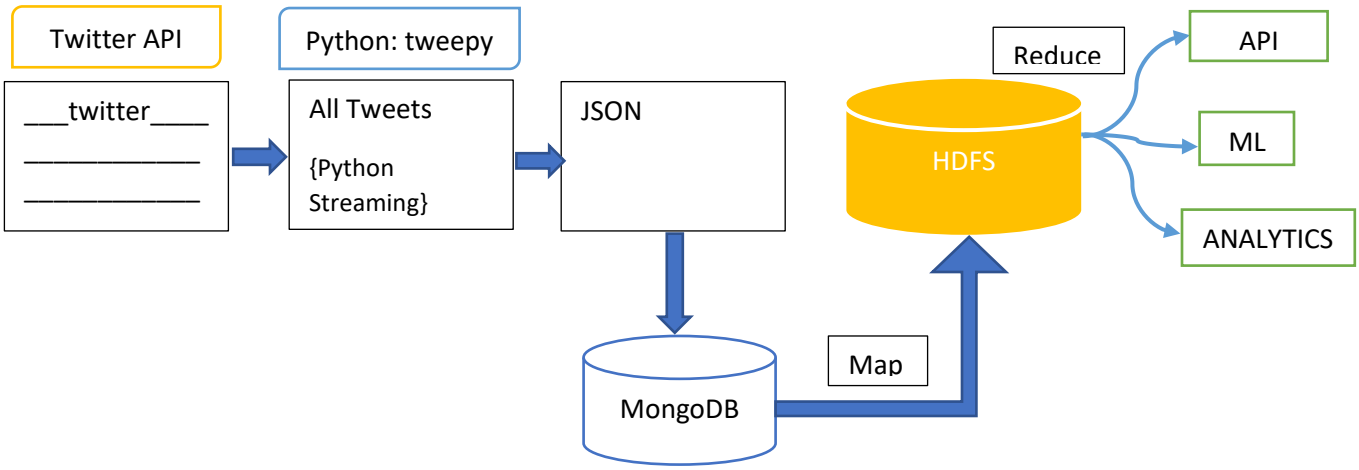
Design:



Twitter:

Twitter API:

Twitter as a platform has a lot of users and it generates a lot of data that can be used for analysis. Data can include user tweets, user profiles, user friends and followers, what's trending, etc. This data can be extracted using twitter Application Programming Interface(API). There are three methods to get this data: the REST API, the search API, and the Streaming API. The Search API is retrospective and allows you search old tweets [with severe limitations], the REST API allows you to collect user profiles, friends, and followers, and the Streaming API collects tweets in real time as they happen. As we are going to do a real time analysis of the data we find the Streaming API most suited to our needs.

The Twitter API requires a few steps:

1) Authenticate with OAuth
2) Make API call
3) Receive JSON file back
4) Interpret JSON file

Authenticate with OAuth:

OAuth is an open standard for access delegation, commonly used as a way for internet users to grant websites or applications access to their information on other websites or applications access to their information on other websites but without giving them the passwords (Gordon, 2012). Authentication with OAuth on Twitter requires you to get keys from the Twitter developers site using a Twitter developer account. There are four keys (Consumer Key, Consumer Secret Key, Access Token, and Secret

Token) that are required to access the API and need to be used during a handshake, once authenticated the program can make API calls.

Make API call:

When making an API call to twitter it has parameters incorporated into the URL, the wrapper looks like this:

https://stream.twitter.com/1.1/statuses/filter.json?track=twitter

This call is being done through the streaming API, where it is asking to connect to Twitter and once connection is established it will track the keyword 'twitter'. We can specify our own keywords to track and if we do it carefully we can filter a lot data at an early stage that is irrelevant in our research.

Since, we will be using a python library called tweepy the working of an API call is abstracted from the user, nevertheless understanding the working of the making a Twitter API call is necessary to extract the relevant data.

Receive JSON file back:

JavaScript Object Notation (JSON) is an open standard file format that has data formatted as attribute-value pairs. The format is language independent and is commonly used for asynchronous browser-server communication for a data request.

JSON files is also the data structure that Twitter returns when an API call is made. The amount of data returned depends how we define our keywords but is usually rather comprehensive and needs to be parsed.

Interpret JSON file:

JSON file can be stored as raw file or can be stored using a SQL/NoSQL database. Since the data received is unstructured the most logical way to store the data would be using a NoSQL database. We will use MongoDB to store the tweets we receive and parse through. We will then run queries based on keywords we need.

Python:

While there are many different programming languages that can be used to interface with the API, the flexibility and huge community support behind python as well as its relevance in data science makes python the ideal choice for our research. Python has many libraries that has different use cases, we are going to use tweepy to stream our data from twitter.

Tweepy:

Python is a versatile language with adaptability to various use cases. These are done by extending the language by using libraries which are community created. One of these libraries is tweepy. Tweepy is open-sourced, hosted on GitHub and enables Python to communicate with Twitter platform and use its API (Novalić, 2013). This makes it easier to access the platform to collect and monitor tweets for analysis.

Using tweepy:

Command to install the tweepy library:

```
$ pip install tweepy
```

Tweepy supports OAuth authentication. Authentication is handled by the tweepy.AuthHandler class. (Roesslein, 2011)

A consumer token and a secret key is needed to connect with the twitter stream API, we can use the keys we generated after we created a twitter developer account.

These keys are a pair of private and public (secret and non-secret) keys and used to maintain security. The consumer key pair authorizes your program to use the Twitter API, and the access token essentially signs you in as your specific Twitter user account. This framework makes more sense in the context of third party Twitter developers like TweetDeck where the application is making API calls but it needs access to each user's personal data to write tweets, access their timelines, etc. (Dolinar, 2015)

We can import the tweepy library as below:

```
from tweepy import Stream
from tweepy import OAuthHandler
from tweepy.streaming import StreamListener
```

The above tweepy class imports will be used to construct the stream listener.

Diving into the code:

Importing the modules:

Apart from the three tweepy class imports that we use to construct the stream listener, the time library will be used to create a time-out feature for the script, and the os library will be used to set your working directory.

```
#!/usr/bin/python3

# Importing modules
import time
from tweepy import Stream
from tweepy import OAuthHandler
from tweepy.streaming import StreamListener
import os
```

Setting the Variables:

```
# API Initialization

ckey = 'TUmVragHR1y******************' #'**CONSUMER KEY**'
consumer_secret =
'bALCr10bNrredcAnKSJqNfoyHB*******************************' #'**CONSUMER
SECRET KEY***'
access_token_key = '106049147-lvkgyFcULiYIBYv0*****************************'
#'**ACCESS TOKEN**'
access_token_secret = 'RfTo8ITCRwjzOcZHue9*********************************'
#'**ACCESS TOKEN SECRET**'

start_time = time.time() #grabs the system time
keyword_list = ['shelter', 'tsunami', 'fire', 'shooting', 'hurricane', 'flood',
'storm'] #track list
```

We have to set the above variables, which will be used in the stream listener by being fed into the tweepy objects.

Using and Modifying the Tweepy Classes:

The code shown below does the following:

- Creates an OAuthHandler instance to handle OAuth credentials
- Creates a listener instance with a start time and time limit parameters passed to it
- Creates an StreamListener instance with the OAuthHandler instance and the listener instance

Before these instances are created, we have to “modify” the StreamListener class by creating a child class to output the data into a .csv file.

We will output the data into MongoDB by reading this csv file.

```
# Listener Class Override


class listener(StreamListener):

    def __init__(self, start_time, time_limit=60):

        self.time = start_time
        self.limit = time_limit
        self.tweet_data = []

    def on_data(self, data):

        saveFile = open('raw_tweets.json', 'a', encoding='utf-8')

        while (time.time() - self.time) < self.limit:

            try:

                self.tweet_data.append(data)

                return True

            except BaseException as e:
                print('failed ondata,', str(e))
                time.sleep(5)
                pass

        saveFile = open('raw_tweets.json', 'w', encoding='utf-8')
        saveFile.write(u'[\n')
        saveFile.write(','.join(self.tweet_data))
        saveFile.write(u'\n]')
        saveFile.close()
        exit()

    def on_error(self, status):

        print(status)
```

To elaborate more on the writing of the data to a file after the StreamListener instance receives data:

```
saveFile = open('raw_tweets.json', 'w', encoding='utf-8')
saveFile.write(u'[\n')
saveFile.write(','.join(self.tweet_data))
saveFile.write(u'\n]')
saveFile.close()
exit()
```

This block of code opens an output file, writes the opening square bracket, writes the JSON data as text separated by commas, then inserts a closing square bracket, and closes the document. This is the standard JSON format with each Twitter object acting as an element in a JavaScript array. If you bring this into Python built-in parser and the json library can properly handle it.

This section can be modified to or modify the JSON file. For example, we can place other properties/fields like a UNIX time stamp or a random variable into the JSON. We can also modify the output file or eliminate the need for a .csv file and insert the tweet directly into a MongoDB database. As it is written, this will produce a file that can be parsed by Python's json class.

After the child class is created we can create the instances and start the stream listener.

Calling the Stream Listener:

```
auth = OAuthHandler(ckey, consumer_secret) #OAuth object
auth.set_access_token(access_token_key, access_token_secret)

twitterStream = Stream(auth, listener(start_time, time_limit=20)) #initialize
Stream object with a time out limit
twitterStream.filter(track=keyword_list, languages=['en'])  #call the filter
method to run the Stream Object
```

Here the OAuthHandler uses your API keys [consumer key & consumer secret key] to create the auth object. The access token, which is unique to an individual user [not an application], is set in the following line. This will take all four of your credentials from the Twitter Dev site. The modified StreamListener class simply called listener is used to create a listener instance. This contains the information about what to do with the data once it comes back from the Twitter API call. Both the listener and auth instances are used to create the Stream instance which combines the authentication credentials with the instructions on what to do with the retrieved data. The Stream class also contains a method for filtering the Twitter Stream. The parameters are passed to the Stream API call.

References:

https://www.pythoncentral.io/introduction-to-tweepy-twitter-for-python/

https://github.com/tweepy/tweepy/blob/v3.6.0/docs/auth_tutorial.rst

https://lifehacker.com/5918086/understanding-oauth-what-happens-when-you-log-into-a-site-with-google-twitter-or-facebook