SAPIENZA
UNIVERSITÀ DI ROMA

Final Project - Cloud Computing

Luong Bang Tran	tran.1956419@studenti.uniroma1.it
Rocco Lo Conte	loconte.1946073@studenti.uniroma1.it
Ahmed Fayez Moustafa Tayel	tayel.1972085@studenti.uniroma1.it
Gaurav Mohan Ramse	ramse.1965564@studenti.uniroma1.it

Rome, July 19 2021

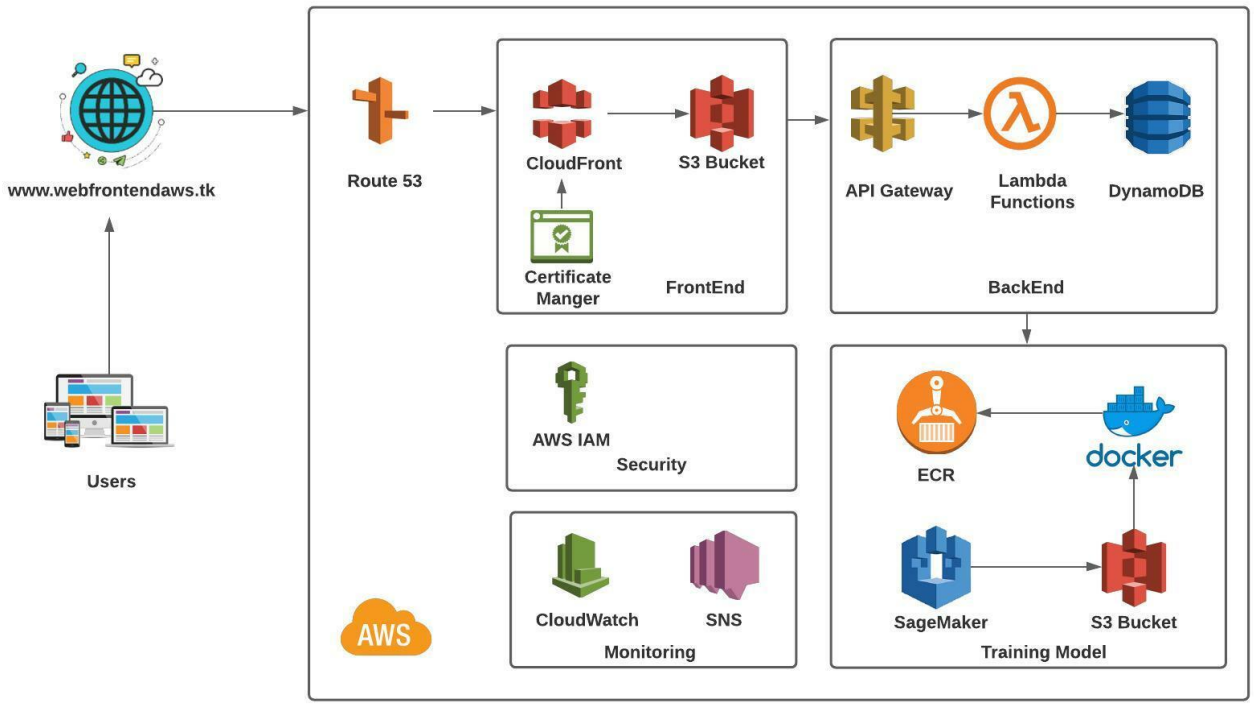
I. Introduction

In this project, we are going to work on Sentiment Analysis using Machine Learning Algorithms. We are going to build a website which allows users to classify a positive or negative comment. This web application will be deployed on AWS cloud platform.

We will also measure the performance and the scalability of webserver (Lambda Functions) by sending multiple requests to our server

II. Design of the Solution

In this project, we try to use services provided by AWS to complete it. The diagram below shows how we are going to use AWS to deploy our web application, monitoring our server and build our model.



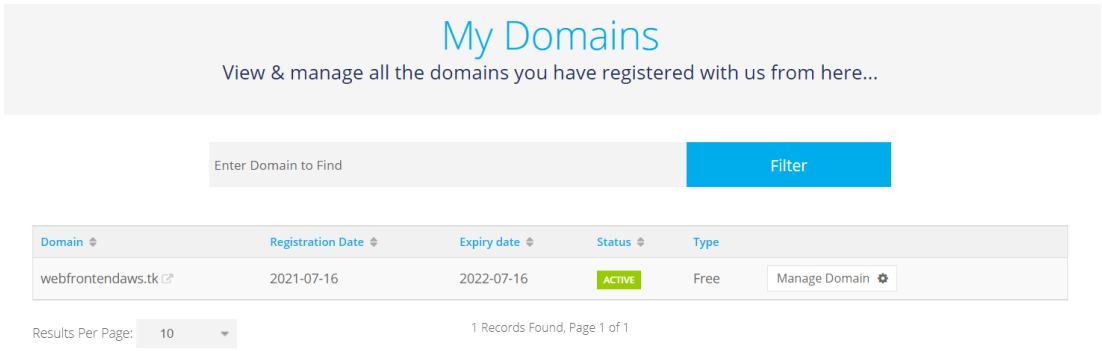
III. Implement and deployment

1. FrontEnd

We provide users a web application with domain name webfrontendaws.tk to access our website from their devices connected to Internet such as computers and mobiles.

DNS

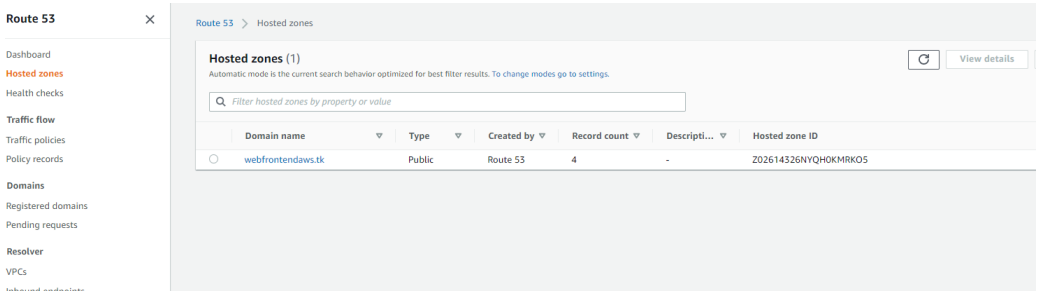
We have registered our domain name webfrontendaws.tk from Freenom World which is a fast and anonymous Public DNS resolver.



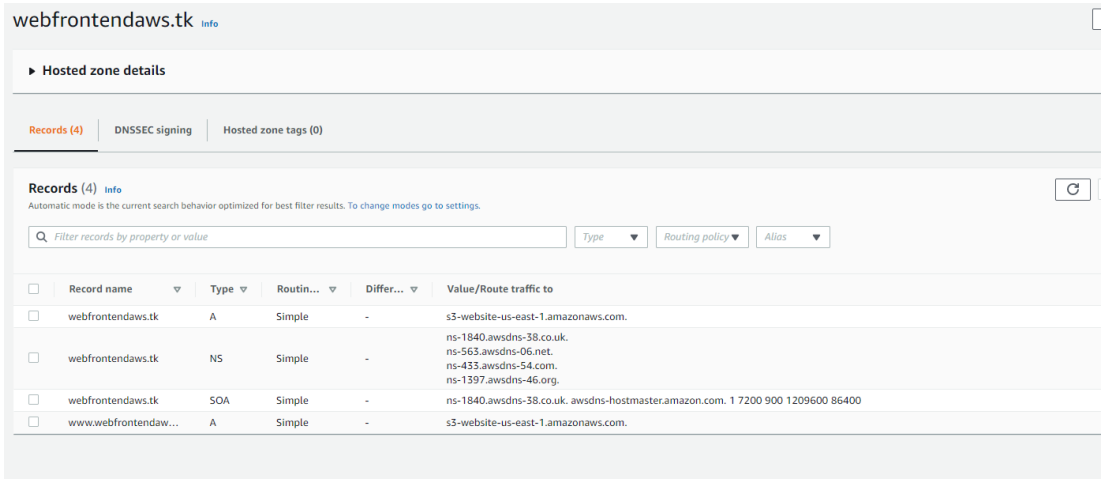
Route 53

Amazon Route 53 is a highly available and scalable cloud Domain Name System web service. Route 53 will redirect requests from users to our website that is originally stored in an Amazon S3 bucket.

We have created a hosted zone on Route 53.



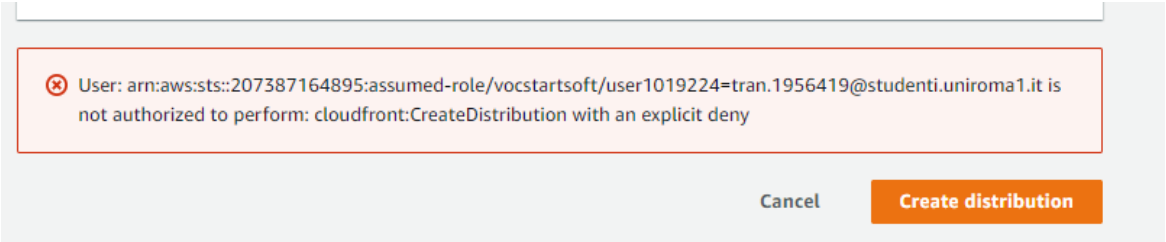
From this hosted zone, we connected it to domain name webfrontendaws.tk and our deployed website stored in Amazon S3 bucket



Cloud Front

Amazon CloudFront is a fast content delivery network (CDN) service that securely delivers data, videos, applications, and APIs to customers globally with low latency, high transfer speeds, all within a developer-friendly environment.

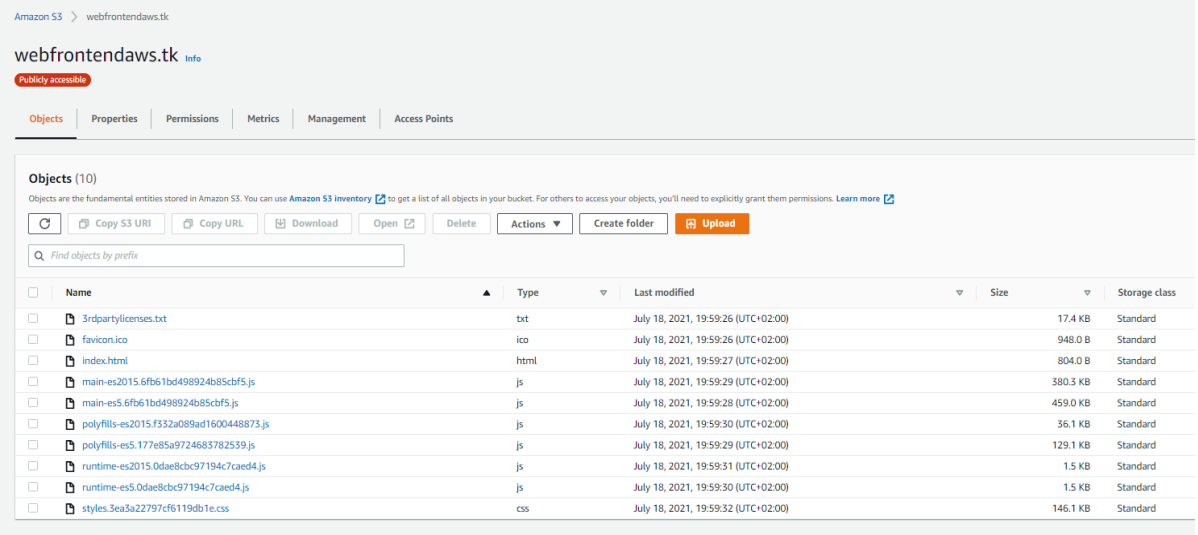
We have tried to create a distribution on CloudFront to speed up our website but we don't have this permission to create it.



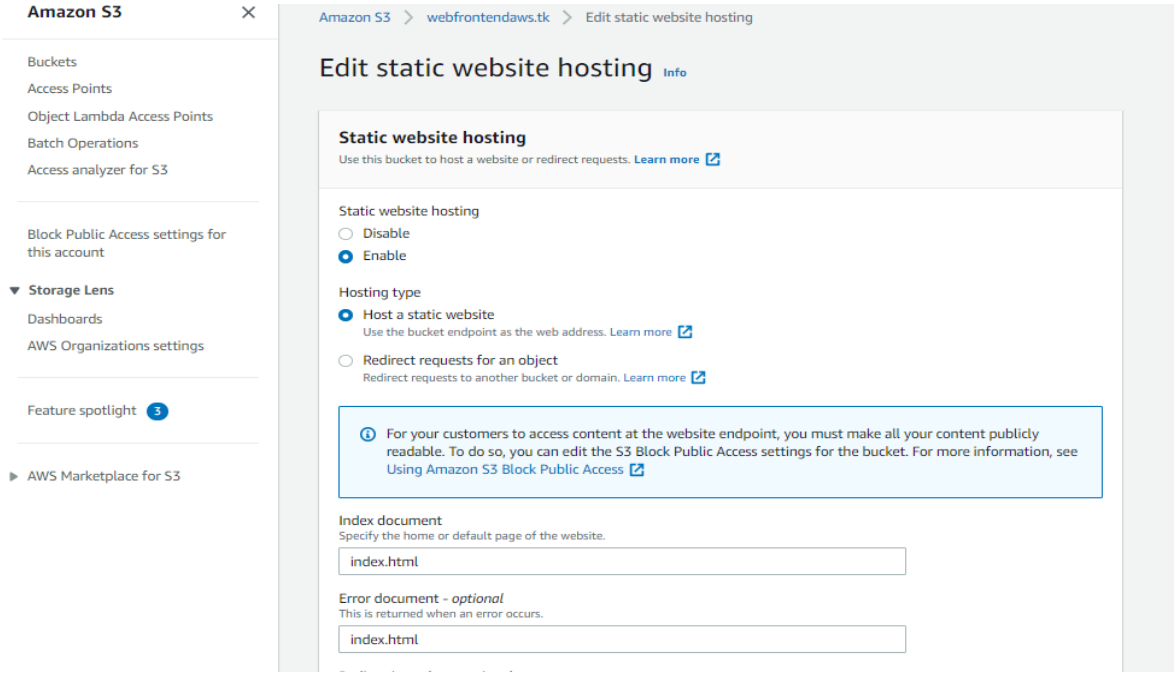
S3 Bucket

Amazon S3 bucket hosts our website which is programmed in Angular.

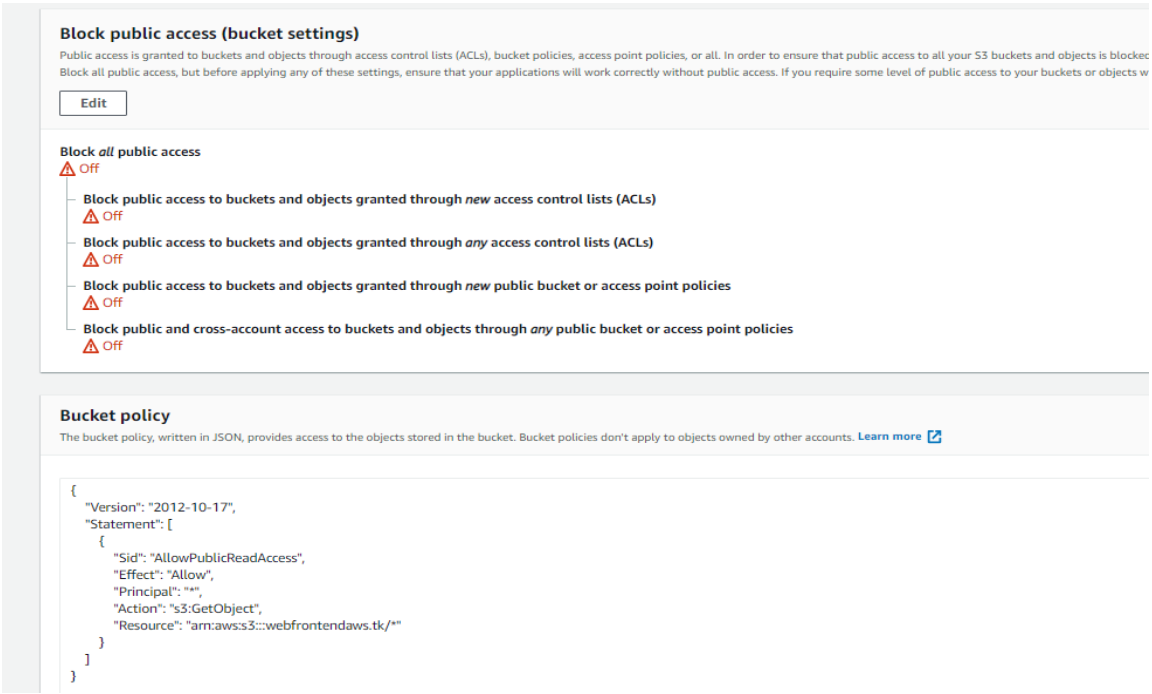
We have uploaded our built website on webfrontedaws.tk S3 bucket.



We have set our web to static website hosting.

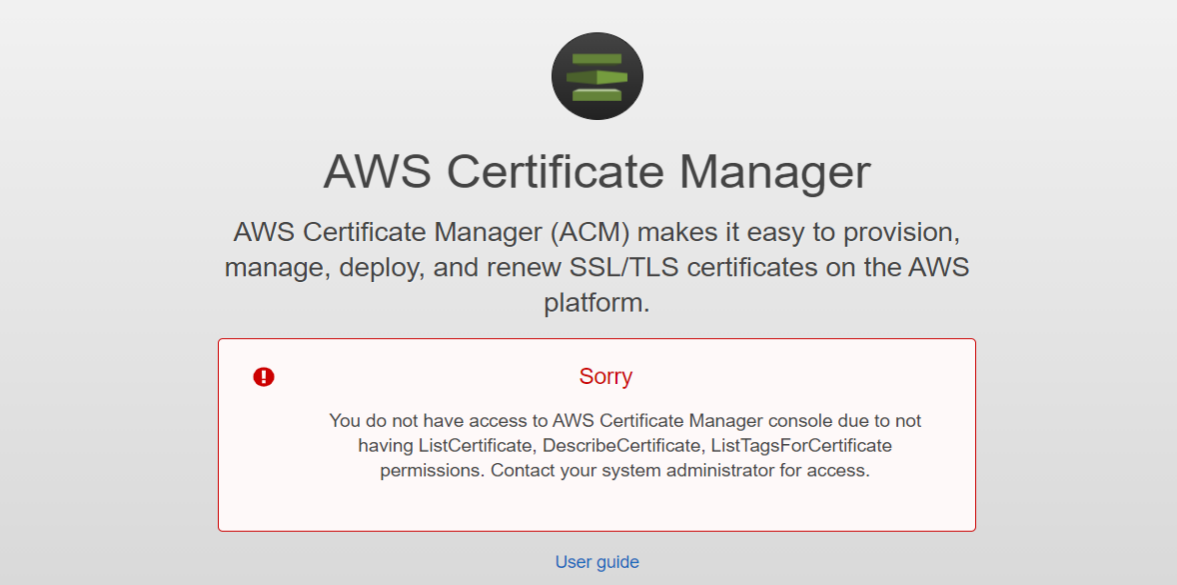


We gave the permission to access this bucket and a policy to getObject from this S3 bucket.



Certificate Manager

We try make our website SSL certified using AWS Certification Manager, which ensures data between our browser and the server is encrypted. Unfortunately, we don't have the permission to use this service.



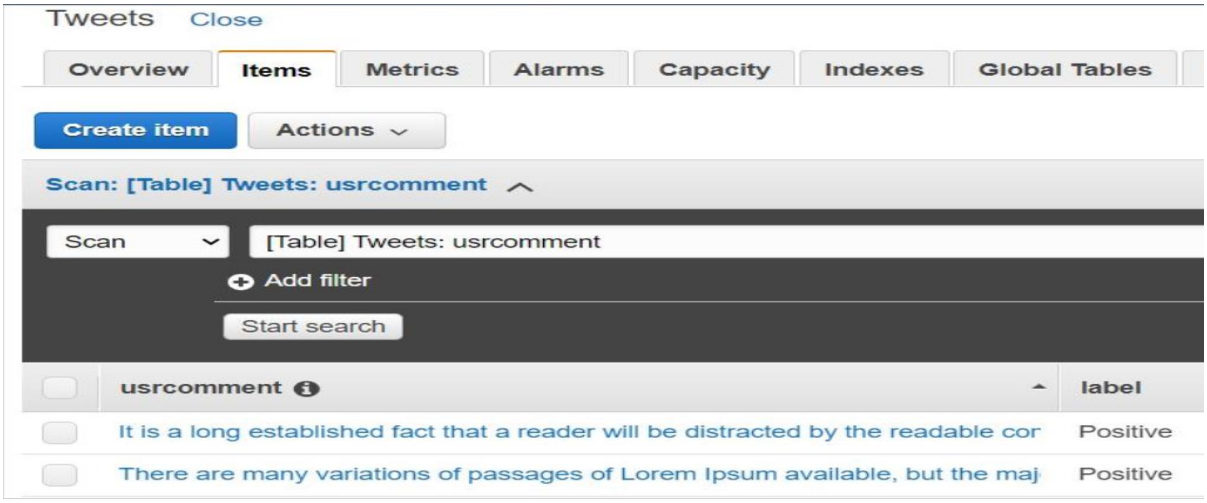
2. Backend

DynamoDB

It consists of only one table (Tweets) which contains two columns (UserComment, Label)
UserComment: comment posted by the user (String)
Label: the comment label {Positive,Negative} (String)

The userComment column acts as the primary key of the table, no secondary keys are specified, as the data in the table should be unique according to only the posted comment, therefore the data is accessed by the comment string

The DynamoDB is created using AWS Console of only 2 columns in one table with one primary key.

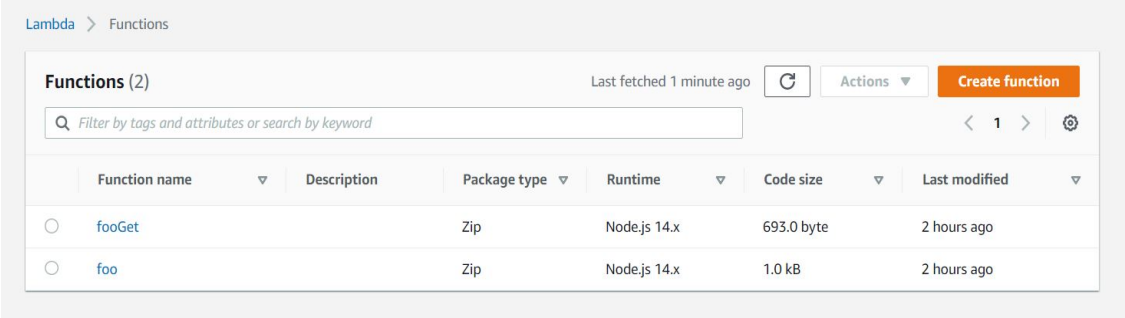


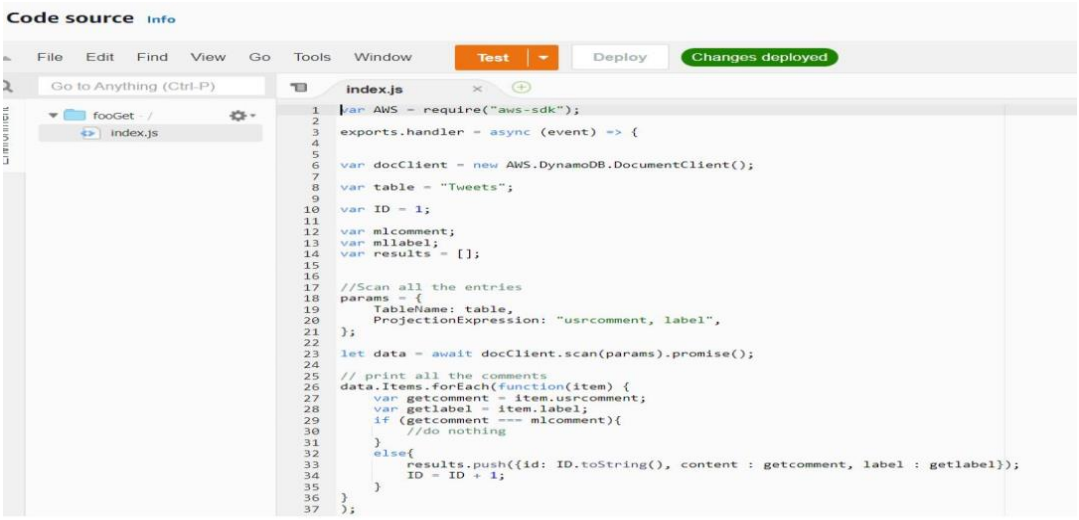
Lambda Function and API Gateway

- It manages the database and links the frontend with the machine learning model
- It is connected to an API Gateway to provide an endpoint to the frontend to send the comment submitted by the user in this format {'content': Comment}
- API gateway provides two methods (GET, POST) requests for two different lambda functions.
- When the frontend invokes the lambda function as a GET request, nothing is sent with the request body and the lambda function only retrieves all the information in the database.
- When the frontend invoke the lambda function as a POST request, the comment is passed to the event variable from the request body and the lambda executes as follows:

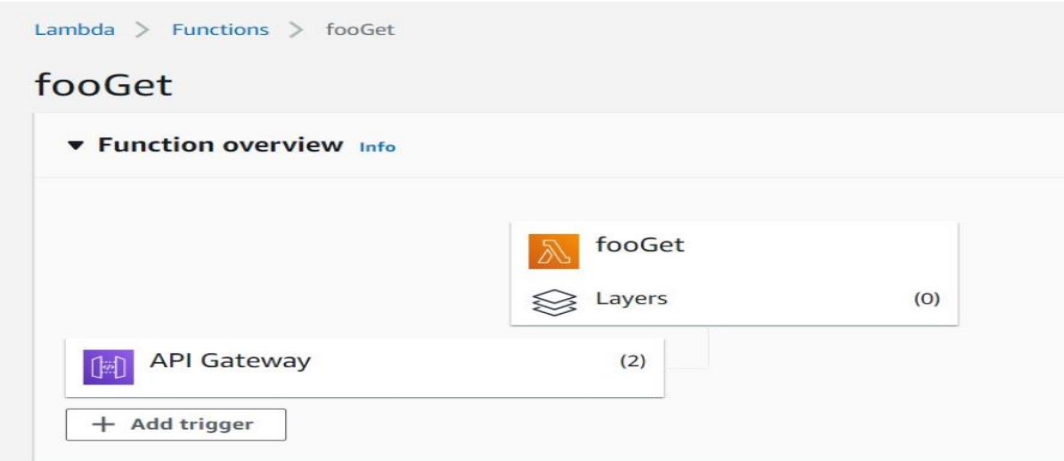
  - It sends the comment to the Machine Learning model for sentiment classification and receives a response (Label)
  - It updates the database with the new label
  - It retrieves all the database information in the response body to the front end in this format {'id': ID, 'content': Comment, 'label': Label}

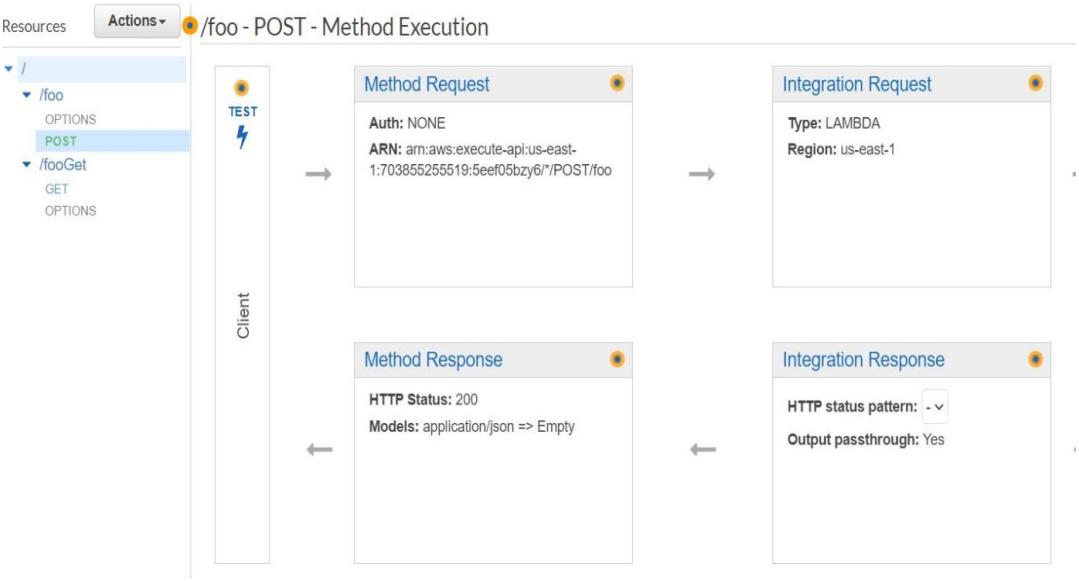
The lambda function is implemented in NodeJS from scratch in a handler function, AWS SDK is used for managing communication with DynamoDB, and a role is assigned to lambda to fully access the database



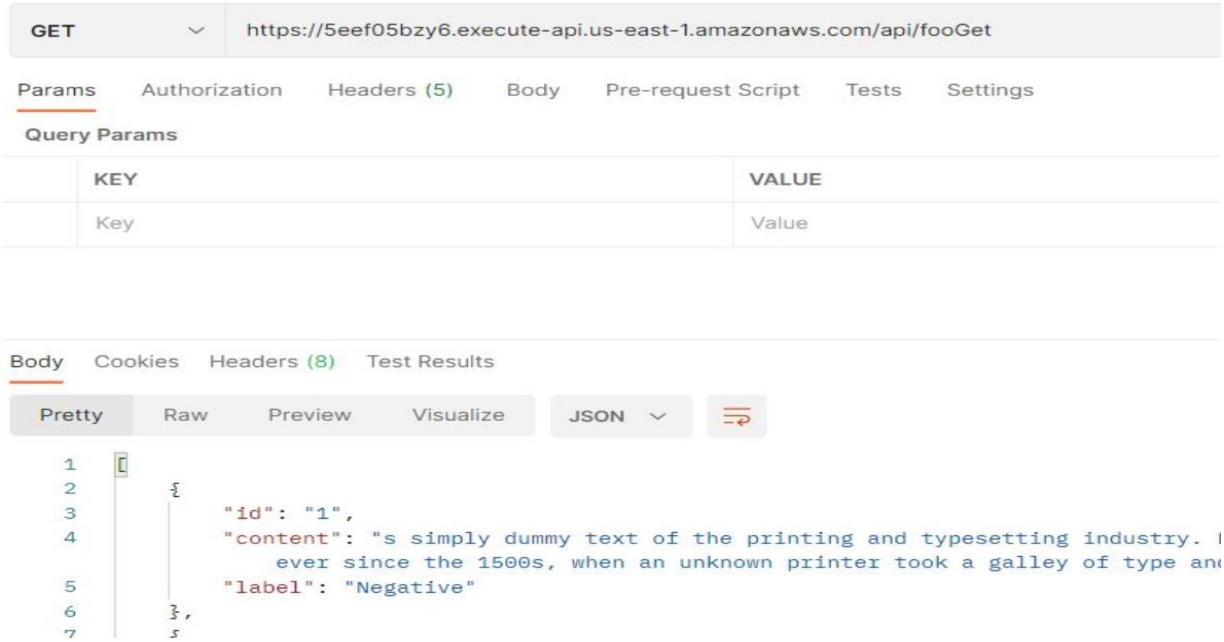


The lambda function is attached to an API Gateway trigger for exposing the end point, the API is public and CORS is enabled.





The API basic functionality is validated by sending GET and POST requests using postman, and the results are compared to the expected results



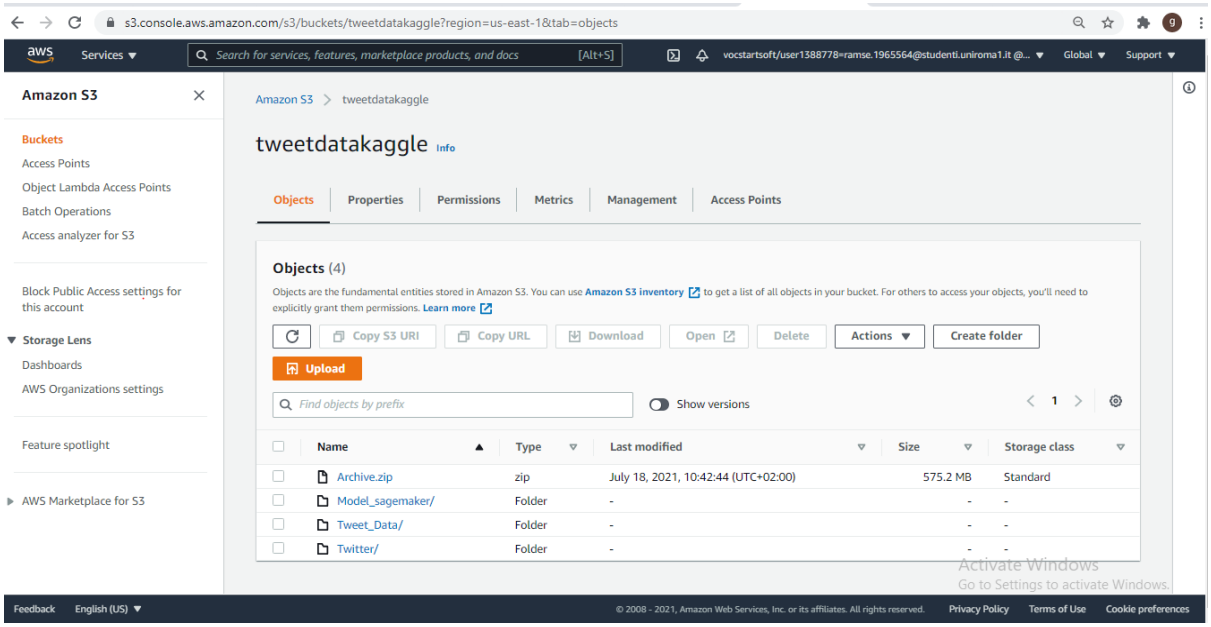
3. Training Model – For Sentiment Analysis

To train model we have used [kaggle_dataset](), In this dataset there are 1.6 million tweets. While training with Bert architecture we got around 84% accuracy.

S3 Bucket

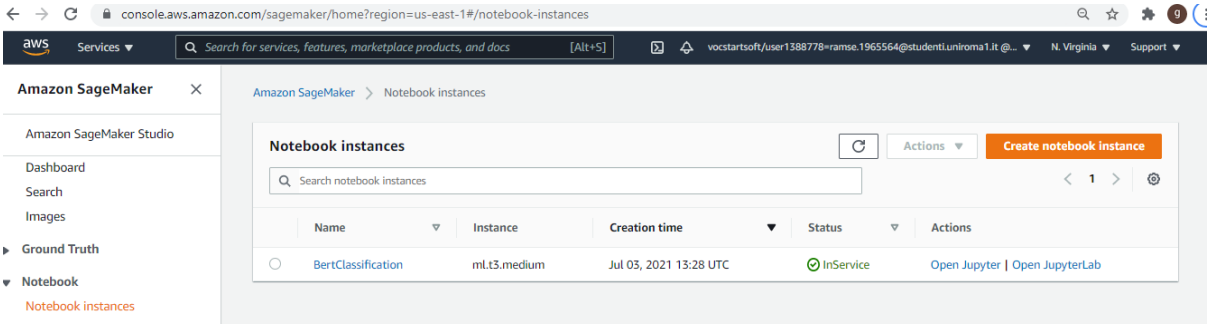
Objects available in s3 is as follows:

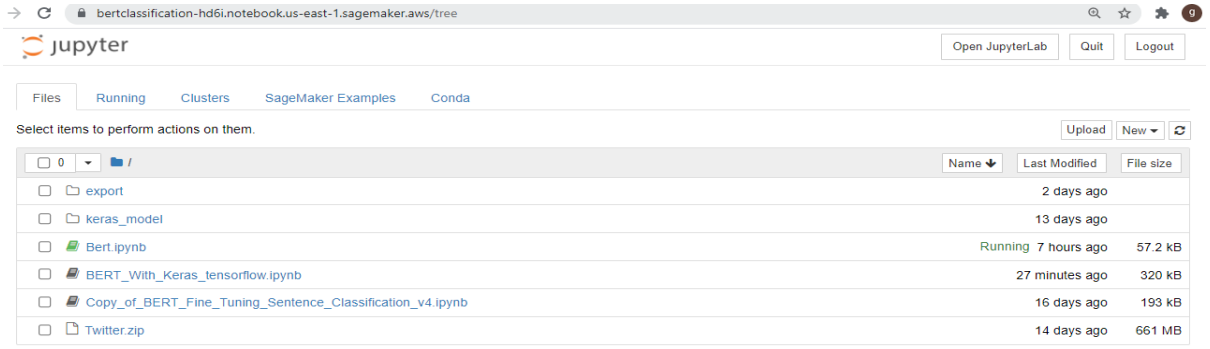
- Tweet_Data – Dataset downloaded from Kaggle for training purposes.
- Twitter- Machine learning Model



SageMaker

We have created ml.t3.medium notebook instance, then used the Pretrained Bert model and finetuned with Tensorflow and Keras. The trained model was saved to the s3 bucket. To connect with s3 in sagemaker, have given public access to the bucket. Then we access the trained model using s3fs (Python Library).



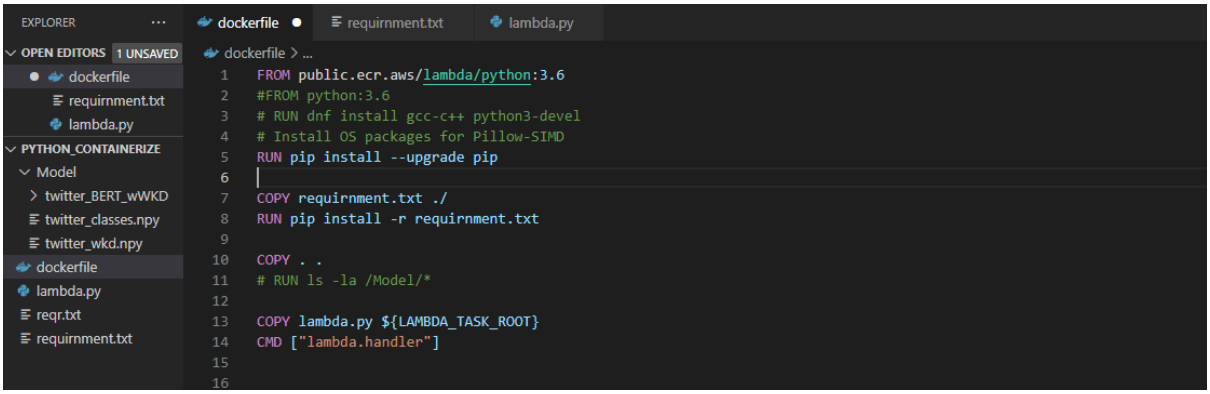


Docker

We decided to use deploying our model using lambda. There are two ways to deploy using lambda one is to use zip (Which has a limit of 250MB) and the other using container (Which has a limit of 10GB).

The code we used to create a Docker image is as shown in a snippet of Vs code.

Commands we used to create docker image: *docker build. -t code_check:v4*



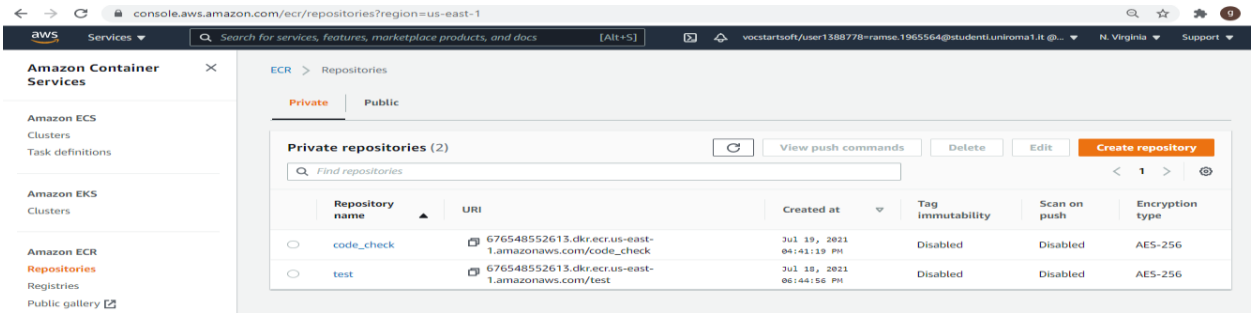
ECR

To use the container we need to upload it to a repository in ECR. We have created a code_check repository and we are trying to add an image we built on a local machine of size 3.83 GB

Steps followed to create repository and push image:

1. aws configure
    i. after executing this command we need to add our credential to connect with aws
2. aws configure set aws_session_token " <token>"
3. aws ecr create-repository --repository-name code_check --region us-east-1
4. aws ecr get-login-password --region us-east-1
5. aws ecr --region us-east-1 I docker login - AWS -p <Above encrypted password> 676548552613.dkr.ecr.us-east-1.amazonaws.com/code_check
6. docker tag code_check:v4 676548552613.dkr.ecr.us-east-1.amazonaws.com/ code_check:v4
7. docker push 676548552613.dkr.ecr.us-east-1.amazonaws.com/ code_check:v4

We faced an issue while uploading large image, So as of now we have tested one test application with lambda and container.



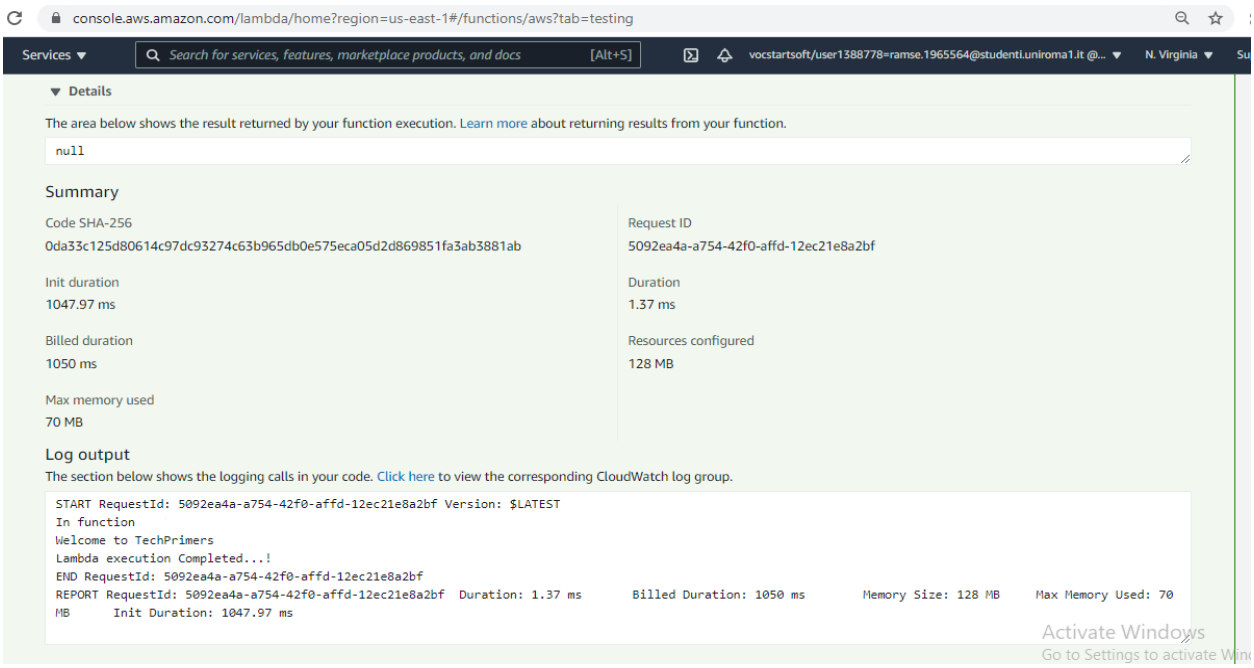

```
Login Succeeded
53fdb38d3a22: Pushing [==========>                                        ]  154.2MB/717.3MB
C:\Users\Gaurav Ramse\Desktop\python_containerize>docker tag code_check:v4 676548552613.dkr.ecr.us-east-1.amazonaws.com/code_check

5e1a8df189c7: Pushing [==>                                                ]    103MB/2.243GB
The push refers to repository [676548552613.dkr.ecr.us-east-1.amazonaws.com/code_check]
8a815ccf85ed: Pushed
53fdb38d3a22: Pushing [==================================================>]  717.3MB/717.3MB
53fdb38d3a22: Pushing [===========>                                       ]  161.5MB/717.3MB
5e1a8df189c7: Pushing [==>                                                ]  94.12MB/2.243GB
222e8e413bbd: Pushed
ebc981406e65: Pushed
5e1a8df189c7: Pushing [==>                                                ]  93.56MB/2.243GB
53fdb38d3a22: Pushing [=============>                                     ]  186.6MB/717.3MB
5e1a8df189c7: Pushing [==>                                                ]  120.9MB/2.243GB
d6fa53d6caa6: Pushed
53fdb38d3a22: Pushing [==========>                                        ]  155.4MB/717.3MB
e62763cfab91: Pushed
652d3dccfc0e: Pushed
write tcp 192.168.65.3:53836->3.86.125.4:443: use of closed network connection

C:\Users\Gaurav Ramse\Desktop\python_containerize>docker push 676548552613.dkr.ecr.us-east-1.amazonaws.com/code_check:v4
The push refers to repository [676548552613.dkr.ecr.us-east-1.amazonaws.com/code_check]
8a815ccf85ed: Layer already exists
53fdb38d3a22: Pushing [==================================================>]  717.3MB/717.3MB
53fdb38d3a22: Pushing [=============>                                     ]  187.7MB/717.3MB
5e1a8df189c7: Pushing [==>                                                ]  92.45MB/2.243GB
222e8e413bbd: Layer already exists
ebc981406e65: Layer already exists
1d39a630092f: Layer already exists
5e1a8df189c7: Pushing [=>                                                 ]  82.42MB/2.243GB
5e1a8df189c7: Pushing [=>                                                 ]  86.88MB/2.243GB
53fdb38d3a22: Pushing [===========>                                       ]  159.8MB/717.3MB
5e1a8df189c7: Pushing [==>                                                ]  91.89MB/2.243GB
e62763cfab91: Layer already exists
652d3dccfc0e: Layer already exists
write tcp 192.168.65.3:47984->52.207.2.251:443: use of closed network connection
```

Lambda Function

We are trying to upload 4 GB of an image to the ECR, It always stops after uploading almost all images. So we have tried to check with a small image size and it worked. As we can see we can execute print statement from the script.

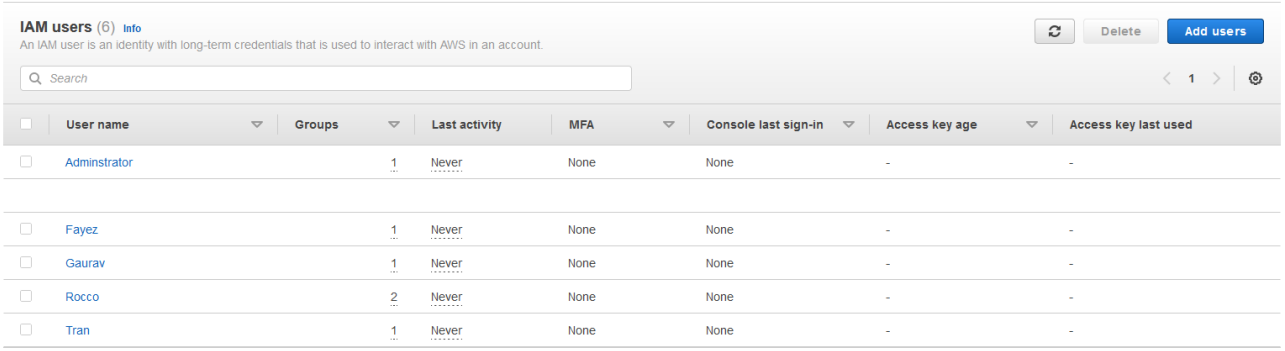


4. Security

IAM

Security of the system was achieved with the IAM service, using the principle of least privilege. The following user were created: 1 admin, 4 users. Each user has a different policy:

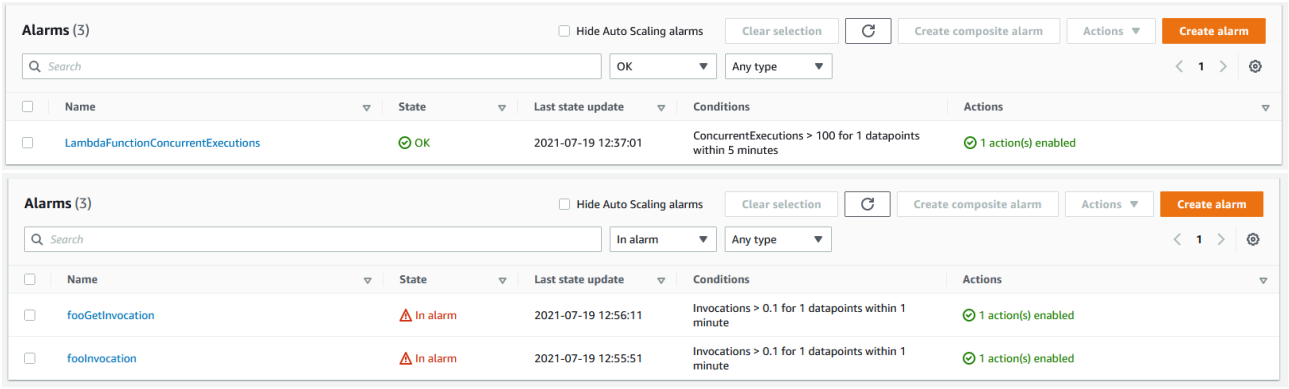
- write access to all services in the frontend part (including Route 53) and read access to everything else (Tran)
- write access to all services in the backend part and read access to everything else (Fayez)
- write access to all services in the monitoring part and read access to everything else (Rocco)
- write access to all services in the training model part and read access to everything else (Gaurav).



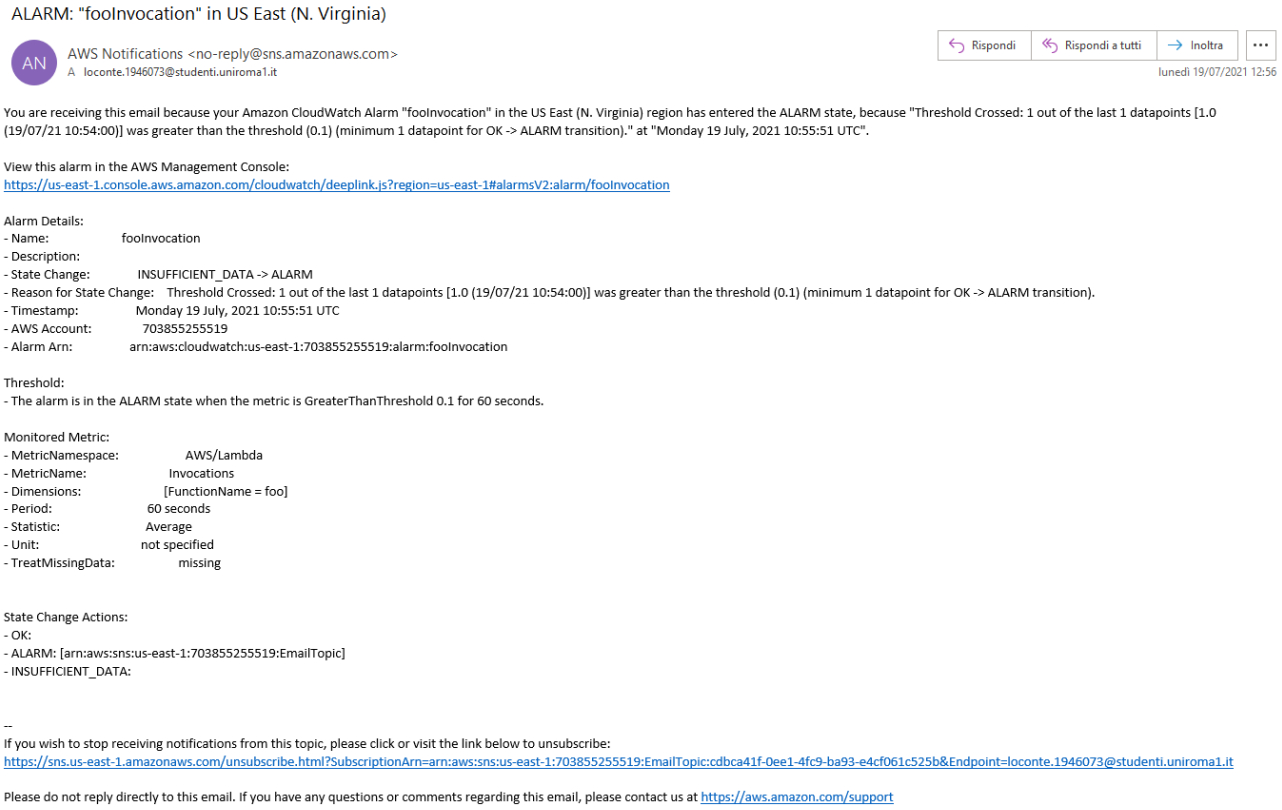
5. Monitoring

CloudWatch

Through CloudWatch, alarms were created to monitor the Lambda's service. An SNS (Simple Notification Service) topic based on email protocol was created, with the creation of two IAM roles (SNSFailureFeedback and SNSSuccessFeedback) for granting permissions and loconte.1946073@studenti.uniroma1.it as endpoint. When a threshold is crossed, an email is sent to the endpoint to notify the alarm.

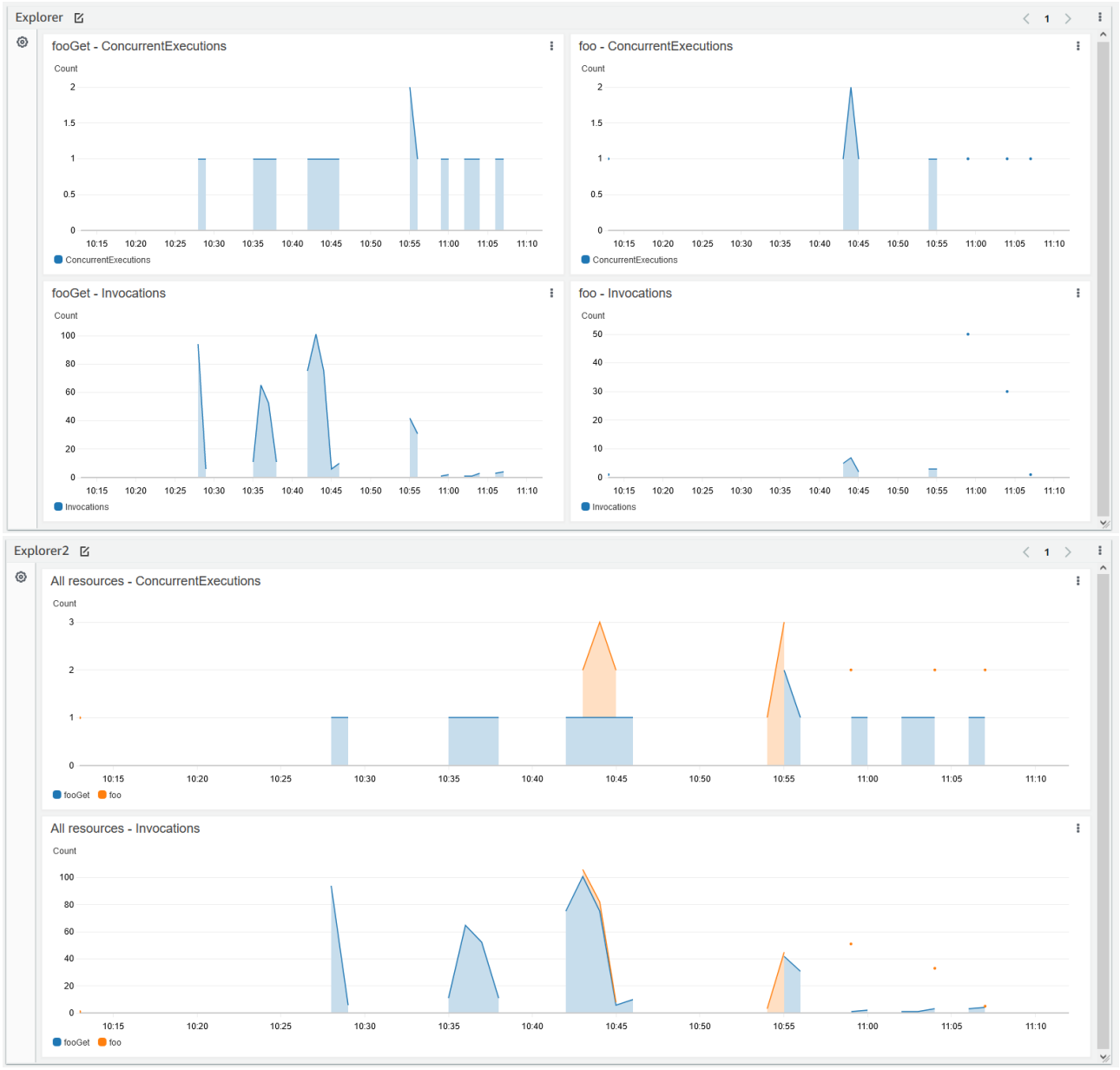


SNS



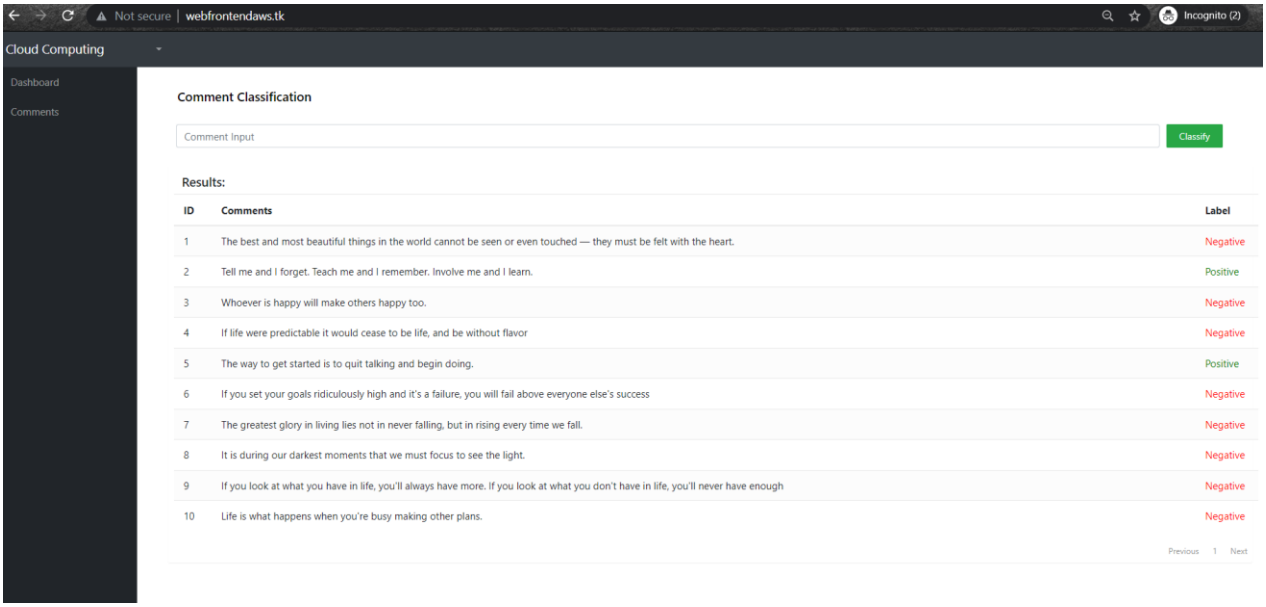
An Auto Scaling policy is set on the concurrency of a Lambda function. When a threshold is crossed (minimum is 100), the function accesses a reserved concurrency. When also the reserved concurrency is full taken, new concurrency is provided based on pricing (pay for the number of new concurrency provided).

In the figure below we can see the monitoring of the invocations and concurrent executions of the two Lambda functions. In the first figure the graph shows the single invocations or concurrent executions, while in the second one the user can compare the data.



IV. Test/validation design and result

After building and deploying our application (Frontend and Backend) on AWS, we can access to website webfrontendaws.tk to classify a comment which is positive or negative.



We aslo write a script to send multiple requests to the web server to test the system behavior in case of traffic and to test the monitoring configuration.

load_POST.py

```python
import requests, time, json

iterations = 30

url = 'https://5eef05bzy6.execute-api.us-east-1.amazonaws.com/api/foo'
myobj = {'content': 1}



for i in range(iterations):
    myobj['content'] = "hello" + str(i)
    x = requests.post(url, data = json.dumps(myobj))
    print(x.status_code, i)
    #time.sleep(0.3)

```

load_GET.py

```python
import requests

url = 'https://5eef05bzy6.execute-api.us-east-1.amazonaws.com/api/fooGet'
iterations = 30


for i in range(iterations):
    x = requests.get(url)
    print(x.status_code, i)

```

Our results of testing are shown in the Monitoring section.