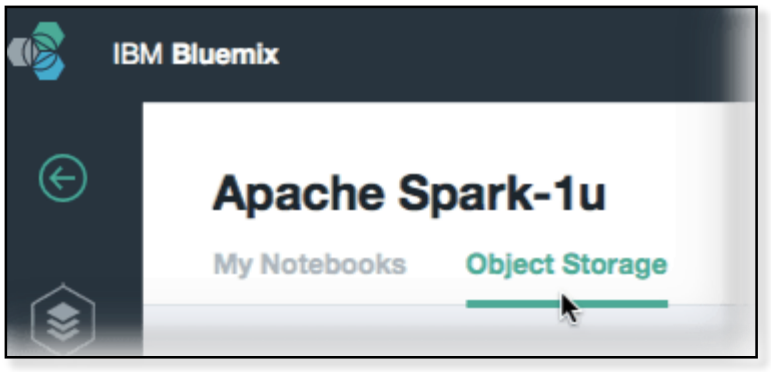
   b. Under **Data and Analytics**, find **Apache Spark**.



   c. Click to open it, and click **Create**.

   d. Click **Open**.

   e. Click the **Object Storage** tab.



f. Click the **Add Object Storage** button and click **Create**.

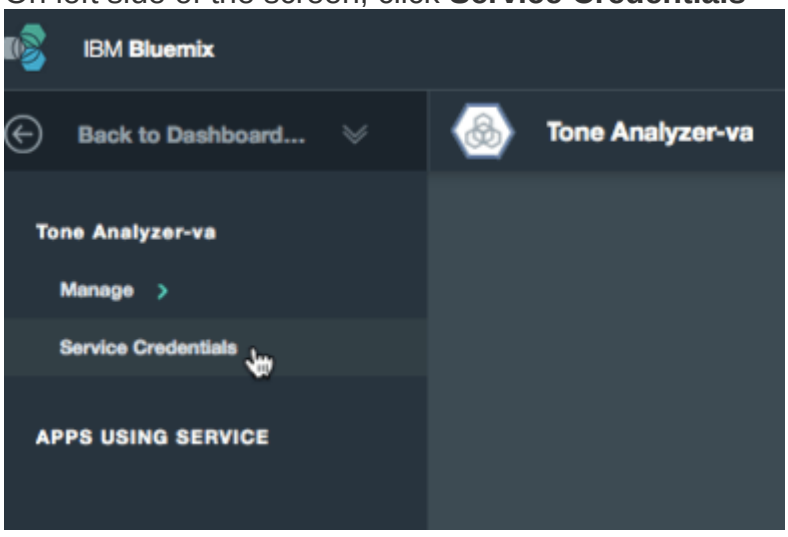
2. Initiate the Watson Tone Analyzer service too. To do so:

. In Bluemix, go to the top menu and click **Catalog**.

a. Scroll down to the bottom of the page and click the **Bluemix Labs Catalog** link.

b. Select the Tone Analyzer service and click **Create**.

3. On left side of the screen, click **Service Credentials**



4. Copy the information (you'll need it later when running the app in a Notebook):

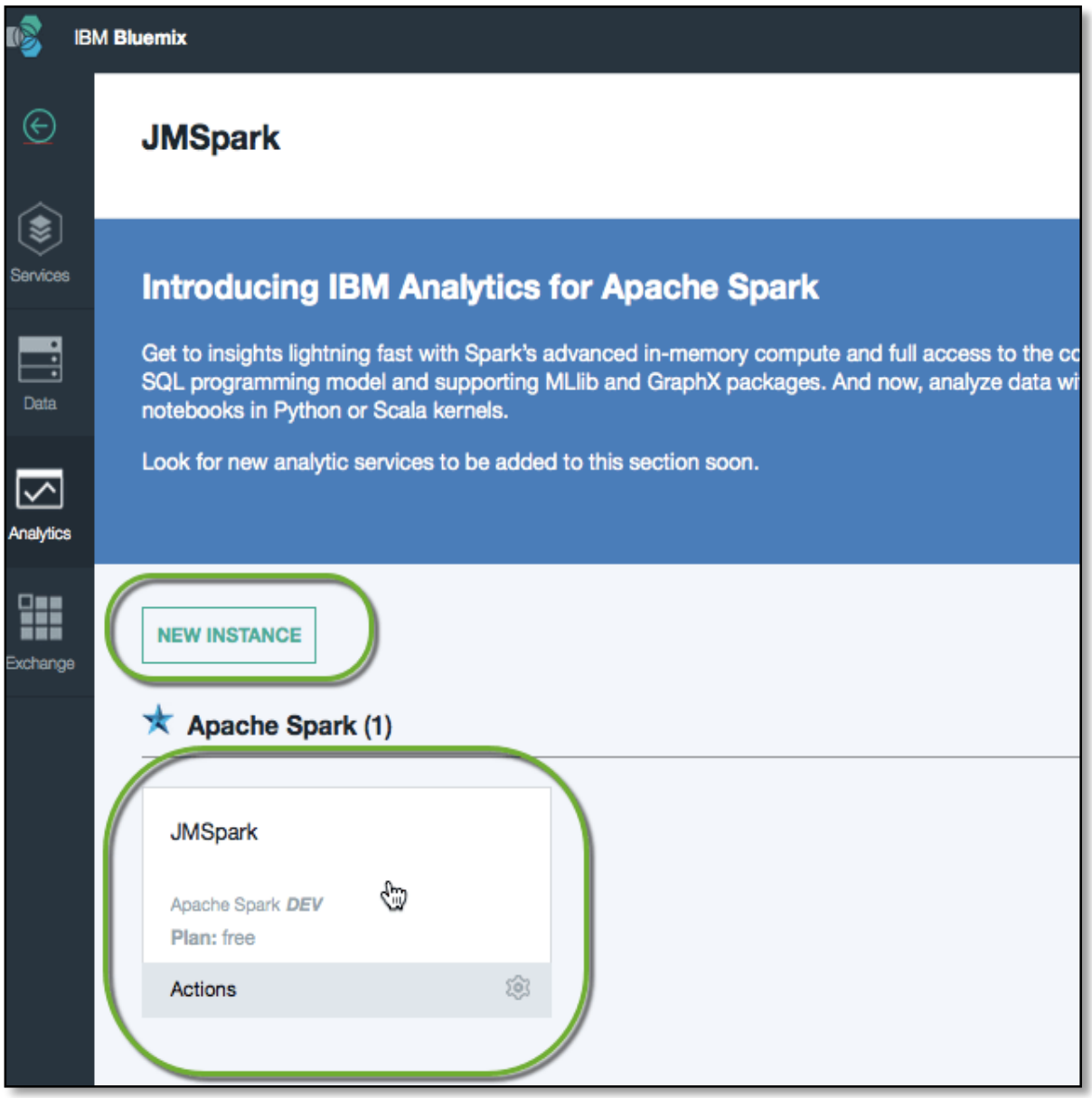
```
  "credentials": {
 "url":"XXXXX",
 "username":"XXXXX",
 "password":"XXXXX"
}
```

5. On the upper left of the screen click **Back to Dashboard**.

6. Create a new Scala notebook.

. In Bluemix, open your Apache Spark service.

a. If prompted, open an existing instance or create a new one.



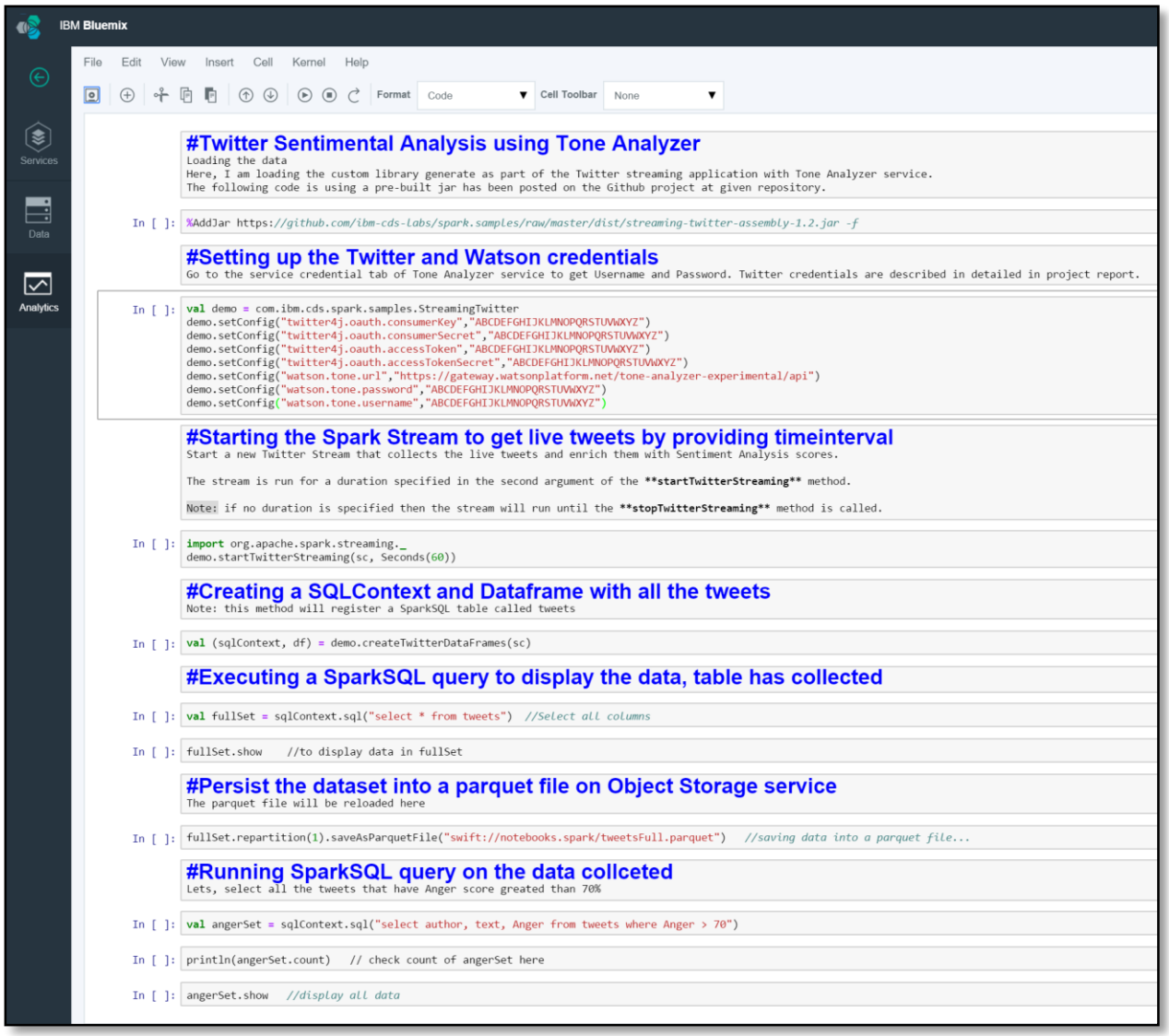
b. Click **New Notebook**.

c. Enter a **Name**, and under **Language** select **Scala**.

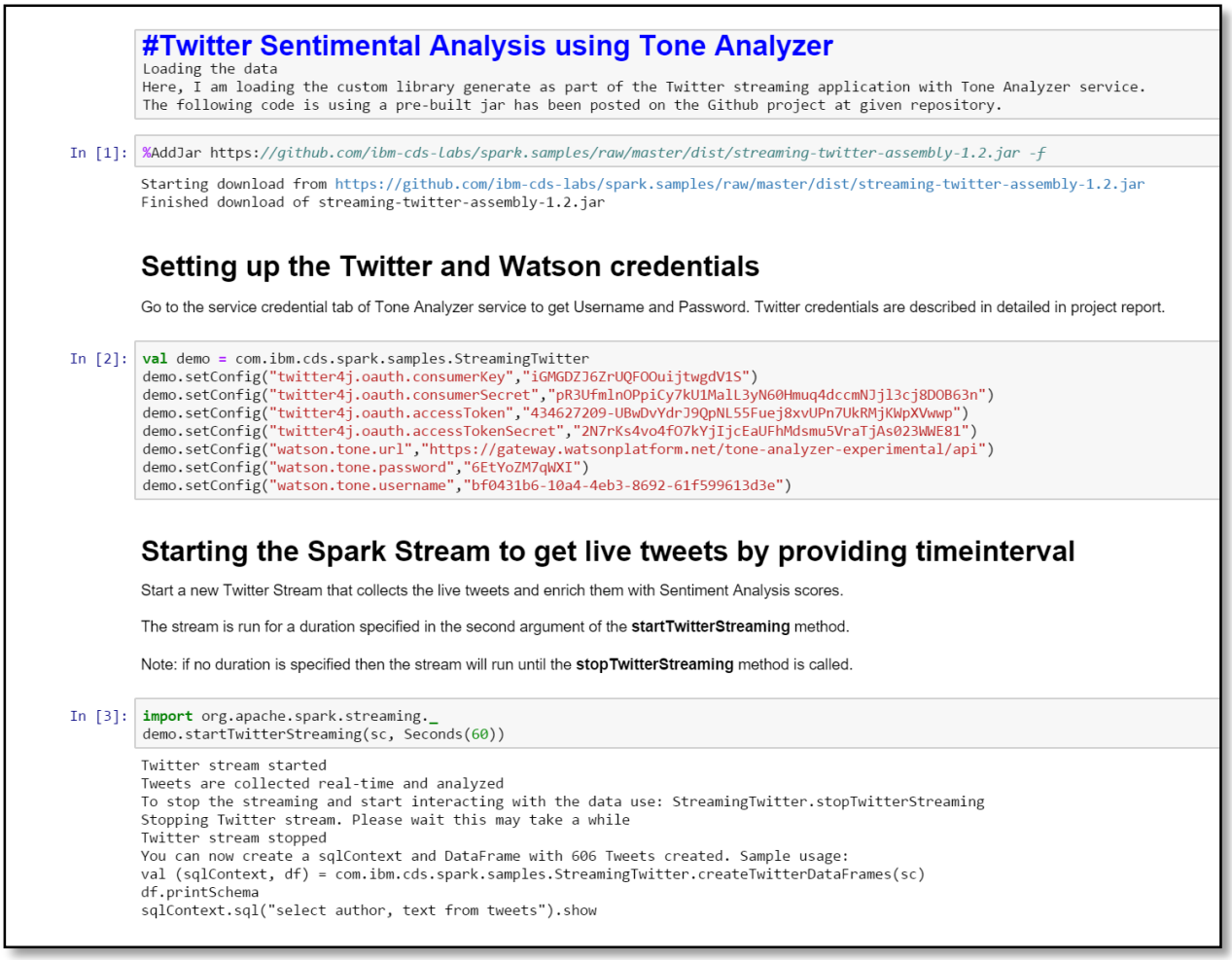
d. Click **Create Notebook**.

PART I: Collecting, processing, storing and retrieving Tweets using Scala Notebook

In this section, using a Scala Notebook, I am going to implement how to run the Twitter Stream to acquire data and enrich it with sentiment scores from Watson Tone Analyzer. I also will execute a command to persist the data in a parquet file on the Object Storage bound to this Spark instance.



Here is explanation and output of each and every step involve is collecting tweets using Twitter APIs and enriching it using Watson Tone Analyzer service. Tone analyzer is going to enrich Twitter data by analyzing it and giving different sentiments as output to input stream.



#Twitter Sentimental Analysis using Tone Analyzer

Loading the data

Here, I am loading the custom library generate as part of the Twitter streaming application with Tone Analyzer service.
The following code is using a pre-built jar has been posted on the Github project at given repository.

In [1]:
```
%AddJar https://github.com/ibm-cds-labs/spark.samples/raw/master/dist/streaming-twitter-assembly-1.2.jar -f
```

```
Starting download from https://github.com/ibm-cds-labs/spark.samples/raw/master/dist/streaming-twitter-assembly-1.2.jar
Finished download of streaming-twitter-assembly-1.2.jar
```

Setting up the Twitter and Watson credentials

Go to the service credential tab of Tone Analyzer service to get Username and Password. Twitter credentials are described in detailed in project report.

In [2]:
```
val demo = com.ibm.cds.spark.samples.StreamingTwitter
demo.setConfig("twitter4j.oauth.consumerKey","iGMGDZJ6ZrUQFOOuijtwgdV1S")
demo.setConfig("twitter4j.oauth.consumerSecret","pR3UfmlnOPpiCy7kU1MalL3yN60Hmuq4dccmNJjl3cj8DOB63n")
demo.setConfig("twitter4j.oauth.accessToken","434627209-UBwDvYdrJ9QpNL55Fuej8xvUPn7UkRMjKWpXVwwp")
demo.setConfig("twitter4j.oauth.accessTokenSecret","2N7rKs4vo4fO7kYjIjcEaUFhMdsmu5VraTjAs023WWE81")
demo.setConfig("watson.tone.url","https://gateway.watsonplatform.net/tone-analyzer-experimental/api")
demo.setConfig("watson.tone.password","6EtYoZM7qWXI")
demo.setConfig("watson.tone.username","bf0431b6-10a4-4eb3-8692-61f599613d3e")
```

Starting the Spark Stream to get live tweets by providing timeinterval

Start a new Twitter Stream that collects the live tweets and enrich them with Sentiment Analysis scores.

The stream is run for a duration specified in the second argument of the **startTwitterStreaming** method.

Note: if no duration is specified then the stream will run until the **stopTwitterStreaming** method is called.

In [3]:
```
import org.apache.spark.streaming._
demo.startTwitterStreaming(sc, Seconds(60))
```

```
Twitter stream started
Tweets are collected real-time and analyzed
To stop the streaming and start interacting with the data use: StreamingTwitter.stopTwitterStreaming
Stopping Twitter stream. Please wait this may take a while
Twitter stream stopped
You can now create a sqlContext and DataFrame with 606 Tweets created. Sample usage:
val (sqlContext, df) = com.ibm.cds.spark.samples.StreamingTwitter.createTwitterDataFrames(sc)
df.printSchema
sqlContext.sql("select author, text from tweets").show
```

In the above code, we have provided account access credential to Spark service to allow Twitter APIs and Tone analyzer to perform assigned actions. Finally, we are collecting tweets for 60sec period of time and saving it as output from Spark Stream.

Creating a SQLContext and Dataframe with all the tweets

Note: this method will register a SparkSQL table called tweets

In [4]:
```
val (sqlContext, df) = demo.createTwitterDataFrames(sc)
```

```
A new table named tweets with 606 records has been correctly created and can be accessed through the SQLContext variable
Here's the schema for tweets
root
 |-- author: string (nullable = true)
 |-- date: string (nullable = true)
 |-- lang: string (nullable = true)
 |-- text: string (nullable = true)
 |-- lat: double (nullable = true)
 |-- long: double (nullable = true)
 |-- Cheerfulness: double (nullable = true)
 |-- Negative: double (nullable = true)
 |-- Anger: double (nullable = true)
 |-- Analytical: double (nullable = true)
 |-- Confident: double (nullable = true)
 |-- Tentative: double (nullable = true)
 |-- Openness: double (nullable = true)
 |-- Agreeableness: double (nullable = true)
 |-- Conscientiousness: double (nullable = true)
```

Executing a SparkSQL query to display the data, table has collected

In [5]:
```
val fullSet = sqlContext.sql("select * from tweets")  //Select all columns
```

In [6]:
```
fullSet.show    //to display data in fullSet
```

author	date	lang	text	lat	long	Cheerfulness	Negative	Anger	Analytical	Confident	Tentative	Openness	Agreeableness	Conscientiousness
Myles	Fri Dec 11 18:37:...	en	@MadSports8 @SKar...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	9.0	1.0	7.000000000000001
isabel	Fri Dec 11 18:37:...	en	@trulyoth jk my r...	0.0	0.0	0.0	0.0	0.0	86.0	0.0	0.0	0.0	100.0	2.0
TTN Charlotte	Fri Dec 11 18:37:...	en	Hit & run acciden...	35.23807	-80.79963	0.0	0.0	0.0	0.0	0.0	0.0	99.0	68.0	100.0
stacee	Fri Dec 11 18:37:...	en	@EloquentDust he'...	0.0	0.0	100.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	68.0
?COCOCANDY??	Fri Dec 11 18:37:...	en	_illniggaalert2x'...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	90.0	68.0
Tinkerbell	Fri Dec 11 18:37:...	en	RT @ipostlovers: ...	0.0	0.0	100.0	0.0	0.0	0.0	0.0	100.0	0.0	99.0	0.0
Jayy	Fri Dec 11 18:37:...	en	it's not even 8 a...	0.0	0.0	100.0	0.0	0.0	0.0	0.0	97.0	0.0	100.0	28.999999999999996
aderktesh	Fri Dec 11 18:37:...	en	List of Slogans A...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	11.0	68.0
Bussy The Bunny?	Fri Dec 11 18:37:...	en	RT @KEEPINUPWITKE...	0.0	0.0	0.0	100.0	100.0	0.0	0.0	0.0	2.0	0.0	0.0
I S S A	Fri Dec 11 18:37:...	en-gb	RT @alldcrew: Ke...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	100.0	68.0
king ni	Fri Dec 11 18:37:...	en	@Loveharrys i for...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	91.0	68.0
kstef	Fri Dec 11 18:37:...	en	RT @fangirlsmum: ...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	97.0	0.0	68.0
Mikay	Fri Dec 11 18:37:...	en	morninggg	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	100.0
Jose Guzman	Fri Dec 11 18:37:...	en	RT @UnitedgamerzH...	0.0	0.0	0.0	0.0	0.0	100.0	0.0	100.0	50.0	0.0	0.0
Alexandre B.	Fri Dec 11 18:37:...	en	@thebullscharge d...	0.0	0.0	100.0	0.0	0.0	93.0	0.0	0.0	0.0	100.0	1.0
??? ? ?!	Fri Dec 11 18:37:...	en	RT @KPOPSECRETFIL...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	97.0	0.0	68.0
Siddhartha Unni	Fri Dec 11 18:37:...	en	RT @shilparathnam...	0.0	0.0	100.0	0.0	0.0	0.0	0.0	0.0	97.0	37.0	100.0
freakyyy_lee	Fri Dec 11 18:37:...	en	RT @__aaakire: LM...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	97.0	0.0	68.0
4sitters.com	Fri Dec 11 18:37:...	en	Panama City / Pen...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	28.000000000000004	98.0	75.0
Deborah E. Whaley	Fri Dec 11 18:37:...	en	Black scientists ...	0.0	0.0	0.0	0.0	0.0	94.0	0.0	0.0	2.0	97.0	61.0

In this step, we are creating SQLContext and converting current RDDs to DataFrames to start our analysis using simple SQL queries.

Persist the dataset into a parquet file on Object Storage service

The parquet file will be reloaded here

In [7]:
```
fullSet.repartition(1).saveAsParquetFile("swift://notebooks.spark/tweetsFull.parquet")   //saving data into a parquet file...
```

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
```

Running SparkSQL query on the data collceted

Lets, select all the tweets that have Anger score greated than 70%

In [8]:
```
val angerSet = sqlContext.sql("select author, text, Anger from tweets where Anger > 70")
```

In [9]:
```
println(angerSet.count)   // check count of angerSet here
```

```
40
```

In [10]:
```
angerSet.show   //display all data
```

```
+--------------------+--------------------+-----+
|              author|                text|Anger|
+--------------------+--------------------+-----+
|    Bussy The Bunny?|RT @KEEPINUPWITKE...|100.0|
|          Bulls 74-8|RT @MachineGunnKi...|100.0|
|              daddy|@StarsAndScars_
...|100.0|
|    Fade VermaVirus |My friend got mad...|100.0|
|     Zachary Daniels|RT @nxlix_: make ...|100.0|
|                 BFD|Yes @celestepewte...|100.0|
|     Jonny Pinciotti|                Shit|100.0|
|       jolly jazmin|my dad fucking te...|100.0|
|               Alan|@AbiRatchford GOD...|100.0|
|           Lindsay?|Hate myself more ...|100.0|
|     Ashley Reynolds|RT @LoveIyCouples...|100.0|
|         Tamara King|lovepleasure0907:...|100.0|
|           anggi_Dot|2 weekly follower...|100.0|
|aggressive zoom blur|@tarotots arbuckl...|100.0|
|                 Jrd|@DieErVonSatan yo...|100.0|
|                  SB|I need all that shit|100.0|
|              Ethan.|EVER SEEN SOMEONE...|100.0|
|          Caaaaaas.|RT @ThomasBangalt...|100.0|
|              carly|ARE Y'ALL FUCKING...|100.0|
|            ?CHROM?|@RaraTrend [[ AAA...|100.0|
+--------------------+--------------------+-----+
```

Above stage describes how we can use simple SQL queries in SparkSQL context to analyze data. We fetched all records having Anger level more than 70% and defining it by Author, Text, and Anger.

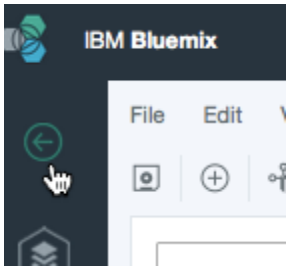
We can see the output in the above screenshot.

PART II: Analyze the data using an IPython Notebook

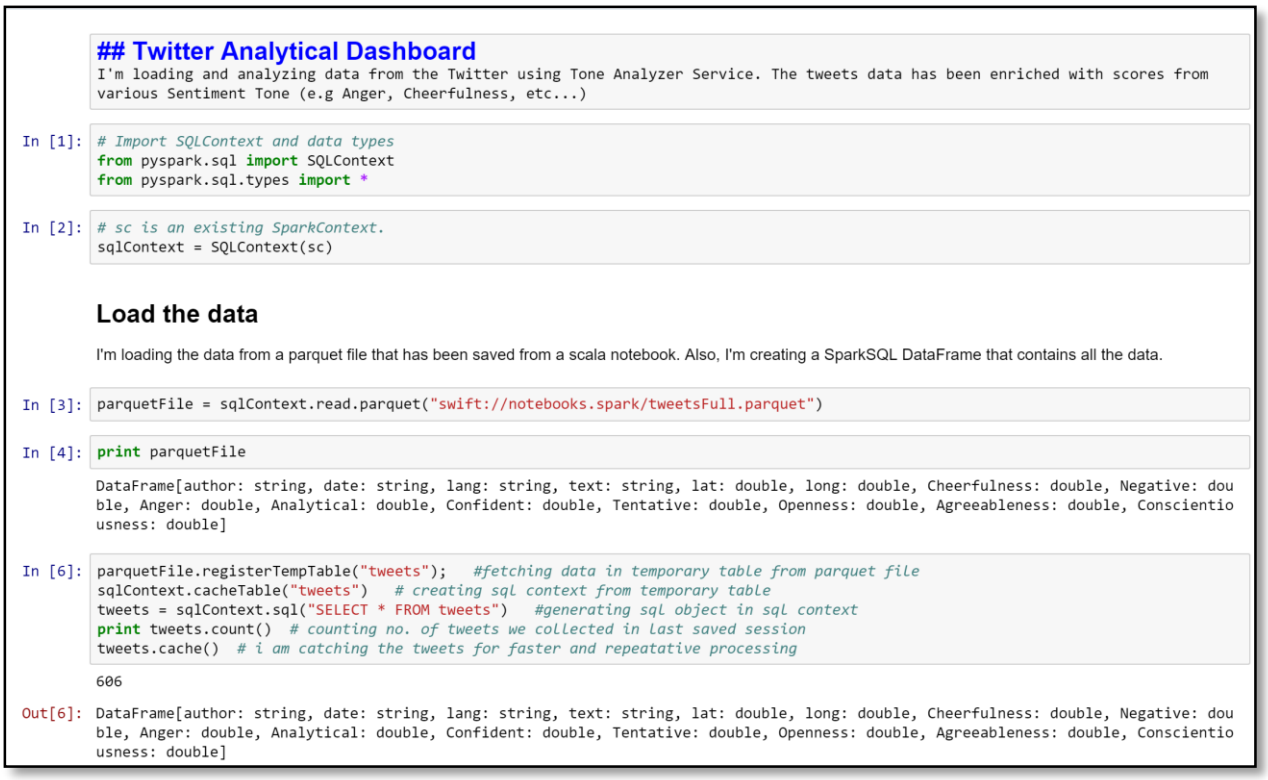
In the previous section, using a Scala Notebook, I have implemented how to run the Twitter Stream to acquire data and enrich it with sentiment scores from Watson Tone Analyzer. I also ran a command to persist the data in a parquet file on the Object Storage bound to this Spark instance. Now, we'll reload this data in an IPython Notebook for further analysis and visualization.

1. From the Notebook main page, create a new Python Notebook.

   From within your Scala notebook, go to the upper left of the screen and click the back button to return to your **My Notebooks** page.



   Click **New Notebook**. Enter a **Name**, and under **Language** select **Python**. Then click **Create Notebook**.



Twitter Analytical Dashboard

I'm loading and analyzing data from the Twitter using Tone Analyzer Service. The tweets data has been enriched with scores from various Sentiment Tone (e.g Anger, Cheerfulness, etc...)

In [1]:
```
# Import SQLContext and data types
from pyspark.sql import SQLContext
from pyspark.sql.types import *
```

In [2]:
```
# sc is an existing SparkContext.
sqlContext = SQLContext(sc)
```

Load the data

I'm loading the data from a parquet file that has been saved from a scala notebook. Also, I'm creating a SparkSQL DataFrame that contains all the data.

In [3]:
```
parquetFile = sqlContext.read.parquet("swift://notebooks.spark/tweetsFull.parquet")
```

In [4]:
```
print parquetFile
```

```
DataFrame[author: string, date: string, lang: string, text: string, lat: double, long: double, Cheerfulness: double, Negative: dou
ble, Anger: double, Analytical: double, Confident: double, Tentative: double, Openness: double, Agreeableness: double, Conscientio
usness: double]
```

In [6]:
```
parquetFile.registerTempTable("tweets");   #fetching data in temporary table from parquet file
sqlContext.cacheTable("tweets")   # creating sql context from temporary table
tweets = sqlContext.sql("SELECT * FROM tweets")   #generating sql object in sql context
print tweets.count()  # counting no. of tweets we collected in last saved session
tweets.cache()  # i am catching the tweets for faster and repeatative processing
```

```
606
```

Out[6]:
```
DataFrame[author: string, date: string, lang: string, text: string, lat: double, long: double, Cheerfulness: double, Negative: dou
ble, Anger: double, Analytical: double, Confident: double, Tentative: double, Openness: double, Agreeableness: double, Conscientio
usness: double]
```

The main purpose of writing this code in python is to show flexibility provided by Bluemix platform to make system open for developers from various background.

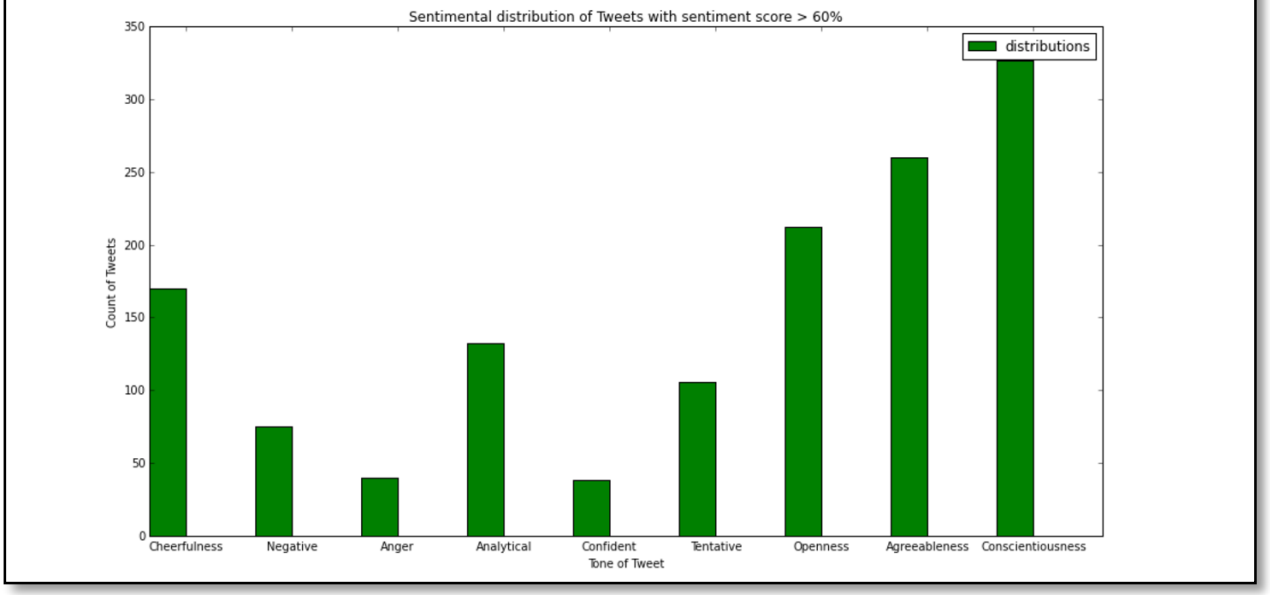
Here, we created SQLContext to load data from parquet file. Later, we created table to analyze data in relational format and cached it to make system faster and better.

Computing the distribution of tweets by sentiments > 60%

Using SparkSQL queries, we can compute for each tone (given by Tone Analyzer) that number of tweets that are greater than 60%

In [7]:
```
#creating an array that will hold the count for each sentiment
sentimentDistribution=[0] * 9
#For each sentiment, im running a sql query that counts the number of tweets for which the sentiment score is greater than 60%
#Storing the data in the array
for i, sentiment in enumerate(tweets.columns[-9:]):
    sentimentDistribution[i]=sqlContext.sql("SELECT count(*) as sentCount FROM tweets where " + sentiment + " > 60")\
        .collect()[0].sentCount
```

In [8]:
```
%matplotlib inline
import matplotlib
import numpy as np
import matplotlib.pyplot as plt

ind=np.arange(9)
width = 0.35
bar = plt.bar(ind, sentimentDistribution, width, color='g', label = "distributions")

params = plt.gcf()
plSize = params.get_size_inches()
params.set_size_inches( (plSize[0]*2.5, plSize[1]*2) )
plt.ylabel('Count of Tweets')
plt.xlabel('Tone of Tweet')
plt.title('Sentimental distribution of Tweets with sentiment score > 60%')
plt.xticks(ind+width, tweets.columns[-9:])
plt.legend()

plt.show()
```



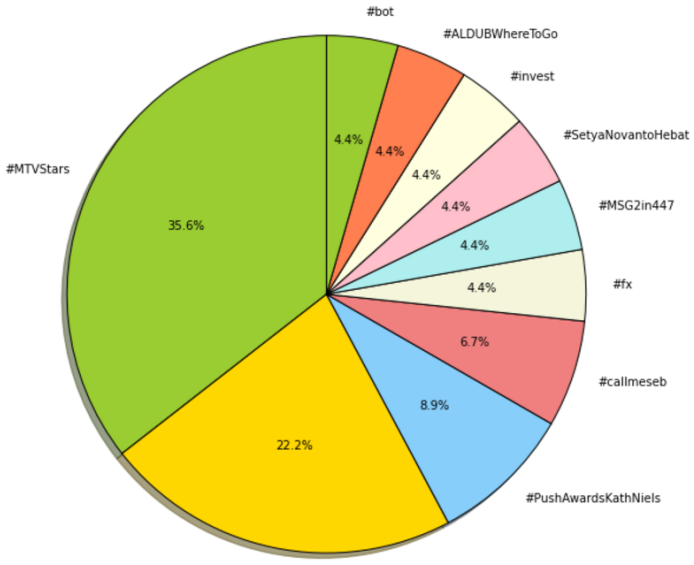
Using simple python matplotlib library APIs and python application, we have created Barchart for showing distribution of tweets by sentiments having score more than 60%. The results can be observed in the above figure given.

#Computing the top 10 hashtags contained in the tweets

Using SparkSQL queries, we can compute top 10 hashtags in collected tweets record.

In [9]:
```
from operator import add
import re
tagsRDD = tweets.flatMap( lambda t: re.split("\s", t.text))\
    .filter( lambda word: word.startswith("#") )\
    .map( lambda word : (word, 1 ))\
    .reduceByKey(add, 10).map(lambda (a,b): (b,a)).sortByKey(False).map(lambda (a,b):(b,a))
top10tags = tagsRDD.take(10)
```

In [10]:
```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt

params = plt.gcf()
plSize = params.get_size_inches()
params.set_size_inches( (plSize[0]*2, plSize[1]*2) )

labels = [i[0] for i in top10tags]
sizes = [int(i[1]) for i in top10tags]
colors = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral', "beige", "paleturquoise", "pink", "lightyellow", "coral"]

plt.pie(sizes, labels=labels, colors=colors,autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')

plt.show()
```



This analysis shows result of top 10 hashtags in corpus of tweets. Using matplotlib, we have drawn a Pie Chart to show contribution of each hashtag in overall tweets.

Here, we needed to convert DataFrames into RDDs to perform map-reduce operation to split, filter by hashtags and then collect top hash tagged tweets in resulting RDD.

Selecting only the top 5 hashtags by sentiment scores

Here, I have built complex analytic which selects the top 5 hashtags by sentiment scores. We can compute the mean of all the sentiment scores and rank them by sentiment score.Then I have shown visualization in a multi-series bar chart

In [11]:
```
cols = tweets.columns[-9:]
def expand( t ):
    ret = []
    for s in [i[0] for i in top10tags]:
        if ( s in t.text ):
            for tone in cols:
                ret += [s + u"-" + unicode(tone) + ":" + unicode(getattr(t, tone))]
    return ret
def makeList(l):
    return l if isinstance(l, list) else [l]
```

In [12]:
```
#Create RDD from tweets dataframe
tagsRDD = tweets.map(lambda t: t )

#Filter to only keep the entries that are in top10tags
tagsRDD = tagsRDD.filter( lambda t: any(s in t.text for s in [i[0] for i in top10tags] ) )

#Create a flatMap using the expand function defined above, this will be used to collect all the scores
#for a particular tag with the following format: Tag-Tone-ToneScore
tagsRDD = tagsRDD.flatMap( expand )

#Create a map indexed by Tag-Tone keys
tagsRDD = tagsRDD.map( lambda fullTag : (fullTag.split(":")[0], float( fullTag.split(":")[1]) ))
```

In [13]:
```
#Call combineByKey to format the data as follow
#Key=Tag-Tone
#Value=(count, sum_of_all_score_for_this_tone)
tagsRDD = tagsRDD.combineByKey((lambda x: (x,1)),
                 (lambda x, y: (x[0] + y, x[1] + 1)),
                 (lambda x, y: (x[0] + y[0], x[1] + y[1])))

#ReIndex the map to have the key be the Tag and value be (Tone, Average_score) tuple
#Key=Tag
#Value=(Tone, average_score)
tagsRDD = tagsRDD.map(lambda (key, ab): (key.split("-")[0], (key.split("-")[1], round(ab[0]/ab[1], 2))))

#Reduce the map on the Tag key, value becomes a list of (Tone,average_score) tuples
tagsRDD = tagsRDD.reduceByKey( lambda x, y : makeList(x) + makeList(y) )

#Sort the (Tone,average_score) tuples alphabetically by Tone
tagsRDD = tagsRDD.mapValues( lambda x : sorted(x) )
```

In [14]:
```
#Format the data as expected by the plotting code in the next cell.
#map the Values to a tuple as follow: ([list of tone], [list of average score])
#e.g. #someTag:([u'Agreeableness', u'Analytical', u'Anger', u'Cheerfulness', u'Confident', u'Conscientiousness', u'Negative', u'Openness', u'Tentative'], [1.0, 0.0, 0.0, 1.0, 0.0, 0.48, 0.0, 0.02, 0.0])
tagsRDD = tagsRDD.mapValues( lambda x : ([elt[0] for elt in x],[elt[1] for elt in x])  )

#Use custom sort function to sort the entries by order of appearance in top10tags
def customCompare( key ):
    for (k,v) in top10tags:
        if k == key:
            return v
    return 0
tagsRDD = tagsRDD.sortByKey(ascending=False, numPartitions=None, keyfunc = customCompare)
```

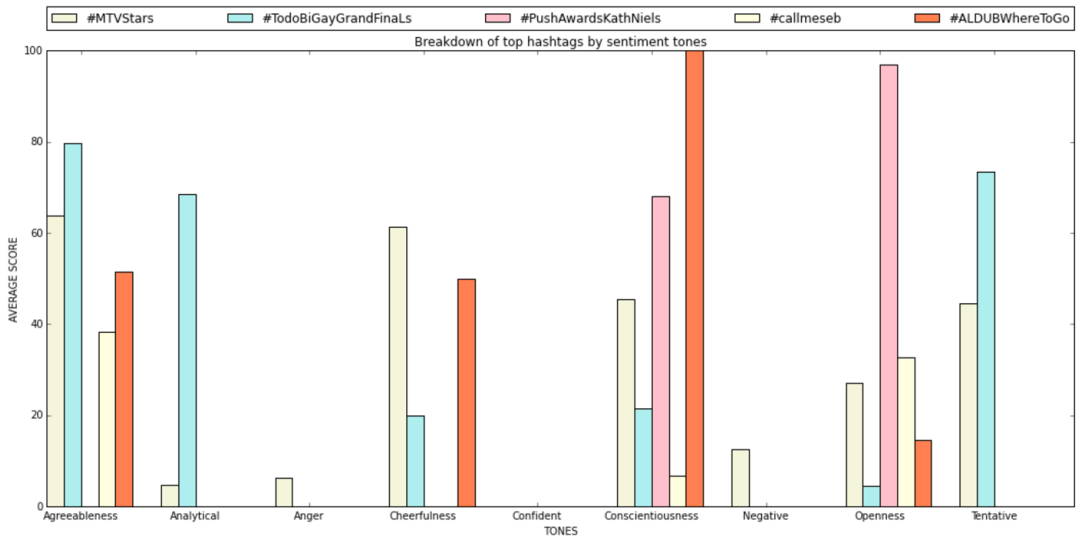
In [15]:
```
#Take the mean tone scores for the top 10 tags
top10tagsMeanScores = tagsRDD.take(10)
```

This is one of the most difficult analysis we have performed. Here, we have successfully collected top 5 hashtags by computing their average sentiment score. We have gone through series of operations to reach conclusion. All the steps are given above.

Finally, using resulting data, we have drawn multi-series Bar chart to see distribution of sentiments involved in each top 5 hashtags. The final results are shown in diagram given below.

In [15]:
```
#Take the mean tone scores for the top 10 tags
top10tagsMeanScores = tagsRDD.take(10)
```

In [16]:
```
%matplotlib inline
import matplotlib
import numpy as np
import matplotlib.pyplot as plt

params = plt.gcf()
plSize = params.get_size_inches()
params.set_size_inches( (plSize[0]*3, plSize[1]*2) )

top5tagsMeanScores = top10tagsMeanScores[:5]
width = 0
ind=np.arange(9)
(a,b) = top5tagsMeanScores[0]
labels=b[0]
colors = ["beige", "paleturquoise", "pink", "lightyellow", "coral", "lightgreen", "gainsboro", "aquamarine","c"]
idx=0
for key, value in top5tagsMeanScores:
    plt.bar(ind + width, value[1], 0.15, color=colors[idx], label=key)
    width += 0.15
    idx += 1
plt.xticks(ind+0.3, labels)
plt.ylabel('AVERAGE SCORE')
plt.xlabel('TONES')
plt.title('Breakdown of top hashtags by sentiment tones')

plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc='center',ncol=5, mode="expand", borderaxespad=0.)

plt.show()
```



Conclusion

- Finally, we built a complex Apache Spark solution that integrates multiple services from Bluemix like Spark service, Object Storage and Tone Analyzer.
- We successfully loaded the data into Spark SQL DataFrames and query the data using SQL.
- We ran complex analytics using RDD transformations and actions.
- We were able to create compelling visualizations using the powerful **matplotlib** Python package provided in the IPython Notebook.

This project reflects the power and potential of the Apache Spark engine and programming model. This project has inspired me to run my own analytics and reports with ‘Spark on Bluemix’ kind of fast and flexible tools

References

1. Spark Steaming Programming Guide
2. Spark SQL and DataFrame Guide
3. Learning Spark: Lightning-Fast Big Data Analysis - Book
4. IBM Bluemix - Analytics for Apache Spark - Documentation
5. IBM Object Storage for Bluemix - Documentation
6. Tone Analyzer - Documentation

Learning Tutorial

1. Introduction to Big Data with Apache Spark - edx online
2. Scalable Machine Learning - edx online

Happy Sparking!!

Presented By

Ashwinkumar Dinoriya