

Creating a single source ETL pipeline to stream tweets for 48 hours and using HiveQL for data management/analysis

This project was done to discover tweets regarding Covid for the next 48 Hours. This is just a simple ETL process without any automation except for a script to let the program know that it is beyond 48 hours and should be stopped. A further modification of this would be to use Apache Airflow to schedule and monitor the streaming/ETL process. On top of that, after doing this project and read how other engineers approach streaming, a lot of unnecessary scripts were done and a lot of automation could be done to improve the pipeline processes.

Part 1 would talk about extracting the tweets from Twitter API.

Part 2 would talk about transforming the tweets based on the given criteria

Part 3 would talk about loading the 48 hours streamed data into DynamoDB

Part 4 would talk about using an EMR clusters on the streamed data.

Part 1: Extract / Streaming Tweets

All scripts would be done in a python notebook called **Stream_tweet.ipynb**.

The streaming was first done by acquiring permissions from the Twitter Developer app as developers requires permission from Twitter to access public tweets. The access would be done by obtaining the **consumer_key, consumer_secret, access_token and access_token_secret**. These four access keys would allow developers to stream tweets. This experiment would then be done by importing the tweepy module.

The requirement for this experiment is to stream public tweets inside the United Kingdom with 5 required hashtags/words. To meet these requirements, first, a bounding-box details of United Kingdom is required as the tweepy module allows location filtration using bounding-box. Figure 1 shows the bounding box of United Kingdom.

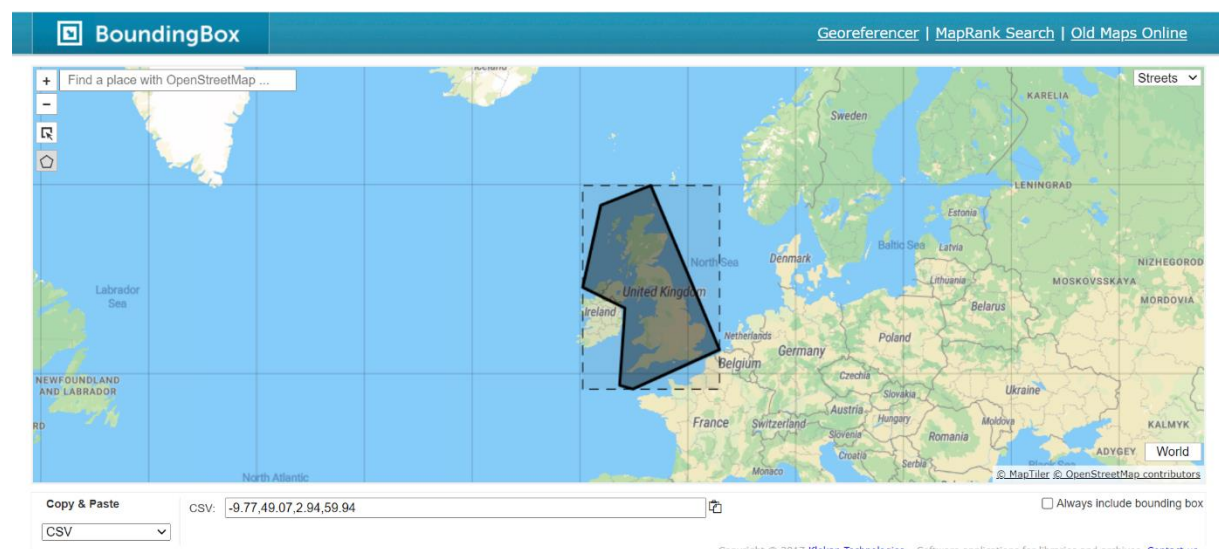


Figure 1: Bounding-Box details of United Kingdom

The second requirement would be achieved by creating a list called hashtags with all the required hashtags/words inside.

The tweepy module would then have a built-in class called myStreamListener where the streaming happens. A snapshot of the class can be seen below:

Figure 2 shows that there are 4 function inside the class myStreamListener. The first function on_connect would just verify that the connection to the streaming API is successful. The second function __init__ would be used to initialize the time limit for the streaming to stop and this experiments time limit would be 48 hours or 172800 seconds. The third function on_error would just be a function that would display the error if needed. The last function on_data is what happen with the streamed data.

```
class myStreamListener(tweepy.StreamListener):
    def on_connect(self):
        # Called initially to connect to the Streaming API
        print("You are now connected to the streaming API.")

    def __init__(self, api=None):
        # Set timer for initial time of reference
        self.start_time = time.time()
        self.limit = 172800 # set time limit in seconds
        super(myStreamListener, self).__init__()
        self.num_tweets = 0

    def on_error(self, status_code):
        # On error - if an error occurs, display the error / status code
        print('An Error has occurred: ' + repr(status_code))
        return False

    def on_data(self, data):
        try:
            # Decode the JSON from Twitter
            datajson = json.loads(data)

            if any(word in datajson['text'] for word in hashtags):
                #load all the required data
                created_at = datajson['created_at']
                location = datajson['user']['location']
                name= datajson['user']['name']
                tweet = datajson['text']
                mention_counts = len(datajson['entities']['user_mentions'])

                #dumping json file for references and backup if needed
                json.dump(datajson, twitter_txt, indent= 3)

                #Writing to csv file the needed attributes
                with open('actual_tweet.csv','a', newline='', encoding = "utf-8") as f:
                    writer = csv.writer(f)
                    writer.writerow([created_at,location,name,tweet,mention_counts])

                #Closing the stream after 2 days
                self.num_tweets += 1
                if (time.time()- self.start_time) < self.limit:
                    print("Tweet ", self.num_tweets)
                    return True
                else:
                    return False
        except Exception as e:
            print(e)
```

Figure 2: myStreamListener class

The function on_data would read the JSON file from the stream and insert it into a JSON variable called datajson. It would then filter the required hashtags/words and only obtain the relevant fields from datajson. The function also included a backup file called **jsondump.txt** to include all the streamed json files. The relevant fields would then be written into a csv file called **actual_tweets.csv**. A snapshot of the csv file can be seen on Figure 3.

```
Create_at,location,Username,Text,Mention_counts
Mon May 03 21:17:16 +0000 2021,"London, England",Kent,For the love of #dogecoin #DogeCoinarmy #DogeDay420 @DogeUK @dogecoin @TegelerSee @DogeUK @DogeCoinM
Mon May 03 21:18:25 +0000 2021,she/her,india-rose 🌸🌸,Petition: Make non-binary a legally recognised gender identity in the UK https://t.co/h1k0dZtVG1,6
Mon May 03 21:19:51 +0000 2021,Tramworld,Neil Hardline Centrist Swan 🐦,"@maverickcalgary Believe me. Alberta at its stupidist, is nowhere near the shits!
Mon May 03 21:20:57 +0000 2021,Blaenau Gwent🐦,Sophie🐦,"🇬🇧 UKIP and Abolish end up splitting the anti devolution vote in Wales so don't gain a single seat

🇬🇧 Lab short of... https://t.co/zyWQCQENGcJ",0
Mon May 03 21:21:16 +0000 2021,Cornwall,dawn portman,"@NLB_UK Haven't been to Scotland for a while but one of my favourite places to sit. Next to lighthou
Mon May 03 21:21:28 +0000 2021,Dublin,Henry Minogue,@Magician147 @Betfred @WeAreHST @RocketYardSport @TileMountainUK @Parriscues Hard luck Shaun. Not to
Mon May 03 21:21:35 +0000 2021,"Armagh, Northern Ireland",John_ni72,@richardtgarland @slay4ever007 The Protocol was 'imposed' via an overwhelming democrat
Mon May 03 21:23:53 +0000 2021,South East London,Kyle Sewell,@NewsForAllUK @DailyMirror Just fuck off,2
Mon May 03 21:23:57 +0000 2021,Lost in cheshire,Russ Jones,"The UK produces 52% of the food required to feed its population."
```

Figure 3: Snapshot on CSV files.

Part 1: Problems

Due to the WIFI connection being disconnected and reconnected constantly, an error as in Figure 4 appeared.

```
201         return six.b('')
202

~\anaconda3\lib\site-packages\urllib3\response.py in read(self, amt, decode_content, cache_content)
539             # raised during streaming, so all calls with incorrect
540             # Content-Length are caught.
--> 541         raise IncompleteRead(self._fp_bytes_read, self.length_remaining)
542
543     if data:

~\anaconda3\lib\contextlib.py in __exit__(self, type, value, traceback)
129         value = type()
130     try:
--> 131         self.gen.throw(type, value, traceback)
132     except StopIteration as exc:
133         # Suppress StopIteration *unless* it's the same exception that

~\anaconda3\lib\site-packages\urllib3\response.py in _error_catcher(self)
453     except (HTTPException, SocketError) as e:
454         # This includes IncompleteRead.
--> 455         raise ProtocolError("Connection broken: %r" % e, e)
456
457         # If no exception is thrown, we should avoid cleaning up

ProtocolError: ('Connection broken: IncompleteRead(2 bytes read)', IncompleteRead(2 bytes read))
```

Figure 4: Error due to inconsistent connection.

Therefore, the error was combated by renaming the finished files and running the python program again for an extra day. Both streamed csv file would then be merged by running an external python script as Figure 5.

Part 2: Pre-processing Tweets

The pre-processing method was done in a python notebook called **Big Data Experiment 2.ipynb**.

The merged files from figure 4 was first converted into a pandas data frame prior to cleaning the textual data. The required module was also imported. This can be seen in Figure 6.

```

▶ M1

#change csv to panda dataframe
import pandas as pd
import re
import emoji
from nltk.corpus import stopwords
stop = stopwords.words('english')
df = pd.read_csv('C:/Users/Asus/Desktop/Backup/All Files/MergedFile.csv')
print("done")

```

Figure 6: CSV to pandas dataframe and importing modules.

The textual cleaning script would be as in Figure 7. The cleaner function will remove any mentions, links, required hashtags/words and emoji. The texts would then be cleaned again by removing all English stopwords from the nltk.corpus module.

```

▶ M1

# To clean all the text data
def cleaner(tweet):
    tweet = tweet.lower()
    tweet = re.sub("@[A-Za-z0-9]+","",tweet) #Remove @ sign
    tweet = re.sub(r"(?:\@|http?\:\/\/|https?\:\/\/|www)\S+", "", tweet) #Remove http links
    tweet = re.sub("#", "", tweet) #Remove hashtags
    tweet = re.sub("covid19","",tweet)
    tweet = re.sub("coronavirus","",tweet)
    tweet = re.sub("unitedkingdom","",tweet)
    tweet = re.sub("uk","",tweet)
    tweet = re.sub("india","",tweet)
    tweet = " ".join(tweet.split())
    tweet = ''.join(c for c in tweet if c not in emoji.UNICODE_EMOJI) #Remove Emojis
    return tweet

▶ M1

tweet_text = df['Text'].map(lambda x: cleaner(x))

▶ M1

# Removing Stopwords (It tooks way too long when doing it on the function cleaner thus doing it outside seems more viable)
tweet_text = tweet_text.str.split()
tweet_text = tweet_text.apply(lambda x: ' '.join([item for item in x if item not in stop]))

```

Figure 7: Textual Cleaning Process

The required hashtags/words would then be counted by running the function wordcount as in Figure 8. All counts would then be inserted into their respective variable. All modified/relevant features would then be merged into a final dataframe before being converted into a csv file called **Clean_tweets_s3.csv**. Figure 9 shows a snapshot on the final dataframe.

```

M1
#Count for all words/hashtags
count1 = df['Text'].apply(wordcount, substring='covid19')
count2 = df['Text'].apply(wordcount, substring='coronavirus')
count3 = df['Text'].apply(wordcount, substring='unitedkingdom')
count4 = df['Text'].apply(wordcount, substring='uk')
count5 = df['Text'].apply(wordcount, substring='india')

M1
df1 = pd.DataFrame()

M1
df1['tweet_text']=tweet_text
df1['Covid19_counts']=count1
df1['CoronaVirus_counts']=count2
df1['UnitedKingdom_counts']=count3
df1['UK_counts']=count4
df1['India_counts']=count5
df1['location']=df['location']
df1['Mention_counts']=df['Mention_counts']

```

Figure 8: Wordcount functions

M1

df1

	tweet_text	Covid19_counts	CoronaVirus_counts	UnitedKingdom_counts	UK_counts	India_counts	location	Mention_counts
0	love dogecoin dogecoinarmy dogeday420 _	0	0	0	2	0	London, England	6
1	petition: make non-binary legally recognised g...	0	0	0	1	0	she/her	0
2	believe me. alberta stupidist, nowhere near sh...	0	0	0	1	0	Tramworld	1
3	ip abolish end splitting anti devolution vot...	0	0	0	1	0	Blaenau Gwent	0
4	_ scotland one favourite places sit. next ligh...	0	0	0	1	0	Cornwall	1
...
4525	loooove kingfisher picture 🐟	0	0	0	1	0	Armley	3
4526	exist australian british cinema hard think equ...	0	0	0	1	0	Eastbourne, UK	0
4527	writing exclusively _nw, says figures show nor...	0	0	0	1	0	North West, England	4
4528	finally got vaccine don't know everyone worrie...	1	0	0	0	0	Belfast	0
4529	great opportunity get involved bssh overseas w...	0	0	0	1	0	North West, England	3

Figure 9: Final Dataframe

Part 3: Importing CSV into DynamoDB

The last part for experiment 2 would be Importing the final-cleaned csv file into DynamoDB. The csv file was uploaded into AWS DynamoDB by using the data model in Figure 10.

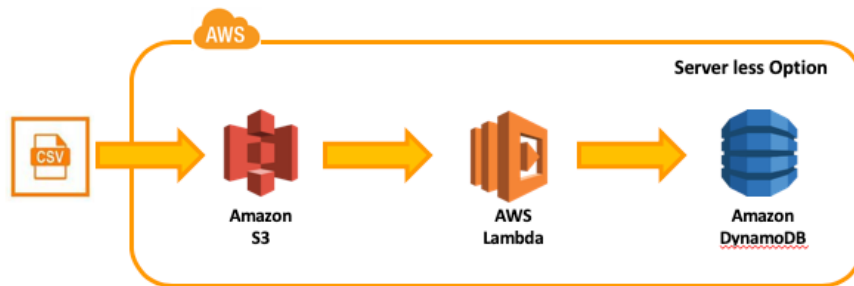


Figure 10: Data Model for CSV files to DynamoDB

The csv file was uploaded into a created S3 bucket called 17114101-bucket. The bucket would then be accessed by a lambda function that would read any uploaded csv files by activating a trigger on the S3 with the Lambda Function. The Lambda function would be called csv-2-dynamodb. A snapshot of the function overview can be seen in Figure 11.

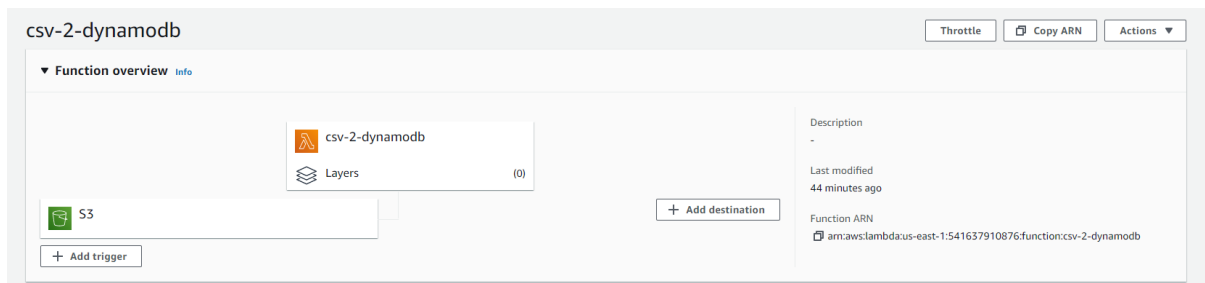


Figure 11: Lambda Function Overview.

The codes for uploading CSV files into DynamoDB can be seen as a snapshot in Figure 12 and the verification can be seen in Figure 13. The Snapshot of the DynamoDB table can be seen on Figure 14.

The full codes to upload the CSV file into DynamoDB is on the text file called **lambda_functions.txt**.

```

for row in csv_reader:
    tweet_num = row[0]
    tweet_text=row[1]
    Covid19_counts = row[2]
    CoronaVirus_counts = row[3]
    UnitedKingdom_counts = row[4]
    UK_counts = row[5]
    India_counts = row[6]
    location = row[7]
    Mention_counts = row[8]

    print('Tweet Num: ', tweet_num, 'tweet_text: ', tweet_text)

    add_to_db = dynamodb.put_item(
        TableName = 'clean_tweet',
        Item = {
            'tweet_num' : {'N': str(tweet_num)},
            'tweet_text' : {'S': str(tweet_text)},
            'Covid19_counts' : {'N': str(Covid19_counts)},
            'CoronaVirus_counts' : {'N': str(CoronaVirus_counts)},
            'UnitedKingdom_counts' : {'N': str(UnitedKingdom_counts)},
            'UK_counts' : {'N': str(UK_counts)},
            'India_counts' : {'N': str(India_counts)},
            'location' : {'S': str(location)},
            'Mention_counts' : {'N': str(Mention_counts)},
        })

    print('Successfully added records to DynamoDB Table')

```

Figure 12: Adding csv to DynamoDB Codes.

Successfully added records to DynamoDB

Figure 13: Verification that the CSV file has been added to DynamoDB.

Scan: [Table] clean_tweet: tweet_num Viewing 1 to 23 items

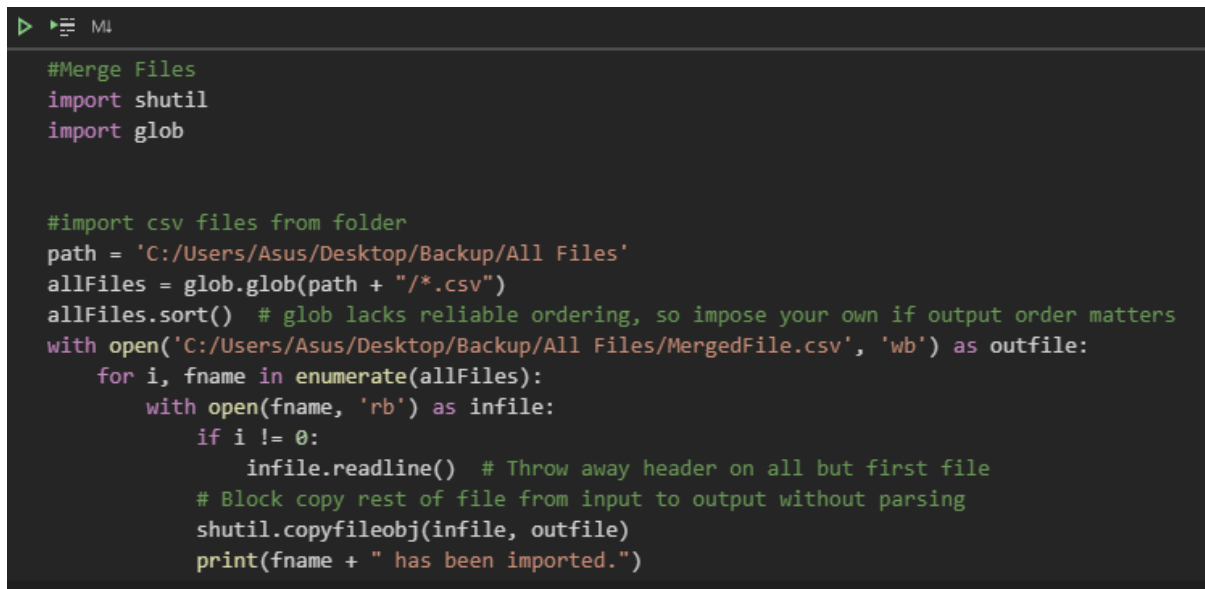
Scan [Table] clean_tweet: tweet_num ^

+ Add filter

Start search

	tweet_num	CoronaVirus_co	Covid19_counts	India_counts	Mention_counts	UK_counts	UnitedKingdom	location	tweet_text
<input type="checkbox"/>	0	0	0	0	6	2	0	London, England	love dogecoin dogecoinarmy doge
<input type="checkbox"/>	1	0	0	0	0	1	0	she/her	petition: make non-binary legally
<input type="checkbox"/>	2	0	0	0	1	1	0	Tramworld	believe me. alberta stupidist, no
<input type="checkbox"/>	3	0	0	0	0	1	0	Blaenau Gwent	ip abolish end splitting anti c
<input type="checkbox"/>	4	0	0	0	1	1	0	Cornwall	_ scotland one favourite places s

Figure 14: Snapshot on DyanamoDB table called clean_tweet.



```

#Merge Files
import shutil
import glob

#import csv files from folder
path = 'C:/Users/Asus/Desktop/Backup/All Files'
allFiles = glob.glob(path + "/*.csv")
allFiles.sort() # glob lacks reliable ordering, so impose your own if output order matters
with open('C:/Users/Asus/Desktop/Backup/All Files/MergedFile.csv', 'wb') as outfile:
    for i, fname in enumerate(allFiles):
        with open(fname, 'rb') as infile:
            if i != 0:
                infile.readline() # Throw away header on all but first file
            # Block copy rest of file from input to output without parsing
            shutil.copyfileobj(infile, outfile)
            print(fname + " has been imported.")

```

Figure 5: Merging streamed files.

This resulted in a file that contains streamed tweets from 3/5/2021 – 6/5/2021 called **MergedFile.csv**. The extra day was inserted to recover the lost time due to downtime.

Part 4: HiveQL

Hadoop is a framework that is used widely to manage Big Data and Hive is an application or a data warehouse system that runs on top of Hadoop to provides SQL like interface for processing/querying big data. Instead of only loading data into HDFS systems, Hive can create external table that could sourced the data from external sources like DynamoDB. This is significant in making the query of big data easier and accessible as the querying of Big Data becomes less daunting. On top of that, Hive architecture has multiple interfaces from Web User Interface to Command Line Interface and this allow seamless user integration into Big Data architecture.

Experiment 3 consist of using the Hive architecture to query the uploaded DynamoDB table called clean_tweets from Experiment 2. The set up was done by creating an Amazon Elastic MapReduce (EMR) cluster that would run Apache Hadoop or Apache Spark. Experiment 3 only emphasis on the usage of Apache Hadoop. The cluster would then be created with details found on Figure 15.

Cluster: My cluster Waiting Cluster ready after last step completed.

Summary Application user interfaces Monitoring Hardware Configurations Events Steps Bootstrap actions

Summary

ID: j-7YYS4YRISU1K

Creation date: 2021-05-21 04:37 (UTC+1)

Elapsed time: 2 hours, 4 minutes

After last step completes: Cluster waits

Termination protection: Off [Change](#)

Tags: -- [View All](#) / [Edit](#)

Master public DNS: ec2-18-208-140-38.compute-1.amazonaws.com [Connect to the Master Node Using SSH](#)

Configuration details

Release label: emr-6.2.0

Hadoop distribution: Amazon 3.2.1

Applications: Hive 3.1.2, Hue 4.8.0, Pig 0.17.0, Tez 0.9.2

Log URI: s3://aws-logs-653080283521-us-east-1/elasticmapreduce/ [View](#)

EMRFS consistent view: Disabled

Custom AMI ID: --

Application user interfaces

Persistent user interfaces [View](#): YARN timeline server, Tez UI

On-cluster user interfaces [View](#): Not Enabled [Enable an SSH Connection](#)

Network and hardware

Availability zone: us-east-1d

Subnet ID: [subnet-a2fef7ef](#) [View](#)

Master: Running 1 m5.xlarge

Core: Running 2 m5.xlarge

Task: --

Cluster scaling: Not enabled

Security and access

Key name: trykeypair

EC2 instance profile: EMR_EC2_DefaultRole

EMR role: EMR_DefaultRole

Visible to all users: All [Change](#)

Security groups for Master: [sg-0d887abdbb3ff50d](#) [View](#) (ElasticMapReduce-master)

Security groups for Core & Task: [sg-03d98260b133c0b34](#) [View](#) (ElasticMapReduce-slave)

Figure 15: Amazon EMR Cluster Console

Figure 15 shows that the cluster was created with these details:

- Release label: emr-6.2.0
- Hadoop distribution: Amazon 3.2.1
- Applications: Hive 3.1.2 and other defaults core applications
- Key/pair name: trykeypair.ppk
- Instances: 1 Master: m5.xlarge, 2 Core: m5.xlarge

The Key/pair is needed to establish SSH connection with the local PC to run EMR cluster and Hive applications. The Key/pair should also be hidden safely to prevent any outside intrusion on the cluster. The EMR Cluster SSH Connection was established using Putty as the cluster was running on Windows 10. Once the connection has been established, the connection between Hive and DynamoDB was created by creating an external Hive Table using these lines of codes in Figure 16.

```
CREATE EXTERNAL TABLE tweets
(
  tweet_num BIGINT,
  corona BIGINT,
  covid BIGINT,
  india BIGINT,
  mention BIGINT,
  uk BIGINT,
  united BIGINT,
  location STRING,
  text STRING)

STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'

TBLPROPERTIES(
  "dynamodb.table.name" = "clean_tweet",
  "dynamodb.column.mapping"="tweet_num:tweet_num,corona:CoronaVirus_counts,
covid:Covid19_counts,
      india:India_counts,mention:Mention_counts,uk:UK_counts,
      united:UnitedKingdom_counts,location:location,text:tweet_text"
);
```

Figure 16: Creating external Hive Table and Connection with DynamoDB

These codes would create a connection with DynamoDB table called clean_tweet and mapped the created Hive table with clean_tweet attributes. The DynamoDB attributes needs to be similar to the one in clean_tweet attributes. This connection would then ease the process of querying the data as HiveQL can be implemented on Hive table. There are 5 required queries:

Query 1: The country which has authored the most tweets.

Query Scripts:

```
SELECT location, COUNT(*) AS magnitude
FROM tweets
GROUP BY location
ORDER BY magnitude DESC;
```

Query Results:

```
OK
      500
London, England 162
London   136
UK       68
```

Majority of the tweets (500 tweets) does not specify their location but the most tweets from specified location is London, England with 162 total tweets.

Query 2: The most frequent hashtags/word mentioned in EXP 2 found in tweets from each country.

Query Scripts:

```
Select SUM(corona) as Corona, SUM(covid) as Covid,
SUM(india) as India, SUM(uk) as UK, SUM(united) as UnitedKingdom, location
FROM tweets
GROUP BY location;
```

Query Results:

```
OK
corona  covid  india  uk      unitedkingdom  location
2       21     15     547     0
0       0      0      4      0
0       0      0      1      0      North Yorkshire
0       0      0      1      0      UK
0       0      0      1      0      | London • Dubai |
0       0      0      1      0      gb Leicester, UK
0       0      0      5      0      #Newbury
```

From the results, it can be deducted that for the unspecified location, the most frequent hashtags/words is UK with 547 counts.

Query 3: The most frequent hashtags/words mentioned in EXP 2 found in all tweets.

Query Scripts:

```
Select SUM(corona) as v1, SUM(covid) as v2,
SUM(india) as v3, SUM(uk) as v4, SUM(united) as v5
FROM tweets;
```

Query Results:

```
OK
v1      v2      v3      v4      v5
5       94      85      4920    0
```

The result shows that there are no tweets with the hashtags/words of UnitedKingdom and the most frequent hashtags/words mentioned would be UK. The naming of each variable should be changed into the one like Query 2 as it would allow the results to be more readable.

Query 4: Total number of users mentions in tweets from each country respectively.

Query Scripts:

```
SELECT location SUM(mention) as mention_sum
FROM tweets
Group BY location;
```

Query Results:

```
location      mention_sum
922
6
North Yorkshire  1
UK              0
| London • Dubai | 3
gb Leicester, UK 0
#Newbury        20
```

As there are many listed countries, the full results can be seen on attached Query 4 Results.txt in the given folder Experiment 3. The script could be improved by arranging it in descending order.

Query 5: Total number of users mentions in all tweets.

Query Scripts:

```
Select SUM(mention) as sum_mention  
FROM tweets;
```

Query Results:

The total number of mentions would be 9850 user mentions.

Termination of clusters

Once all query on the data has been executed, the EMR Cluster should be terminated instantly as to avoid excessive billing as the bill will run in parallel with the cluster. A script could be included to automatically terminate the cluster either based on certain met conditions or certain time limit as extra precautions.