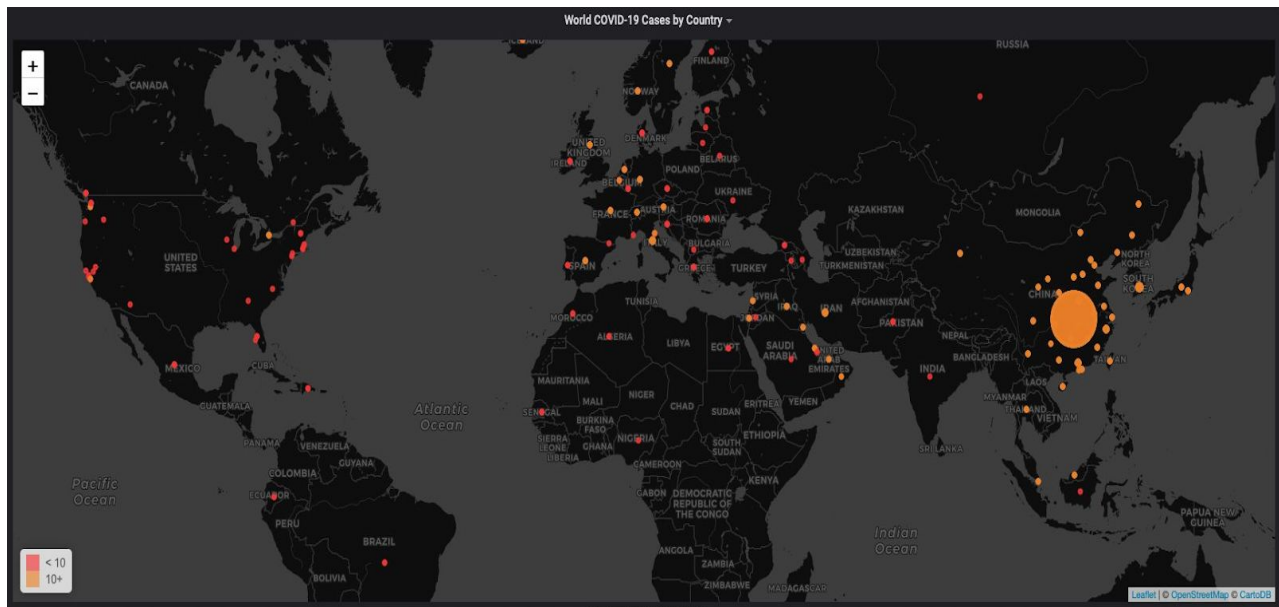


# Assignment-2

## Cloud Computing Summer Sem 2021

### Covid-19 Policy Tracker



Live Project:

<http://flask-env.eba-au6i98zk.ap-southeast-2.elasticbeanstalk.com/>

Github Repository:

<https://github.com/s3691487/covid-policy-tracker.git>

Authors:

s3808528 Vaishali Wahi



s3691487 Yanying Xu

# Project title here

Covid-19 Policy Tracker

## Group name here

Wahi-Xu

Student Name: Yanying Xu	Student Name: Vaishali Wahi
Student ID: s3691487	Student ID: s3808528
Contributions: 1. Backend Developer 2. Report writing	Contributions: 1. Frontend developer 2. Report writing
Contribution Percentage: 50%	Contribution Percentage: 50%
<i>By signing below, I certify all information is true and correct to the best of my knowledge.</i>  Signature:   Date: 30/01/2021	<i>By signing below, I certify all information is true and correct to the best of my knowledge.</i>  Signature:   Date: 30/01/2021

## Contents

<b>1. Summary.....</b>	<b>Page 3</b>
<b>2. Introduction.....</b>	<b>Page 3</b>
<b>3. Related work.....</b>	<b>Page 4</b>
<b>4. High-Level Architecture.....</b>	<b>Page 4</b>
<b>5. Data Structure and Software Architecture.....</b>	<b>Page 6 - 9</b>
5.1 Google BigQuery	
5.2 Data collection and cron scheduler	
5.3 DynamoDB	
5.4 AWS Lambda	
5.5 API Gateway	
5.6 Front-end	
5.6.1 Flask	
5.6.2 Elastic Beanstalk	
5.6.3 S3	
5.6.4 Google Charts	
5.6.5 Google Maps API	
<b>6. Implementation- Developer Manual.....</b>	<b>Page 10 - 18</b>
6.1 Front-end	
6.2 Backend	
<b>7. User Manual.....</b>	<b>Page 19 - 25</b>
<b>8. Future Improvements.....</b>	<b>Page 26</b>
<b>9. References.....</b>	<b>Page 27</b>

## **Project Summary**

The covid-19 policy tracker is a mobile friendly web app developed and hosted in the cloud platform that retrieves and presents real time data for covid cases and government policies in different countries. The purpose of this project is to help the users to understand and track the government policies in different countries, including what level of restrictions are being placed in different countries, whether they are effective to control virus outbreaks.

Through this web app, users can gather information about the current government policies in different countries. It also tracks previous government policies back from 2020, when covid first started. To help users understand the effectiveness of the policies, the app shows users the number of covid cases in different countries on different dates, which can be used as a clear indicator for policy effectiveness. Detailed government policies are listed so users can find out what the exact policies are implemented on different dates. It also dynamically generates a covid restriction level chart to help users estimate what restriction levels are implemented on a monthly time frame.

## **Introduction**

As cited on the UpToDate website: 'Coronaviruses are important human and animal pathogens. At the end of 2019, a novel coronavirus was identified as the cause of a cluster of pneumonia cases in Wuhan, a city in the Hubei Province of China. It rapidly spread, resulting in an epidemic throughout China, followed by a global pandemic. In February 2020, the World Health Organization designated the disease COVID-19, which stands for coronavirus disease 2019. The virus that causes COVID-19 is designated severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2); previously, it was referred to as 2019-nCoV'. (McIntosh 2020, para 1)

While this virus has infected nearly a hundred people and claimed over 2 millions lives around the world at the time of writing this report, governments from all around the world have set up different policies during different stages of the pandemic to protect their citizens. While these public health policies have served the purpose of protecting the health of the citizens, it is recognized that they can have different effectiveness. Since many of these policies or restrictions have greatly impacted our daily lives since last year, we decided to develop a web app to track them. Hopefully from the current data we have collected, we can find some close answers to the most effective policies.

We would like to point out, in this project, the policy data we used is sourced from the Oxford Coronavirus Government Response Tracker (OxCGRT). The Oxford COVID-19 Government Response Tracker (OxCGRT) systematically tracks and records information on different common policy responses that the governments had implemented to response to the pandemic on 18 indicators such as school closures and travel restrictions. We have selected 3 data sets for our current project. (University of Oxford, 2020) The covid cases data we are sourced from the Johns Hopkins University's coronavirus dataset. (Johns Hopkins University of Medicine 2020) All data we sourced can be found in the Google BigQuery public dataset.

## **Related work**

We have done some research online, there is a research project that is using the same dataset named Policy Responses to the Coronavirus Pandemic on Our World in Data (Our World in Data 2020). However, in our webapp, we have utilized the data in a different way and our project emphasis more on the details in different policies from different countries rather than conducting research and analysis on the dataset, which has been done in the Policy Responses to the Coronavirus Pandemic.

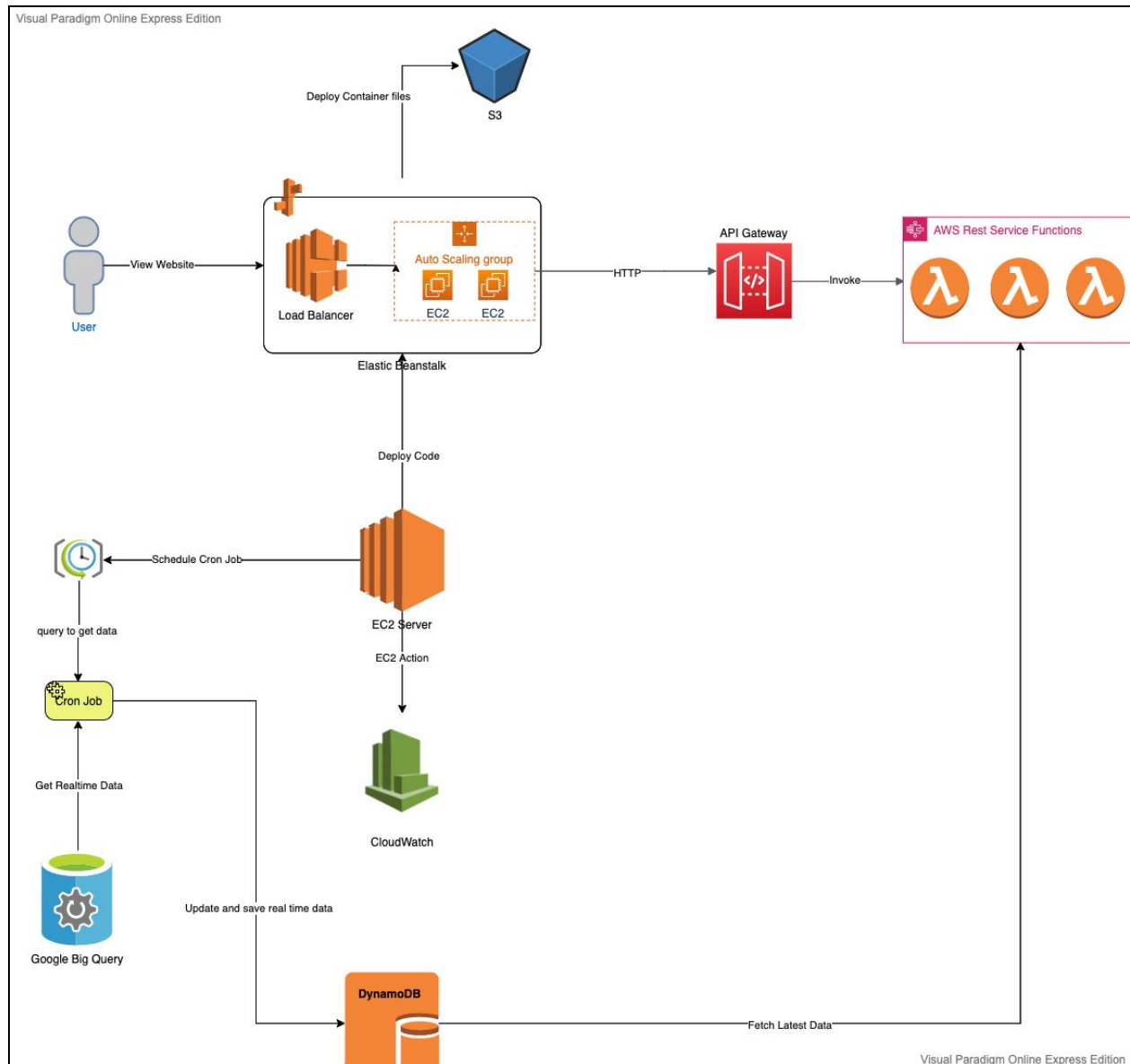
## **High-Level Architecture**

Traditional Monolithic Architecture was built using numerous layers - UI Layer, persistence Layer and Business layer. Whereas, the Modern Microservices approach removes the overhead of layers and divides the functionalities into different domains. We decided to opt for the Modern approach and built our application using microservices.

Our architecture starts by the user calling the https url which again takes the image from elastic beanstalk and sends the response to the user. Elastic beanstalk is an orchestration service provided by AWS which works on orchestration of various services provided by AWS. It maintains different versions of the application deployed in the S3 bucket. We have used EBS for deploying our flask app over the server and to increase the scalability of our project. Through HTTPS, we have connected our flask application to the API gateway which works in connection with all our REST lamda functions. The lambda functions gets the data from DynamoDB using 'Boto3' library of DynamoDB. We have written our lamda functions in python3.6 and have enabled CORS in our API Gateway. Our DynamoDb is updated every one hour by the microservice(CRON job) we have written on our EC2 instance which is

getting the Data through GoogleBigQuery.We have used open Dataset to fed our BigQuery Data.

Below is a detailed flow chart that illustrates our architecture design:



## Data Structure and Software Architecture

Now we are going to discuss different services that we decided to use for this project, in the session below, we will list out each one of them and explain why we think they are the best choice for our project.

## Google BigQuery

All the data that is used in this project has been sourced from the public datasets from Google BigQuery. A public dataset is a dataset that has been hosted by BigQuery that is available for public access. There are a few reasons that we decided to choose this service: through this service, we can retrieve data through the use of standard sql and test our queries in the google federated cloud sql connection. This saves time and effort to first copy and save a massive amount of raw data in order to just extract the data we actually need for the project. Google BigQuery is also a live dataset, the covid policy data and covid cases data we collect from it are being updated on a daily basis. Therefore we are also able to update our data on a daily basis to serve the most recent data to the users. In regards to pricing: users only need to pay by query basis and the first 1TB per month is free of charge, which suits our project because we are not querying over 1TB in this project. (Jones 2019)

## Data collection and cron scheduler

The two datasets that we are using are the covid19\_govt\_response and covid19\_jhu\_case. With the covid19\_govt\_response there is only one schema, which is the oxford\_policy\_tracker, this schema has over 50 fields, this project only selects 19 fields which we think it's the most important policies for the general public. These fields include: country\_name, region\_name, date, school\_closing\_notes, workplace\_closing, workplace\_closing\_notes, cancel\_public\_events, cancel\_public\_events\_notes, restrictions\_on\_gatherings, restrictions\_on\_gatherings\_notes, close\_public\_transit, close\_public\_transit\_notes, stay\_at\_home\_requirements, stay\_at\_home\_requirements\_notes, contact\_tracing, contact\_tracing\_notes, confirmed\_cases, deaths international\_travel\_controls and international\_travel\_controls\_notes. We choose 3 different countries from the dataset which are Australia, United Kingdom and China, these 3 countries have relatively different policies during the pandemic. For the covid statistics we selected 5 fields from the summary table in the covid19\_jhu\_case schema, which are: country\_region, date, confirmed, deaths, and active. The timeframe we choose for this project is from the beginning of 2020 until the most recent day, new data will be added every day as the pandemic is still ongoing at time of writing this report. In terms of technology choice for data querying, we made a decision to use python and the boto3 library for the BigQuery, which has a syntax that is very easy to understand and can deal with large amounts of data.

To constantly update the data, we set up a cron job which runs the python script on an hourly basis that queries data from the BigQuery. The more detailed implementation will be discussed in the implementation session below.

## DynamoDB

After successfully querying the data from Google BigQuery, we then store this data into the AWS DynamoDB. DynamoDB has been chosen as our main data storage instead of a relational database. We decided to use a NoSQL database because the dataset we have chosen is relatively simple and there is no relationship between the tables. One of the other main reasons is DynamoDB can provide very high throughput at a low latency. Although currently we only include data from 3 countries, we have considered the possibility of scaling up the dataset for future improvements and development, using DynamoDB will always make sure a high performance for data query compared to traditional relational databases. We also considered that the effort of initial set up and maintenance of DynamoDB is very little since it is serverless and handled by AWS. The scale up and scale down of the database will also be much easier than hosting a database as it is all maintained by AWS in the cloud environment.(Crosett 2019)

In our python script, we have done a once only data ingestion for the previous data into the DynamoDB. Then the code will query data from the most recent days and insert into the database. We will list the table structure and the more detailed code demo in the implementation session later in this report.

## AWS Lambda

Since our website will require us to provide data to the front end and handle user requests, we decided to use AWS lambda to build our back-end services. There are a few reasons that we have selected AWS lambda in this case. First AWS lambda is again serverless, therefore we did not need to deploy any infrastructure for the services. All we need to do is to write code on the AWS console or upload a zip file of our code. It also scales automatically depending on the size of the workload, i.e. it will scale up with more requests and vice versa and the performance is consistent with any scaling. More importantly, because it is a AWS service, it integrates very well with DynamoDB and our API gateway which we are going to discuss below. (AWS 2021)

## API Gateway

The project has used the AWS API Gateway to deploy the required APIs for the front end. The API Gateway is very easy to use, we were able to do rapid code development on Lambda and deployment on API Gateway simultaneously as they integrate very well. We were also able to monitor our services on CloudWatch. (AWS 2021)



## Frontend

The Frontend of the website consists of one single page website built on the technologies Flask, python, HTML, Javascript and CSS. We have integrated the frontend with google charts and Google Maps Api to visualize the Covid data over various countries and compare them.

The webapp utilized the flask framework which was built on the AWS Elastic Beanstalk service. Using the flask framework, we can call the endpoint that we deployed previously and retrieve the data on the webapp for the users.

Below is a table that listed all the technologies used in frontend:

Technology	Description
Flask	Flask is a framework which uses python to build web applications. It is imported in the application by using the python import statement. It has several other features including a 'request' library which we have used to fetch the data from REST API. It also increases the scalability and maintainability of the code.
Elastic Beanstalk	Elastic Beanstalk is a kind of all rounder service for deploying and scaling Web applications using various languages including python. It itself manages the load balancing, auto scaling as well as application health monitoring. It creates a container environment for the app deployed to the web using the framework. It also decreases the chance for failure of system
S3	S3 is the type of AWS storage which is used by elastic beanstalk in our application to store resources, containerized code and all other files related to the code deployed.
Google Charts	Google Charts is one of the widely used web services which is used to display the graphical representation of user supplied data. We have used google charts to display the latest level of restrictions in various countries. The data is connected to the real time database and could be analyzed till the current date.
Google Maps API	Google Maps API is the RESTful API used to display information over the map or using map. We have used Google Maps to display the cluster of regions affected by the COVID-19 and have formed the cluster over those positions. The input given to this API is json format longitude and latitude locations over the map.

## Implementation - Developer Manual

In this session, we will discuss the implementation of all services used for this project.

### Front-end

#### Develop web app on Cloud 9 environment and set up flask

##### Prerequisite:

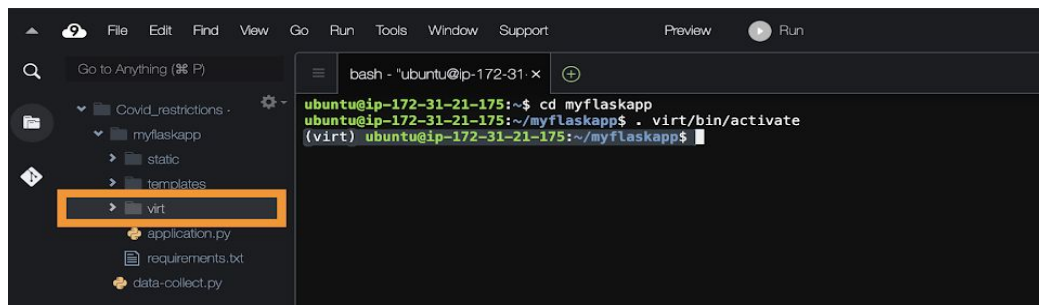
- Connect to EC2 instance through terminal or Putty or Cloud 9.
- Install python on EC2 Instance.

*(we used cloud 9 service on aws to exclude the overhead of connecting to instance. Steps to initialize cloud 9 is explained on the [link](#))*

1. Clone the repository from the github link above in your ec2 server using ssh, aws cli or cloud9.
2. Install Flask web framework for easing the server response using microframeworks.
  - a. (optional) Before installing flask we installed a virtual environment package (venv library) in python to specify versions of python and other dependencies for our project.

Steps to include virtual environment:

- a. Install - `pip install virtualenv`
- b. Create - `virtualenv virt`  
*("Virt" folder is created shown in image below in image below)*
- c. Activate - `. virt/bin/activate`  
*(Virtual environment activated shown in image below)*



*(look for img\_F.1 in folder report\_images)*

For more information go to link - [venv Env](#).

Steps to install Flask:

- a. `pip install flask`

Run application on localhost:

a. `python application.py`

To deploy the website on a public URL we used elastic beanstalk to create a flask container over the server.

'application.py' contains the python script which imports flask and tells the server how to respond to different requests.

```
from flask import Flask,render_template, redirect, url_for, request, session,flash
import requests

# EB looks for an 'application' callable by default.
application = Flask(__name__)

#tells the server the response to request '/' and '/index'
@application.route('/')
@application.route('/index')
def index():
    return render_template('index.html')

@application.route('/maps')
def maps():
    #fetch data from API and send it to the website frontpage
    r =
requests.get('https://xk4w3gnxqb.execute-api.ap-southeast-2.amazonaws.com/beta_0_0/allcountrylocations')
    j = r.json()
    return render_template('maps.html',location = j)

if __name__ == "__main__":
    # Setting debug to True enables debug output. This line should be
    # removed before deploying a production app.
    application.debug = True
    application.run()
```

Here Flask made it all easier to scale the code.

*For more information about flask please visit this [link](#).*

Now we have deployed the app. Next we will discuss the AWS service to use the flask framework over the server.

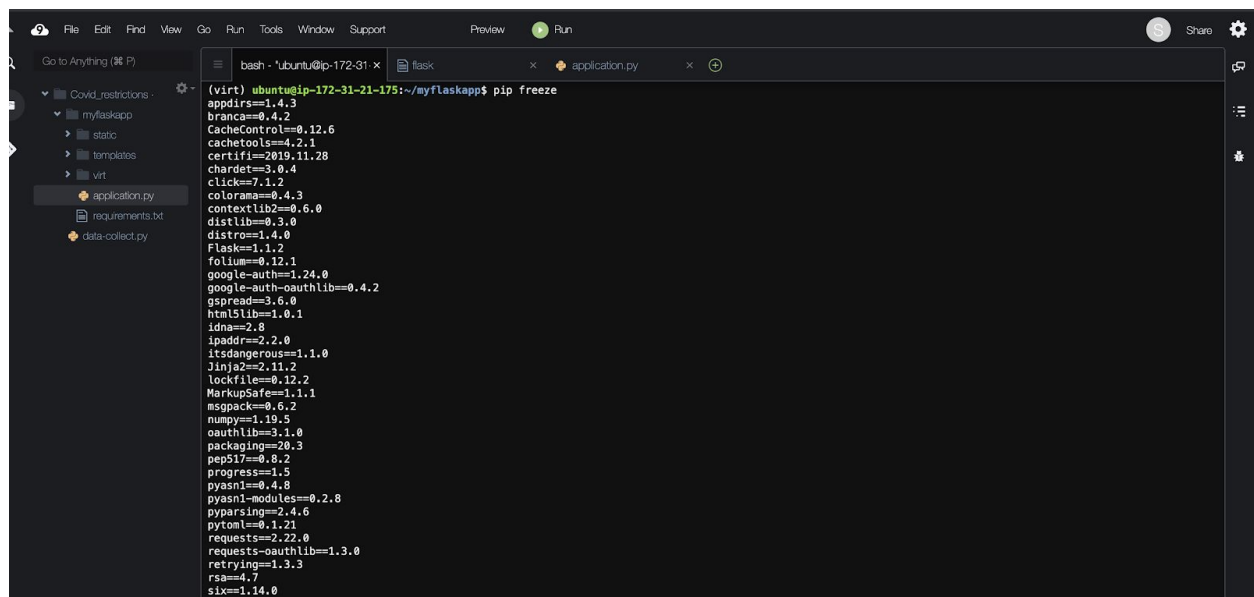
## Initialising and installing AWS elastic beanstalk

Elastic Beanstalk created a virtual environment for flask application i.e it makes sure that the environment in which the website is deployed have all the requirements met according to the specificity of the project. It will automatically take care of scaling,load balancing, automatic health checkup and several other tasks. [For more information refer to link](#)

To set up elastic beanstalk environment get the specific requirements file by running command:

Get the requirements: `pip freeze`

Output would look something like this:



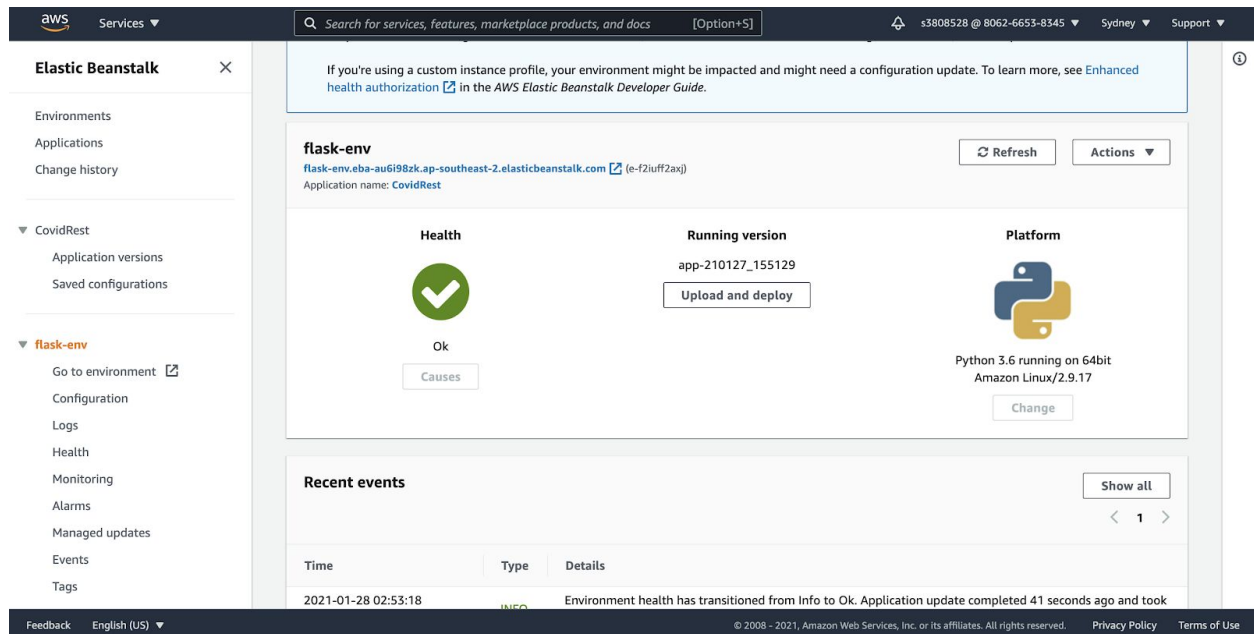
```
(virt) ubuntu@ip-172-31-175:~/myflaskapp$ pip freeze
appdirs==1.4.3
branca==0.4.2
CacheControl==0.12.6
cachetools==4.2.1
certifi==2019.11.28
chardet==3.0.4
click==7.1.2
colorama==0.4.3
contextlib2==0.6.0
distlib==0.3.0
distro==1.4.0
Flask==1.1.2
folium==0.12.1
google-auth==1.24.0
google-auth-oauthlib==0.4.2
gspread==3.6.0
httplib==1.0.1
idna==2.8
ipaddr==2.2.0
itsdangerous==1.1.0
Jinja2==2.11.2
lockfile==0.12.2
MarkupSafe==1.1.1
msgpack==0.6.2
numpy==1.19.5
oauthlib==3.1.0
packaging==20.3
pep517==0.8.2
progress==1.5
pyasn1==0.4.8
pyasn1-modules==0.2.8
pyparsing==2.4.6
pytoml==0.1.21
requests==2.22.0
requests-oauthlib==1.3.0
retrying==1.3.3
rsa==4.7
six==1.14.0
```

To save the requirements in a file run the following command:

`pip freeze > requirements.txt`

Now follow the elastic beanstalk initialization given on the following [link](#).

After creating the flask env, go and see the status of environment of elastic beanstalk



Just make sure to use the region specific to the region you are hosting the server and want to deploy the website.

To deploy any changes to the server follow the following steps:

1. Update and save the code in the EC2 server.
2. Deploy the latest code from EC2 server to elastic beanstalk server using command:

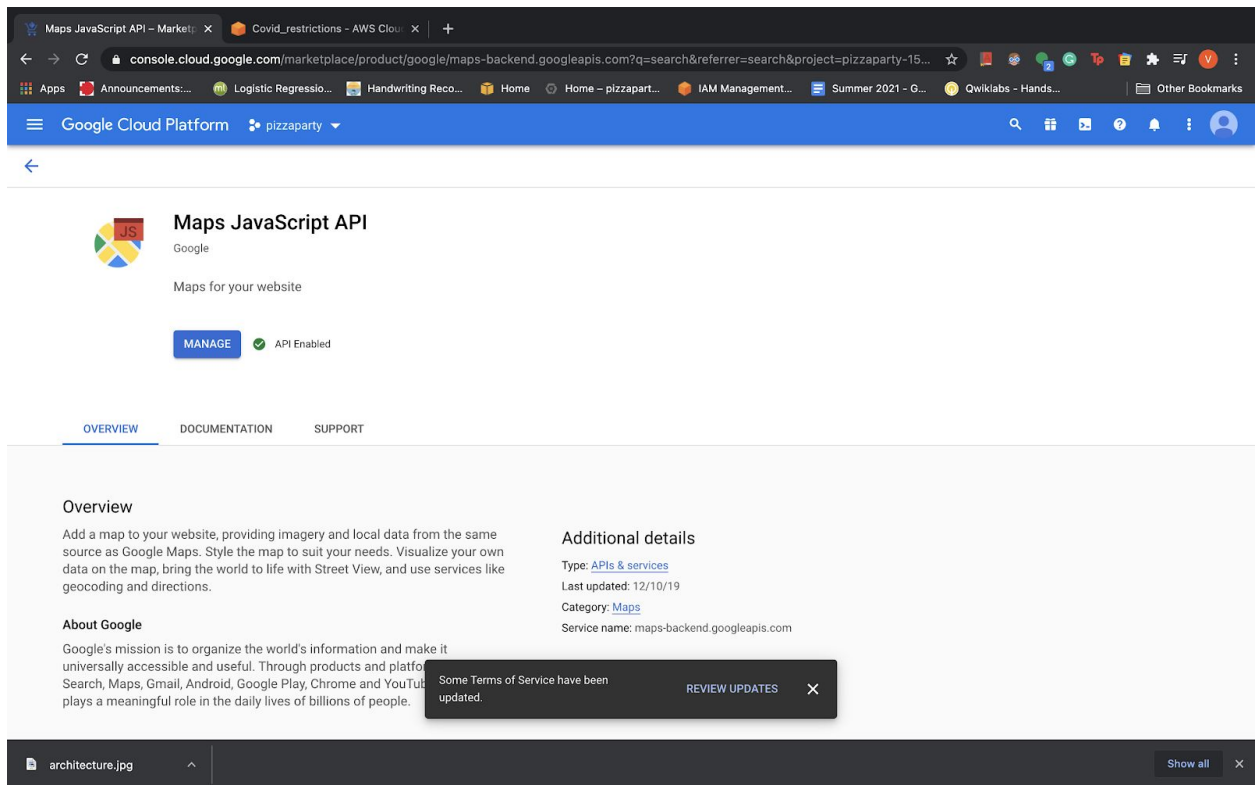
```
eb deploy
```

(make sure to be in the same directory as application.py).

## Set up Google Maps API

1. Log on to google cloud console and search for google maps API.
2. Go to API's section and search for 'maps javascript API'.
3. Enable the API and reload the page.
4. On this page you would get a personal API key for using google maps. Store the key in somewhere safe.

When you reload the page it should have a manage option just like the image shown below.



5. Replace the api key in maps.html file in templates folder, the next image shows the exact position.

```
<script
src="https://maps.googleapis.com/maps/api/js?key=_YOUR_API_KEY_&aShBJ4bcallba
ck=initMap&libraries=&v=weekly"
defer
></script>
```

6. The Clusters are formed using the positions (longitude and latitudes) of the locations and the data is coming through the API gateway which calls the lambda function getCountryLocation and returns the data.

## Back-end

### Retrieve live data from Google BigQuery

There are a few prerequisites before setting up the data retrieving process, all of them are listed below, we will include some external links for instructions:

- Gcloud account and project setup (see: [link](#))
- EC2 instance deployment (see: [link](#))
- Python installation (see: [link](#))
- Pip installation (see: [link](#))

Both the retrieving and the storage are automated through a python script we developed called data-collect.py and we have stored the script on the EC2 instance. While the code can also be found on GitHub, there are a few set up steps and some key codes we would like to discuss in the report.

As this script uses the pandas-gbq library, before we run the code we need to install the pandas library then upgrade it to use the pandas-gbq library through this command:

```
- pip install --upgrade pandas-gbq.
```

While we have excluded all the google cloud authentication/credentials in the code in github for security purposes, they are very important to the connection. Because the python script has been deployed on one of the EC2 instances, it will not connect to google cloud service unless we provide the credentials. Each time we query the BigQuery, we have to include the credentials as shown in the code snippet below.

```
credentials = service_account.Credentials.from_service_account_info(
    {
        "type": "service_account",
        "project_id": "<project_id>",
        "private_key_id": "<private_key_id>",
        "private_key": "-----BEGIN PRIVATE KEY-----\n<private_key>\n-----END PRIVATE KEY-----\n",
        "client_email": "<client_email>",
        "client_id": "<client_id>",
        "auth_uri": "<auth_uri>",
        "token_uri": "<token_uri>",
        "auth_provider_x509_cert_url": "<auth_provider_x509_cert_url>",
        "client_x509_cert_url": "<client_x509_cert_url>"
    },)

df = pd.read_gbq(query, dialect='standard', project_id=project_id, credentials=credentials)
```

While working on the script, we have been testing our query on the google federated cloud sql connection before including the query in our code as shown in the image below:

The screenshot shows the Google Cloud SQL query editor interface. At the top, there's a toolbar with buttons for RUN, SAVE, SCHEDULE, and MORE. A status bar indicates "This query will process 20.8". The query text is as follows:

```
1 SELECT country_name, region_name, date, school_closing_notes, workplace_closing, workplace_closing_notes,
2 cancel_public_events, cancel_public_events_notes, restrictions_on_gatherings, restrictions_on_gatherings_notes,
3 close_public_transit, close_public_transit_notes, stay_at_home_requirements, stay_at_home_requirements_notes,
4 contact_tracing, contact_tracing_notes, confirmed_cases, deaths international_travel_controls,
5 international_travel_controls_notes FROM `bigquery-public-data.covid19_response.oxford_policy_tracker`
6 where ( country_name like 'United Kingdom' or country_name like 'China' or country_name like 'Australia') and date = '2020-01-15'
```

Below the query, the "Query results" section shows "Query complete (0.6 sec elapsed, 20.8 MB processed)". The results are displayed in a table with 11 columns: Row, country\_name, region\_name, date, school\_closing\_notes, workplace\_closing, workplace\_closing\_notes, cancel\_public\_events, cancel\_public\_events\_notes, restrictions\_on\_gatherings, and restrictions\_on\_gatherings\_notes. The table contains 7 rows of data.

Row	country_name	region_name	date	school_closing_notes	workplace_closing	workplace_closing_notes	cancel_public_events	cancel_public_events_notes	restrictions_on_gatherings	restrictions_on_gatherings_notes
1	United Kingdom	England	2020-01-15	null	0.00	null	0.00	null	0.00	null
2	United Kingdom	Northern Ireland	2020-01-15	null	0.00	null	0.00	null	0.00	null
3	United Kingdom	Scotland	2020-01-15	null	0.00	null	0.00	null	0.00	null
4	United Kingdom	Wales	2020-01-15	null	0.00	null	0.00	null	0.00	null
5	Australia	null	2020-01-15	null	0.00	null	0.00	null	0.00	null
6	United Kingdom	null	2020-01-15	null	0.00	null	0.00	null	0.00	null
7	China	null	2020-01-15	null	0.00	null	0.00	null	0.00	null

## Data storage in DynamoDB

To store the data in DynamoDB, we first create the tables in the DynamoDB and specify its partition key and sort key, detailed steps can be found on [here](#). As the image below shown, we have created 3 tables in this project.

The screenshot shows the AWS DynamoDB console. At the top, there are buttons for "Create table" and "Delete table". Below, there's a search bar "Filter by table name" and a dropdown "Choose a table group". The main area displays a list of three tables:

	Name	Status	Partition key	Sort key
<input type="radio"/>	country-location	Active	country_region (String)	province_state (String)
<input type="radio"/>	covid-cases	Active	country_region (String)	date (String)
<input type="radio"/>	covid-policy	Active	country_name (String)	date (String)

Then in the python script, we used the boto3 library to put items into the table, below is some code snippet:



```
def load_daily_data_to_table(dataframe, dynamodb=None):
    if not dynamodb:
        dynamodb = boto3.resource('dynamodb', aws_access_key_id='<aws_access_key_id>', aws_secret_access_key='<aws_secret_access_key>', region_name='ap-southeast-2')

    table_policy = dynamodb.Table('covid-policy')
    policy_data = dataframe.T.to_dict().values()
    for policy in policy_data:
        table_policy.put_item(Item=policy)

    table_country = dynamodb.Table('covid_cases')
    country_data = query_yesterday_country_data().T.to_dict().values()
    for country in country_data:
        table_country.put_item(Item=country)
```

As the image shows on the table below, all the items are now created in the table.

country_name	date	cancel_public_events_note	cancel_public_events	close_public_transit_notes	close_public_transit	confirmed_cases	contact_tracing_notes	international_travel_controls	international_travel_controls_notes	region_name	restrictions_on_gatherings
China	2020-02-01	different cities have different policies	2.00	Public transport is closed	0.00	0.00	0.00	0.00	0.00	China	0.00
China	2020-02-02	different cities have different policies	0.00	Public transport is closed	0.00	0.00	0.00	0.00	0.00	China	0.00
China	2020-02-03	different cities have different policies	0.00	Public transport is closed	0.00	0.00	0.00	0.00	0.00	China	0.00
China	2020-02-04	different cities have different policies	0.00	Public transport is closed	0.00	0.00	0.00	0.00	0.00	China	0.00
China	2020-02-05	different cities have different policies	0.00	Public transport is closed	0.00	0.00	0.00	0.00	0.00	China	0.00
China	2020-02-06	different cities have different policies	0.00	Public transport is closed	0.00	0.00	0.00	0.00	0.00	China	0.00
China	2020-02-07	different cities have different policies	0.00	Public transport is closed	0.00	0.00	0.00	0.00	0.00	China	0.00
China	2020-02-08	different cities have different policies	0.00	Public transport is closed	0.00	0.00	0.00	0.00	0.00	China	0.00
China	2020-02-09	different cities have different policies	0.00	Public transport is closed	0.00	0.00	0.00	0.00	0.00	China	0.00

## Cron scheduling

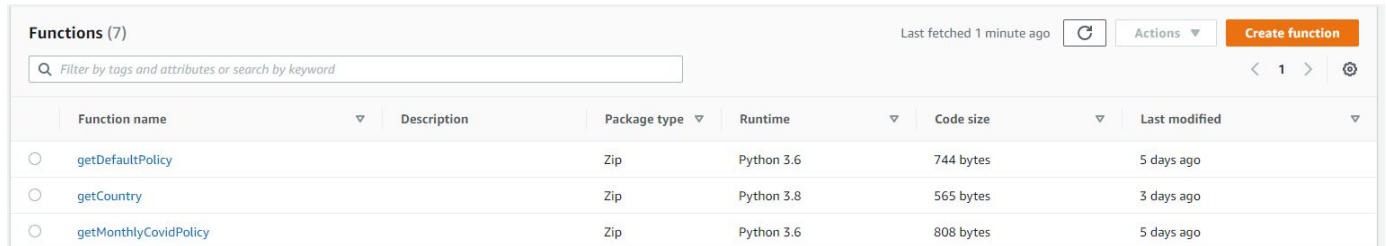
We also set up the cron job on the EC2 instance so the data-collect.py can be triggered every hour to retrieve the latest data. To set up the cron job, ssh into the EC2 instance, then run the command: `crontab -e`.

Then add two lines to the crontab as shown below, this means we are setting up cron to run the command `python3 /data-collect/data-collect.py` every hour and no logging required if successful.

```
MAILTO=""
0 * * * * python3 /data-collect/data-collect.py /dev/null
```

## Lambda functions and API endpoints

As shown in the image below, we have implemented several lambda functions. To create a new function: select 'Create Function' >> enter the Function name and select the runtime (python 3.6) >> select 'Create a new role with basic Lambda permissions' >> then hit 'Create Function'

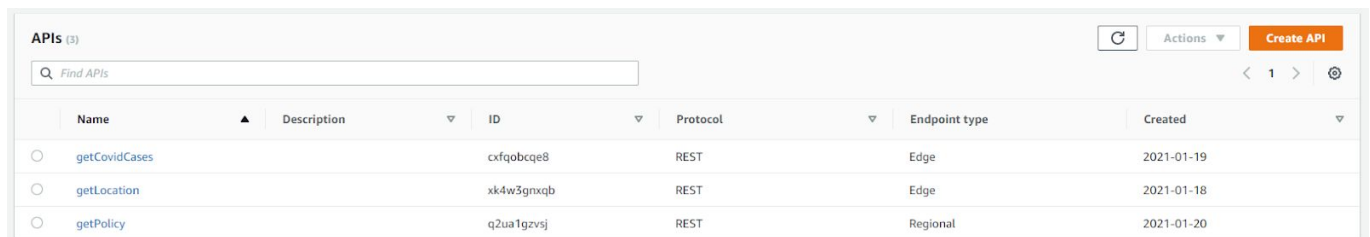


The screenshot shows the AWS Lambda console 'Functions' page. At the top, it says 'Functions (7)' and 'Last fetched 1 minute ago'. There is a search bar with the placeholder 'Filter by tags and attributes or search by keyword'. On the right, there are 'Actions' and 'Create function' buttons. Below the header is a table with the following columns: Function name, Description, Package type, Runtime, Code size, and Last modified. Three functions are listed:

Function name	Description	Package type	Runtime	Code size	Last modified
<a href="#">getDefaultPolicy</a>		Zip	Python 3.6	744 bytes	5 days ago
<a href="#">getCountry</a>		Zip	Python 3.8	565 bytes	3 days ago
<a href="#">getMonthlyCovidPolicy</a>		Zip	Python 3.6	808 bytes	5 days ago

After creating a new function, then we will start coding on the console or upload our code, if a function is required to access another aws service, we then need to select the role and attach a new access policy, for example. Some of our lambda roles have AmazonDynamoDBFullAccess as they need to query DynamoDB.

Then we can use the Lambda functions in our API Gateway, as shown below, we have deployed a few APIs. To integrate a lambda function with an API, we first create an API, then we create new resources and methods, follow the prompt and select the lambda integration.



The screenshot shows the AWS API Gateway console 'APIs' page. At the top, it says 'APIs (3)'. There is a search bar with the placeholder 'Find APIs'. On the right, there are 'Actions' and 'Create API' buttons. Below the header is a table with the following columns: Name, Description, ID, Protocol, Endpoint type, and Created. Three APIs are listed:

Name	Description	ID	Protocol	Endpoint type	Created
<a href="#">getCovidCases</a>		cxfgbcqe8	REST	Edge	2021-01-19
<a href="#">getLocation</a>		xk4w3grxqb	REST	Edge	2021-01-18
<a href="#">getPolicy</a>		q2ua1gzvsj	REST	Regional	2021-01-20

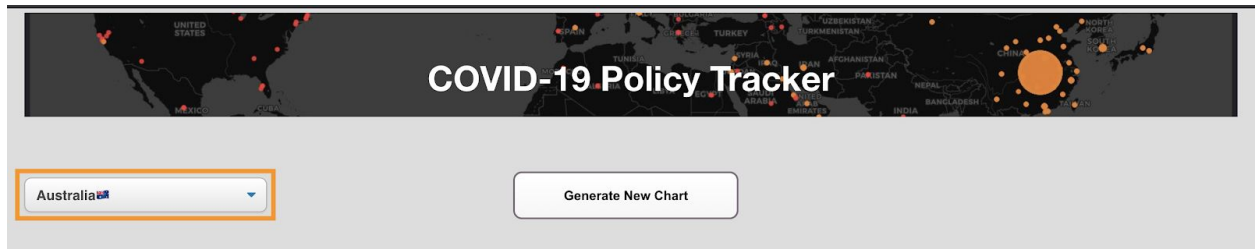
## User Manual

Below is a user manual for our COVID-19 Policy Tracker web app:

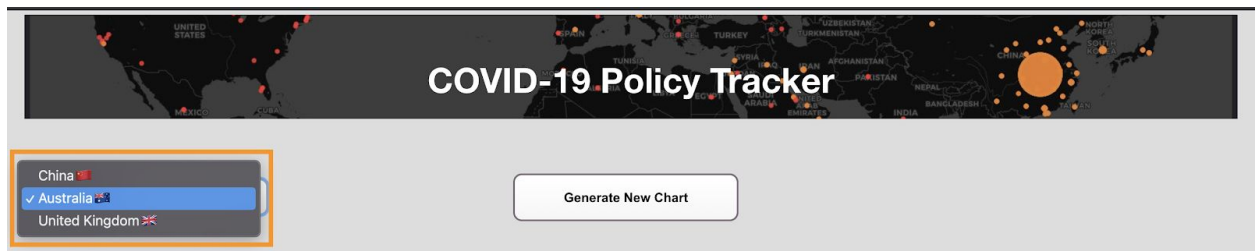
1. Find out what restrictions/policy has been placed in different countries

### Desktop/Tablet View

- 1.1 Click on the options bar marked under the orange rectangle.



- 1.2 You will see the option to select different countries from the selection. Select the desired country to get the information.



- 1.3 By default it will show the main policies in the year 2021 for the selected country.

Eg: Here it is showing for country australia



### Mobile View

1.1 Click on the options bar marked under the orange rectangle.

1.2 Select the desired country to get the information.



1.3 By default it will show the main policies in the whole year 2020 for the selected country.

Eg: Here it is showing for country australia



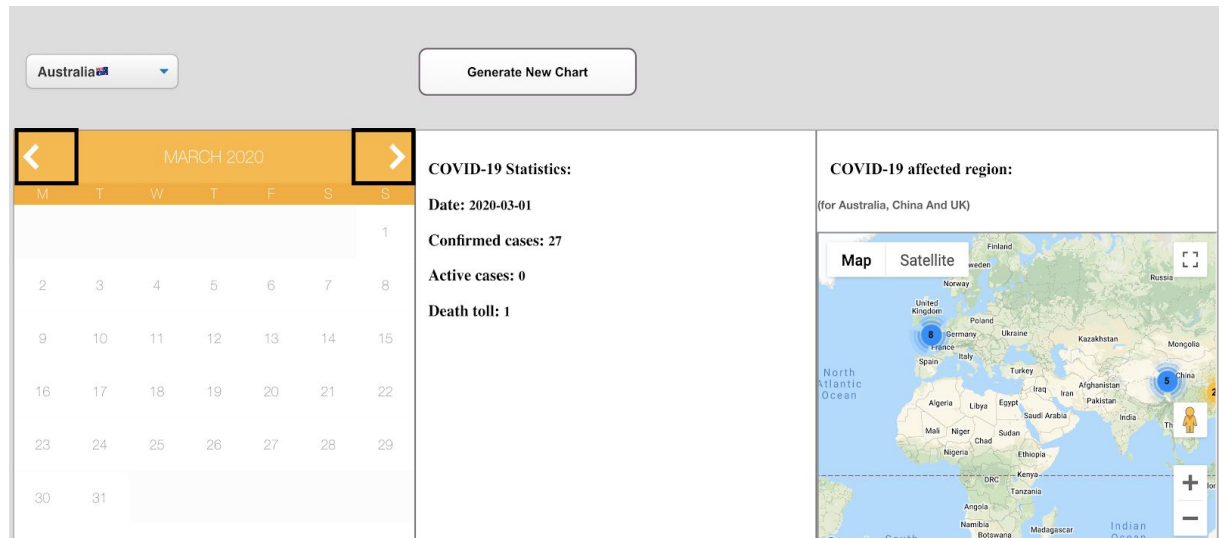
Through this you can easily get to know different rules placed at different times in different countries and be able to follow them.

2. To know what restrictions levels have been placing on a particular month.

### Desktop/Tablet View

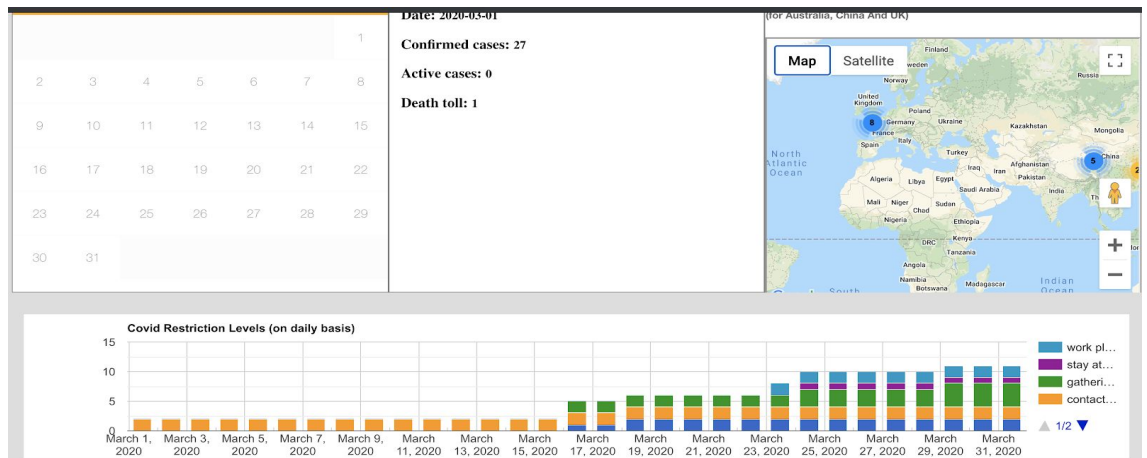
2.1 Select the country.

2.2 Select the month in the calendar as shown below.



2.3 Click on Generate New Chart Button shown in above image.

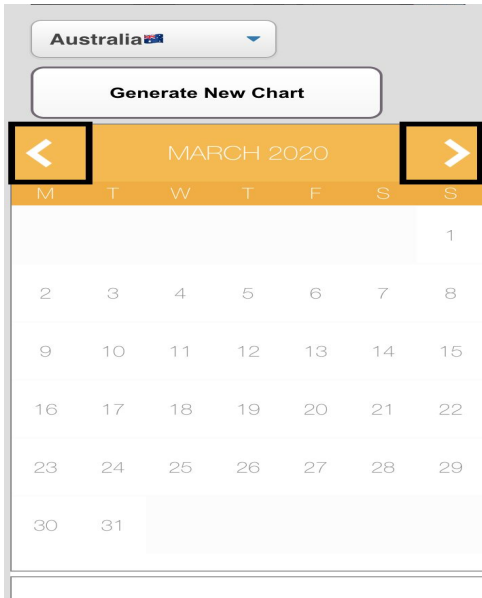
Eg: on selecting month 'March', a graph showing march restriction level is generated.



## Mobile View

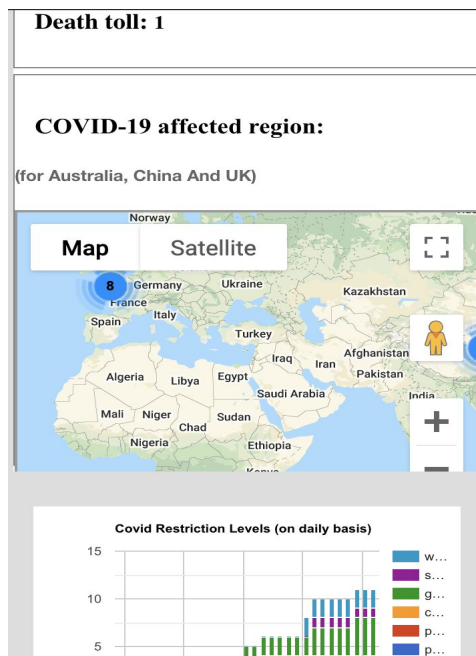
2.1 Select the country.

2.2 Select the month in the calendar as shown below.



2.3 Click on Generate New Chart Button shown in above image.

Eg: on selecting month 'March', a graph showing march restriction level is generated.



3. To know what restrictions levels have been placed on a particular Date in the selected country.

#### Desktop/Tablet View

3.1 Select the country as shown previously

3.2 Select the month on the calendar

3.3 Select the date from the calendar

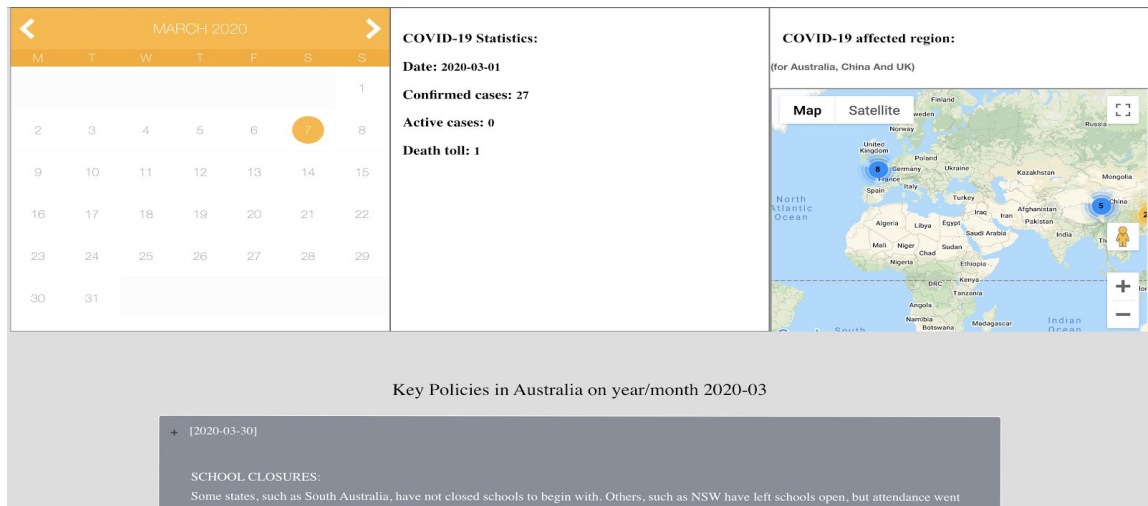


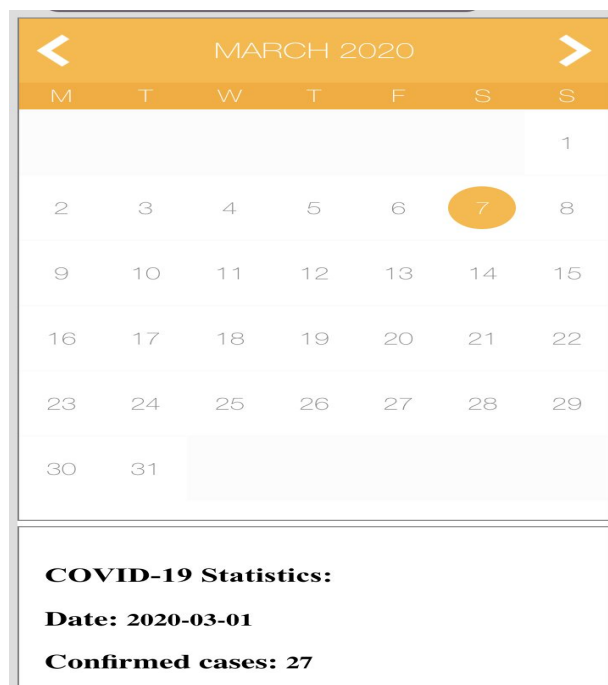
Image 3.3

#### Mobile View

3.1 Select the country as shown previously

3.2 Select the month on the calendar

3.3 Select the date from the calendar.



4. Check details of policies that have been placed for a country.

#### Desktop/Tablet View

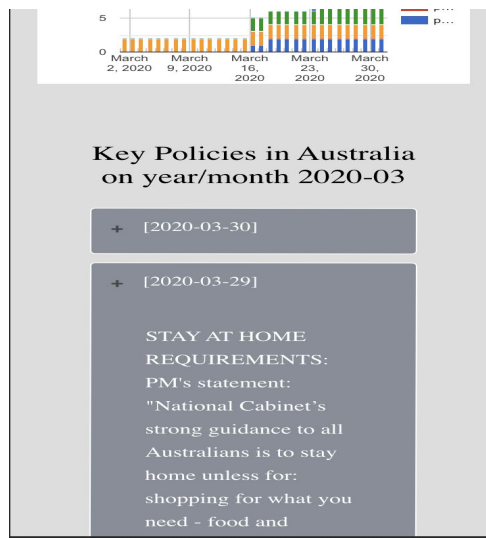
4.1 Repeat all steps from session 3 above

4.2 Click on the tiles to expand it and see the information.(refer to image 3.3)

#### Mobile View

4.1 Follow step 3

4.2 Click on the tiles to get the information.

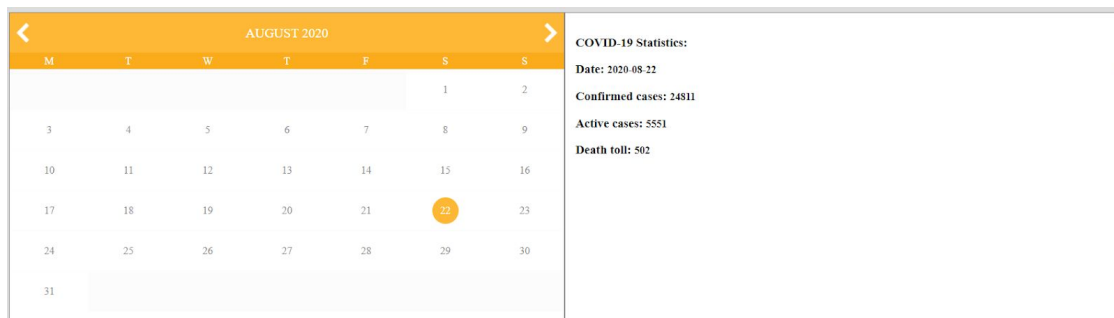


5. Track the number of COVID-19 cases/death toll a country

#### Desktop/Tablet View

5.1 Follow Step 1.1-1.2

5.2 Select month and date from calendar and see the covid statistics

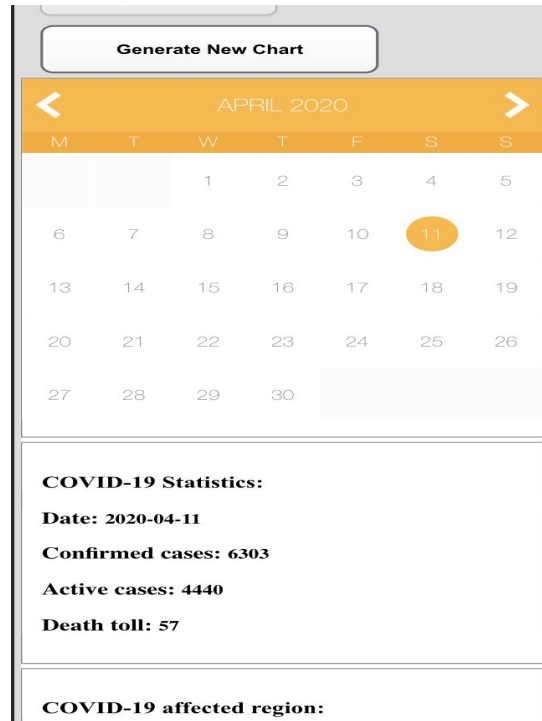




## Mobile View

5.1 Follow Step 1.1-1.2

5.2 Select month and date from calendar and see the covid statistics.



6. To Know the Covid Affected Region.

(This is limited to 3 countries, i.e Australia, China and UK).

6.1 Go to the map loaded on the frontpage.

6.1.1 You can zoom in and out or expand the maps by just clicking on it.

## Potential Future Improvements

The current version of *Covid Policy Tracker* remarkably satisfies its sole purpose to be built but this project has a diverse range of features which could be added to increase the level of information which our current version is giving. As we have shown before, there are only datasets from 3 countries over a one year period but we are already processing over 2000 data items in DynamoDB. Although, we have built the project in a way which is easily scalable. Hence, it is possible to scale up and the process should be very easy.

With our data sourcing we have used data from two highly regarded sources, however we feel that if this project is for public use, it will be better to conduct more research on the datasets being used on the webapp. To ensure the accuracy of our data.

Another area that we can potentially work on is to combine the restriction level data and the covid case data to provide even better visualization for the users. It will help the users to understand more about how the government policies affect the cases, which hopefully in return, this project can help the community to fight the virus.

The Google Maps Visualization could be more explored and could be used to increase the usability of the website. We can have more data visualization on the maps rather than just the quantity of areas majorly affected by Covid-19.

By implementing these features we believe that our project could be more easy to use and be more distinguished to our competitive sites.

## References

Amer, A 2020, *Creating AWS EC2 and connecting it with AWS Cloud9 IDE and AWS S3*, viewed 28 January 2021, <<https://towardsdatascience.com/creating-aws-ec2-and-connecting-it-with-aws-cloud9-ide-and-aws-s3-a6313aa82ec>>

AWS 2021, *Amazon API Gateway*, viewed 28 January 2021, <<https://aws.amazon.com/api-gateway/>>

AWS 2021, *AWS Lambda*, viewed 28 January 2021, <<https://aws.amazon.com/lambda/>>

AWS 2021, *Creating Tables and Loading Data for Code Examples in DynamoDB*, viewed 29 January 2021, <<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SampleData.html>>

AWS 2021, *Deploy Code to a Virtual Machine*, viewed 29 January 2021, <<https://aws.amazon.com/getting-started/tutorials/deploy-code-vm/>>

AWS 2021, *Developer Guide - Prerequisites*, viewed 29 January 2021, <<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create-deploy-python-flask.html>>

AWS 2021, *What is AWS Elastic Beanstalk?*, viewed 29 January 2021, <<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html>>

Crosett, L 2019, *How to determine if Amazon DynamoDB is appropriate for your needs, and then plan your migration*, viewed 26 January 2021, <<https://aws.amazon.com/blogs/database/how-to-determine-if-amazon-dynamodb-is-appropriate-for-your-needs-and-then-plan-your-migration/>>

Full Stack Python 2021, *Flask*, viewed 29 January 2021, <<https://www.fullstackpython.com/flask.html>>

Google Cloud 2020, *Creating and managing projects*, viewed 29 January 2021, <[https://cloud.google.com/resource-manager/docs/creating-managing-projects#creating\\_a\\_project\\_using\\_a\\_service\\_account](https://cloud.google.com/resource-manager/docs/creating-managing-projects#creating_a_project_using_a_service_account)>

Johns Hopkins University of Medicine 2020, *TRACKING Follow global cases and trends, Updated daily*, viewed 25 January 2021, <<https://coronavirus.jhu.edu/data>>

Jones, E 2019, *What's happening in BigQuery: Adding speed and flexibility with 10x streaming quota, Cloud SQL federation and more*, viewed 26 January 2021,

<<https://cloud.google.com/blog/products/data-analytics/whats-happening-bigquery-adding-speed-and-flexibility-10x-streaming-quota-cloud-sql-federation-and-more>>

Linuxize 2019, How to Install Pip on Ubuntu 18.04, viewed 27 January 2021,  
<<https://linuxize.com/post/how-to-install-pip-on-ubuntu-18.04/>>

McIntosh, K 2020, *Coronavirus disease 2019 (COVID-19): Epidemiology, virology, and prevention*, viewed 27 January 2021,  
<<https://www.uptodate.com/contents/coronavirus-disease-2019-covid-19-epidemiology-virology-and-prevention#:~:text=In%20February%202020%2C%20the%20World,of%20COVID%2D19%20is%20evolving>>

Our World in Data 2020, Policy Responses to the Coronavirus Pandemic, viewed 29 January 2021, <<https://ourworldindata.org/policy-responses-covid>>

phoenixNAP 2019, How To Install Python 3 On Ubuntu 18.04 Or 20.04, viewed 25 January 2021, <<https://phoenixnap.com/kb/how-to-install-python-3-ubuntu>>

Python 2021, *venv — Creation of virtual environments*, viewed 28 January 2021,  
<<https://docs.python.org/3/library/venv.html#:~:text=A%20virtual%20environment%20is%20a,part%20of%20your%20operating%20system>>

University of Oxford 2020, *CORONAVIRUS GOVERNMENT RESPONSE TRACKER*, viewed 25 January 2021,  
<<https://www.bsg.ox.ac.uk/research/research-projects/coronavirus-government-response-tracker>>