

ITEC5207 Project 4 - Backend

Georges Ankenmann

Nnadozie Iwunze

Feifei Pang

October 20, 2022

1 High Level Design

At a high level the project consists of a backend database with a frontend client that displays the dataset along with accompanying scripts to simplify the ingest and management of data. Figure 1 is a diagram of the high level backend design. All the required code to create a functional replication of the work that has been done for this project has been included with this document, for reference the Python code used for the initial sanity check has been provided in section 2.2 of the document while the Python code used for the CSV data ingest has been provided in section 2.3 of the document.

ITEC5207 Project Diagram

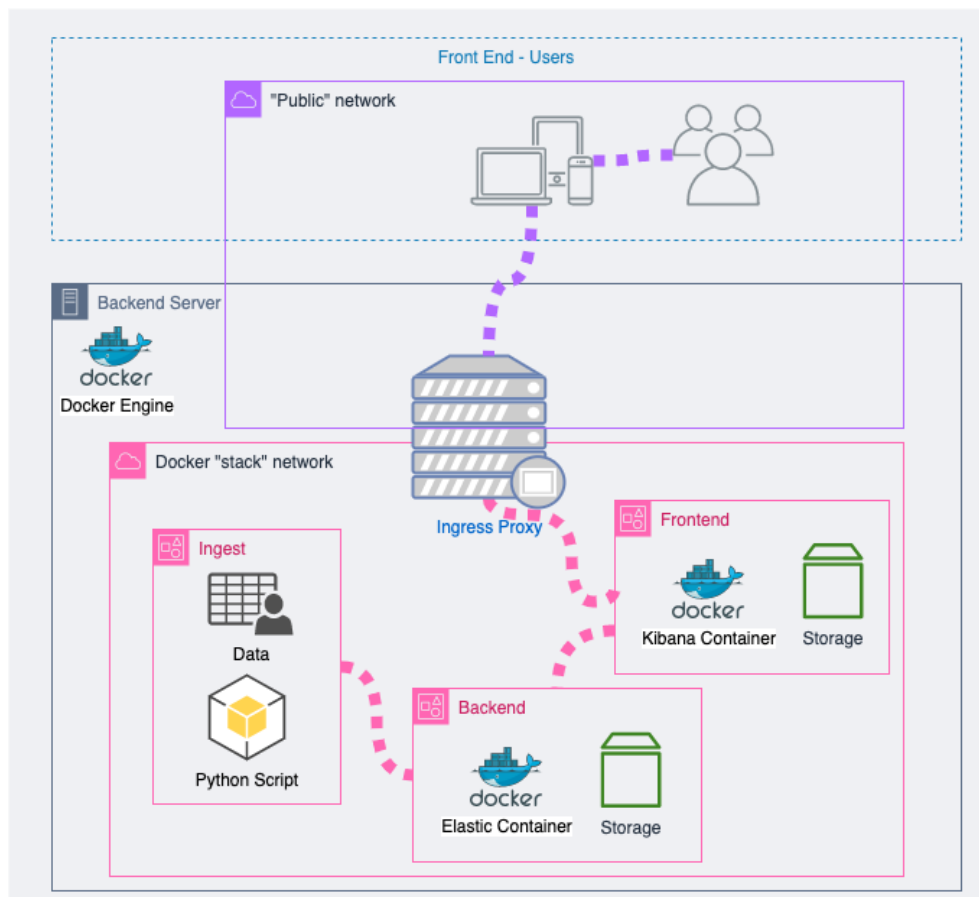


Figure 1: High level diagram of the backend design.

1.1 Elasticsearch

For the database Elasticsearch (<https://www.elastic.co/what-is/elasticsearch>) was chosen as it can easily store JSON objects.

1.2 Kibana

Kibana (<https://www.elastic.co/what-is/kibana>) was chosen as the front end as it can easily intergrate with Elasticsearch and has powerful built in tools for visualisation.

1.3 Python

Python was chosen as the language to write the scripts to aid in data visualisation as there is Python library for Elasticsearch (<https://elasticsearch-py.readthedocs.io/en/v8.4.3/>) and it's easy to write a converter that takes in the CSV dataset and output JSON objects directly into Elasticsearch. A test client was also written in Python to perform basic connection testing between the Python scripts and the Elasticsearch container.

1.4 Docker

For simplicity and reproducibility, the Docker container engine was chosen as Elastic (the parent company to Elasticsearch and Kibana) has containers already available. Python also publishes containers for most versions of Python and are readily available. The containers are also OS agnostic as long as the operating system in question can run the Docker engine. Docker Compose was chosen to simplify the coordination of the Elasticsearch and Kibana containers.

1.5 Running the Code

1.5.1 Minimum Requirements

- Modern Computer (4 cores, 8GB of RAM, 20GB storage)
- Internet connection
- Latest Docker Engine

- Latest Docker Compose

1.5.2 Container Networking

You need a docker network called `stack` and **MUST** be created externally. To create this network run `docker network create stack` in a terminal on a computer with the latest Docker Engine installed.

1.5.3 Running the Backend and Frontend

In a terminal switch to the `Backend Stack` folder and run `docker-compose up`. This will spin up the Elastic database on port 9200 along with Kibana (the frontend) on 5601. Wait a few minutes and you can run a few sanity checks by opening a browser to `http://localhost:9200` and `http://localhost:5601`, which will open Elastic and Kibana respectively. The environmental and docker configurations are mostly defaults from Elastic from their website. It is really important to note that most security and data resiliency features that are part of the core Elastic offering have been removed to simplify this project and should not be used in **ANY** production environment.

1.5.4 Stopping the Backend and Frontend

If you want to stop the stack, in the terminal where you ran the command in section 1.5.3, issue the following key stroke combination and command.

1. `control + c`
2. `docker-compose down`

The Docker engine can use volumes for storage. In this project volumes are used to allow for Elastic and Kibana to store data between restarts of the stack. If it is desired to remove the saved data (for a full reset), you can run the following command `docker-compose down -v` in the same folder where you ran the `docker-compose up` command from section 1.5.3.

1.5.5 Data Ingest and Sanity Check

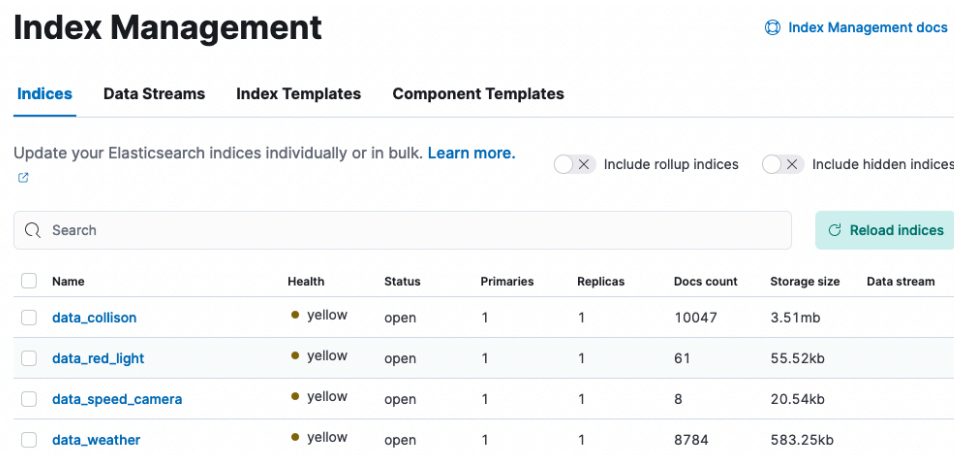
A data ingest container has been provided to simplify index creation and document ingest. It's a Python script that takes in the CSVs and for each record, it generates a JSON document and adds it to the Elasticsearch database. To

build the container, in the Backend Data Ingest folder, run the following command `docker build -t backend . -.`

To run the container, in the Backend Data Ingest folder, run the following command `docker run --rm --network="stack" -it -v "$(PWD):/opt/" backend /bin/bash` You will be presented with a Linux terminal prompt, as a sanity check, while in this prompt run the following command `python sanity.py`. If it does not fail, then you are safe to ingest data. Do do this, run `python run.py`

1.5.6 Checking if the data is in Elastic

From Kibana (<http://localhost:5601>), select the "hamburger" icon at the top left of the Kibana interface. Select **Stack Management**, then under the **Data** header, select **Index Management**. In here you should see something similar to figure 2.



<input type="checkbox"/>	Name	Health	Status	Primaries	Replicas	Docs count	Storage size	Data stream
<input type="checkbox"/>	data_collison	● yellow	open	1	1	10047	3.51mb	
<input type="checkbox"/>	data_red_light	● yellow	open	1	1	61	55.52kb	
<input type="checkbox"/>	data_speed_camera	● yellow	open	1	1	8	20.54kb	
<input type="checkbox"/>	data_weather	● yellow	open	1	1	8784	583.25kb	

Figure 2: Screenshot of the Index Management prompt.

If the Kibana interface is similar to the screen shot in figure 2, then the data was correctly ingested.

1.5.7 Importing the Kibana Saved Objects

Kibana has a concept of "saved objects", which are basically the Elastic flavour of a JSON object that represents things such as dashboards, indexes, configurations, etc. For Kibana to "know" how to display the data from the Elasticsearch database, it must have instructions on how to correctly pull data from Elasticsearch indexes. Saved objects have been provided to provide the index display configuration along with a sample dashboard to verify the data has been correctly ingested.

To import the saved objects, select the "hamburger" from the top left of the Kibana and select **Stack Management**. Then under the **Kibana** header, select the **Saved Objects** prompt. Click on the **Import** prompt and select the provided saved objects file which is located inside of the backend folder. Then click on **Import** and then **Done**. Now you should see something similar to figure 3.

Saved Objects

Refresh

Import

Export 6 objects

Manage and share your saved objects. To edit the underlying data of an object, go to its associated application.

















Search...

Type

Tags

Delete

Export

<input type="checkbox"/>	Type	Title	Tags	Spaces	Last updated	Actions
<input type="checkbox"/>		Collision Data			31 seconds ago	
<input type="checkbox"/>		Weather Data			31 seconds ago	
<input type="checkbox"/>		A dashboard		—	31 seconds ago	
<input type="checkbox"/>		Speed Camera Data			31 seconds ago	
<input type="checkbox"/>		Red Light Data			31 seconds ago	
<input type="checkbox"/>		Advanced Settings [8.4.3]		—	31 seconds ago	

Rows per page: 50

<

1

>

Figure 3: Screenshot of the Saved Objects prompt.

To test the sample dashboard, click the **A dashboard** link which is in the **Saved Objects** interface. If the dashboard and data was loaded correctly you should see something similar to figure 4.

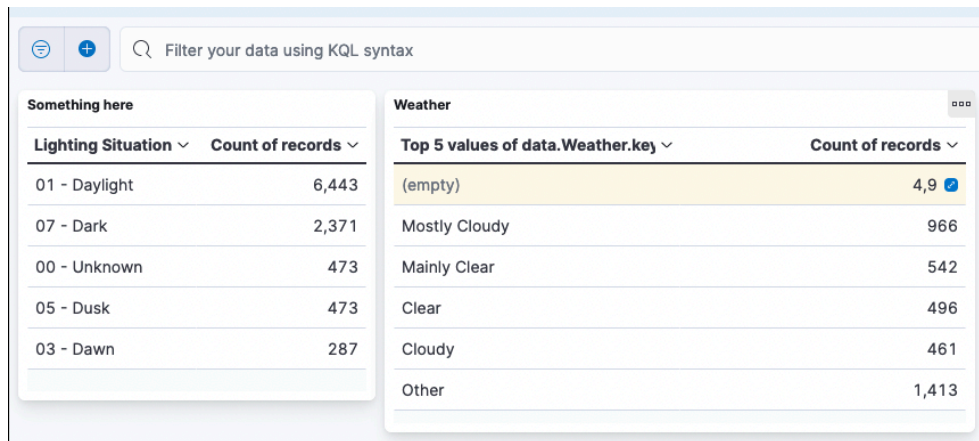


Figure 4: Screenshot of the sample Dashboard.

2 Annex

2.1 Team Members

- Georges Ankenmann - 100 935 237 - georgesankenmann@cmail.carleton.ca
- Nnadozie Iwunze - 101 213 820 - nnadozieiwunze@cmail.carleton.ca
- Feifei Pang - 101 244 289 - feifeipang@cmail.carleton.ca

2.2 Python Sanity Code

```
1 from elasticsearch import Elasticsearch
2 es = Elasticsearch("http://es01:9200")
3 print ("If there is something that is printed after this and it
4       does not fail, then the Elastic database works.")
5 print (es.info())
```

2.3 Python Ingest Code

```
1 import csv
2 from elasticsearch import Elasticsearch
3 import os
4 import json
```

```

5 import pandas as pd
6 import uuid
7 es = Elasticsearch("http://es01:9200")
8 input_folder = "/opt/input/"
9
10 # get all the files in the input directory
11 data_files = []
12 for root, dirs, files in os.walk(input_folder):
13     for name in files:
14         data_files.append(os.path.join(root, name))
15
16 # Define an Elastic index mapping, to force the location field
17 # to be a "geo_point", which is a Elastic/Kibana specific field
18 # that can be plotted on the internal map in Kibana. Required.
19 mapping = {
20     "mappings": {
21         "properties": {
22             "location": {
23                 "type": "geo_point"
24             }
25         }
26     }
27 }
28
29 # create the indexes with teh specific mapping.
30 datasets_with_geo = ["data_red_light", "data_collison", "
31                     data_speed_camera"]
32 for dataset_name in datasets_with_geo:
33     response = es.indices.create(
34         index=dataset_name,
35         body=mapping,
36         ignore=400 # ignore 400 already exists code
37     )
38
39 # add data from the CSVs to Elastic
40 datasets_with_geo_lowercase = ["collison", "speed_camera"]
41 datasets_with_geo_uppercase = ["red_light"]
42 for file in data_files:
43     name = file.split("/")[-1].split(".")[0]
44     tmp = {}
45     with open(file, 'r', encoding='utf-8-sig') as csvfile:
46         reader = csv.DictReader(csvfile)
47         for row in reader:
48             #this if can be removed with further data cleaning.
49             if name in datasets_with_geo_lowercase:

```



```

47         tmp = {
48             "dataset":name,
49             "data":row,
50             "location":{
51                 "lat":float(row["Latitude"]),
52                 "lon":float(row["Longitude"])
53             }
54         }
55     elif name in datasets_with_geo_uppercase:
56         tmp = {
57             "dataset":name,
58             "data":row,
59             "location":{
60                 "lat":float(row["LATITUDE"]),
61                 "lon":float(row["LONGITUDE"])
62             }
63         }
64     # not everything has GPS coordinates (weather data)
65     else:
66         tmp = {
67             "dataset":name,
68             "data":row,
69         }
70     resp = es.index(index="data_"+name,document=tmp)

```