Twitter Sentiment Analysis

Big Data & Cloud Based Tools

By: Deepti Joshi, Elsa Yammin, Satadipa Sarkar

Seattle Pacific University

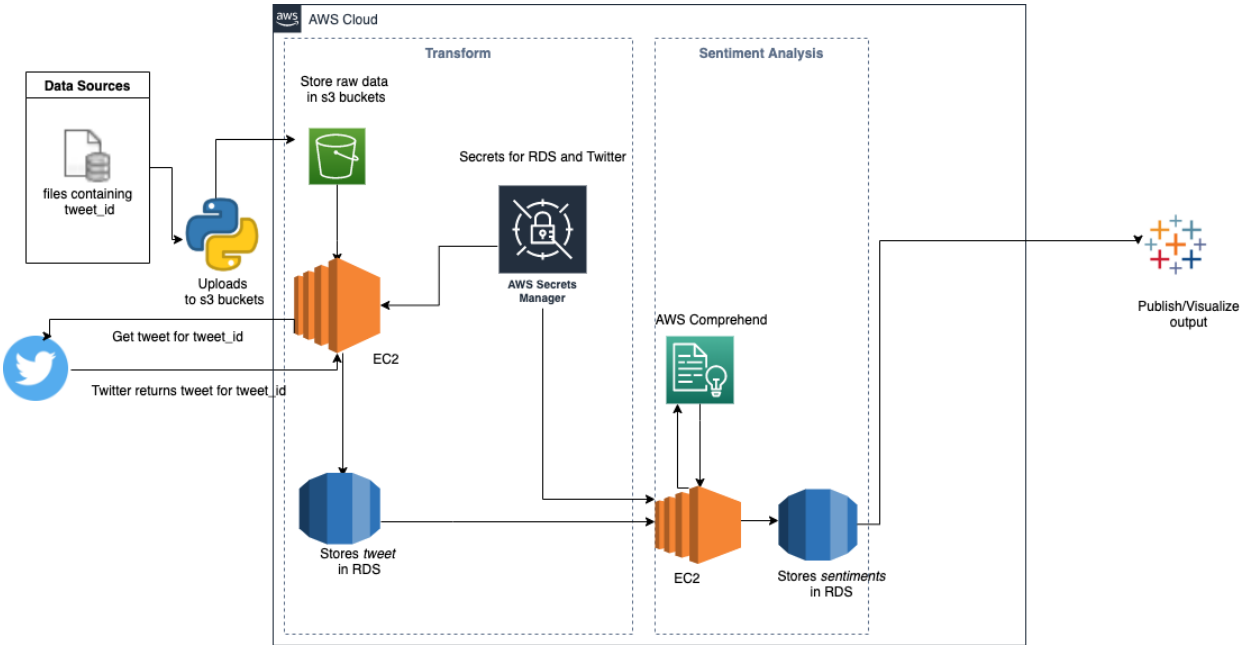
Goal

Sentiment analysis on the various tweets for certain global events of importance in the past

Dataset

- The project dataset contains 30 different Twitter datasets, associated with real world events from the year (2012-2016). This dataset cannot be used for commercial purposes.
- Reference: https://onlinelibrary.wiley.com/doi/full/10.1002/asi.24026
- The dataset comprise of only Tweet_id compressed in a GZ format. Tweeter developer account was created for API key, API Secret Key, Access token which will be used later for cloud formation stack.

Architecture Diagram



Downstream Flow:

1. The input compressed files are decompressed and uploaded to AWS S3 to be processed by downstream systems.
2. *tweet_ids* are read from the S3 bucket and corresponding tweets are fetched from Twitter API.
3. The *tweet_ids* along with the tweets are stored in RDS.

4. The tweets are read from RDS and AWS Comprehend is used to derive the sentiments for the tweets.
5. *tweet_ids* along with the sentiment are stored in RDS.
6. The sentiments are read from RDS and transformed into the required output data model-connected to Tableau.
7. Tableau gets the data from RDS by connetcing directly with the PostgreSQL Server.

Different AWS services and tools used are as follows:

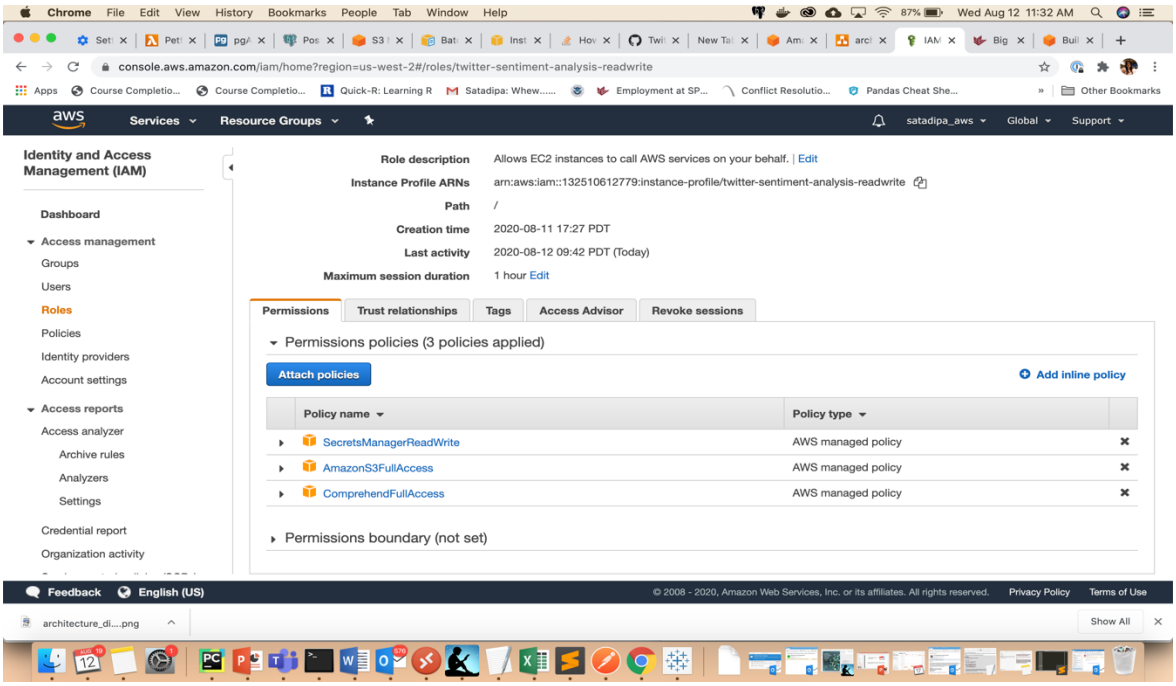
We have followed the following steps to configure the following services in AWS.

IAM

Create IAM Role to allow EC2 access to all the services that you will use.

- Select type of trusted entity to be `AWS service`.
- Select `EC2` as the use case.
- For permissions, select the following
- AmazonS3FullAccess
- SecretsManagerReadWrite
- ComprehendFullAccess
- Next, add tags if needed.
- Set role name as `twitter-sentiment-analysis-readwrite`.
- Fill in the role description.
- Create role.

Following is the screenshot of the IAM roles:



EC2

- Launch EC2 instance
- From AWS console EC2 page, select Instances and click Launch Instance.
- Select `Free tier only` checkbox from the left side bar and choose `Amazon Linux 2 AMI (HVM), SSD Volume Type` 64-bit (x86).
- Next screen, choose the processor that you need (or the free tier eligible t2.micro).
- Configure Instance Details.
- Change the IAM Role to use `twitter-sentiment-analysis-readwrite`.
- Check `Protect against accidental termination`.
- Storage: Leave default.
- Tags: Add if needed.
- Security groups: Leave default.
- Next screen, leave default and Launch.
- EC2 will ask you to create a new key-pair. Choose "Create a new key-pair", give it a good name and download the .pem file. Keep the file safe. We will need it to connect to the EC2.
- The .pem file is called the key file. The downloaded key file has permissions which are "too open". We want to restrict these permissions so that the file is read only and therefore secure from any tampering. Run `chmod 400 /path/key-pair.pem`.
- Click View Instance and wait for the state to become running.

Connecting to EC2 instance

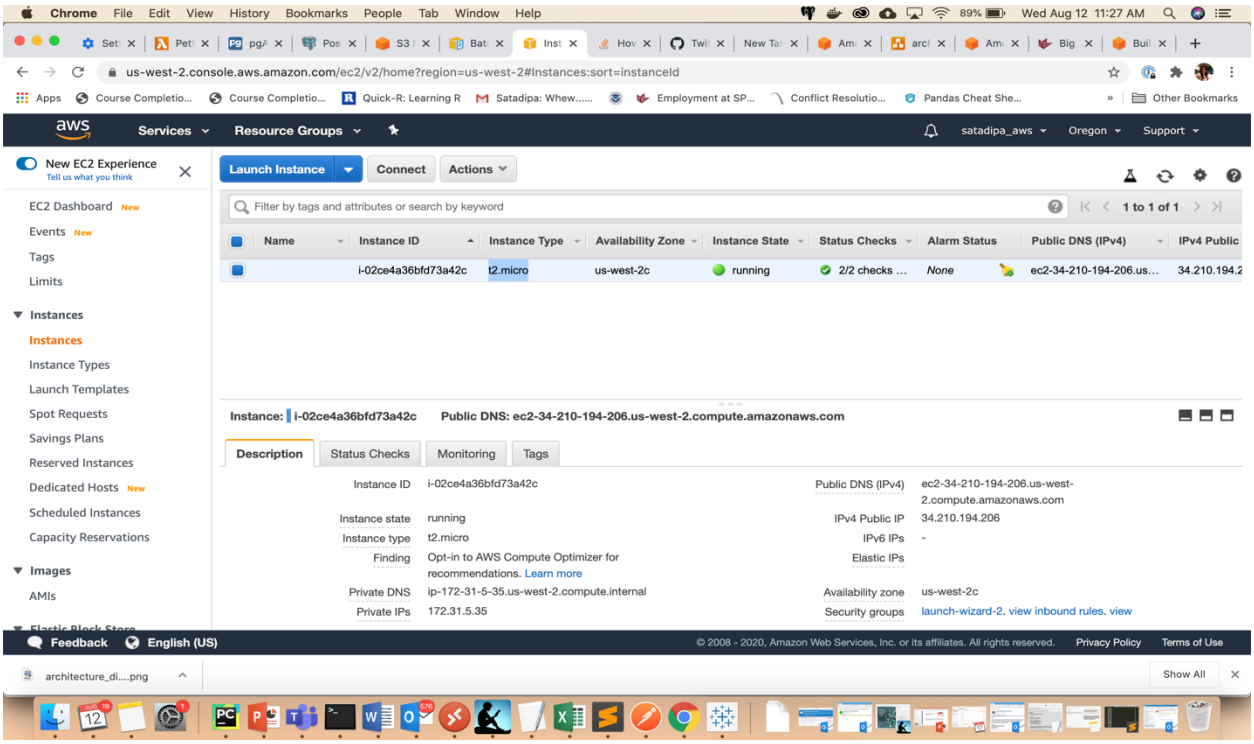
- Grab the Public DNS (IPv4) address from the EC2 AWS Console. Located under EC2 -> Instances.
- Ensure that the instance state is running.
- Run `ssh -i /path/key-pair.pem ec2-user@my-instance-public-dns-name`

The terminal will ask for confirmation to connect to the EC2 instance, type `Yes`.
First time logging in, run a system update for Linux. Run `sudo yum update`. Follow the prompts and ensure system is up to date.

- Install Python
- Script to test connection

- Some of the steps may not be needed.
- Run `python --version`. If not 3.7 or higher, install Python 3 using the following steps.
- Run `sudo amazon-linux-extras install python3.8`.
- Verify install by running `python3.8 -V`.
- Download pip. Run `curl -O https://bootstrap.pypa.io/get-pip.py`.
- Install pip. Run `python3.8 get-pip.py --user`.
- Install boto3. Run `pip install boto3`.
- Install python psql connector. Run `pip install psycopg2-binary`.
- Set up RDS and then come back here.
- Create a new python file using terminal
- Run `vim rds_connection_test.py`
- Copy the code from [rds_connection_test.py](#rds_connection_test.py)
- Run this file `python3.8 rds_connection_test.py`

Following is the screenshot of the EC2 instance created:

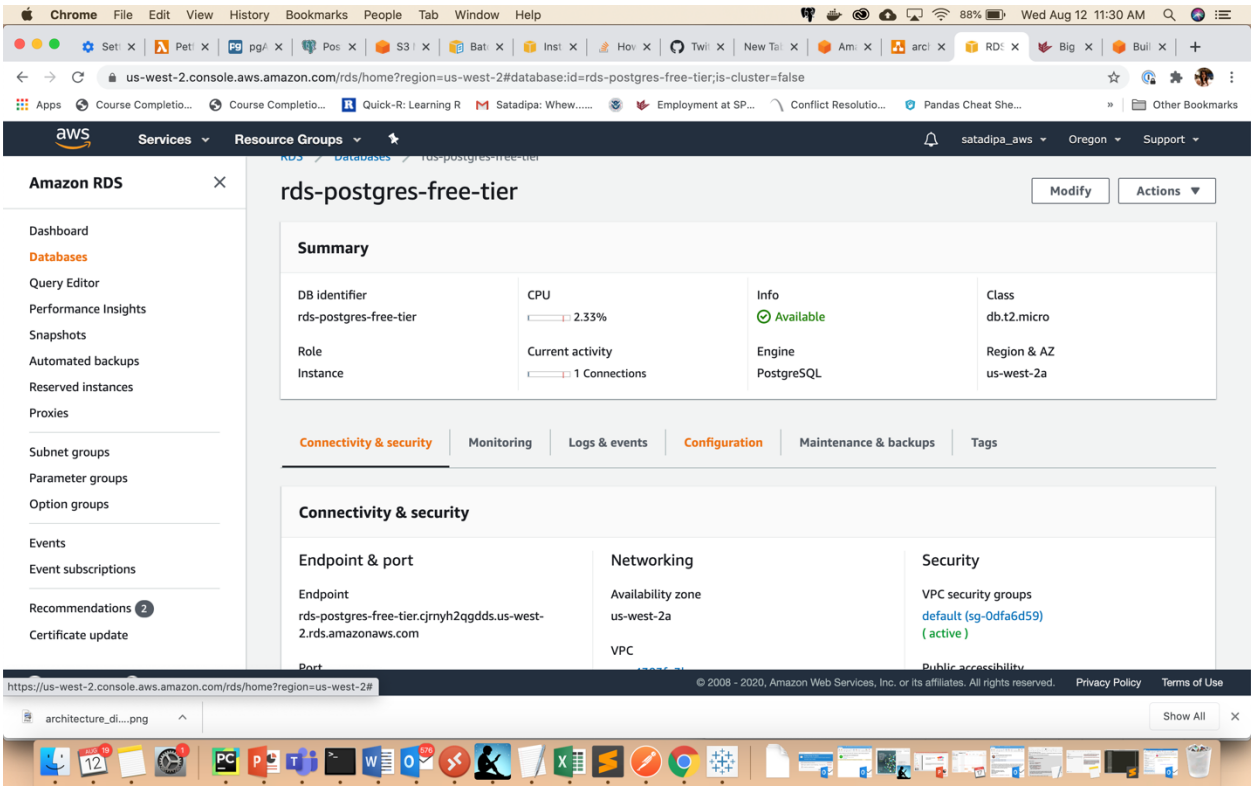


RDS

- Launch RDS **Postgres Instance**
- Standard Create
- PostgreSQL
- Free Tier
- Set DB Instance identifier `rds_postgres_free_tier`
- Credential Settings
- Master username: `postgres`
- For password: Generate a password in 1Password and save it, keep the username as `postgres`. Copy the password from 1Password and apply it to the RDS postgres configuration.
- DB instance size: No change
- Storage: No change
- Connectivity: Should be set to `Default VPC`
- Database authentication: Password authentication
- Addtional configuration
- Initial database name : `big_data_project` .Create database
- Wait for the database status to become Available.
- Go to EC2 instance and copy the `Security groups` for it.
- Go to RDS database and click `Connectivity & security`.
- Click on the `default` `VPC security groups` listed.
- Click `Edit inbound rules`.

- For the rule which has, `Type=All traffic`, `Protocol=All`, `Port range=All`, `Source=Custom`
- Add the EC2 instance's security group.
- Click save rule.

Following is the snapshot of the RDS instance created:

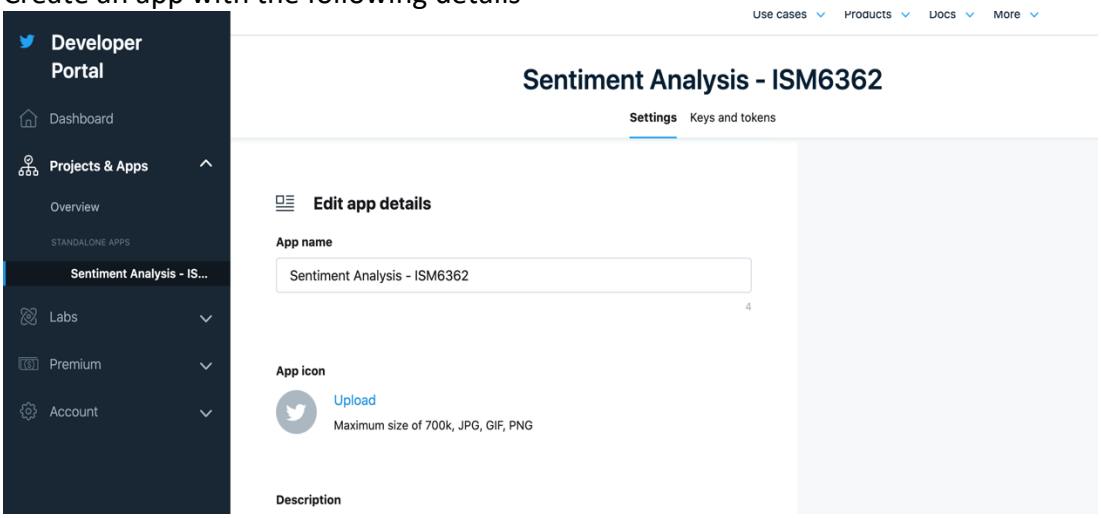


Secrets Manager

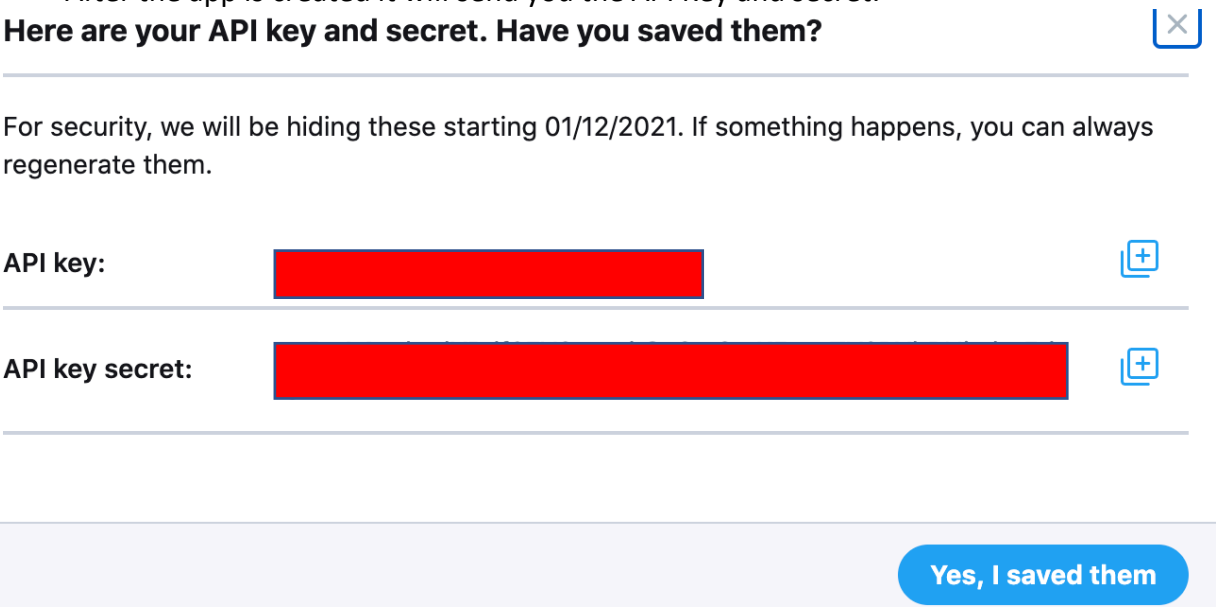
- Storing the RDS password
- Click `Store a new secret`
- Choose `Other type of secrets`.
- Choose `Plaintext`
- Paste the database password that you stored in 1Password as a simple string without quotes.
- Click `Next`
- Set `Secret Name` to `rds_postgres_free_tier`. Click `Next`.
- Click `Next`.
- Click `Store`.

Twitter API Connection check

- Created a developer account.
- Create an app with the following details

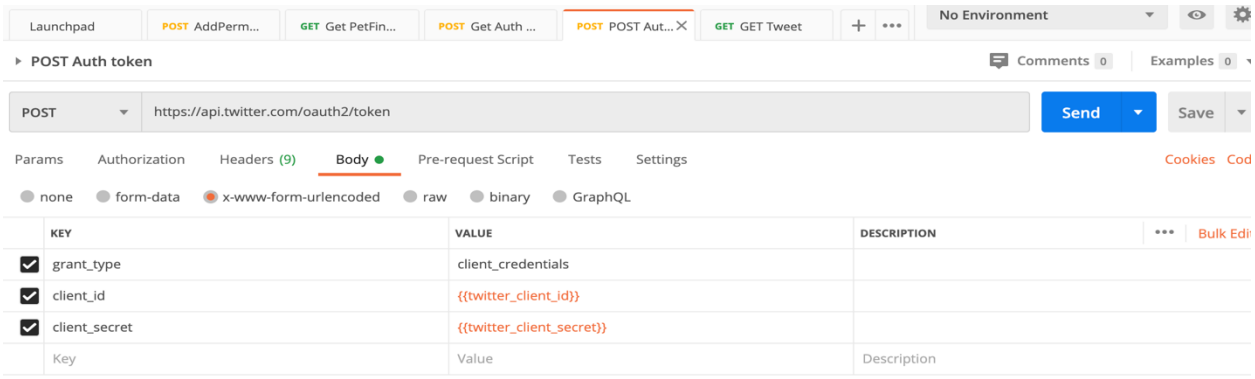


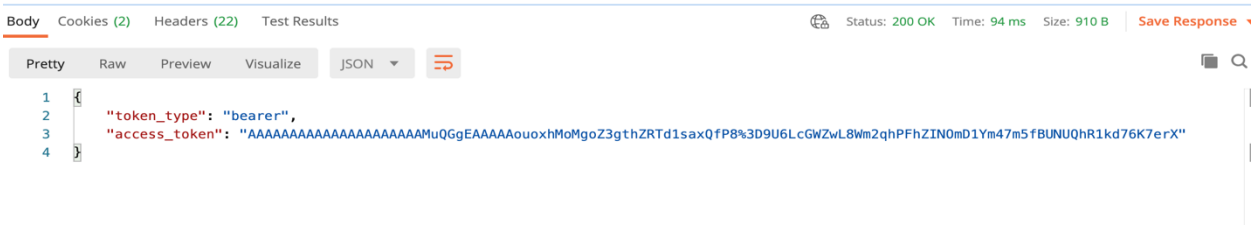
- After the app is created it will send you the API Key and secret.



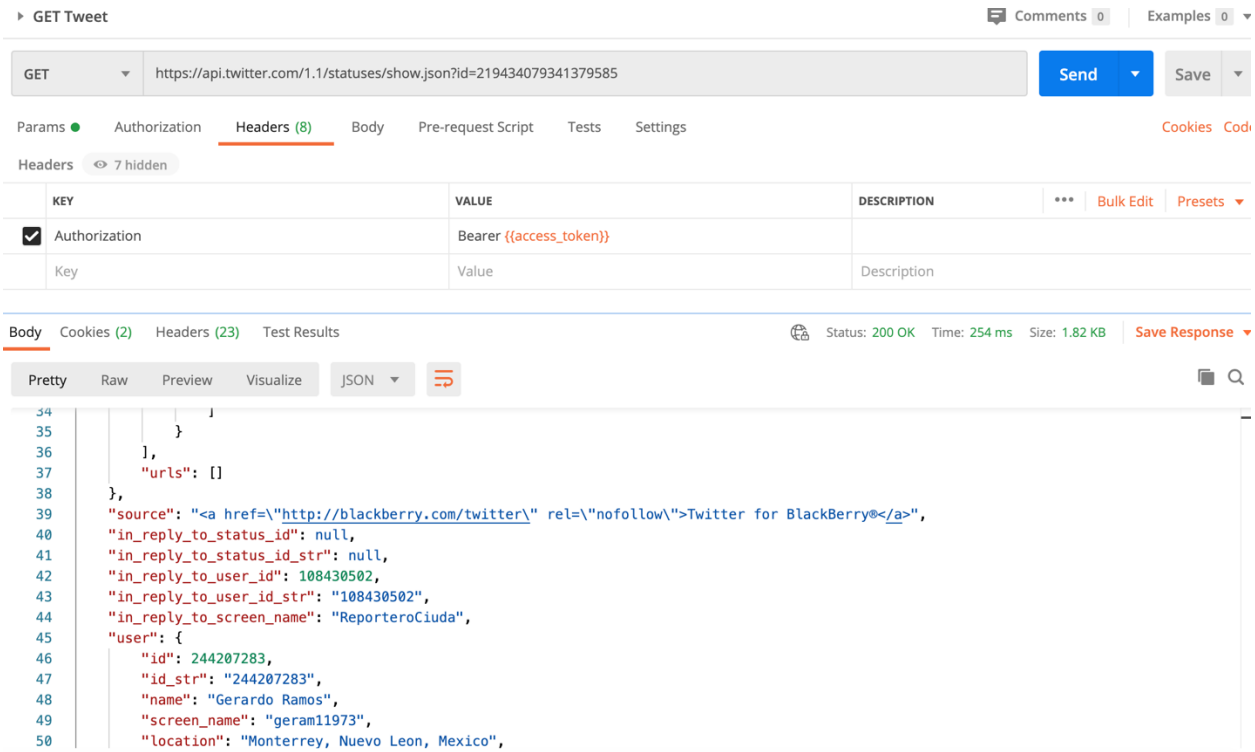
- We will need access tokens to get the tweets for the tweet_id stored in amazon s3.

- Using Postman create a new collection called Twitter API, and create an add request: POST Auth token, with the following parameters in the body to obtain a request token.





- Create an add request: GET Tweet, with the following parameters in the headers to obtain a request token as shown below



Successfully! got tweets for the tweet_id's stored in s3. But for few there was no access.

Python script steps:

- Unzip the tweet_id files with .gz extension and upload to S3.
- For each file from S3, load tweet_id into RDS table in batch.
- Create a table in RDS with table schema as follows:
-

tweet_id	tweet_response	tweet_text	sentiment_score_positive	sentiment_score_negative	sentiment_score_neutra	sentiment_score_mixed	sentiment	workflow_status
[PK] character varying	json	character varying	numeric	numeric	numeric	numeric	character varying	character varying

- For each unprocessed tweet_id, fetch the tweet, update row in PostgreSQL with the tweet.
- Update the workflow_status =”TWEET_FETCHED”
- Get batch of tweet with workflow_status “TWEET_FETCHED” and it calls Comprehend.
- Store the response from Comprehend into RDS.And update the status as “COMPLETED”

Why we did not do run all the tweet_id’s using AWS Comprehend?

- Number of tweet_id = 4.02 million
- Approximate character count = 300
- Approximate units per tweet =300/100 = 3
- Chargeable units = 12.06 million
- Cost up to 10 million units = 0.0001/unit
- Free tier =50K units
- Billable units = 12.06 million -50,000 = ~12 million
- Approximate cost of the project = 0.0001 * 12 * 10^6 = $1200

Why EC2 not Lambda?

- EC2 is free.
- Lambda does not support most of the libraries. And installing different libraries was difficult and cumbersome using Layers.
- The development could be done on personal laptop and same code was executed on EC2.
- Development and deployment are much more streamlined.
- No need of ant special code, while Lambda requires special entry methods which is hard to replicate on local machine.
- No way to connect directly to GitHub repository.

Why RDS and not DynamoDB?

- It is a relational database.

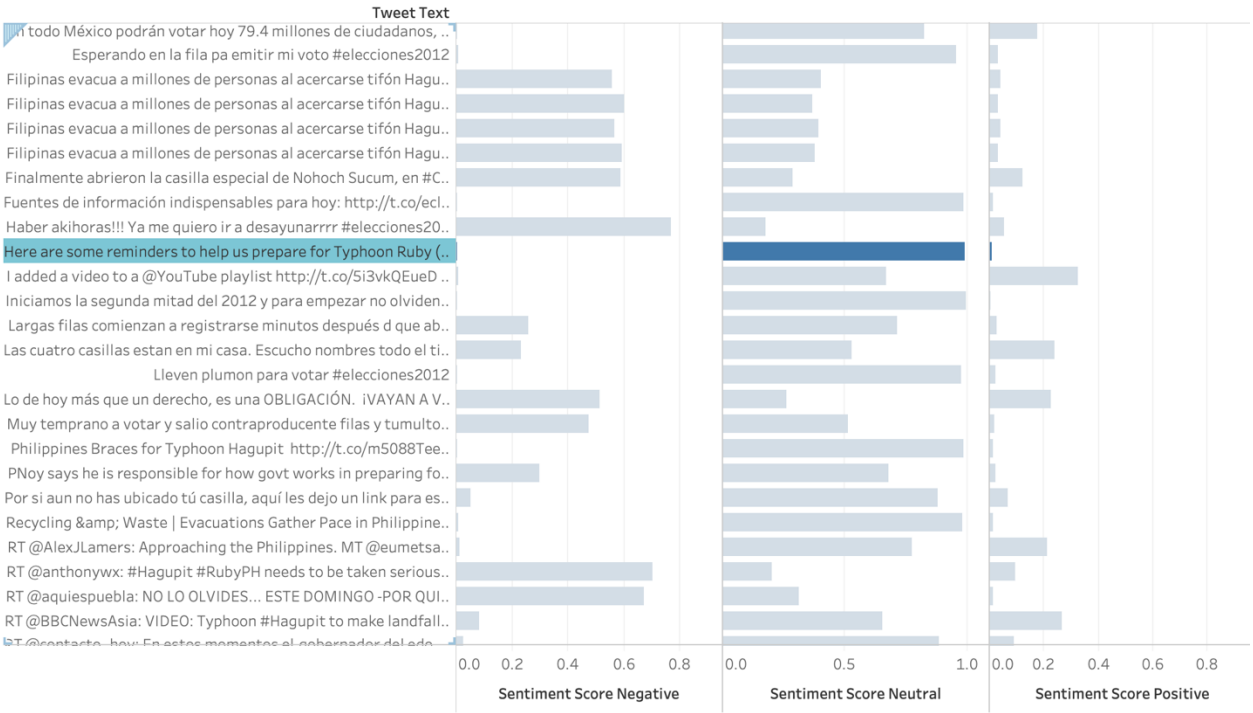
- DynamoDB is good for writing once and reading multiple times on the primary key. However, in this project the workflow was on reading the status of the record.
- Relational database is superior in read -write scenario. In this project (2 Reads and 3 Writes).
- Easier to connect with data visualization tools (Tableau).
- RDS cost is lesser.

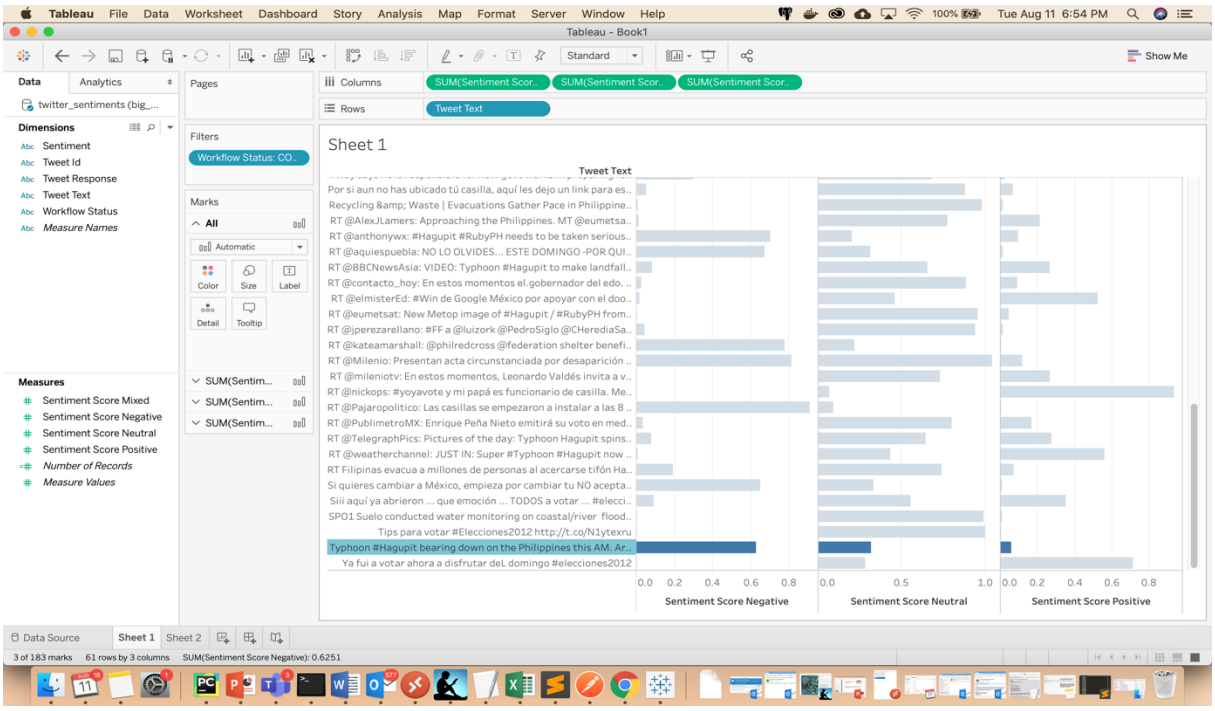
Following is the run output:

Data Visualization with Tableau:

- Connect to a server :
- Select PostgreSQL
- Go to RDS:
- Select Endpoint: rds-postgres-free-tier.cjrnyh2qgdds.us-west-2.rds.amazonaws.com
- Database: rds-postgres-free-tier

Visualization below shows the sentiment of the highlighted tweet. The visualization chart also shows various tweets, with positive, negative and neutral sentiment score as illustrated below.





Scope of Improvements:

- **Cloud formation can be used in future for scaling-** It can be definitely be useful when we are scaling out to multiple regions, in our case it is only one region-US-West-2. We wanted to learn the nitty-gritty of various services and components used by hand.
- Leverage **Amazon Kinesis Data Firehose** to easily capture, prepare, and load real-time data streams into data stores, data warehouses, and data lakes.