Twitter Sentiment Analysis

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

Presented By
#AshwinDinoriya

Overview

- To search trending discussion on Twitter using hashtags to analyze opinion/emotions/moods of people by performing sentiment analysis
- Sentiments talks more about view/intention/opinion of people towards certain topic
- 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

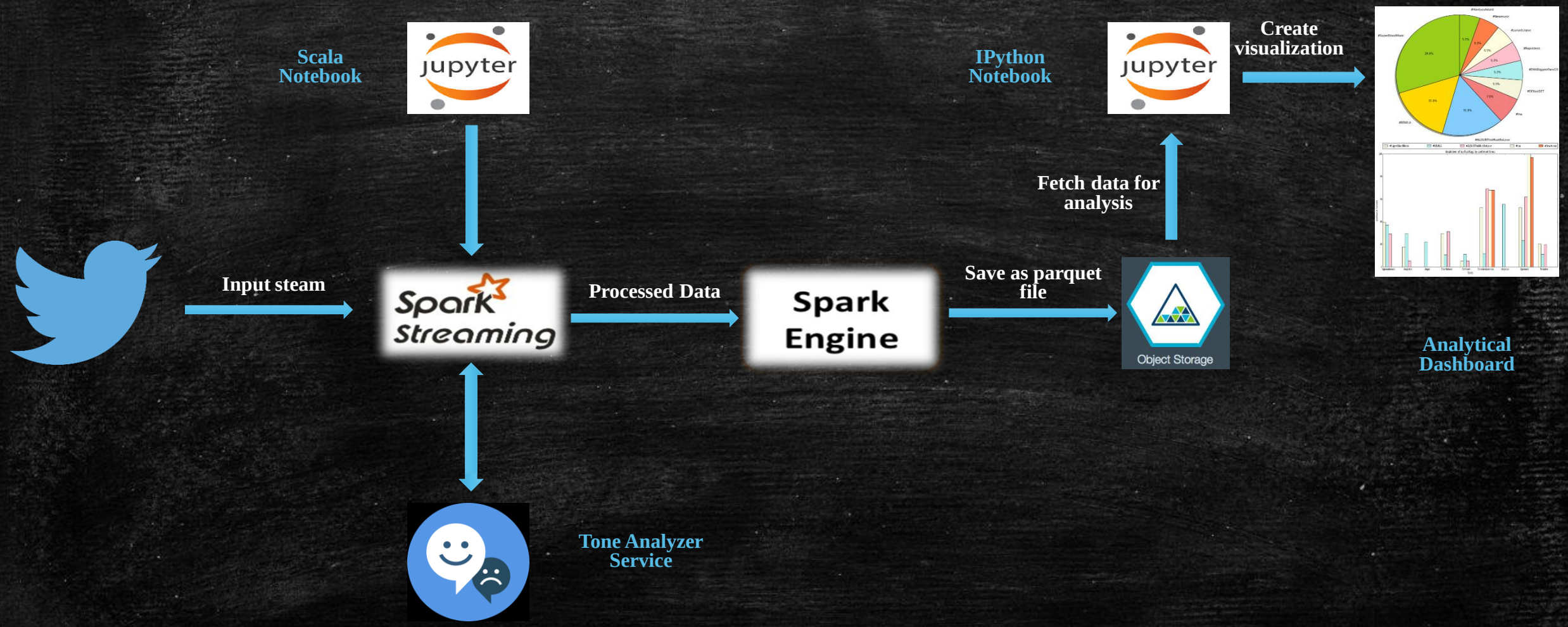
Aim of sentiment analysis is to…

- Analytic 1: Compute the distribution of tweets by sentiment scores greater than 60%
- Analytic 2: Compute the top 10 hashtags contained in the tweets
- Analytic 3: Visualize aggregated sentiment scores for the top 5 hashtags

REAL TIME STREAMING

WITH Apache Spark

The Basic Architecture

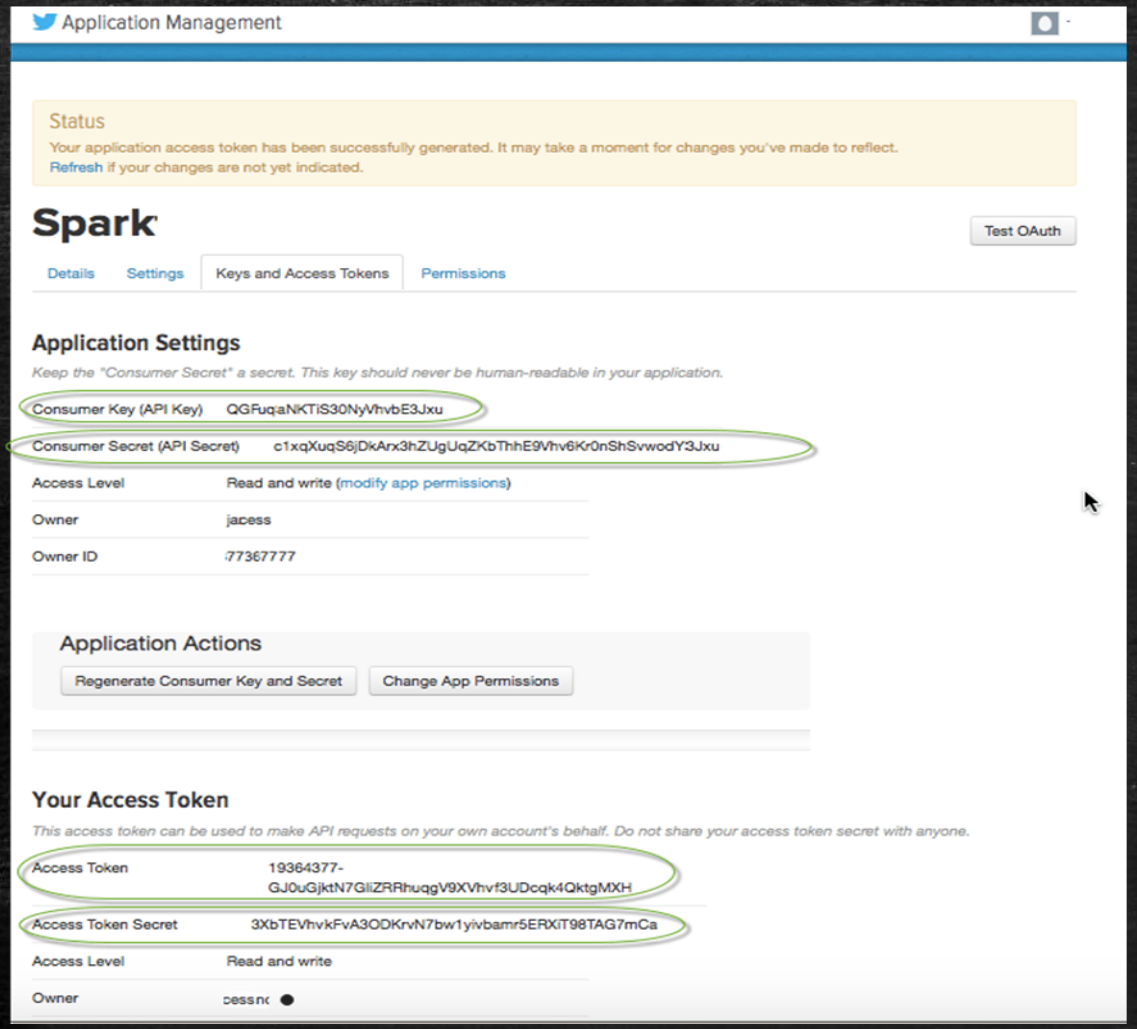


Tools & Technologies

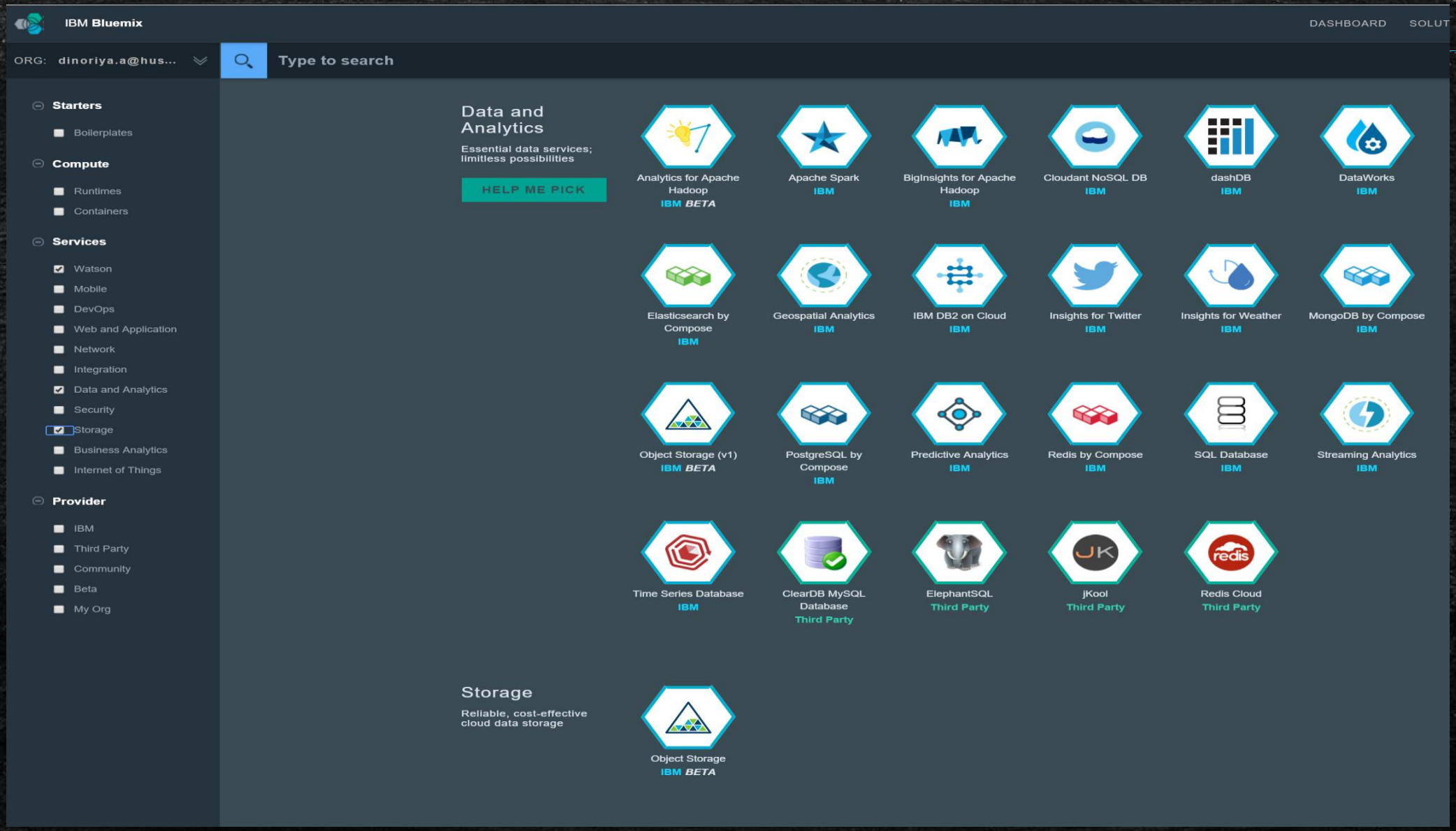
- ✓ Twitter Streaming APIs
- ✓ IPython and Scala Notebook (jupyter notebook)
- ✓ Parquet Data Object Storage
- ✓ Matplotlib python library
- ✓ IBM Bluemix
    - a. Object Storage Service
    - b. Tone Analyzer Service
    - c. Apache Spark Service
        - c.1. Spark Steaming
        - c.2. SparkSQL

Implementation

1. Twitter Developer Account Credential



2. IBM Bluemix Catalog – Sign Up



IBM Bluemix Services

Services

Apache Spark-Service
Apache Spark
Unbound Service
Plan: **ibm.SparkService.PayGoPer...**

Apache Spark-Service_objec...
Object Storage
Unbound Service
Plan: **Free**

Tone Analyzer-Final Project
Tone Analyzer
Unbound Service
Plan: **experimental**

IBM Bluemix – Apache Spark Analytics

https://console.ng.bluemix.net/data/analytics/

IBM Bluemix

DASHBOARD SOLUTIONS CATALOG PRICING DOCS COMMUNITY

Services

Data

Analytics

Exchange

Analytics

Introducing IBM Analytics for Apache Spark

Get to insights lightning fast with Spark's advanced in-memory compute and full access to the complete Spark SQL programming model, as well as MLlib and GraphX packages. And now, analyze data with interactive notebooks in Python or Scala kernels.

To learn more about using the Apache Spark service and interactive notebooks, check out Getting started with IBM Analytics for Apache Spark.

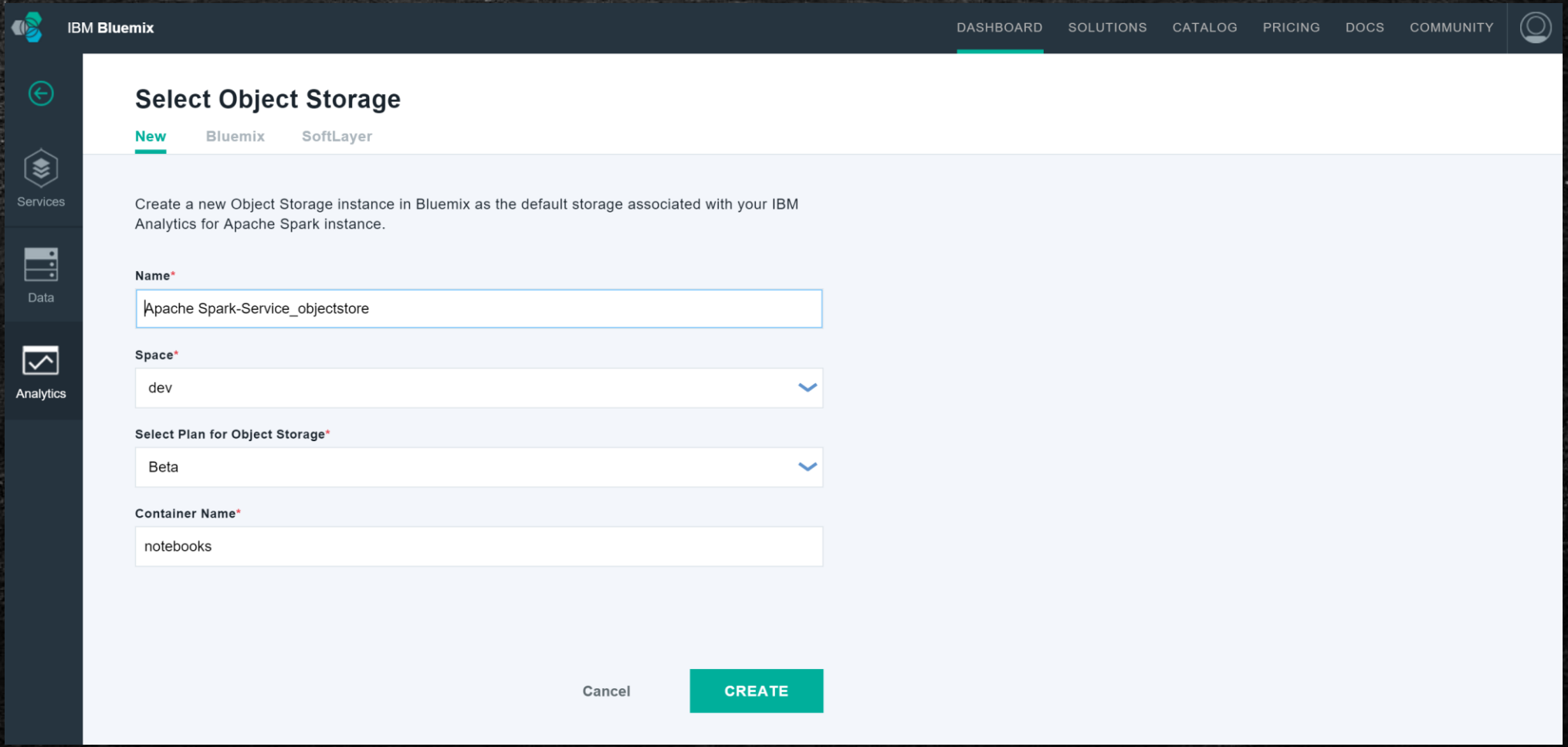
IBM Analytics for Ap...

NEW INSTANCE

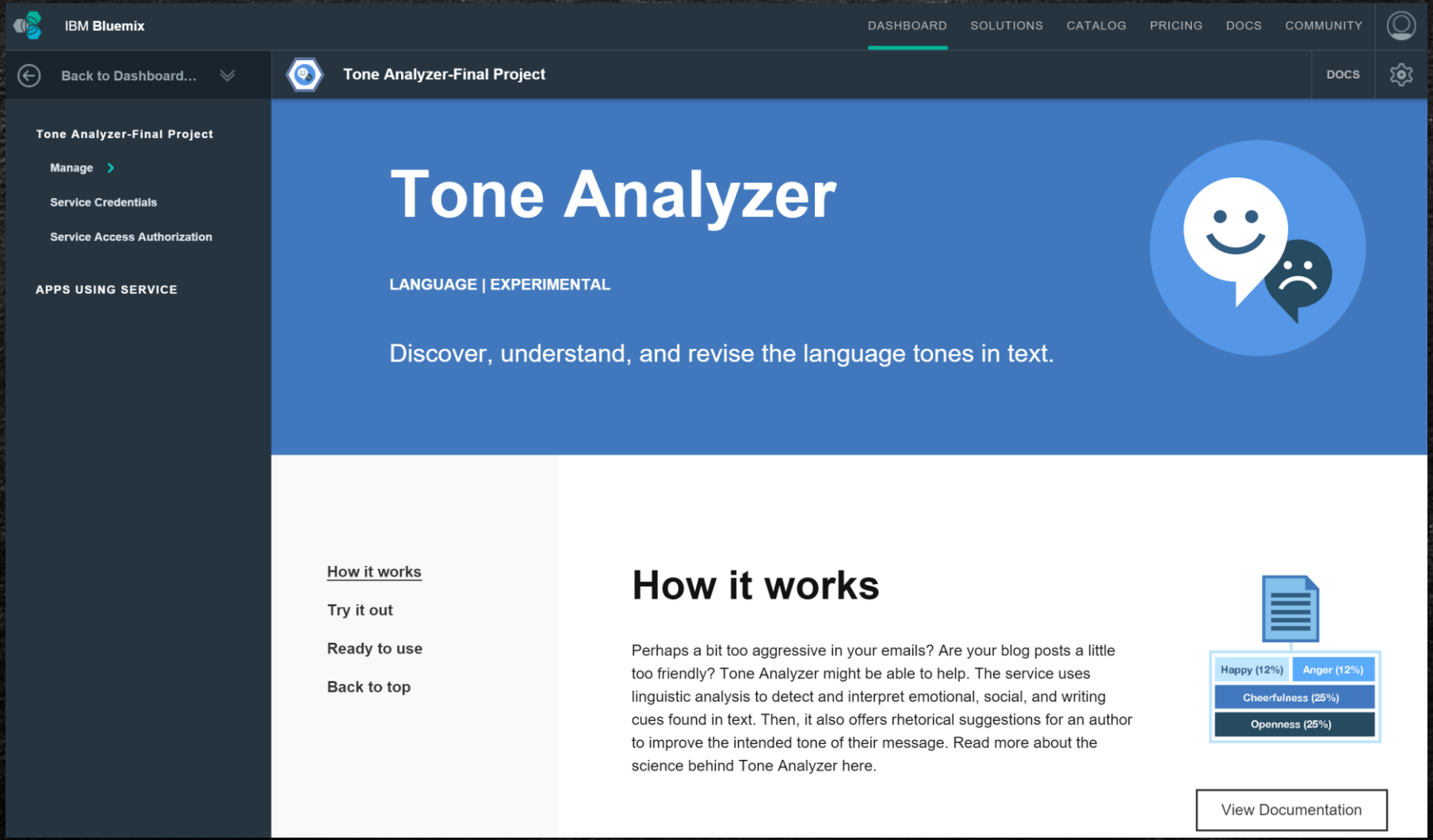
Apache Spark (4)

Apache Spark-f9	DTAIEB-Insight-Demo	spark-test-spark
Apache Spark *DEV*	Apache Spark *DEV*	Apache Spark *DEV*
Plan: free	**Plan:** free	**Plan:** free
Actions	Actions	Actions

IBM Bluemix – Object Storage



IBM Bluemix –Tone Analyzer



IBM Bluemix – Tone Analyzer Credentials

IBM Bluemix

DASHBOARD SOLUTIONS CATALOG PRICING DOCS COMMUNITY

Back to Dashboard...

Tone Analyzer-Final Project

DOCS

Tone Analyzer-Final Project

Manage

Service Credentials

Service Access Authorization

APPS USING SERVICE

Service Credentials

ADD CREDENTIALS

Cloud Foundry provides your credentials in JSON format. The JSON snippet lists credentials, such as the API key and secret, as well as connection information for the service.

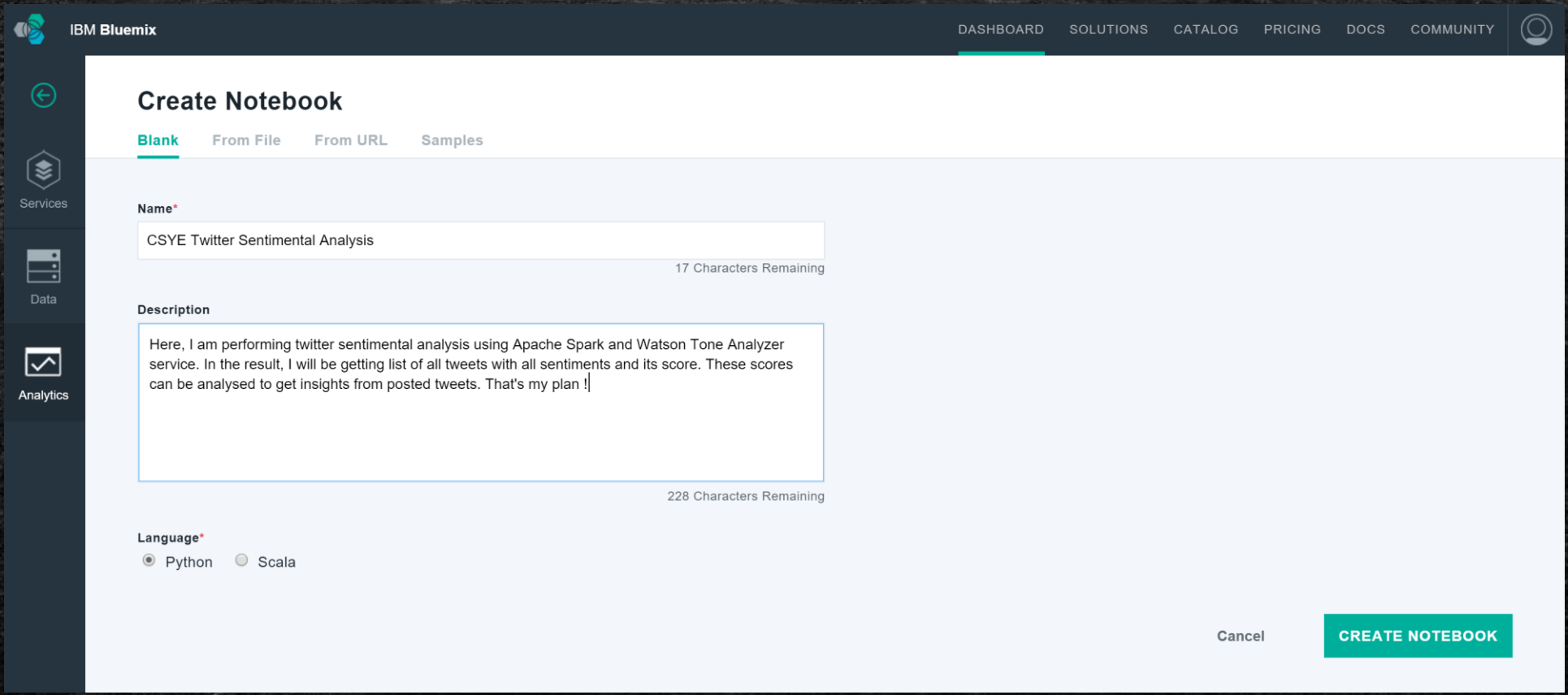
NAME ashwin

DELETE

SERVICE CREDENTIALS


```
{
  "credentials": {
    "url": "https://gateway.watsonplatform.net/tone-analyzer-experimental/api",
    "username": "[redacted]",
    "password": "[redacted]"
  }
}
```

IBM Bluemix – Creating Notebook in Scala/Python



Sentiment Analysis using Tone Analyzer

IBM Bluemix

File Edit View Insert Cell Kernel Help

Format Code Cell Toolbar None

Services

Data

Analytics

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

#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 [ ]:
```
val demo = com.ibm.cds.spark.samples.StreamingTwitter
demo.setConfig("twitter4j.oauth.consumerKey","ABCDEFGHIJKLMNOPQRSTUVWXYZ")
demo.setConfig("twitter4j.oauth.consumerSecret","ABCDEFGHIJKLMNOPQRSTUVWXYZ")
demo.setConfig("twitter4j.oauth.accessToken","ABCDEFGHIJKLMNOPQRSTUVWXYZ")
demo.setConfig("twitter4j.oauth.accessTokenSecret","ABCDEFGHIJKLMNOPQRSTUVWXYZ")
demo.setConfig("watson.tone.url","https://gateway.watsonplatform.net/tone-analyzer-experimental/api")
demo.setConfig("watson.tone.password","ABCDEFGHIJKLMNOPQRSTUVWXYZ")
demo.setConfig("watson.tone.username","ABCDEFGHIJKLMNOPQRSTUVWXYZ")
```

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

#Creating a SQLContext and Dataframe with all the tweets

Note: this method will register a SparkSQL table called tweets

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

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

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

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

#Persist the dataset into a parquet file on Object Storage service

The parquet file will be reloaded here

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

#Running SparkSQL query on the data collceted

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

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

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

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

Twitter Analytical Dashboards

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
```

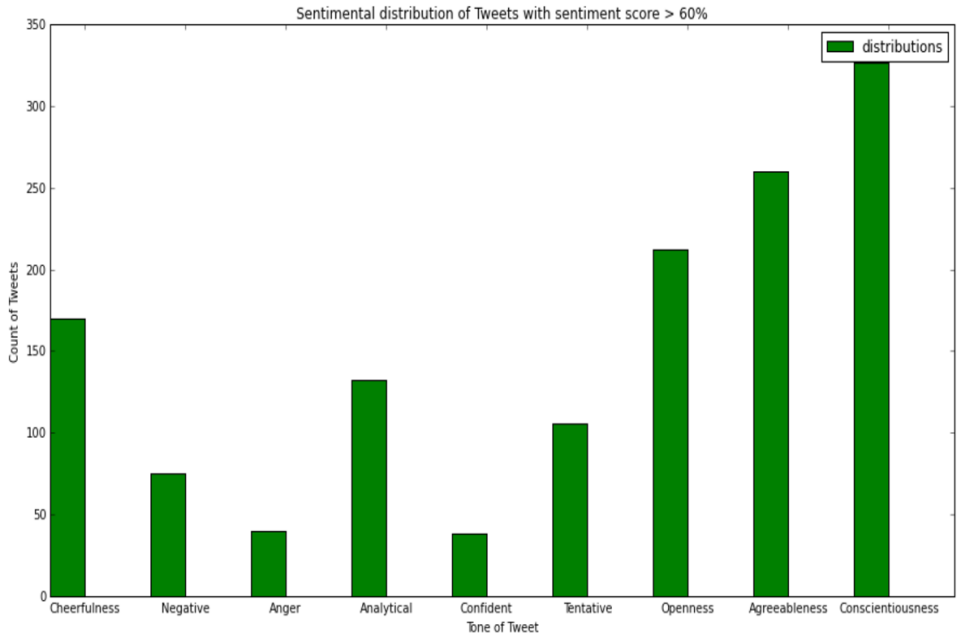
Out[6]:
```
DataFrame[author: string, date: string, lang: string, text: string, lat: double, long: double, Cheerfulness: double, Negative: dou
ble, Anger: double, Analytical: double, Confident: double, Tentative: double, Openness: double, Agreeableness: double, Conscientio
usness: double]
```

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()
```

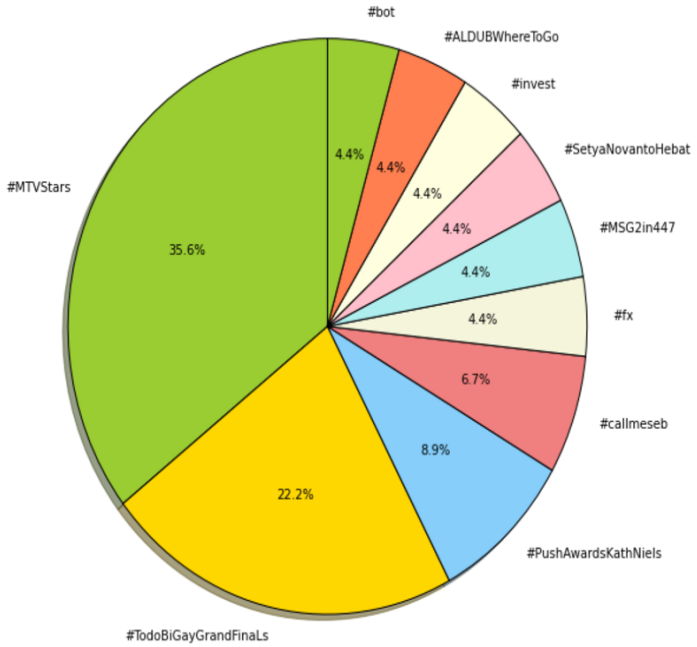


#Computing the top 10 hashtags contained in the tweets

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()
```



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]) ))
```

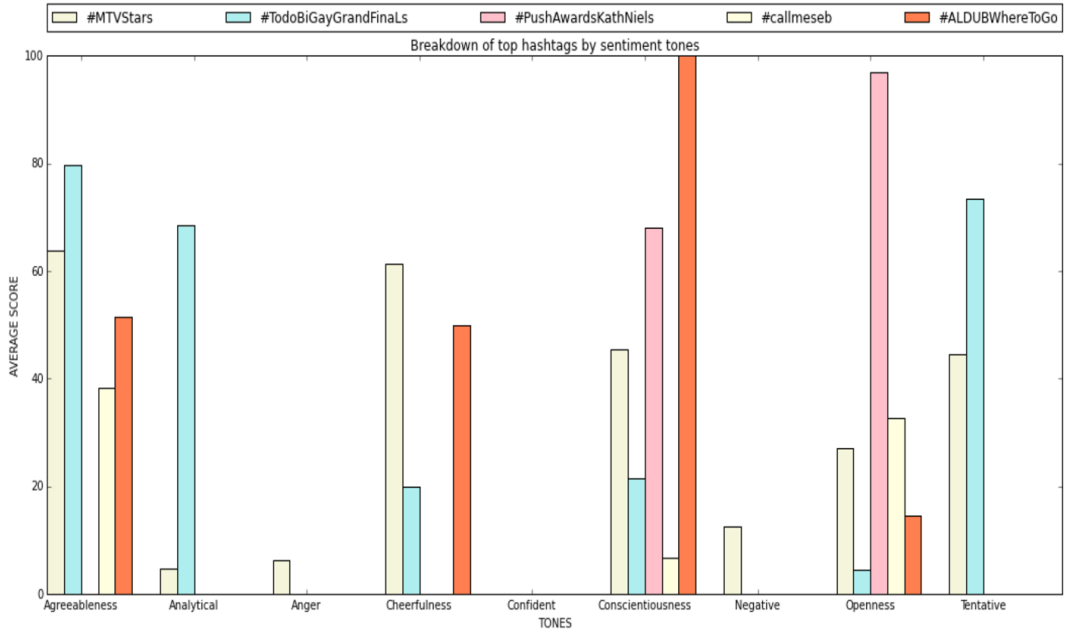
In [13]:
```
#Call combineByKey to format the data as follow
#Key=Tag-Tone
#Value=(count, sum_of_all_score_for_this_tone)
tagsRDD = tagsRDD.combineByKey((lambda x: (x,1)),
                (lambda x, y: (x[0] + y, x[1] + 1)),
                (lambda x, y: (x[0] + y[0], x[1] + y[1])))

#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)
```

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()
```



Learning Outcome

- Able to implement most of the concepts in Apache Spark
- Incorporated all the knowledge gained through special topic presentation like Cloud Services, NoSQL Connection, Streaming APIs
- Planned and created task flow using various technologies like Spark Streaming, external NLP APIs, NoSQL databases and Jupyter Notebook, etc. to build a business model

Follow

#ThankYou