Project 1 - Analyzing Millions of NYC Fire Incident Dispatch Data

For this project, you will be tasked with loading and then analyzing a dataset containing millions of NYC fire incident dispatch data. In completing this exercise, you will demonstrate mastery of the principles of containerization, terminal navigation, Python scripting, and AWS EC2 provisioning.

[**Overview and Requirements 2**](#_Toc129692910)

[**Project Context 2**](#_Toc129692911)

[**Before You Begin 2**](#_Toc129692912)

[**Python Scripting 3**](#_Toc129692913)

[**Inputs/Outputs 3**](#_Toc129692914)

[**Libraries 4**](#_Toc129692915)

[**Visualizing and Analysis on OpenSearch Dashboard 7**](#_Toc129692916)

[**Submission and Deadlines 8**](#_Toc129692917)

[**Rubric 9**](#_Toc129692918)

# 

# 

# 

# 

# Overview and Requirements

In this project, you will apply what we have learned about EC2, the terminal, Docker, and Elasticsearch to ingest and analyze a dataset that is “too large” to fit into a single machine. (Technically, this is not entirely true, but for the sake of practice/argument let us assume it is).

As such, you will write a python script that runs in docker to consume data from the [**NYC Open Data**](https://opendata.cityofnewyork.us/) project and pushes that information into an OpenSearch cluster provisioned via AWS. This way, the data is never “saved” into your EC2 instance but instead streamed directly to OpenSearch.

Once the data is loaded and available on OpenSearch, you will create a few visualizations to help better explain/understand the data.

## Project Context

The [**NYC Open Data**](https://opendata.cityofnewyork.us/)project makes freely available data published by NYC agencies and other partners. These datasets range from a few thousand rows to millions, depending on department and time frame.

For the really large datasets, attempting to download is not feasible, as some of these files are upwards of 5-10 GBs in size. As such, this service offers an Application Programming Interface - known colloquially as an **API** - for ease of querying and loading the data via web requests in terminal (*via curl*) or code (*via python, javascript, etc*).

Furthermore, these APIs are made available via the [**Socrata Open Data API**](https://dev.socrata.com/)**,** which provides a well-established and easy-to-use set of conventions for querying public datasets such as the one we will be using.

For this project, we will leverage a python client of the Socrata API to connect to the [**Fire Incident Dispatch Data**](https://dev.socrata.com/foundry/data.cityofnewyork.us/8m42-w767) API, load data into an OpenSearch cluster, and visualize/analyze with OpenSearch Dashboards.

To accomplish this, we will leverage our knowledge of containerization, working with the terminal, and python scripting.

## Before You Begin

Make sure you have the following:

* An [**app token**](https://data.cityofnewyork.us/profile/edit/developer_settings)for the NYC Open Data API
* An AWS account (for running EC2, etc)

# Python Scripting

You must develop a python **command line interface** that can connect to the **Fire Incident Dispatch Data** API and demonstrate that the data is accessible via Python.

Your script must be able to run within docker but take parameters from the command line. Each line of data consumed by your script must then be passed appropriately into OpenSearch.

It is up to you to determine which of the fields you get back from the API should be pushed into your OpenSearch instance. You may choose to pass everything in or only a few fields in but there are a few requirements:

1. Include **incident\_datetime** as your time field to create an index pattern in OpenSearch Dashboard.
2. Include at least one numeric variable (*i.e incident\_response\_seconds\_qy, engines\_assigned\_quantity*) to be used in one of your visualizations.
3. Include at least one categorical variable (*i.e. incident\_borough, incident\_classification*) to be used in one of your visualizations.
4. In the data collection process (*client.get() function*), you should filter the data where **starfire\_incident\_id** and **incident\_datetime** have null values (*see https://dev.socrata.com/docs/queries/*)

It is the students’ responsibility to navigate the [dataset](https://data.cityofnewyork.us/Public-Safety/Fire-Incident-Dispatch-Data/8m42-w767) and learn what those fields are for.

## Inputs/Outputs

Here are all the command line arguments your script must support:

|  |
| --- |
| **docker run \**  **-e INDEX\_NAME="dispatch" \**  **-e DATASET\_ID="8m42-w767" \**  **-e APP\_TOKEN= "your\_token" \**  **-e ES\_HOST= "your\_domain" \**  **-e ES\_USERNAME="your\_username" \**  **-e ES\_PASSWORD="your\_password" \**  **bigdataproject1:1.0 --page\_size=1000 --numpages=4** |

Some key arguments here:

* **APP\_TOKEN:** This is how a user can pass along an **APP\_TOKEN** for the API in a safe manner. **APP\_TOKEN** should not be “hardcoded” anywhere in your source code.
* **bigdataproject1:1.0** This is the name of your docker image.
* **ES\_HOST:** This is the domain endpoint of your Elasticsearch cluster. **ES\_HOST** should not be “hardcoded” anywhere in your source code.
* **ES\_USERNAME:** This is the master username. **ES\_USERNAME** should not be “hardcoded” anywhere in your source code.
* **ES\_PASSWORD**: This is the master password. **ES\_PASSWORD** should not be “hardcoded” anywhere in your source code.
* **--page\_size:** This command line argument is **required** for the command line. It will ask for how many records to request from the API per call. If not provided, your program will give you an error.
* **--num\_pages:** This command line argument is **optional** for the command line.[[1]](#footnote-1) If not provided, your script should continue requesting data until the entirety of the content (*except for the rows with the null values of starfire\_incident\_id and incident\_datetime*) has been exhausted and the total number of rows should be printed out. If this argument is provided, continue querying for data **num\_pages** times (*except for the rows with the null values of starfire\_incident\_id and incident\_datetime*). You will need to navigate [**Paging through data**](https://dev.socrata.com/docs/paging.html#2.1) to incorporate **num\_pages** to your script.

**Example: 1**

If you pass the following arguments to the command line: --page\_size=1000 --numpages=4, your script should get 4000 data points.

**Example: 2**

If you pass the following argument to the command line without –numpages argument: --page\_size=1000, it does not mean that your script will only get 1000 data points. It should find a way to collect entire dataset.

## Libraries

In order to accommodate the command line arguments (like the --page\_size, etc), you will probably want to use the [**argparse**](https://docs.python.org/3/library/argparse.html)library. Feel free to use a different library of your choice if you’d like.

**All third-party libraries that you use must be tracked in the requirements.txt** **file**.

For this script, you must use PyPI’s [**sodapy**](https://github.com/xmunoz/sodapy)module. This module makes it seamless to connect to your Socrata API and provides a high-level interface for providing additional parameters such as page offsets.

Your project should have an **src** folder containing functions for interacting with the **Fire Incident Dispatch Data** API and for managing the API response. Your **main.py** script should live in the **src** folder at the same level as your **Dockerfile** and **requirements.txt.**

Below, you find an example of the **sodapy** module being used to make a call out to the **Fire Incident Dispatch Data** API.

Graphical user interface, text

Description automatically generated

Note: **8m42-w767** is the **id** for our dataset.

**Food for Thought** 🤔🤔: Why would you ever need to use the **select=’COUNT(\*)’** argument? Refer to the Example 2.

**Example: 3**

If you pass the following arguments to the command line: --page\_size=500 --numpages=4, your script should retrieve the results in 4 batches. Each batch will retrieve 500 data points. You use “limit” parameter to set the size of the batch and you use “offset” parameter to set the index of the next batches.

First batch: client.get(“8m42-w767”, limit=500, offset=0) #By default, the starting index is zero.

Second batch: client.get(“8m42-w767”, limit=500, offset=500)

Third batch: client.get(“8m42-w767”, limit=500, offset=1000)

Fourth batch: client.get(“8m42-w767”, limit=500, offset=1500)

**Example: 4**

If you pass the following arguments to the command line: --page\_size=100 --numpages=6, your script should retrieve the results in 6 batches. Each batch will retrieve 100 data points.

First batch: client.get(“8m42-w767”, limit=100, offset=0)

Second batch: client.get(“8m42-w767”, limit=100, offset=100)

Third batch: client.get(“8m42-w767”, limit=100, offset=200)

Fourth batch: client.get(“8m42-w767”, limit=100, offset=300)

Fifth batch: client.get(“8m42-w767”, limit=100, offset=400)

Sixth batch: client.get(“8m42-w767”, limit=100, offset=500)

**Example: 5**

If you pass the following arguments to the command line: --page\_size=1000, your script should retrieve the entire dataset. You do not have the num\_pages, but you still need to page through the batches as you cannot get the entire dataset with one batch. What will you do?

First batch: client.get(“8m42-w767”, limit=100, offset=0)

Second batch: client.get(“8m42-w767”, limit=1000, offset=1000)

Third batch: client.get(“8m42-w767”, limit=1000, offset=2000)

Fourth batch: client.get(“8m42-w767”, limit=1000, offset=3000)

…

…

…

…

…

Last batch: client.get(“8m42-w767”, limit=1000, offset=??)

For your project, you will need to automate your application for any given scenario to work. These are a few “hardcoded” examples. **Arguments will not be hardcoded in your script.**

**YOU MUST LOAD AT LEAST 100,000 ROWS** of data into OpenSearch. But, when you test your script, keep it small! Load maybe 100 to 1000 rows tops and test that your OpenSearch connection works, etc. To demonstrate (to yourself) that the application is working, it is sufficient to pass in a few combinations of **page\_size** and **num\_pages.**

Create a new EC2 instance which has the following configuration: **Storage = 30 GB.** All other configurations will remain the same. This is pivotal if you especially want to collect more than 500K data points. Otherwise, your EC2 instance will crash.

Graphical user interface, text, application, email

Description automatically generated

For testing purposes, you can use the same configuration for OpenSearch from “*Connecting to Elasticsearch via Docker on EC2 with Python*”. In this way, you will keep your costs at minimum. I’d also suggest navigating OpenSearch Dashboard to learn about how to do visuals using this configuration so that you will not be charged a lot.

For data collection, create an OpenSearch cluster where the instance type is **r6g.large.search**, availability zones is **3-AZ**, and the number of nodes is **3**. Its price per hour is $0.167. Since you will create 3 nodes, the hourly cost will be $0.167\*3=$0.501. Collecting the entire dataset takes around 2 hours with the correct script.

**NOTE:** In order to prove that you’ve loaded 100K or more rows, please ensure that you provide a Gauge chart that shows the number of data that you collect. This will be an addition to the visuals you will be producing. Based on this proof you will be awarded extra credit for each order of magnitude of rows extra you can achieve.

* *If you get between 300K – 500K, you’ll get 0.5 pts of extra credit.*
* *If you get between 500K – 1M, you’ll get 1 pts of extra credit.*
* *If you get between 1M – 3M, you’ll get 1.5 pts of extra credit.*
* *If you get between 3M – 5M, you’ll get 2 pts of extra credit.*
* *If you get between 5M – 8M, you’ll get 2.5 pts of extra credit.*
* *If you get the entire dataset (around 8.4M), you’ll get 3 pts of extra credit.*

# Visualizing and Analysis on OpenSearch Dashboard

In order to properly index information on OpenSearch Dashboard, we will want to properly define a **time field**. The **Fire Incident Dispatch Data** dataset does contain an **incident\_datetime** field (as mentioned above) that would be a good candidate for this definition.

As far as output goes, configure OpenSearch Dashboard to pull items from the index defined when loading data into OpenSearch. This should load up the resulting data into the OpenSearch Dashboard API and allow you to do some interesting analysis.

Have some fun with this and try to come up with a unique analysis as you explore the capabilities of Kibana - some questions to consider answering (in the form of visualizations or graphs):

* What is the average incident response time per borough?
* Which borough had the highest number of fire incidents? The second highest number of fire incidents?
* Etc.

**Create four different visualizations[[2]](#footnote-2)** in OpenSearch Dashboard that analyze the data loaded and present the analysis in graphical form. Here is the [inspiration](https://datascience-enthusiast.com/Miscellaneous/NYC311calls_Elastic_stack.html) for this.

You can just take a screenshot of each visualization and submit those. Keep in mind that these four visualizations should be different from the gauge chart where you show the number of rows that you collect.

Do not ask questions like “*What is the maximum incident response time?*”. Instead ask “*what is the average incident response time per borough/zipcode*?” or “*What is the maximum incident response time per borough/zipcode*?

If you want to really go above and beyond, you can write a “blog post” in your readme incorporating your OpenSearch Dashboard visualizations and explain some trend or feature of the data you loaded. Here is a very in-depth [example](https://toddwschneider.com/posts/a-tale-of-twenty-two-million-citi-bikes-analyzing-the-nyc-bike-share-system/). Of course, you are not expected to make your “blog post” this intense.

# Submission and Deadlines

There are two main parts you must submit:

* Your source code, which will include a Dockerfile and your python script.
* A screenshot of your OpenSearch Dashboard visuals (A gauge chart and 4 visuals).

You will zip your files and submit the zipped artifact through a Blackboard submission link. I would also like a README that explains how to build and run the docker image, provide some background on the project, and the list of visuals-related questions and their answers. This can be a plain text file or a PDF or [markdown](https://www.markdowntutorial.com/).

Expected Zip file structure:

**project01/**

**+-- Dockerfile**

**+-- requirements.txt**

**+-- src/**

**+-- +-- main.py**

**+-- assets/**

**+-- +-- gauge.png**

**+-- +-- visual01.png**

**+-- +-- visual02.png**

**+-- +-- visual03.png**

**+-- +-- visual04.png**

**+-- README**

Please submit your zipped folder to the Blackboard by **Apr 17th** at midnight.

# Rubric

|  |  |
| --- | --- |
| **RUBRIC** | |
| **Part 1 - Structure & Naming** | **8** |
| 1.1) Parent folder is named “project01” and is exposed when unzipped | 2 |
| 1.2) Dockerfile and requirements.txt exist under project01 folder | 2 |
| 1.3) README exists under project01 folder | 1 |
| 1.4) “src” folder exists and contains main.py | 2 |
| 1.5) “assets” folder exists | 1 |
| **Part 2 - Code** | **22** |
| 2.1) If num\_pages is not provided, the script continues requesting data until the entirety of the content has been exhausted (2pts) (except for the rows with the null values of starfire\_incident\_id and incident\_datetime) (1pt) and the total number of rows should be printed out (1pt). | 4 |
| 2.2) If num\_pages is provided, the script continues querying for data num\_pages times (3pts) (except for the rows with the null values of starfire\_incident\_id and incident\_datetime) (1pt) | 4 |
| 2.3) Used argparse library | 1 |
| 2.4) Used sys library | 1 |
| 2.5) Used os library | 1 |
| 2.6) Used sodapy library | 1 |
| 2.7) Used Bulk API and action is defined properly | 2 |
| 2.8) Docker build command works | 2 |
| 2.9) Script runs without errors | 3 |
| 2.10) At least 100k records loaded (2pts) (**and** there is proof, gauge chart, documenting the count of all rows in a graph visualization) (1pt) | 3 |
| **Part 3 - Analysis** | **10** |
| 3.1) README contains info about how to build docker and run the docker image | 1 |
| 3.2) README provide background information about the project | 1 |
| 3.3) README lists all four visuals-related questions and their answers | 2 |
| 3.4) At least 4 OpenSearch Dashboard charts exist under “assets” folder | 4 |
| 3.5) Kibana charts make sense and are “interesting” (see my comments above) | 2 |

1. This does not mean that it is optional for the project. [↑](#footnote-ref-1)
2. You should use a different graph for each one (*i.e. Do not use a bar chart four times*). [↑](#footnote-ref-2)