CIS 9760 Project 1

This is a CLI application to extract Open Parking and Camera Violations dataset from NYC open data to an Elasticsearch instance hosted on AWS. Once the data has been extracted, it can be visualized in Kibana.

Pre-Requisites:

1. **Machine with docker installed. Eg: EC2 instance or localhost with docker installed.**
2. **Elasticsearch instance up and running. This could be in the cloud or on a machine. The important factors are the availability of the ES host address for access, and a username and password for authentication.**
3. **APP token generated from the Socrata open data API to access and extract data.**

How to run the application:

1. **Extract the zipped files.**
2. **In the terminal, navigate to the directory of the unzipped file.**
3. **Run the following command:**

   docker build -t bigdata1:1.0 project01/

   This will build a docker image based on the instructions in the Dockerfile which exists in the project01 folder.
4. **Once the docker image is built, we can run a docker container which will execute the application. In the application, we will have to pass a few environment variables. They are:**
   a. **DATASET_ID: For OPCV, the id is 'nc67-uf89'**
   b. **ES_USERNAME: The ES username for access**
   c. **ES_PASSWORD: The ES password for access**
   d. **APP_TOKEN: APP token generated for the Socrata Open Data API**
   e. **ES_HOST: The address of the elasticsearch host to be used.**
5. **In addition to the above environment variables, we will also pass 2 command line arguments:**
   a. **Num_pages: Number of pages of the data to fetch**
   b. **Page_size: Size of each page which is to be fetched.**

6. In order to run the docker container, the following command must be executed and the variable values from step 4 and 5 must be substituted:

```
docker run \
     -e DATASET_ID="nc67-uf89" \
     -e APP_TOKEN="<YOUR APP_TOKEN>" \
     -e ES_HOST="<YOUR ES HOST>" \
     -e ES_USERNAME="<USER ID>" \
     -e ES_PASSWORD="<PASSWORD>" \
     --network="host" \
     bigdata1:1.0 --num_pages=<NUMBER OF PAGES NEEDED>
--page_size=<REQUIRED PAGE SIZE>
```

On running the above command the program will begin execution and will extract the data from NYC Open data.

7. In order to view this data, you will need to login to the Elasticsearch instance.
8. Once logged in to the elasticsearch instance, navigate to 'Stack Management' and click on Index Patterns -> Create Index pattern
9. Enter the characters 'my*' and you will see an index by the name 'my-index-1'. This is the index to which the program will write the data to. Click on next and select issue_date as the time field with which to view the data.
10. Click on create index pattern.
11. Navigate to Discover and search for the index pattern 'my*' from the top left corner.
12. Once you have navigated to Discover, you can view the data as it is being loaded in Elasticsearch. (Kindly adjust the time filter to the past 22 years to view the data)

Major Challenges and Overall Approach to the project:

1. Overall approach:

This project was tested first on a jupyter notebook. The first approach for extracting the data and uploading it to Elasticsearch(ES) was done using the Socrata API and the requests library from python. The drawback with this method was that only one document was uploaded per request and even with multithreading, it would not be enough to

load millions of records. Thus, I read the documentation for the elasticsearch python client library and the elasticsearch helpers library. Based on the information gained from the documentation, I was able to experiment with the es.bulk() function. This function allows me to send multiple documents to be indexed at once. The documents need to be in an iterable. The documentation also mentions that if the documents are sent using a python generator, it is ideal as it avoids loading all the data in memory at once. Thus, I researched on how to build a generator and pass the data into it. After testing the code in the jupyter notebook, I moved my execution to AWS EC2 and docker where I implemented the environment variables and the command line arguments using argparse. Using the bulk() and paralled_bulk() functions, I was able to test-load over a million records on the free tier ES instances in approximately 10 minutes.

2. Data Preprocessing

Another aspect of the data loading was the preprocessing. All the data fields were in text format. In order to make any sense of the data, I needed to parse them in their correct data types before sending them to ES. This was accomplished in part by specifying the ES mappings while creating the index. While specifying the mappings, I skipped the fields which were strings as they would be saved as text in ES by dynamic mapping. For the numeric fields, since they were denoting cash, I used the scaled float with a scaling factor of 10 as it is better suited for memory management. I also had to parse the 'issue_date' field, which was done using the pd.to_datetime() function from pandas.

There were also many records which had a few missing fields. I could have skipped those rows entirely but chose not to. In order to accomplish this, I converted my extracted data into a pandas dataframe. Thus the missing fields became NaN values. Since ES does not recognize NaN, I had to then convert these into None values. This way, when I sent the data to ES, the missing fields were saved with a '-' value. Finally, since the data is being indexed by the issue_date field, I could not have any missing fields for it. So I used the dropna() function to get rid of any rows with a missing issue_date field.

3. Main Execution:

For the main run of the application, I used a r5.large.elasticsearch instance. Using the parallel_bulk() with 20 threads, each with a chunk size of 500 documents, I sent a page size 75000 records and was able to achieve a write speed of less than 5 minutes for a million records. At least initially. In the first 40 minutes, I was able to write over 8 million records. After that, the next 2 hours had only approximately 2.5 million document writes. I was getting ClusterIndexWritesBlocked errors in addition to other write errors. I also noticed that the cluster health was Red. After investigating for over an hour, I noticed that my ES instance had only 1 node. Thus, the cluster was at capacity and I was unable to add more documents.

In order to remedy this, I edited my ES domain and added 6 nodes, then later 12 nodes and finally settled at 9 since 12 nodes were at less than 50% utilization. On observing and troubleshooting, I also noticed a bottle neck at the Socrata API end after loading 20 million records. The time taken to fetch the records form socrata was running in minutes and I started getting timeout errors. Thus, I had to adjust the client timeout for socrata and restart the upload from the last record written in elasticsearch. I never got back the write speed of the first 40 minutes and an indexing rate of around 80,000-100,000 documents was the average for the rest of the process. As the speed kept getting slower, I decided to stop the execution at around the 43 million mark as the deadline for submission was uncomfortably close.

Overall Thoughts:

The elasticsearch project was extremely enjoyable and I was forced to mine the corners of the web for any resource which would enable me to get all the 60 million records loaded. I could have done it if I let my program run for another 6 hours but I was satisfied with loading over 43 million records in about 8 hours with an average of 5 million per hour. Thank you for designing this project, the sense of satisfaction (though 20 million short), was worth the effort.