DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

February - May 2023

Report on

Master of Technology
in
Computer Science & Engineering

Fundamentals of Scalable Computing Project

FACIAL RECOGNITION SERVICE USING AWS REKOGNITION AI SERVICE

Submitted by:

AISHWARYA GIRISH MENON	**PES1PG22CS002**
ADITHI S	**PES1PG22CS001**
SHRESTA L	**PES1PG22CS048**

1st Semester M.Tech

INTRODUCTION

Facial recognition is a technology feature that enables computers to recognize and identify individuals based on their facial features. This technology has numerous applications, ranging from security and surveillance to marketing and personalization. Amazon Web Services (AWS) offers a powerful facial recognition AI service called AWS Rekognition, that can build sophisticated facial recognition models. In this project, we will be using AWS Rekognition service to create a working AI model that can perform facial recognition on input images and classify the individuals with their respective class label names. This model uses Convolutional neural networks (Deep Learning Models) to analyze facial features such as the distance between the eyes, the shape of the nose and the mouth, and the contours of the face, to identify individuals in the input images.

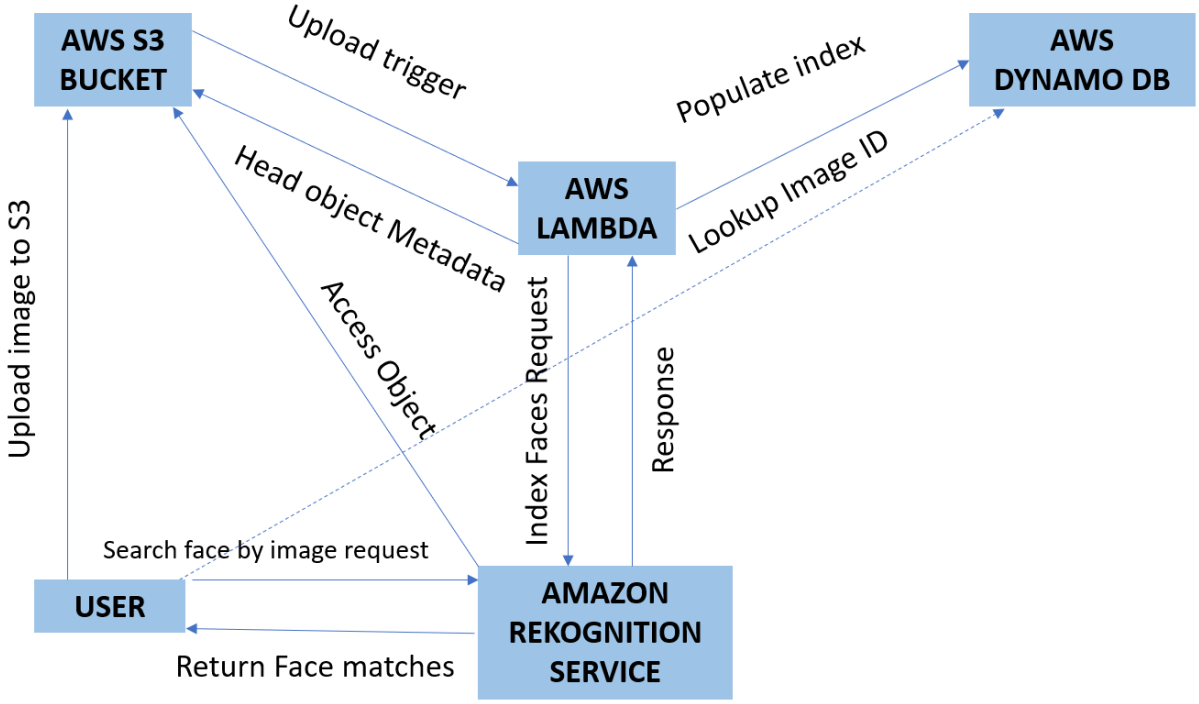
PROBLEM STATEMENT

Create a working AI model using Amazon Web Services (AWS) Facial Rekognition AI Service to perform Facial recognition on input images and classify the individual with their respective names.

OBJECTIVES

- To train and test a working AI model that recognizes facial images fed to it.
- Accurately identify and verify individuals based on their facial features. This can be useful in a variety of settings, such as security systems, access control, and surveillance.
- To learn how to use AWS services for building and deploying AI models.
- To create a web service that can take in input images, perform facial recognition, and return the names of the individuals in the images in real-time using AWS Rekognition.
- To gain experience working with labeled and unlabeled datasets and using machine learning algorithms to analyze and classify data.

AWS STACK COMPONENTS AND USES



COMPONENTS:

- Amazon S3 (Simple Storage Service) is a cloud-based object storage service that allows users to store and retrieve data from anywhere on the web. It provides a highly scalable, reliable, and cost-effective solution for storing and archiving data.
- Amazon Lambda is a serverless compute service that enables users to run code without provisioning or managing servers. It allows users to build and run applications without worrying about the infrastructure required to run them.
- Amazon DynamoDB is a NoSQL database service that provides fast and flexible storage for applications. It is a fully managed service that provides high availability and automatic scaling, making it an ideal choice for applications that require low latency and high performance.
- Amazon Rekognition is a cloud-based image and video analysis service that provides developers with the ability to add intelligent image and video analysis to their applications. It can be used to automatically detect objects, scenes, and faces in images and videos, as well as to analyze and classify visual content.

USES:

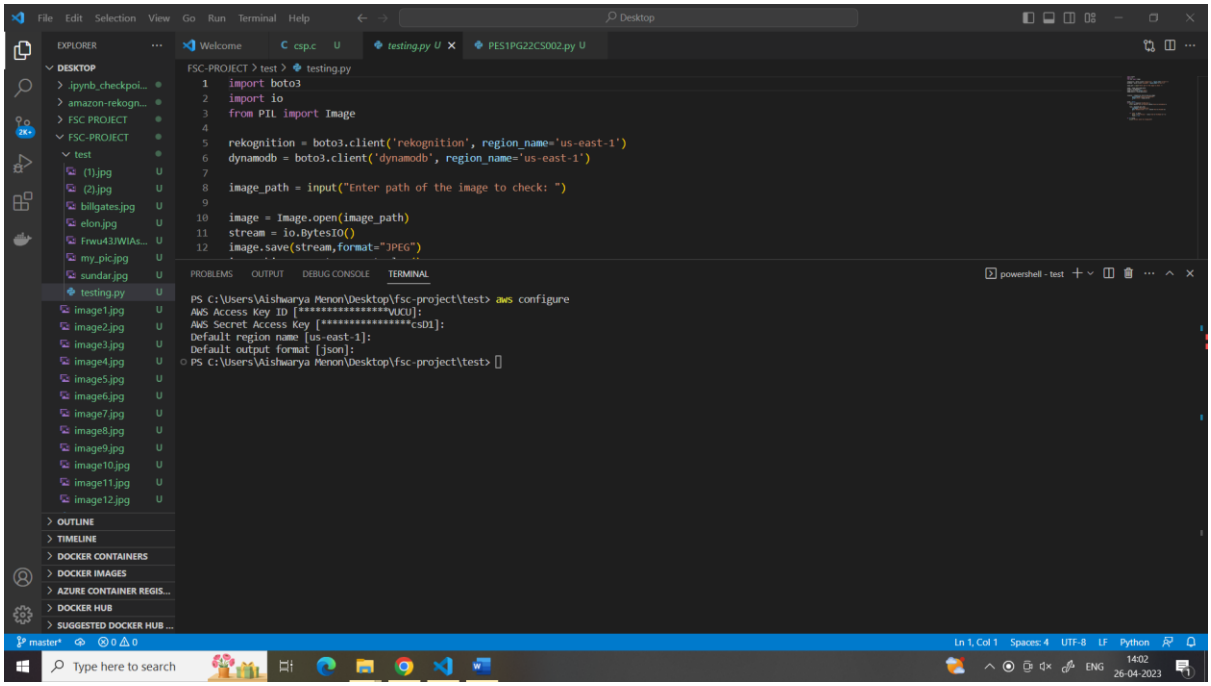
- An AWS stack can be used to define and manage the resources required for the project.
- Once the stack is created, the resources can be managed as a single unit, which simplifies the deployment and management of the resources required for the project.
- Using an AWS stack ensures consistency and scalability of the resources, reduces the risk of configuration errors, and can make the deployment process more efficient.

PROCEDURE

Amazon Rekognition automates the image and video analysis with machine learning. This project uses AWS S3, AWS Lambda, AWS DynamoDB and AWS IAM with the AI Model.

Install aws shell on windows powershell using below command: -
>>pip install aws-shell

Configure aws to set access key, secret ID, region name and output format
>> aws configure

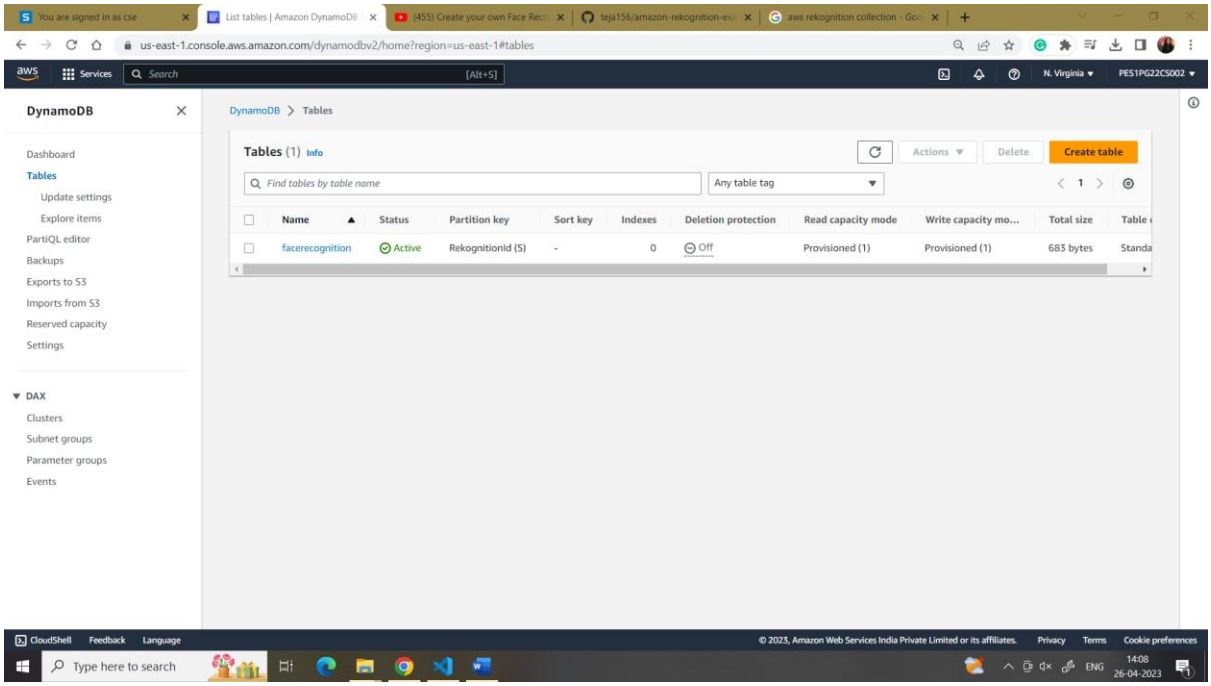


Create a collection in AWS Rekognition using below command. Amazon Rekognition can store information about detected faces in server-side containers known as collections. You can use the facial information that's stored in a collection to search for known faces in images, stored videos, and streaming videos.

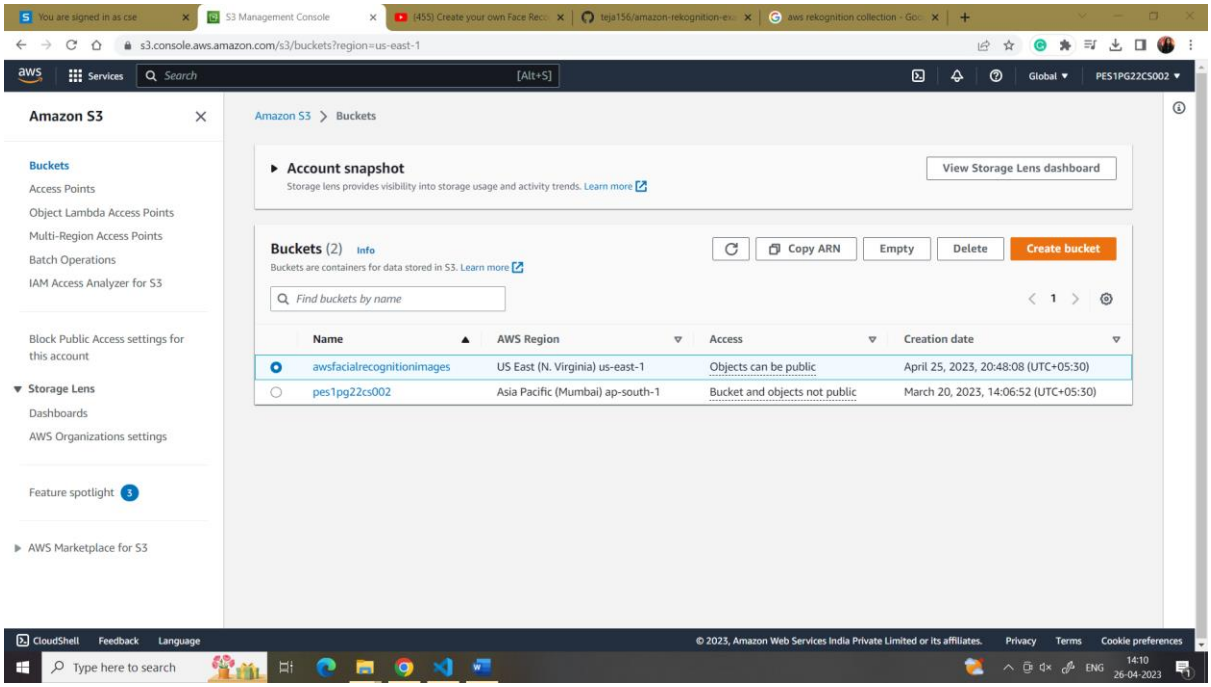
>> aws rekognition create-collection --collection-id awsfacialrecognition --region us-east-1

Create AWS DynamoDB table with name facerecognition to store the image metadata.

>>aws dynamodb create-table --table-name facerecognition --attribute-definitions AttributeName=RekognitionId,AttributeType=S --key-schema AttributeName=RekognitionId,KeyType=HASH --provisioned-throughput ReadCapacityUnits=1,WriteCapacityUnits=1 --region us-east-1



Create an AWS S3 bucket to store the images to be used for training the Amazon Rekognition model. Bucket name is awsfacialrecognitionimages.



Create IAM role to allow lambda function to access AWS services like DynamoDB and Amazon Rekognition. Set the policies required for the IAM role using the below JSON code.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents"
         ],
         "Resource": "arn:aws:logs:*:*:*"
      },
      {
         "Effect": "Allow",
         "Action": [
            "s3:GetObject"
         ],
```

```
            "Resource": [
                "arn:aws:s3:::awsfacialrecognitionimages/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:548342900319:table/facerecognition"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "rekognition:IndexFaces"
            ],
            "Resource": "*"
        }
    ]
}
```

This is an AWS IAM policy document in JSON format. It grants permissions to a user or a group of users to perform specific actions on AWS resources.

The policy consists of four statements, each with an "Effect" (Allow or Deny), a list of "Action" permissions, and a list of "Resource" ARNs (Amazon Resource Names) specifying the AWS resources to which the permissions apply.

Here's a breakdown of each statement:

Statement 1:

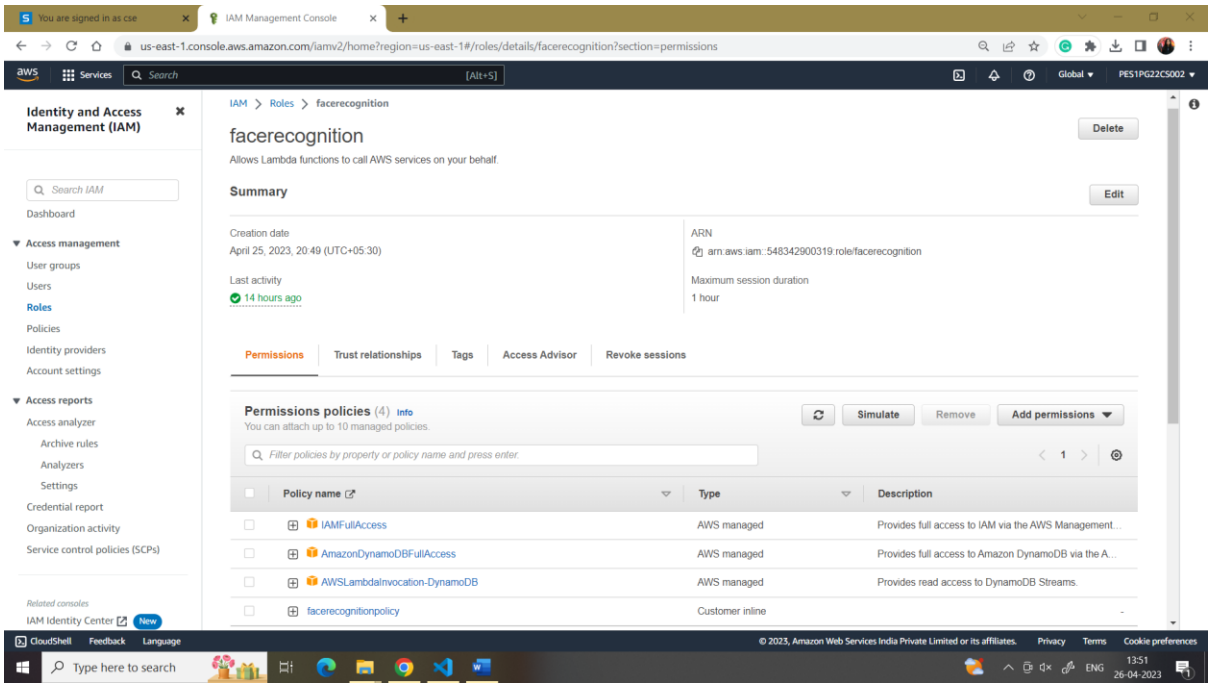
Effect: Allow
Actions: logs:CreateLogGroup, logs:CreateLogStream, logs:PutLogEvents
Resource: arn:aws:logs:::*
This statement allows the user/group to create log groups, log streams, and put log events in all AWS regions and accounts.

Statement 2:

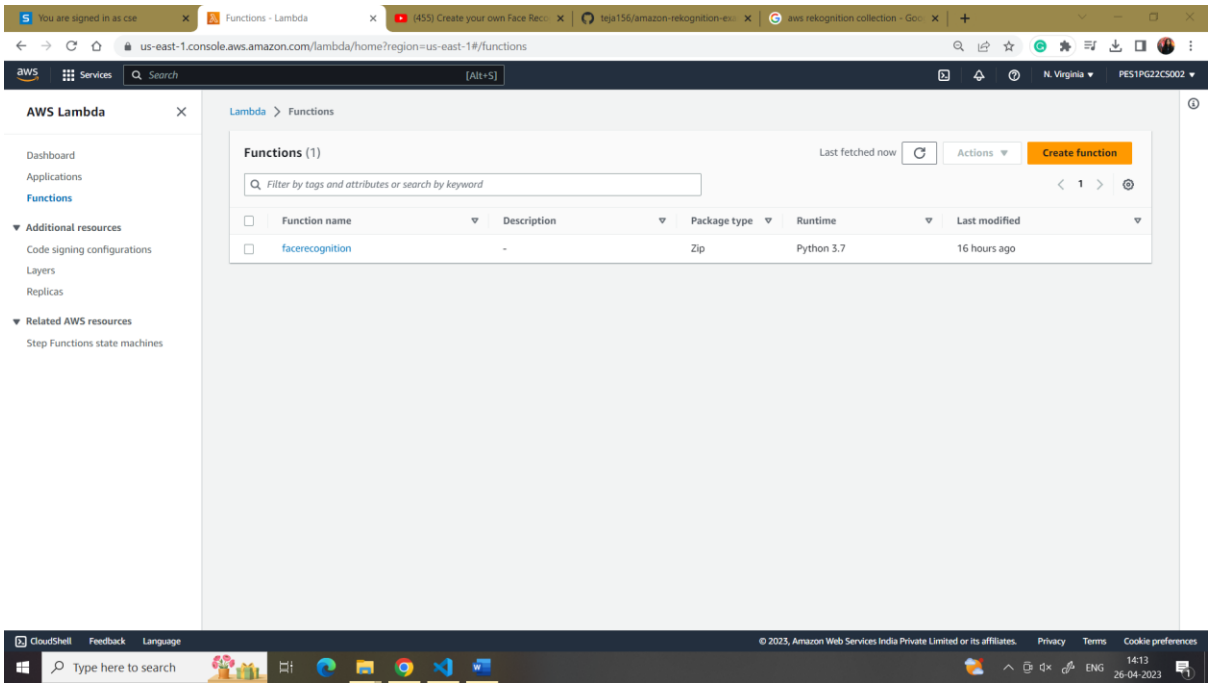
Effect: Allow
Actions: s3:GetObject
Resource: arn:aws:s3:::awsfacialrecognitionimages/*
This statement allows the user/group to get objects (i.e., files) from an S3 bucket named "awsfacialrecognitionimages". The asterisk (*) at the end of the ARN allows access to all objects in the bucket.
Statement 3:

Effect: Allow
Actions: dynamodb:PutItem
Resource: arn:aws:dynamodb:us-east-1:548342900319:table/facerecognition
This statement allows the user/group to put (i.e., insert) items into a DynamoDB table named "facerecognition" in the US East (N. Virginia) region.
Statement 4:

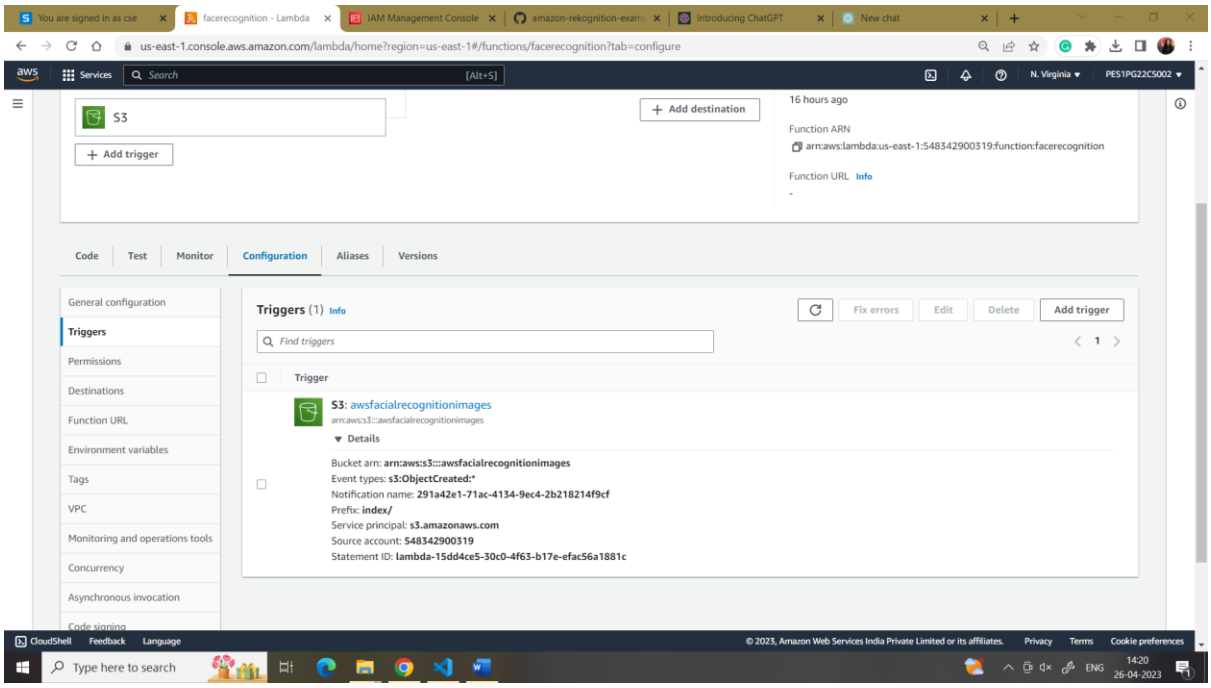
Effect: Allow
Actions: rekognition:IndexFaces
Resource: *
This statement allows the user/group to index faces using Amazon Rekognition, a deep learning-based image and video analysis service. The asterisk (*) in the Resource field allows access to all resources.



Create a lambda function that gets triggered whenever S3 bucket is inserted with images. The lambda function will populate the DynamoDB with the metadata of images. The lambda function will populate the DynamoDB with the metadata and faceprint of the images.The faceprint uniquely identifies each image.



Set the Trigger for the lambda function to be called when new images are pushed to s3 bucket.



The lambda function performs the training of the AI model using AWS Rekognition function that takes the input as images and metadata from the S3 bucket. The faceprint obtained for each image is populated to the index of DynamoDB. The DynamoDB data is also linked to the collection on server side through the attribute RekognitionID. Write the code for the lambda function and deploy the code
.
This code is a Python script that is intended to be executed in AWS Lambda, a serverless compute service. The script uses Amazon Web Services (AWS) APIs to detect faces in images and store information about the faces in a DynamoDB database. The script first imports necessary libraries and initializes AWS clients for Amazon DynamoDB, Amazon S3, and Amazon Rekognition. The helper functions are defined next. The index_faces function uses the rekognition.index_faces API to detect faces in an image stored in an S3 bucket and add them to a collection. The update_index function adds metadata to the facerecognition DynamoDB table for each detected face, including the faceId and the fullName.

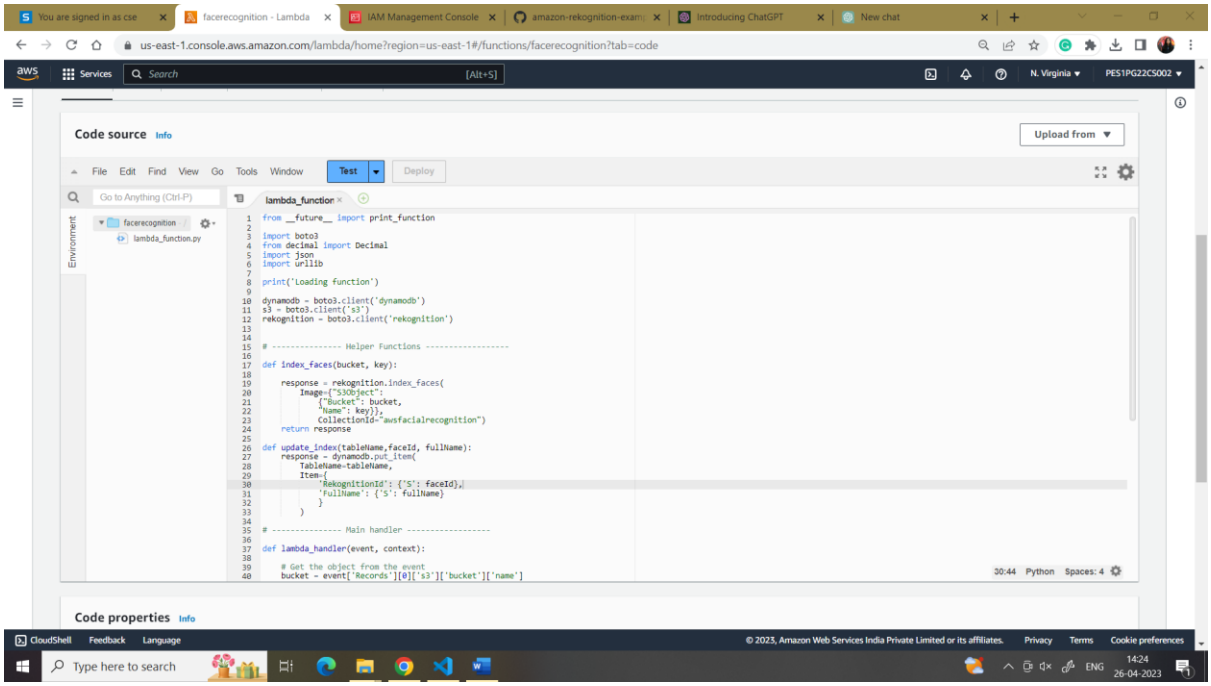
The lambda_handler function is the main entry point for the Lambda function. It is executed when an object is created or updated in an S3 bucket. The function first extracts the bucket and key values from the Lambda event object. It then calls the index_faces function to detect faces in the image and adds them to a collection. If the API call is successful, it extracts the faceId and fullName metadata for the face from the S3 object and calls the update_index function to add the metadata to the DynamoDB table. If any exceptions are raised during the execution of the function, they are caught and re-raised along with an error message. The function also logs various messages to the console using the print function.

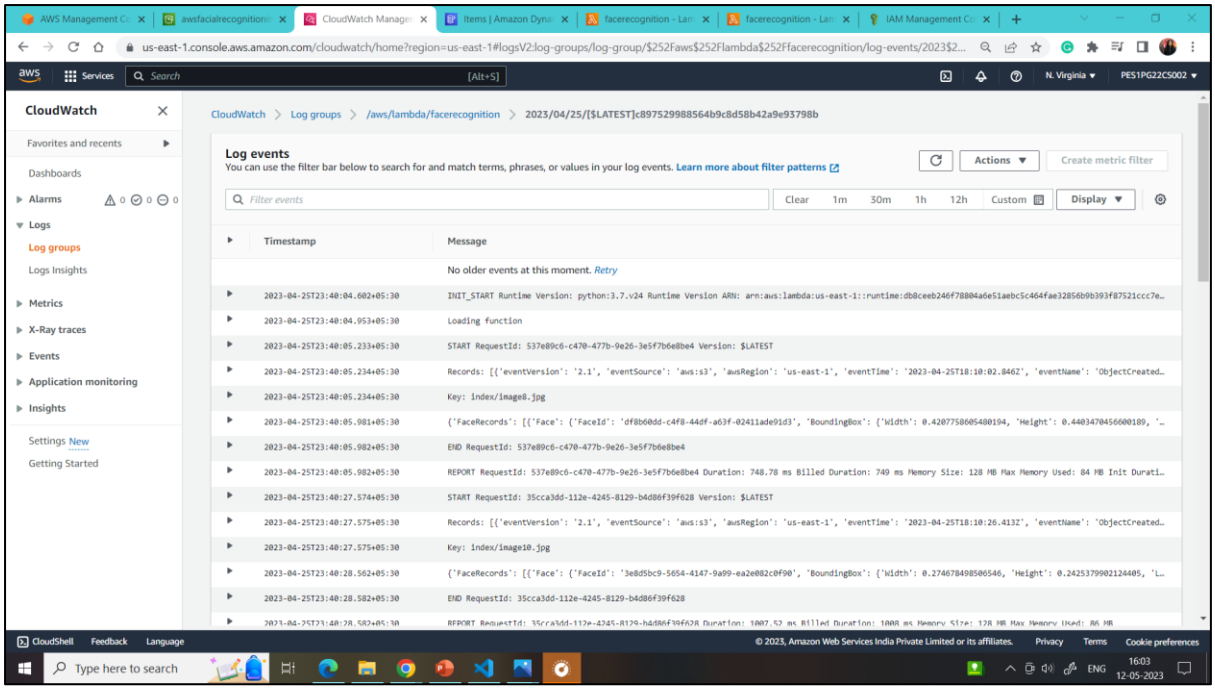
lambdafunction.py

```
from __future__ import print_function

import boto3
from decimal import Decimal
import json
import urllib

print('Loading function')

dynamodb = boto3.client('dynamodb')
s3 = boto3.client('s3')
```

```
rekognition = boto3.client('rekognition')


# ---------------- Helper Functions ------------------

def index_faces(bucket, key):

    response = rekognition.index_faces(
        Image={"S3Object":
            {"Bucket": bucket,
            "Name": key}},
            CollectionId="awsfacialrecognition")
    return response

def update_index(tableName,faceId, fullName):
    response = dynamodb.put_item(
        TableName=tableName,
        Item={
            'RekognitionId': {'S': faceId},
            'FullName': {'S': fullName}
            }
        )

# ---------------- Main handler ------------------

def lambda_handler(event, context):

    # Get the object from the event
    bucket = event['Records'][0]['s3']['bucket']['name']
    print("Records: ",event['Records'])
    key = event['Records'][0]['s3']['object']['key']
    print("Key: ",key)
    # key = key.encode()
    # key = urllib.parse.unquote_plus(key)

    try:

        # Calls Amazon Rekognition IndexFaces API to detect faces in S3 object
        # to index faces into specified collection

        response = index_faces(bucket, key)

        # Commit faceId and full name object metadata to DynamoDB
```

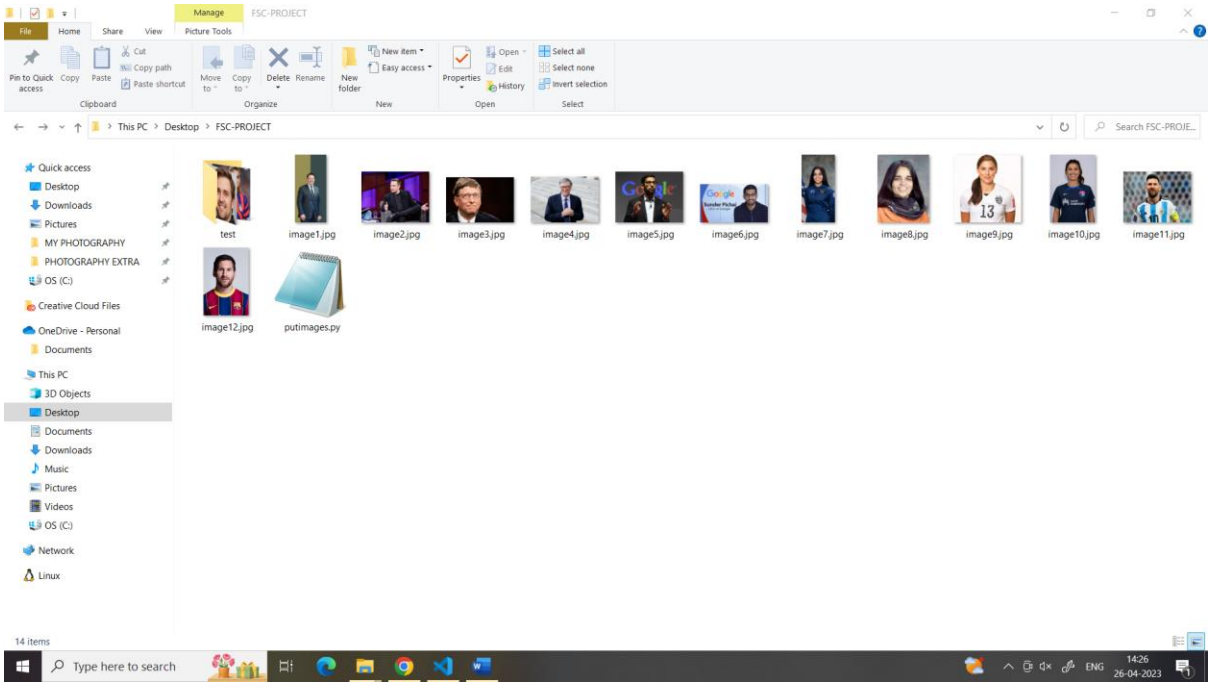
```
    if response['ResponseMetadata']['HTTPStatusCode'] == 200:
        faceId = response['FaceRecords'][0]['Face']['FaceId']

        ret = s3.head_object(Bucket=bucket,Key=key)
        personFullName = ret['Metadata']['fullname']

        update_index('facerecognition',faceId,personFullName)

    # Print response to console
    print(response)

    return response
except Exception as e:
    print(e)
    print("Error processing object {} from bucket {}. ".format(key, bucket))
    raise e
```





AWS Cloudwatch holds all the logs from training process of AWS Rekognition

Now collect the dataset required for training and testing the model. In this case the dataset has images of 6 different persons with two images to train for each person.



Run the below python code to push the images to S3 bucket with the respective metadata that will be used as label to classify the individuals in the images.

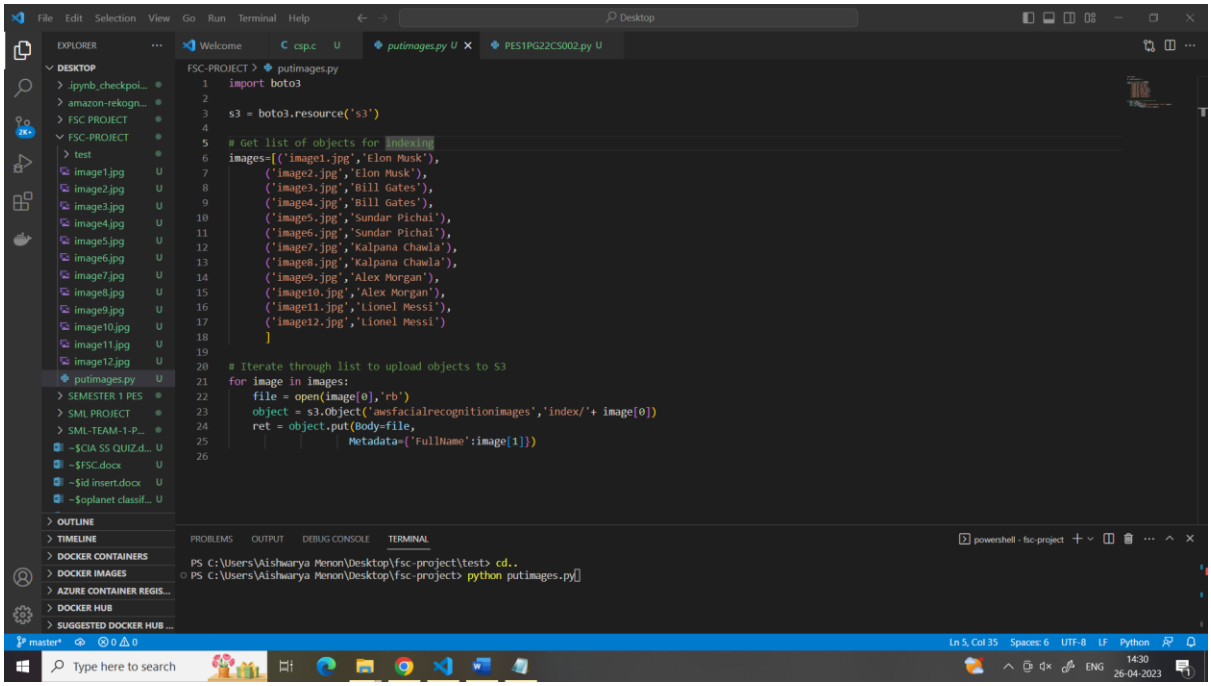
putimages.py

```
import boto3

s3 = boto3.resource('s3')

# Get list of objects for indexing
images=[('image1.jpg','Elon Musk'),
      ('image2.jpg','Elon Musk'),
      ('image3.jpg','Bill Gates'),
      ('image4.jpg','Bill Gates'),
      ('image5.jpg','Sundar Pichai'),
      ('image6.jpg','Sundar Pichai'),
      ('image7.jpg','Kalpana Chawla'),
      ('image8.jpg','Kalpana Chawla'),
      ('image9.jpg','Alex Morgan'),
      ('image10.jpg','Alex Morgan'),
      ('image11.jpg','Lionel Messi'),
      ('image12.jpg','Lionel Messi')
      ]

# Iterate through list to upload objects to S3
for image in images:
   file = open(image[0],'rb')
   object = s3.Object('awsfacialrecognitionimages','index/'+ image[0])
   ret = object.put(Body=file,
                    Metadata={'FullName':image[1]})
```

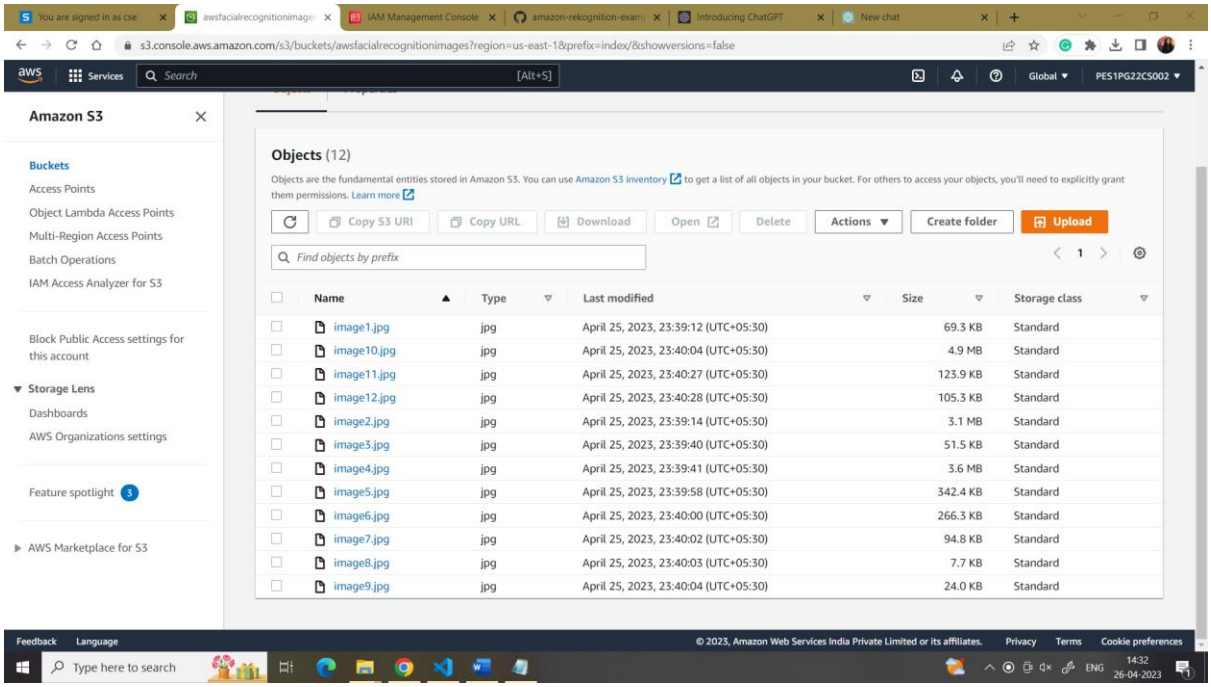
The code above uses the Boto3 Python library to interact with the Amazon S3 (Simple Storage Service) to upload images and metadata. The metadata acts as the label for supervised training of the AI model using the images.

First, the code imports the Boto3 library and creates an S3 resource object. The resource object provides an interface to interact with Amazon S3 using Python code. Next, a list of tuples called "images" is created, which contains the file names and corresponding names of people in the images.The code then iterates through each tuple in the "images" list and opens the corresponding image file in read-only binary (rb) mode. It then creates an S3 object with the file's name prefixed with "index/" under a bucket named "awsfacialrecognitionimages".

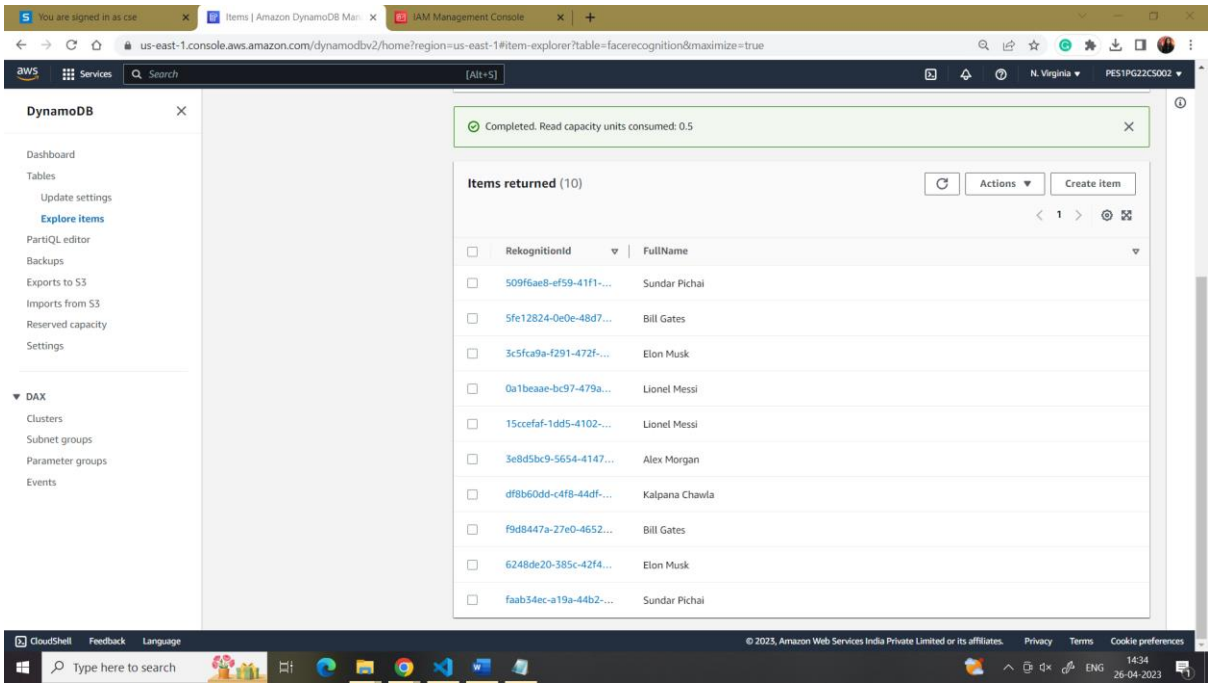
The "object" variable is then used to upload the file to S3 using the "put" method. The "put" method takes the file object as "Body" and adds the person's full name as a "Metadata" key-value pair. The metadata can be used to store additional information about the object. By uploading images to S3 with metadata, we can use Amazon Rekognition, a deep learning-based image and video analysis service, to index faces and search for faces that match a given image or person's name. This can be useful for various applications, such as security systems, social media platforms, and e-commerce.



All images are now inserted to the S3 bucket with respective metadata.

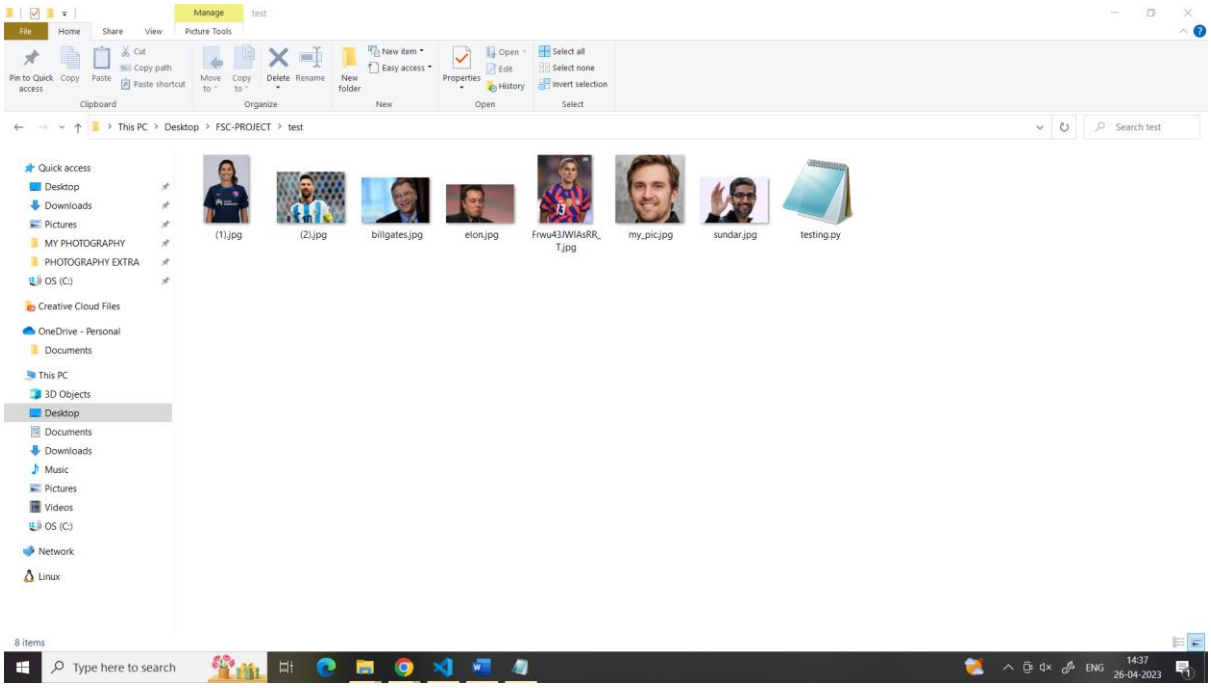


The metadata of each image is pushed into the table as well. This shows that the dataset was successfully passed to Amazon Rekognition service for training the ML model.



Now test the efficiency of the model to gain the resultant output using the below python code. The testing dataset consists of random images of the individuals

on which model was trained and of individuals whose faces were not fed to the model for training.



Testing.py

```
import boto3
import io
from PIL import Image

rekognition = boto3.client('rekognition', region_name='us-east-1')
dynamodb = boto3.client('dynamodb', region_name='us-east-1')

image_path = input("Enter path of the image to check: ")

image = Image.open(image_path)
stream = io.BytesIO()
image.save(stream,format="JPEG")
image_binary = stream.getvalue()


response = rekognition.search_faces_by_image(
        CollectionId='awsfacialrecognition',
        Image={'Bytes':image_binary}
        )

found = False
```

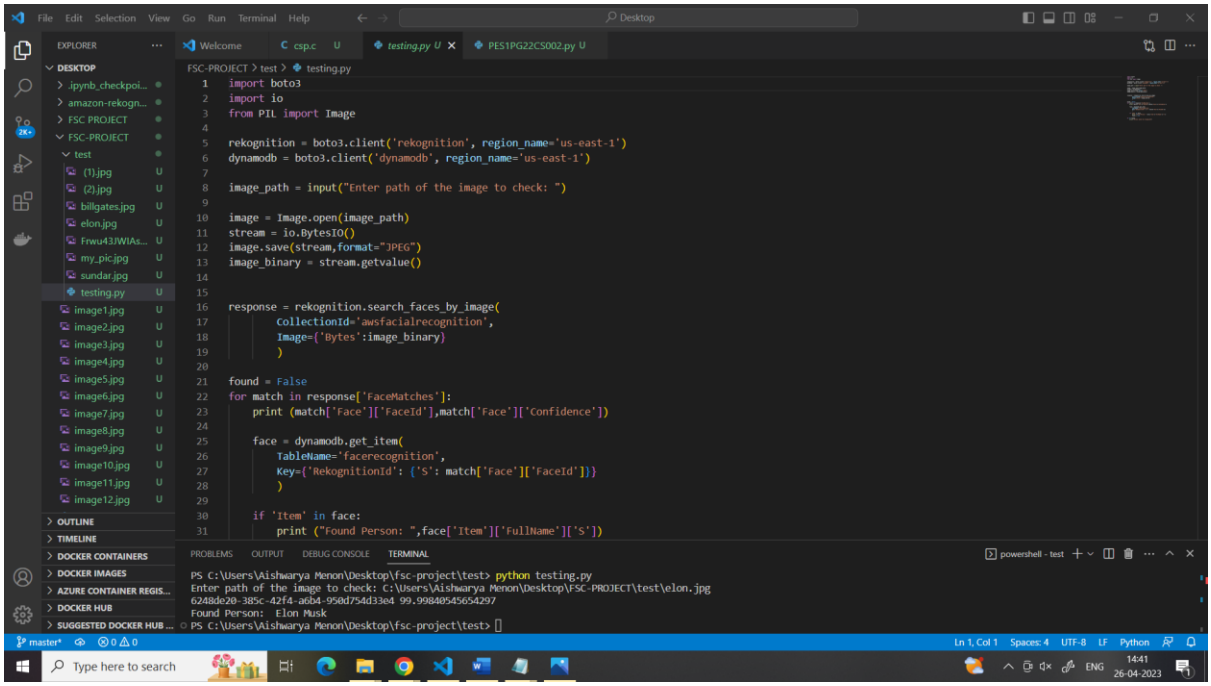
```
for match in response['FaceMatches']:
    print (match['Face']['FaceId'],match['Face']['Confidence'])

    face = dynamodb.get_item(
        TableName='facerecognition',
        Key={'RekognitionId': {'S': match['Face']['FaceId']}}
        )

    if 'Item' in face:
        print ("Found Person: ",face['Item']['FullName']['S'])
        found = True

if not found:
    print("Person cannot be recognized")
```

The code is a Python script that uses the Boto3 library to integrate with Amazon Web Services (AWS) Rekognition and DynamoDB services. The script performs facial recognition on an input image and searches for a match in a collection of faces stored in AWS Rekognition. If a match is found, it retrieves the associated name from a DynamoDB table and outputs it. If no match is found, it outputs a message indicating that the person cannot be recognized.

The script begins by importing the necessary libraries: Boto3, io, and PIL (Python Imaging Library). It then creates Boto3 clients for Rekognition and DynamoDB services in the us-east-1 region. The user is prompted to enter the path of the input image, which is then opened using the PIL Image.open() method. The image is then converted to binary format using io.BytesIO() and saved in a stream object. The binary image data is obtained using the stream.getvalue() method.

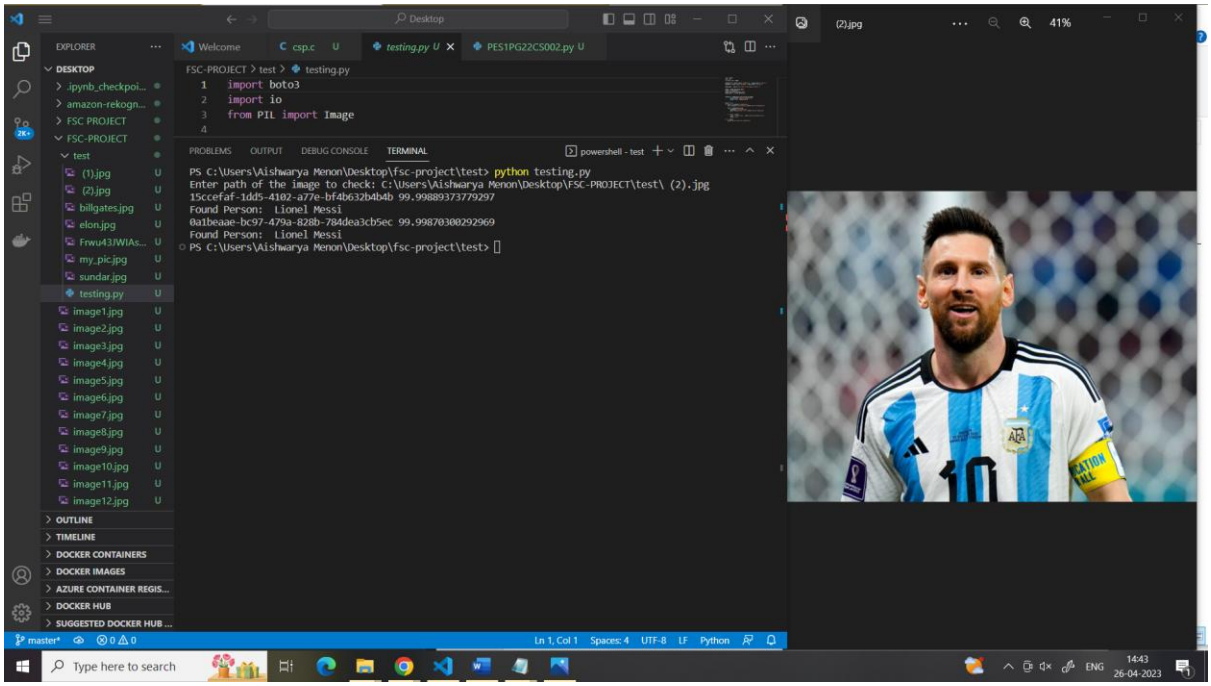
The script then calls the search_faces_by_image() method of the Rekognition client to search for faces in the "awsfacialrecognition" collection that match the input image. If a match is found, the script retrieves the face ID and confidence score from the response and prints them. It then calls the get_item() method of the DynamoDB client to retrieve the associated name from the "facerecognition" table using the face ID as the primary key. If a match is found, the script prints the full name of the person and sets the found flag to True.

Finally, if the found flag is not set to True, the script prints a message indicating that the person cannot be recognized.

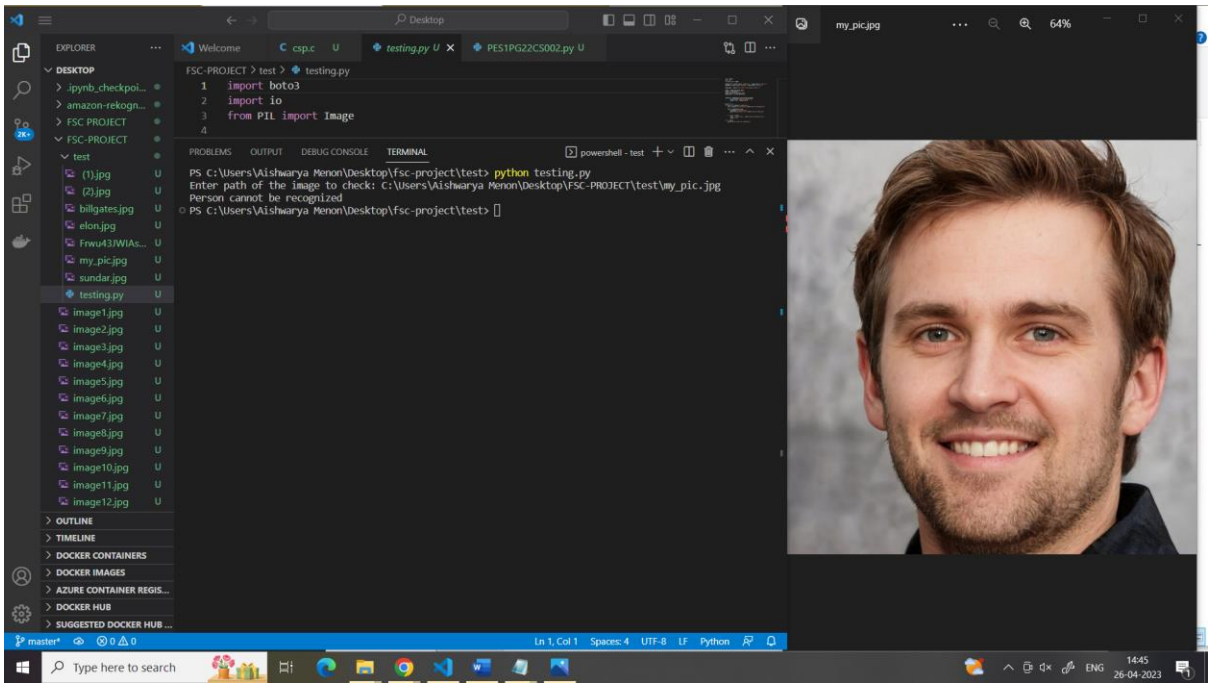

File Edit Selection View Go Run Terminal Help
Desktop
EXPLORER
DESKTOP
.ipynb_checkpoi...
amazon-rekogn...
FSC PROJECT
FSC-PROJECT
test
(1).jpg
(2).jpg
billgates.jpg
elon.jpg
Frwu43IWIAs...
my_pic.jpg
sundar.jpg
testing.py
image1.jpg
image2.jpg
image3.jpg
image4.jpg
image5.jpg
image6.jpg
image7.jpg
image8.jpg
image9.jpg
image10.jpg
image11.jpg
image12.jpg
OUTLINE
TIMELINE
DOCKER CONTAINERS
DOCKER IMAGES
AZURE CONTAINER REGIS...
DOCKER HUB
SUGGESTED DOCKER HUB ...
Welcome
csp.c U
testing.py U
PES1PG22CS002.py U
FSC-PROJECT > test > testing.py
import boto3
import io
from PIL import Image
rekognition = boto3.client('rekognition', region_name='us-east-1')
dynamodb = boto3.client('dynamodb', region_name='us-east-1')
image_path = input("Enter path of the image to check: ")
image = Image.open(image_path)
stream = io.BytesIO()
image.save(stream,format="JPEG")
image_binary = stream.getvalue()
response = rekognition.search_faces_by_image(
CollectionId='awsfacialrecognition',
Image={'Bytes':image_binary}
)
found = False
for match in response['FaceMatches']:
print (match['Face']['FaceId'],match['Face']['Confidence'])
face = dynamodb.get_item(
TableName='facerecognition',
Key={'RekognitionId': {'S': match['Face']['FaceId']}}
)
if 'Item' in face:
print ("Found Person: ",face['Item']['FullName']['S'])
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
powershell - test
PS C:\Users\Aishwarya Menon\Desktop\fsc-project\test> python testing.py
Enter path of the image to check: C:\Users\Aishwarya Menon\Desktop\FSC-PROJECT\test\elon.jpg
6248de20-385c-42f4-a6b4-950d754d33e4 99.99840545654297
Found Person: Elon Musk
PS C:\Users\Aishwarya Menon\Desktop\fsc-project\test>
master*
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python
Type here to search
ENG
14:41
26-04-2023

RESULTS

Person Recognised by Amazon Rekognition Model



Person Unrecognised by Amazon Rekognition Model



The python script Testing.py performs facial recognition on an input image and searches for a match in a collection of faces stored in AWS Rekognition.

When a match is found :

->The output will indicate that a match has been found.

->The output include information such as the name or ID of the matched individual from DynamoDB table

If no match is found :

->It outputs a message indicating that the person cannot be recognized.
The AI model has 99.9% accuracy rate

CONCLUSION

- AWS Rekognition has a reputation for being highly accurate in detecting and recognizing faces. In various benchmark tests, it has achieved high levels of accuracy, outperforming other commercial facial recognition services.

- The performance of AWS Rekognition in recognizing faces can be affected by various factors, including lighting conditions, image quality, and the size of the dataset. However, with proper configuration and optimization, the service can achieve fast and reliable performance.

- AWS Rekognition provides several features for facial recognition, including face detection, comparison, search, analysis, and recognition. These features can be used for various use cases, such as security and surveillance, user verification, and customer experience enhancement.

- AWS Rekognition can be easily integrated with other AWS services, such as Amazon S3 and Amazon Kinesis, as well as third-party applications through APIs. This makes it easier to build custom solutions and workflows that utilize facial recognition.

- This project uses AWS S3, DynamoDB, Rekognition AI, Lambda in sync with one another