



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

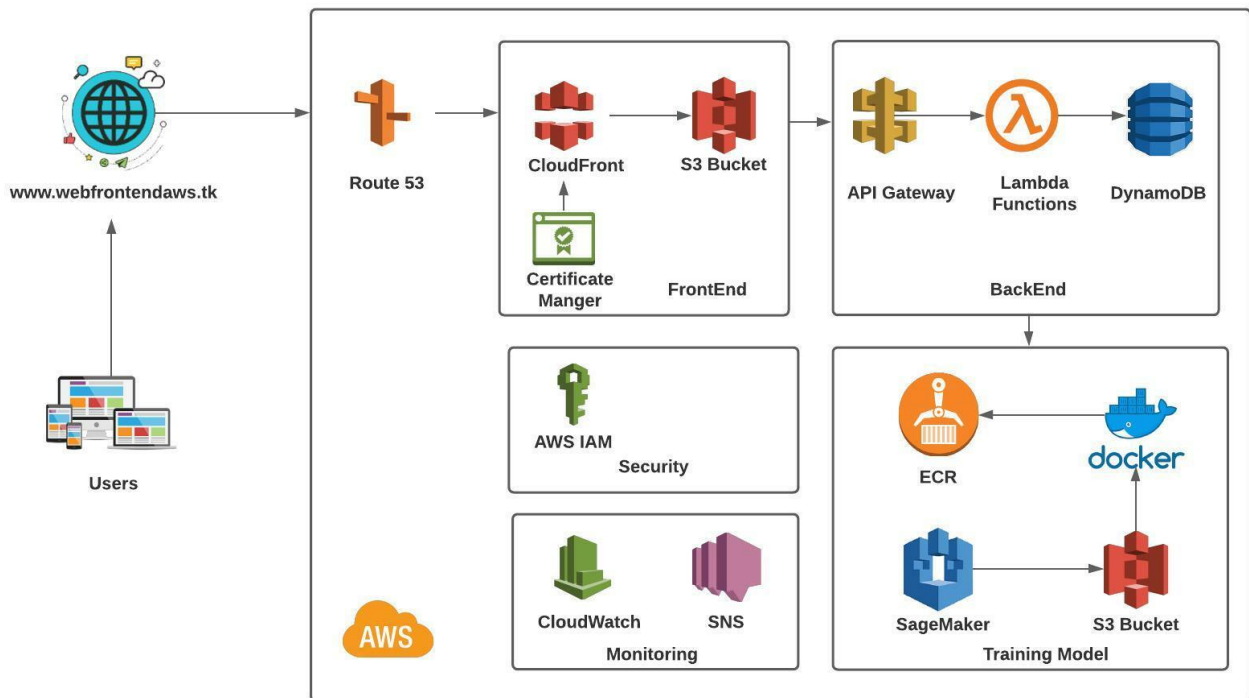
I. Introduction

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.

My Domains

View & manage all the domains you have registered with us from here...

Filter

Domain	Registration Date	Expiry date	Status	Type	
webfrontendaws.tk	2021-07-16	2022-07-16	ACTIVE	Free	Manage Domain

Results Per Page: 10

1 Records Found, Page 1 of 1

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.

Route 53

Dashboard

Hosted zones

Health checks

Traffic flow

Traffic policies

Policy records

Domains

Registered domains

Pending requests

Resolver

VPCs

Route 53 > Hosted zones

Hosted zones (1)

Automatic mode is the current search behavior optimized for best filter results. To change modes go to settings.

Domain name	Type	Created by	Record count	Descripti...	Hosted zone ID
webfrontendaws.tk	Public	Route 53	4	-	Z02614326NYQHOKMRK05

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

webfrontendaws.tk

Hosted zone details

Records (4)

DNSSEC signing

Hosted zone tags (0)

Records (4)

Automatic mode is the current search behavior optimized for best filter results. To change modes go to settings.

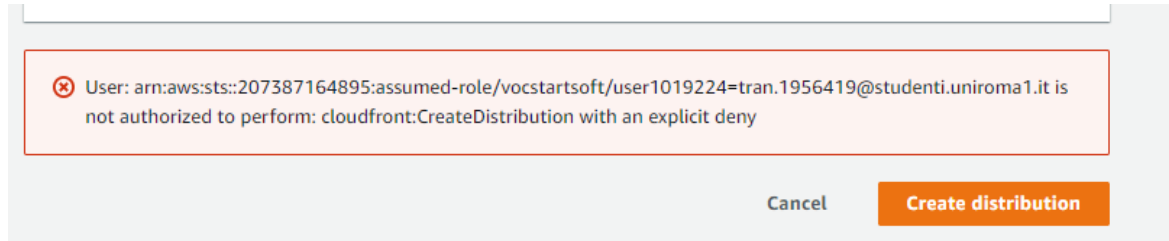
Type Routing policy Alias

	Record name	Type	Routin...	Differ...	Value/Route traffic to
<input type="checkbox"/>	webfrontendaws.tk	A	Simple	-	s3-website-us-east-1.amazonaws.com.
<input type="checkbox"/>	webfrontendaws.tk	NS	Simple	-	ns-1840.awsdns-38.co.uk. ns-563.awsdns-06.net. ns-433.awsdns-54.com. ns-1397.awsdns-46.org.
<input type="checkbox"/>	webfrontendaws.tk	SOA	Simple	-	ns-1840.awsdns-38.co.uk. awsdns-hostmaster.amazon.com. 1 7200 900 1209600 86400
<input type="checkbox"/>	www.webfrontendaw...	A	Simple	-	s3-website-us-east-1.amazonaws.com.

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.

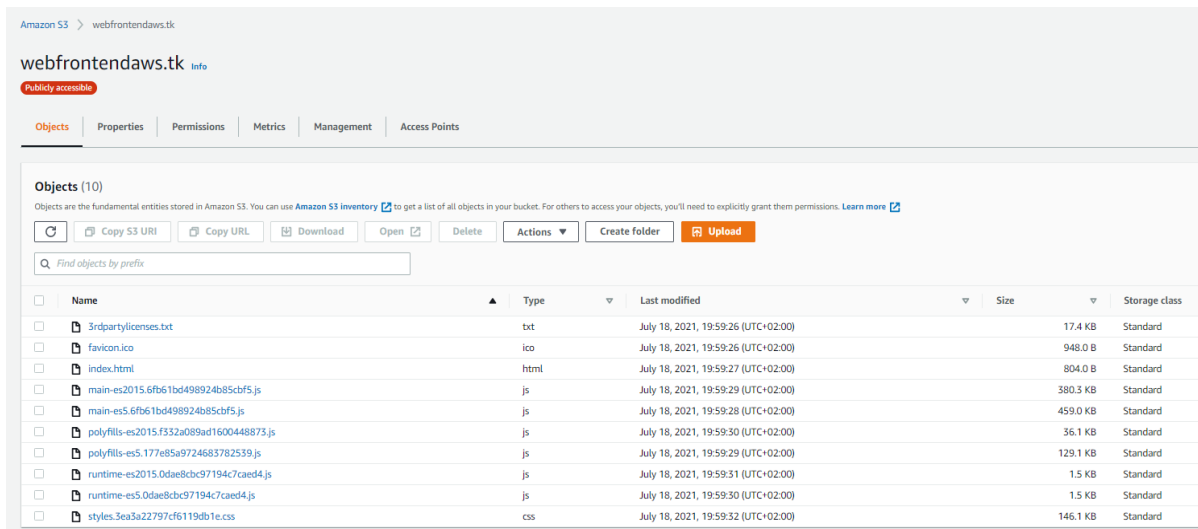
We have tried to create a distribution on CloudFront to speed up our website but we don't have this permission to create it.



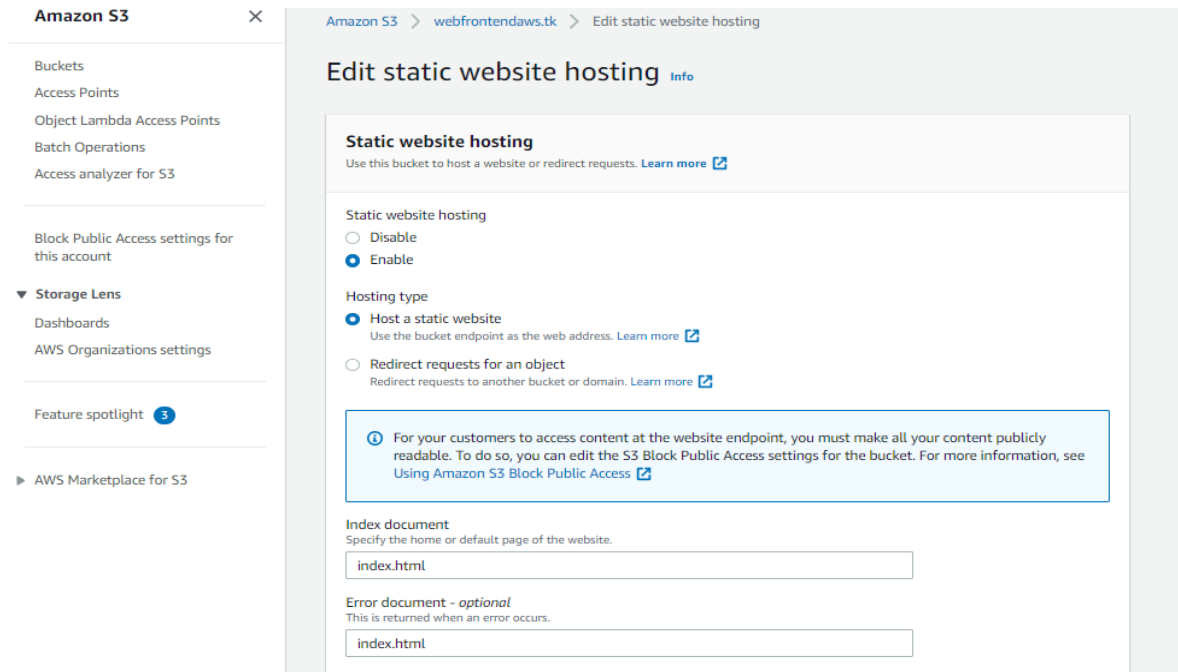
S3 Bucket

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

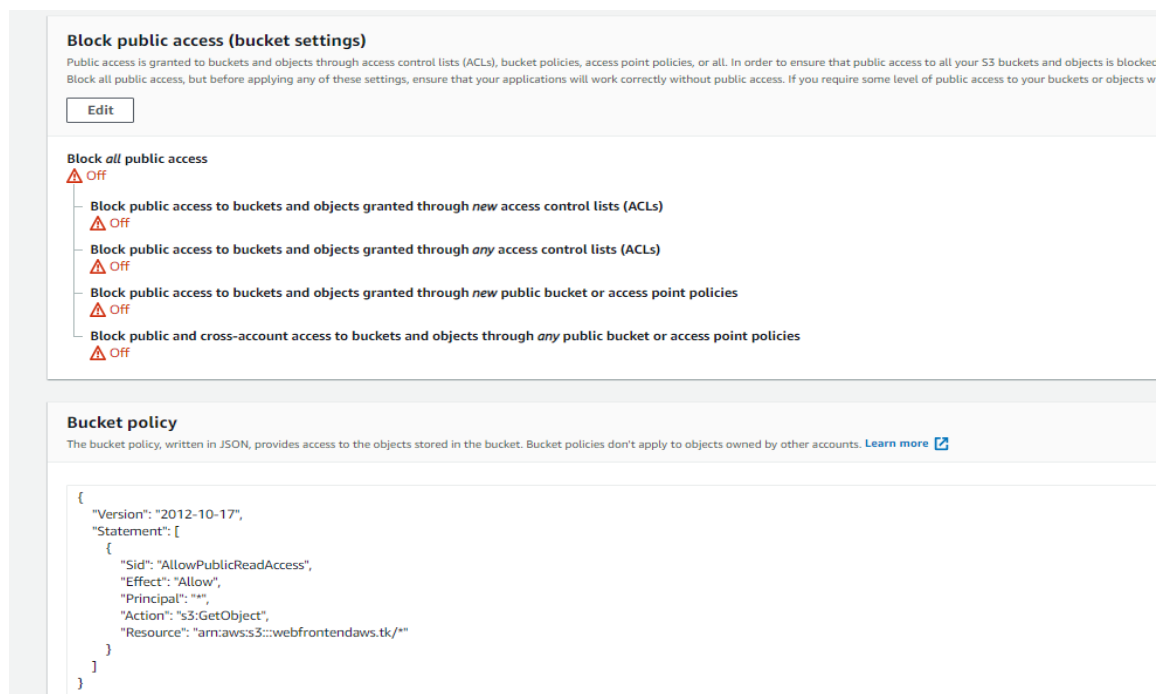
We have uploaded our built website on webfrontedaws.tk S3 bucket.



We have set our web to static website hosting.

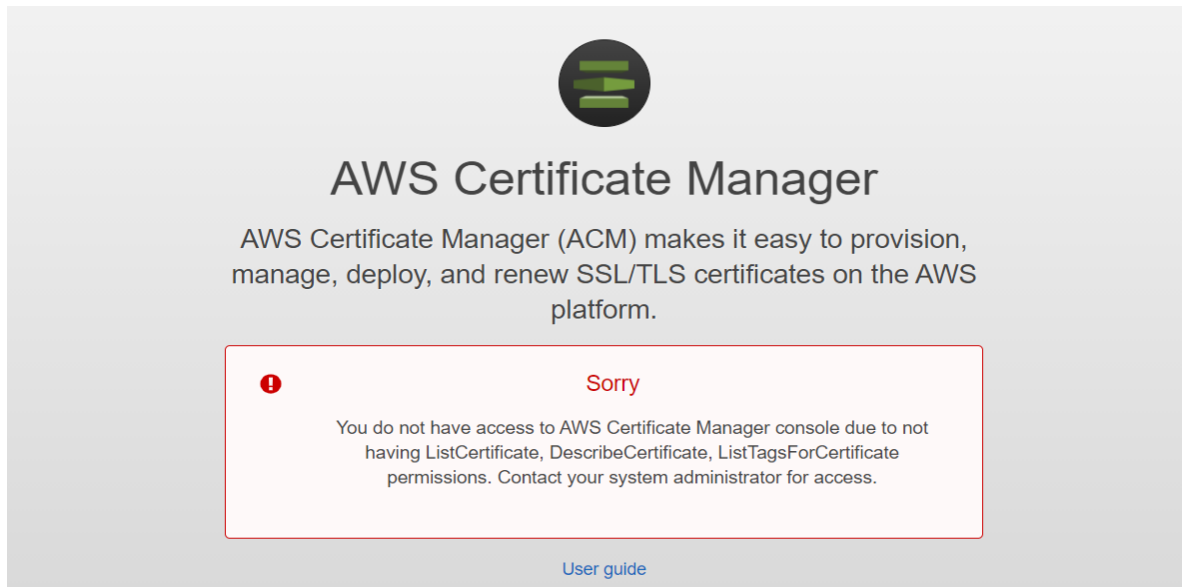


We gave the permission to access this bucket and a policy to getObject from this S3 bucket.



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.

The screenshot shows the AWS DynamoDB console interface for a table named 'Tweets'. At the top, there is a 'Tweets' header with a 'Close' link. Below the header, there are several tabs: 'Overview', 'Items', 'Metrics', 'Alarms', 'Capacity', 'Indexes', and 'Global Tables'. The 'Items' tab is currently selected. Below the tabs, there is a 'Create item' button and an 'Actions' dropdown menu. Below these, there is a search bar with the text 'Scan: [Table] Tweets: usrcomment' and a dropdown menu set to 'Scan'. Below the search bar, there is a text input field containing '[Table] Tweets: usrcomment', an 'Add filter' button, and a 'Start search' button. Below the search bar, there is a table with two columns: 'usrcomment' and 'label'. The 'usrcomment' column has a checkbox and an information icon. The 'label' column has a dropdown arrow. The table contains two rows of data. The first row has the value 'It is a long established fact that a reader will be distracted by the readable cor' in the 'usrcomment' column and 'Positive' in the 'label' column. The second row has the value 'There are many variations of passages of Lorem Ipsum available, but the maj' in the 'usrcomment' column and 'Positive' in the 'label' column.

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 screenshot displays the AWS Lambda console. The top section shows a list of functions under the heading 'Functions (2)'. Below this is a table with columns for Function name, Description, Package type, Runtime, Code size, and Last modified. Two functions are listed: 'fooGet' and 'foo'. The bottom section shows the 'Configuration' tab for a selected function, with a sidebar on the left containing 'General configuration', 'Triggers', and 'Permissions'. The main content area is titled 'Execution role' and shows the 'Role name' as 'lambda_to_Dynamo' with a link to view the role.

Function name	Description	Package type	Runtime	Code size	Last modified
fooGet		Zip	Node.js 14.x	693.0 byte	2 hours ago
foo		Zip	Node.js 14.x	1.0 kB	2 hours ago

Execution role

Role name
lambda_to_Dynamo

Code source Info

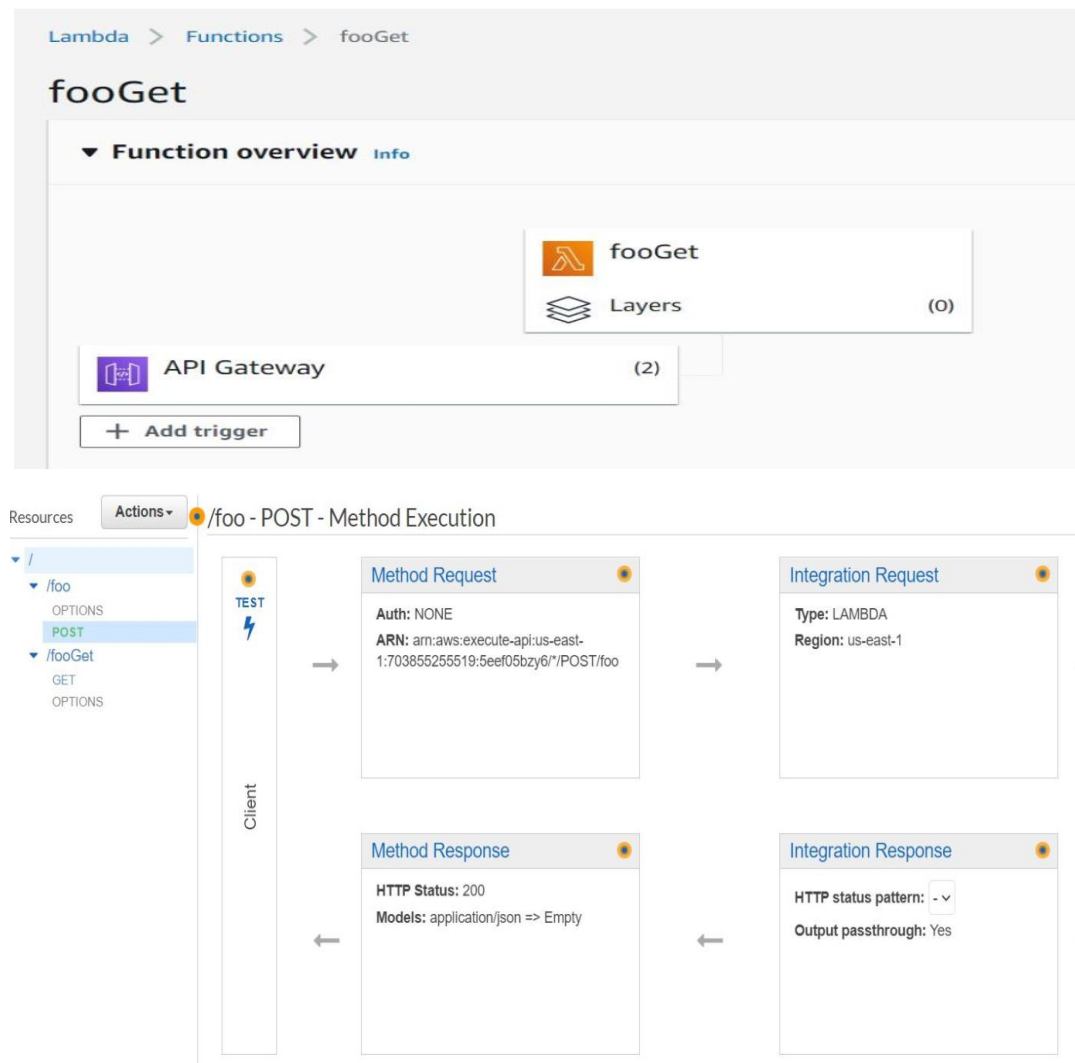
File Edit Find View Go Tools Window Test Deploy Changes deployed

Go to Anything (Ctrl-P)

fooGet - /
index.js

```
1 var AWS = require("aws-sdk");
2
3 exports.handler = async (event) => {
4
5   var docClient = new AWS.DynamoDB.DocumentClient();
6   var table = "Tweets";
7   var ID = 1;
8   var mlcomment;
9   var mlcomment;
10  var mlcomment;
11  var mlcomment;
12  var mlcomment;
13  var mlcomment;
14  var results = [];
15
16  //Scan all the entries
17  params = {
18    TableName: table,
19    ProjectionExpression: "usrcomment, label",
20  };
21
22  let data = await docClient.scan(params).promise();
23
24  // print all the comments
25  data.Items.forEach(function(item) {
26    var getcomment = item.usrcomment;
27    var getlabel = item.label;
28    if (getcomment !== mlcomment){
29      //do nothing
30    }
31    else{
32      results.push({id: ID.toString(), content : getcomment, label : getlabel});
33      ID = ID + 1;
34    }
35  })
36
37  ;
```

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

GET <https://5eef05bzy6.execute-api.us-east-1.amazonaws.com/api/fooGet>

Params Authorization Headers (5) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```

1  {
2    "id": "1",
3    "content": "s simply dummy text of the printing and typesetting industry. I
4              ever since the 1500s, when an unknown printer took a galley of type and
5    "label": "Negative"
6  },
7  ]

```

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

← → ↻ s3.console.aws.amazon.com/s3/buckets/tweetdatakaggle?region=us-east-1&tab=objects

Amazon S3 Services Search for services, features, marketplace products, and docs [Alt+S] vocstartsoft/user1388778@ramse.19655564@studenti.uniroma1.it @... Global Support

Amazon S3 ×

Buckets

Access Points

Object Lambda Access Points

Batch Operations

Access analyzer for S3

Block Public Access settings for this account

Storage Lens

Dashboards

AWS Organizations settings

Feature spotlight

▶ AWS Marketplace for S3

Amazon S3 > tweetdatakaggle

tweetdatakaggle Info

Objects Properties Permissions Metrics Management Access Points

Objects (4)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Copy S3 URI Copy URL Download Open Delete Actions Create folder

Upload

Find objects by prefix Show versions

	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	Archive.zip	zip	July 18, 2021, 10:42:44 (UTC+02:00)	575.2 MB	Standard
<input type="checkbox"/>	Model_sagemaker/	Folder	-	-	-
<input type="checkbox"/>	Tweet_Data/	Folder	-	-	-
<input type="checkbox"/>	Twitter/	Folder	-	-	-

Activate Windows Go to Settings to activate Windows.

Feedback English (US) © 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use Cookie preferences

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).

The image shows two screenshots. The top screenshot is the Amazon SageMaker console, displaying the 'Notebook instances' page. A table lists one instance named 'BertClassification' with type 'ml.t3.medium', created on 'Jul 03, 2021 13:28 UTC', and status 'InService'. The bottom screenshot is the JupyterLab interface, showing a file browser with a list of files and folders: 'export', 'keras_model', 'Bert.ipynb', 'BERT_With_Keras_tensorflow.ipynb', 'Copy_of_BERT_Fine_Tuning_Sentence_Classification_v4.ipynb', and 'Twitter.zip'. The 'Bert.ipynb' file is highlighted, indicating it is the active notebook.

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`

```
1 FROM public.ecr.aws/lambda/python:3.6
2 #FROM python:3.6
3 # RUN dnf install gcc-c++ python3-devel
4 # Install OS packages for Pillow-SIMD
5 RUN pip install --upgrade pip
6
7 COPY requirement.txt ./
8 RUN pip install -r requirement.txt
9
10 COPY . .
11 # RUN ls -la /Model/*
12
13 COPY lambda.py ${LAMBDA_TASK_ROOT}
14 CMD ["lambda.handler"]
15
16
```

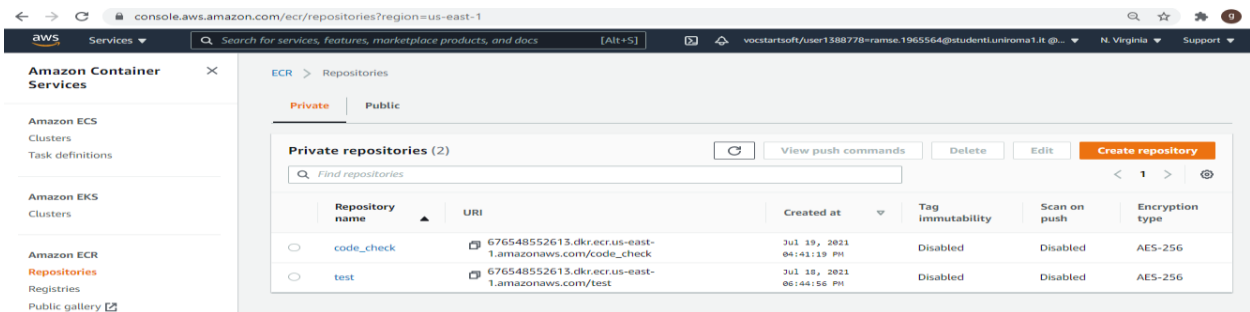
ECR

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 | 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
652d3dcccfc0e: 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
652d3dcccfc0e: 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.

The screenshot shows the AWS Lambda console interface. At the top, the URL is `console.aws.amazon.com/lambda/home?region=us-east-1#/functions/aws?tab=testing`. The main content area displays the execution details for a function named `aws`. The execution result is `null`. Below this, a 'Summary' section provides key metrics: Code SHA-256 (`0da33c125d80614c97dc93274c63b965db0e575eca05d2d869851fa3ab3881ab`), Request ID (`5092ea4a-a754-42f0-affd-12ec21e8a2bf`), Init duration (`1047.97 ms`), Duration (`1.37 ms`), Billed duration (`1050 ms`), Resources configured (`128 MB`), and Max memory used (`70 MB`). The 'Log output' section shows the following log entries: `START RequestId: 5092ea4a-a754-42f0-affd-12ec21e8a2bf Version: $LATEST`, `In function`, `Welcome to TechPrimers`, `Lambda execution Completed...`, `END RequestId: 5092ea4a-a754-42f0-affd-12ec21e8a2bf`, and a detailed report: `REPORT RequestId: 5092ea4a-a754-42f0-affd-12ec21e8a2bf Duration: 1.37 ms Billed Duration: 1050 ms Memory Size: 128 MB Max Memory Used: 70 MB Init Duration: 1047.97 ms`. At the bottom right, there is a watermark that says 'Activate Windows Go to Settings to activate Windows'.

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).

IAM users (6) Info								
An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.								
<input type="text" value="Search"/>								
<input type="checkbox"/>	User name	Groups	Last activity	MFA	Console last sign-in	Access key age	Access key last used	
<input type="checkbox"/>	Administrator	1	Never	None	None	-	-	
<input type="checkbox"/>	Fayez	1	Never	None	None	-	-	
<input type="checkbox"/>	Gaurav	1	Never	None	None	-	-	
<input type="checkbox"/>	Rocco	2	Never	None	None	-	-	
<input type="checkbox"/>	Tran	1	Never	None	None	-	-	

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.

Alarms (3)						
<input type="checkbox"/> Hide Auto Scaling alarms		Clear selection		Create composite alarm	Actions	Create alarm
Search		OK	Any type	< 1 >		
<input type="checkbox"/>	Name	State	Last state update	Conditions	Actions	
<input type="checkbox"/>	LambdaFunctionConcurrentExecutions	OK	2021-07-19 12:37:01	ConcurrentExecutions > 100 for 1 datapoints within 5 minutes	1 action(s) enabled	

Alarms (3)						
<input type="checkbox"/> Hide Auto Scaling alarms		Clear selection		Create composite alarm	Actions	Create alarm
Search		In alarm	Any type	< 1 >		
<input type="checkbox"/>	Name	State	Last state update	Conditions	Actions	
<input type="checkbox"/>	fooGetInvocation	In alarm	2021-07-19 12:56:11	Invocations > 0.1 for 1 datapoints within 1 minute	1 action(s) enabled	
<input type="checkbox"/>	fooInvocation	In alarm	2021-07-19 12:55:51	Invocations > 0.1 for 1 datapoints within 1 minute	1 action(s) enabled	

SNS

ALARM: "fooInvocation" in US East (N. Virginia)



AWS Notifications <no-reply@sns.amazonaws.com>
A loconte.1946073@studenti.uniroma1.it

Rispondi Rispondi a tutti Inoltra ...
lunedì 19/07/2021 12:56

You are receiving this email because your Amazon CloudWatch Alarm "fooInvocation" in the US East (N. Virginia) region has entered the ALARM state, because "Threshold Crossed: 1 out of the last 1 datapoints [1.0 (19/07/21 10:54:00)] was greater than the threshold (0.1) (minimum 1 datapoint for OK -> ALARM transition)." at "Monday 19 July, 2021 10:55:51 UTC".

View this alarm in the AWS Management Console:

<https://us-east-1.console.aws.amazon.com/cloudwatch/deeplink.js?region=us-east-1#alarmsV2:alarm/fooInvocation>

Alarm Details:

- Name: fooInvocation
- Description:
- State Change: INSUFFICIENT_DATA -> ALARM
- Reason for State Change: Threshold Crossed: 1 out of the last 1 datapoints [1.0 (19/07/21 10:54:00)] was greater than the threshold (0.1) (minimum 1 datapoint for OK -> ALARM transition).
- Timestamp: Monday 19 July, 2021 10:55:51 UTC
- AWS Account: 703855255519
- Alarm Arn: arn:aws:cloudwatch:us-east-1:703855255519:alarm:fooInvocation

Threshold:

- The alarm is in the ALARM state when the metric is GreaterThanThreshold 0.1 for 60 seconds.

Monitored Metric:

- MetricNamespace: AWS/Lambda
- MetricName: Invocations
- Dimensions: [FunctionName = foo]
- Period: 60 seconds
- Statistic: Average
- Unit: not specified
- TreatMissingData: missing

State Change Actions:

- OK:
- ALARM: [arn:aws:sns:us-east-1:703855255519:EmailTopic]
- INSUFFICIENT_DATA:

--

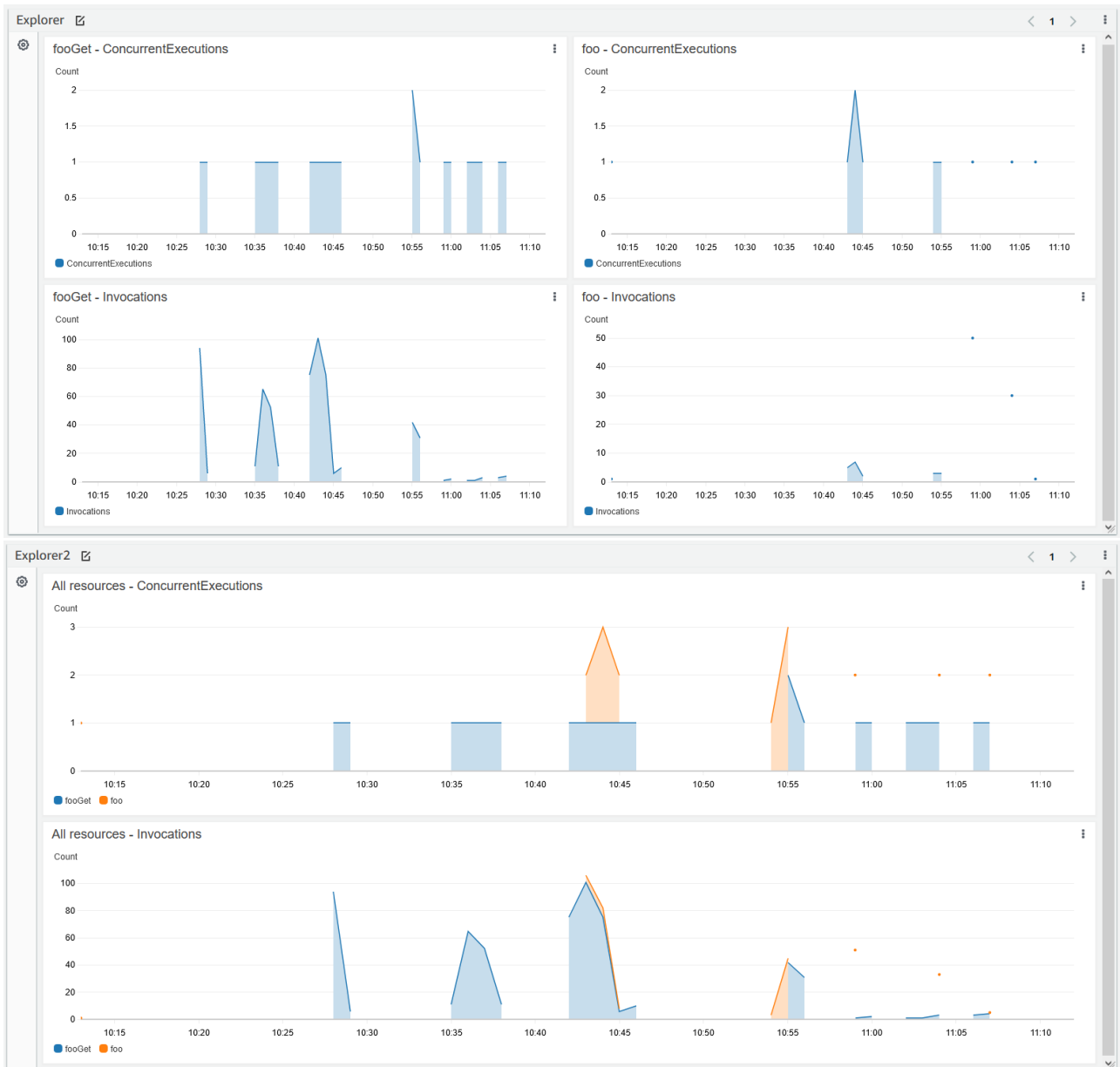
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:

<https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:703855255519:EmailTopic:cdbca41f-0ee1-4fc9-ba93-e4cf061c525b&Endpoint=loconte.1946073@studenti.uniroma1.it>

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at <https://aws.amazon.com/support>

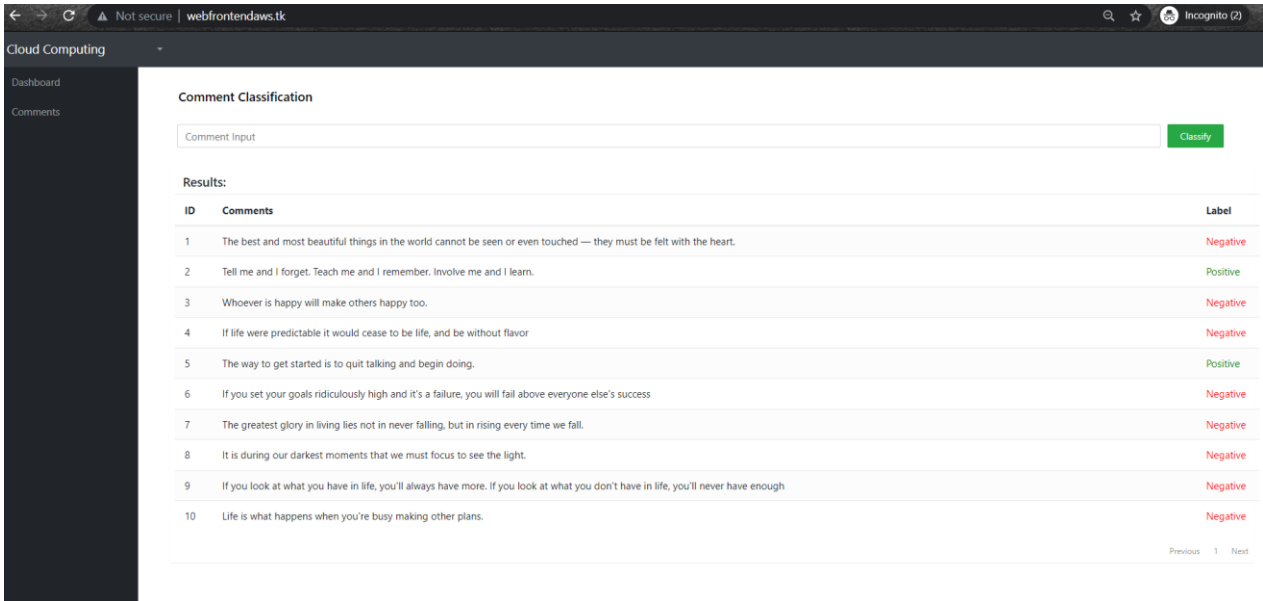
An Auto Scaling policy is set on the concurrency of a Lambda function. When a threshold is crossed (minimum is 100), the function accesses a reserved concurrency. When also the reserved concurrency is full taken, new concurrency is provided based on pricing (pay for the number of new concurrency provided).

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 also 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 X load_GET.py X
load_POST.py
1 import requests, time, json
2
3 iterations = 30
4
5 url = 'https://5eef05bzy6.execute-api.us-east-1.amazonaws.com/api/foo'
6 myobj = {'content': 1}
7
8
9
10 for i in range(iterations):
11     myobj['content'] = "hello" + str(i)
12     x = requests.post(url, data = json.dumps(myobj))
13     print(x.status_code, i)
14     #time.sleep(0.3)
15
load_POST.py load_GET.py X
load_GET.py
1 import requests
2
3 url = 'https://5eef05bzy6.execute-api.us-east-1.amazonaws.com/api/fooGet'
4 iterations = 30
5
6
7 for i in range(iterations):
8     x = requests.get(url)
9     print(x.status_code, i)
10
```

Our results of testing are shown in the Monitoring section.