USGS Project Report

João Felício

January 2022

1 Introduction

In this document I will describe how I implemented the app that provides real-time earthquake data from the USGS API (`https://earthquake.usgs.gov/fdsnws/event/1/`). I will start by illustrating how my app will receive a request, get the data and return the response (section 2), then I will go through each endpoint specifying which methods and parameters were used in the USGS API and whether I would need to still filter that data or not (section 3). I will also talk about the process of making the app scalable and highly available (section 4), and finally, a step by step instructions to have the app up and running (section 5).

2 Project's Overview

In this project I was asked to implement a scalable and highly available service that would provide real-time earthquake data from the USGS API.

First I was thinking about having a database and retrieving all the data to store in it. Then my app would query the requested data in my database. However this must be stateless, so I can't store any data (only if it is for a short period of time). For this reason, when a user makes a request, my app must do another request to the USGS API to get the data requested by the user. If needed, some data filtering is still done, and finally the user gets its response.

The Figure bellow shows how this whole process is done step by step:

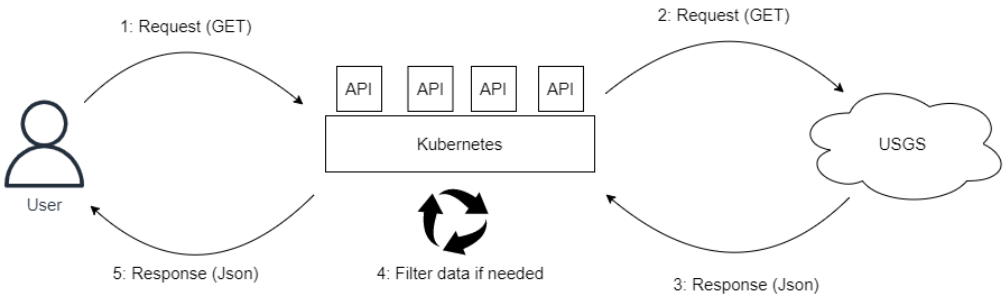

Figure 1: Project's overview illustration.

To implement my application I used the Flask framework (`https://flask.palletsprojects.com/en/2.0.x/`). Here I define my methods and its parameters in order to query the data requested by the user and filter the data received by the USGS API.

To make the requests in the USGS API I implemented a class called *Query*. This class receives a certain number of parameters and makes a request to the USGS API using the method *query* with format = *geojson* returning the response back to my application file. The parameters I use from the USGS API are described in section 3.

3 Endpoints

I was asked to do three endpoints:

- Retrieve all earthquakes M2.0+ for the San Francisco Bay Area during a specific time range (subsection 3.1).
- Retrieve all earthquakes M2.0+ that have 10+ felt reports for the San Francisco Bay Area during a specific time range (subsection 3.2).
- Retrieve all earthquakes M2.0+ for the past day that had tsunami alerts for any given US state (subsection 3.3).

3.1 First Endpoint

On the first endpoint I have to retrieve all earthquakes M2.0+ for the San Francisco Bay Area during a specific time range. To do so I used the following parameters from the USGS API:

- **starttime** - This is the date of the first day the user wants to search for earthquakes in the format: yyyy-mm-dd.
- **endtime** - This is the date of the last day the user wants to search for earthquakes in the same format as the starttime.

- **minmagnitude** - This is the minimum magnitude of the earthquakes (it has value 2 in all endpoints).
- **maxlatitude** - Maximum latitude in degrees.
- **minlatitude** - Minimum latitude in degrees.
- **maxlongitude** - Maximum longitude in degrees.
- **minlongitude** - Minimum longitude in degrees.

Because it was asked to retrieve all earthquakes in the San Francisco Bay Area, I did a search by coordinates making a rectangle that has San Francisco Bay Area in it as shown in Figure 2.

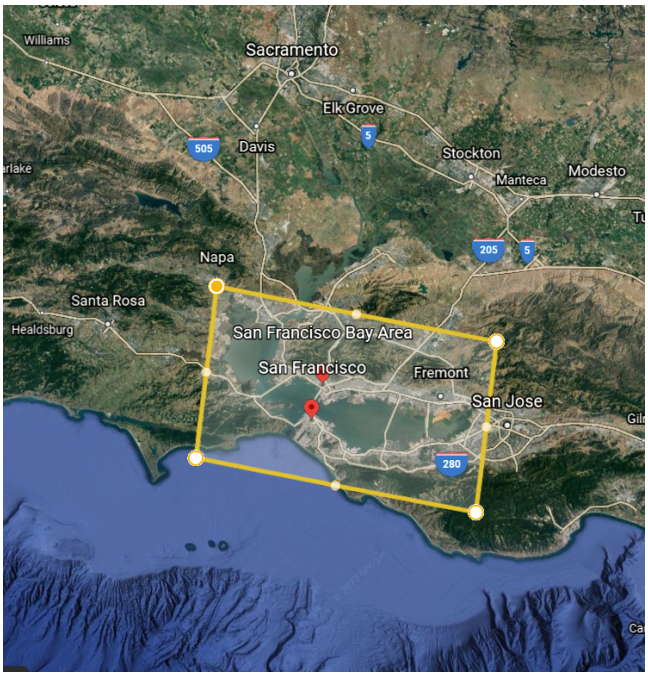

Figure 2: San Francisco Bay Area coordinates.

In this case I get the data already filtered from the request done on the USGS API.

To make this query the user has to go to the following url:

[base_url]/first?starttime=[date]&endtime=[date]

Note: Section 5 explains how to get the base url.

3.2 Second Endpoint

On the second endpoint I was asked to retrieve all earthquakes M2.0+ that have 10+ felt reports for the San Francisco Bay Area during a specific time range. So I used the following parameters from the USGS API:

- **starttime**
- **endtime**
- **minmagnitude**
- **maxlatitude**
- **minlatitude**
- **maxlongitude**
- **minlongitude**
- **minfelt** - Minimum felt reports.

This endpoint uses the same rectangular coordinates as the first one to get all earthquakes for the San Francisco Bay Area.

It also gets the data already filtered from the request done on the USGS API.

To make this query the user has to go to the following url:

[base_url]/second?starttime=[date]&endtime=[date]

3.3 Third Endpoint

On the third endpoint I have to retrieve all earthquakes M2.0+ for the past day that had tsunami alerts for any given US state. I used the following parameters from the USGS API:

- **starttime**
- **endtime**
- **minmagnitude**

To get the **starttime** I used the present day when the user is making the request and for the *endtime* I used the day before.

In this endpoint I still need to filter the data received by the request from the USGS to get only the earthquakes for the given US state. To do so I stored a dictionary that has all the states stored having the key as the abbreviation of the state and the value as the name of the state (for example: "CA":"California") and searched in the "place" key of the json file if, for example, "CA" or "California" were in it.

To make this query the user has to go to the following url:

[base_url]/third?state=[state]

It is important to have into account that the user must write the abbreviation of the desired state in the url. So if the user wants to get the earthquakes from the California state he must write:

[base_url]/third?state=CA

3.4 Bonus Endpoint

I also have another endpoint that retrieves all earthquakes M2.0+ during a specific time range for any given US state.

I did this because, usually, it is hard to get data from the third endpoint so I did another to show that it actually can query all earthquakes for a given US state.

To make this query the user has to go to the following url:

[base_url]/bonus?starttime=[date]&endtime=[date]&state=[state]

The same rules are applied for the format of the *state* parameter.

4 Kubernetes Cluster

In order to make my application scalable and highly available I am using Kubernetes.

Because this is my first time using it and to make it simple, I used the Minikube tool which runs a single-node Kubernetes cluster, having the master node and worker node in a single node. So I can say that this is not properly highly available. However I can still scale out my application by creating more replicas.

If I were to make it better in the future, I would probably choose a topology where it has the master and worker nodes separated. Also I would have at least three replicas of the master node so that I don't have a single point of failure making my application always up and running (ensuring high availability). It would probably be a good idea to also add auto-scaling so that my master nodes could add more replicas of the worker nodes when there is an increased pressure from more requests as well as to scale down to reduce unused resources.

To be able to have all these features I would probably use kubernetes from a service provider like AWS that makes my life a lot easier to manage the kubernetes cluster. Kubedeam is also a good tool to create a minimum viable kubernetes cluster.

5 Instructions to Run the Project

In this last section I will go through all the steps to test the application.

First we need to have docker, kubectl and minikube installed.

1. Go to the directory USGS/api and open a terminal
2. Run the command: “minikube start” to start the kubernetes cluster
3. If you’re in Mac or Windows, make sure it is using the Docker for Desktop context by running the following: eval $(minikube docker-env)
4. Build the docker image by running the command: docker build -t usgs .
5. Start the kubernetes: kubectl apply -f deployment.yaml
6. Wait until all pods are ready. You can check by running (Figure 3 shows an example of the output): kubectl get deployment
7. Run and get the base url by typing (Figure 5 shows where to get the url): minikube service –url usgs-service
8. Go to the browser and use the url as explained in section 3 having base_url as the one you got from the previous step

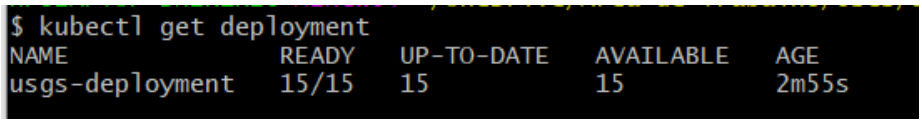

Figure 3: kubectl get deployment

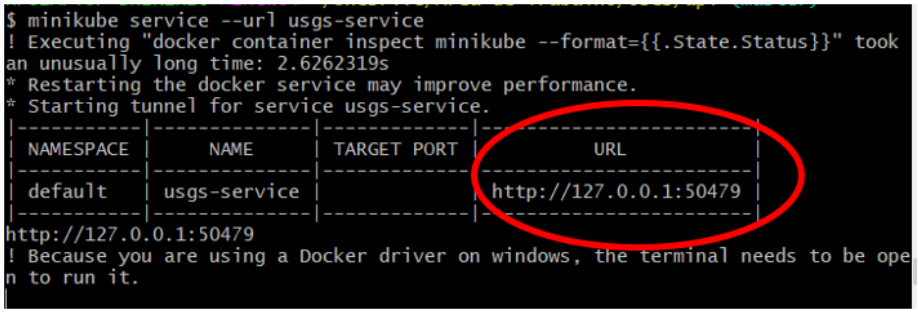

Figure 4: minikube service –url usgs-service

I have also done a script where I send 500 requests at once to see if any request fails.

To run it go to the directory USGS/test and run the command: “python send_requests.py”. It will then ask for the base url where you have to write the one that you got from step 7. If nothing is printed, it means that every response got the status = 200 (success), otherwise it prints the response’s status.

It is also possible to see how it runs by changing the number of replicas in the deployment.yaml file:

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: usgs-deployment
  labels:
    app: usgs
spec:
  selector:
    matchLabels:
      app: usgs
  replicas: 15
  template:
    metadata:
      labels:
        app: usgs
    spec:
      containers:
      - name: usgs
        image: usgs:latest
        imagePullPolicy: Never
        ports:
        - containerPort: 5000
```

Figure 5: Yaml file.