PROJECT ENTITLED

Face Recognition Attendance Management System With AWS

Group Members:

Priyanka Korde

Aniket Singh

Kiran Watole

Project Guide:

Dheerendra Solanki

Table of Contents

INTRODUCTION

1.1) Introduction:

The Face Recognition Attendance Management System is software that tracks attendance using facial recognition technologies. It records attendance by capturing and analyzing facial features in real time, eliminating the need for manual registers.

The application shall capture attendance without any manual intervention. Developing a smart device that can be integrated with a camera that will capture the images of student and send the images to model. Then the model will use AWS Rekognition Service to recognize the student's faces & push the images to S3(Simple Storage Service) for storage and also updates the attendance automatically in a database After that, they will receive an email informing them that their attendance has been successfully marked. We created a web-based dashboard to view all of the students' attendance data.

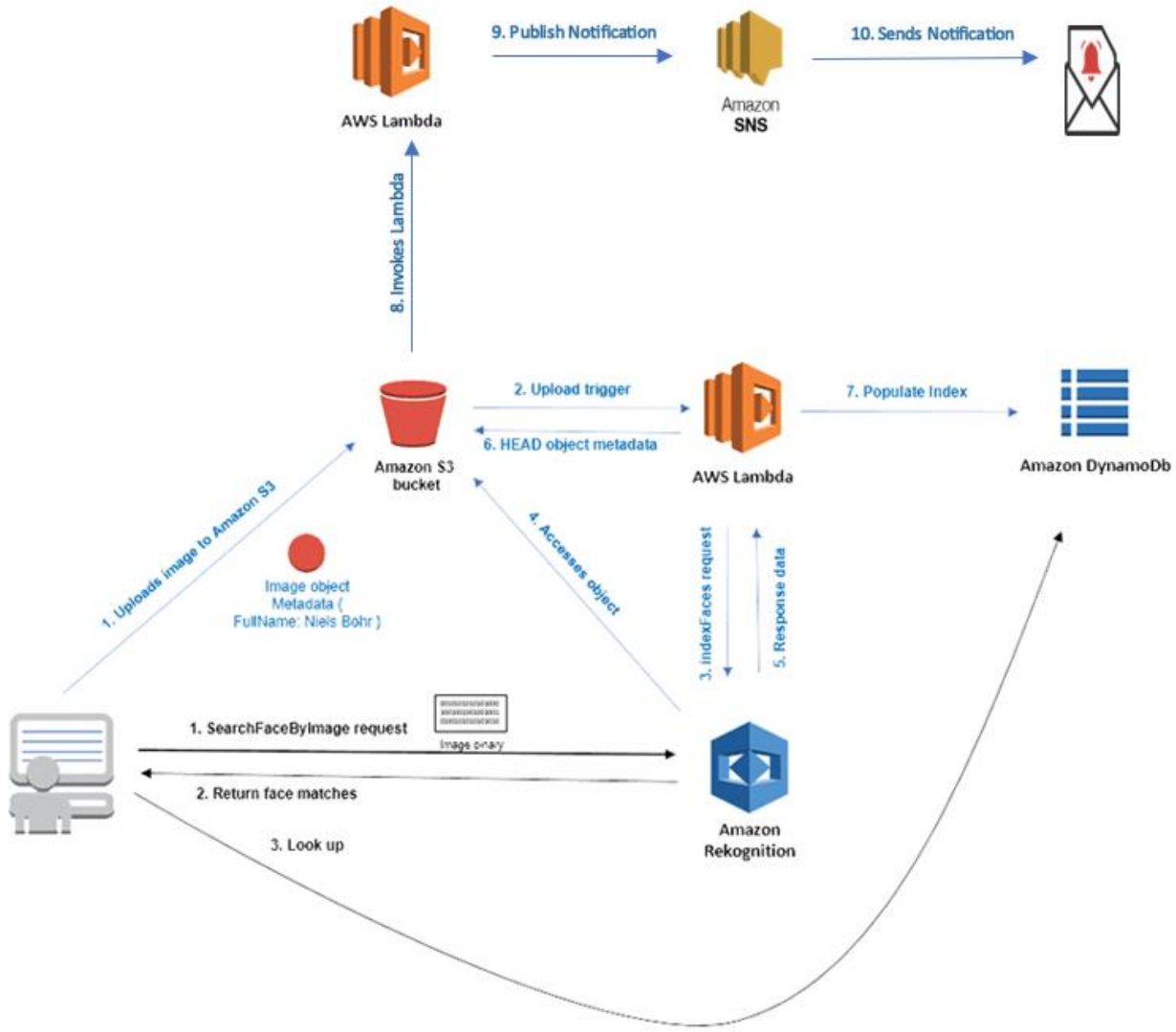
1.2) Problem Statement:

Traditional attendance systems sometimes suffer from inefficiencies caused by human entry and time-consuming verification processes. To overcome these issues, an automated and reliable attendance management system that utilizes face recognition technology is required.

The goal is to develop a face recognition attendance system capable of accurately identifying individuals in real-time, recording their attendance, and ensuring a seamless and fraud-resistant process.

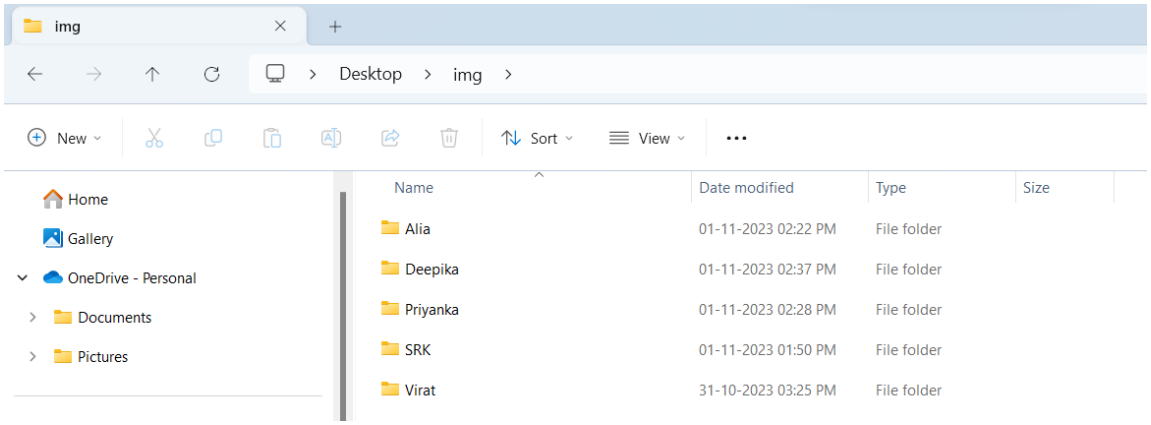
PROPOSED SYSTEM

2.1) System Architecture:



2.2) Datasets:

Dataset is created manually by collecting images of minimum 5 persons in our local system. We will take a minimum 50-60 images of faces for each person. If you take only 5-15 images for each person then it will not detect correctly, so you have to take at least 50 images for each person.



2.3) Requirements:

Software Requirements:

1. VSCode

Hardware Requirements:

1. RAM: 4 GB or above
2. Storage: 30 to 50 GB
3. Processor: Any Processor above 500MHz
4. Windows XP and above

2.4) AWS Services Used:

1. AWS Rekognition:

Amazon Rekognition makes it easy to add image and video analysis to your applications. You just have to provide an image or video to the Amazon Rekognition API, and the service can: Identify labels (objects, concepts, people, scenes, and activities) and text, Detect inappropriate content, Provide highly accurate facial analysis, face comparison, and face search capabilities

Amazon Rekognition is based on the same proven, highly scalable, deep learning technology developed by Amazon's computer vision scientists to analyze billions of images and videos daily. It requires no machine learning expertise to use. Amazon Rekognition includes a simple, easy-to-use API that can quickly analyze any image or video file that's stored in Amazon S3. It's always learning from new data, and we're continually adding new labels and facial comparison features to the service.

2. AWS Lambda

AWS Lambda is a compute service that lets you run code without provisioning or managing servers. Lambda runs your code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, and logging. With Lambda, all you need to do is supply your code in one of the language runtimes that Lambda supports.

3. API Gateway

You can create a web API with an HTTP endpoint for your Lambda function by using Amazon API Gateway. API Gateway provides tools for creating and documenting web APIs that route HTTP requests to Lambda functions. You can secure access to your API with authentication and authorization controls. Your APIs can serve traffic over the internet or can be accessible only within your VPC.

4. Amazon Dyanmo DB

Amazon DynamoDB is a fully managed, serverless, key-value NoSQL database designed to run high-performance applications at any scale. DynamoDB offers built-in security, continuous backups, automated multi-Region replication, in-memory caching, and data import and export tools.

5. Simple Storage Service (S3)

Amazon S3 is an object storage service that offers industry-leading scalability, data availability, security, and performance. Store and protect any amount of data for a range of use cases, such as data lakes, websites, cloud-native applications, backups, archive, machine learning, and analytics. Amazon S3 is designed for 99.999999999% (11 9's) of durability, and stores data for millions of customers all around the world.

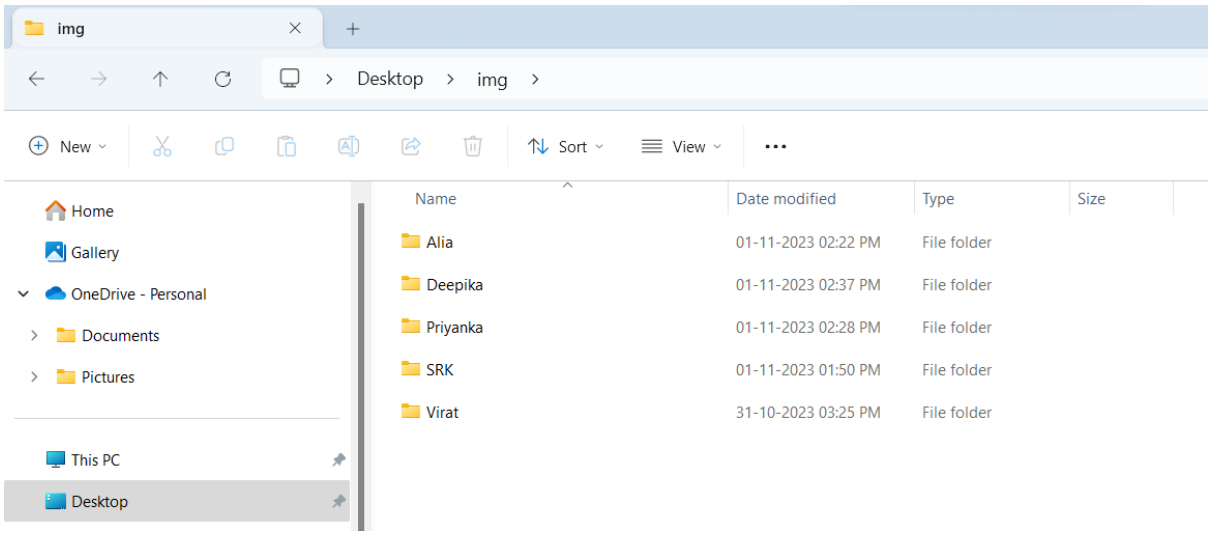
6. Simple Notification Service (SNS)

Amazon Simple Notification Service (Amazon SNS) sends notifications two ways, A2A and A2P. A2A provides high-throughput, push-based, many-to-many messaging between distributed systems, microservices, and event-driven serverless applications. These applications include Amazon Simple Queue Service (SQS), Amazon Kinesis Data Firehose, AWS Lambda, and other HTTPS endpoints. A2P functionality lets you send messages to your customers with SMS texts, push notifications, and email.

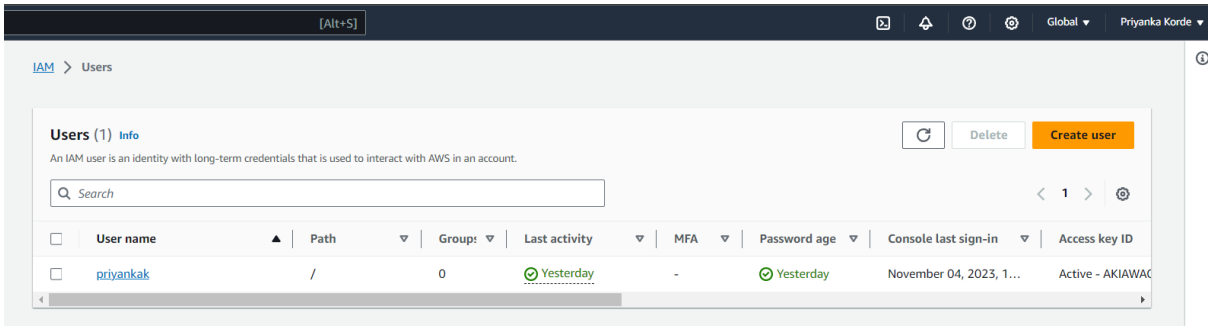
IMPLEMENTATION

3.1) Steps:

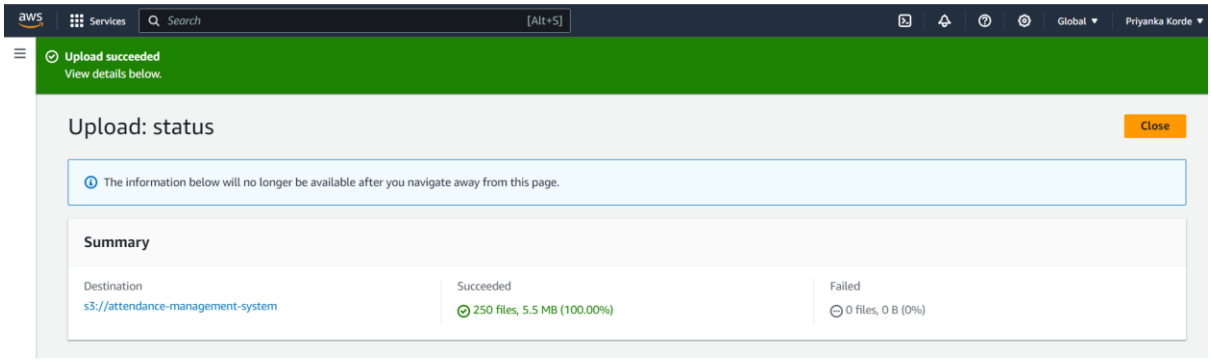
1) First we will create a training dataset in our local system. We will take a min 50-60 images of faces for each person.

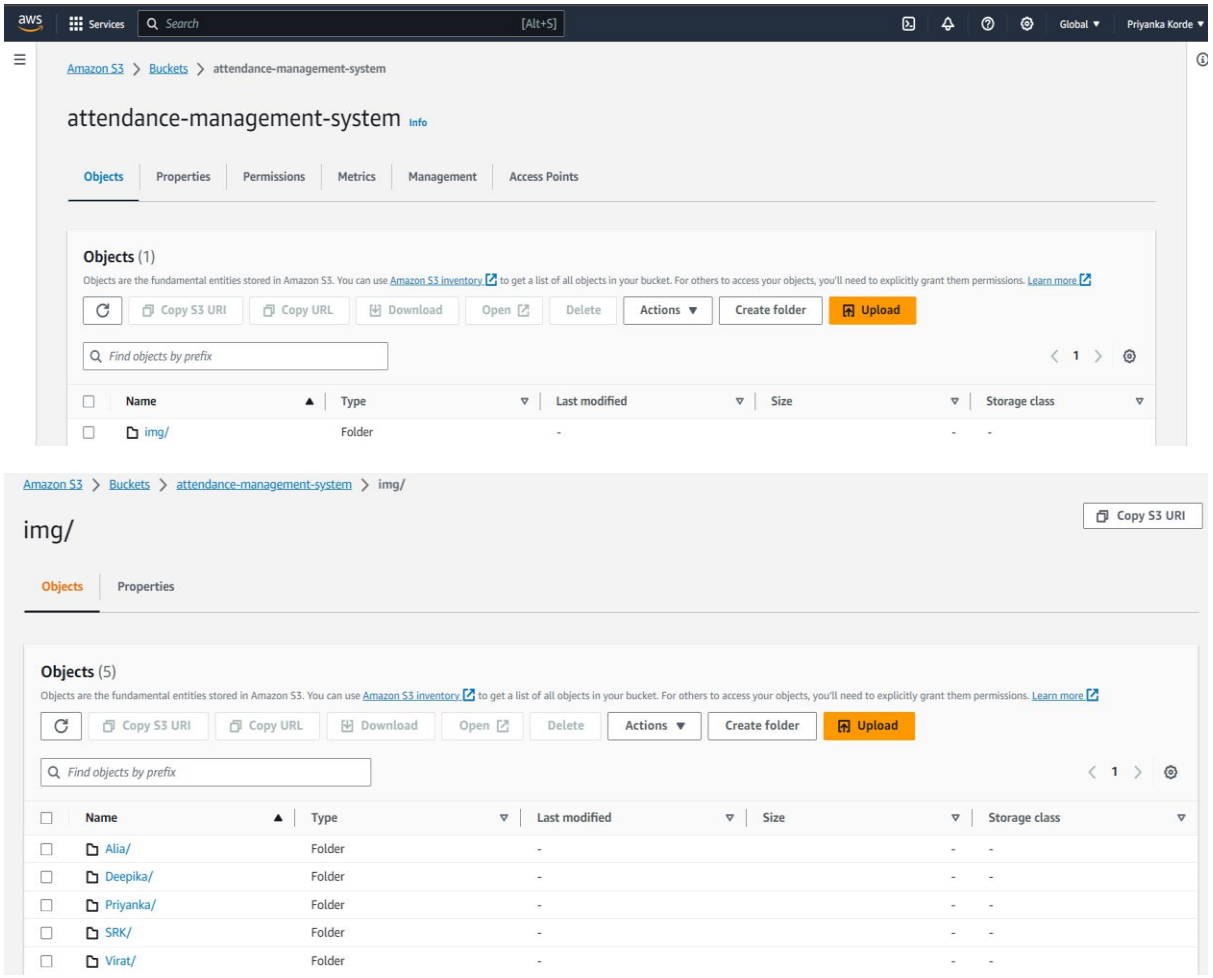


2) In AWS Account we will create one IAM User. And define policy → Do AWS Login using IAM User.



3) Then we will create S3 Bucket in AWS in Region you want → Upload folder of images of our system in S3.

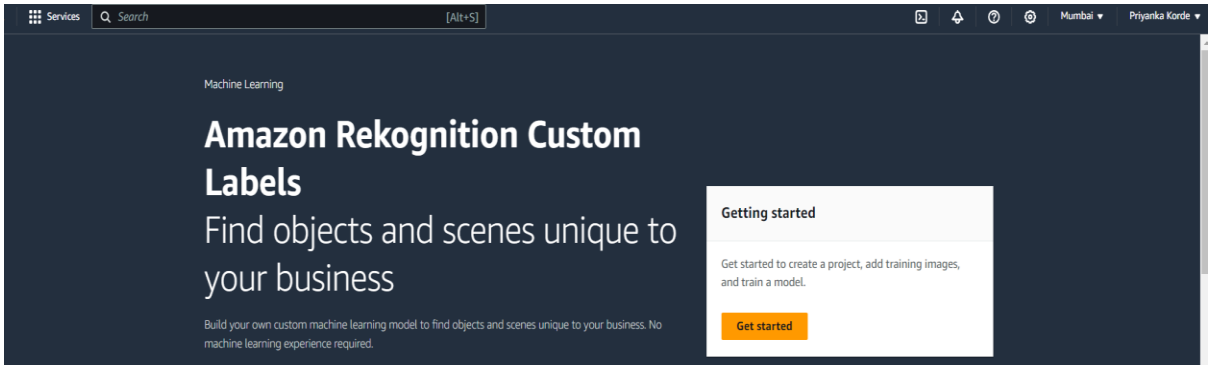


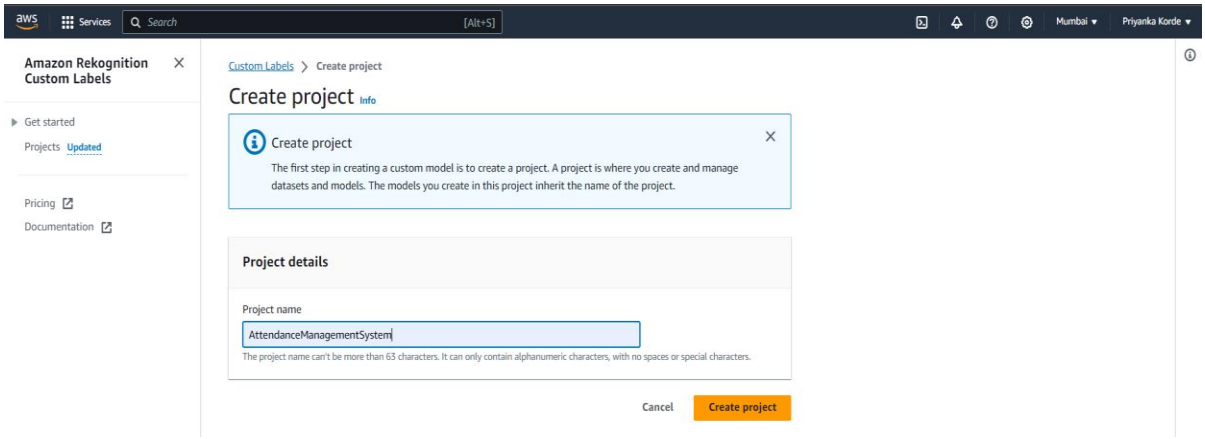


4) Go to AWS Rekognition Service in the same region where you have created S3 bucket.

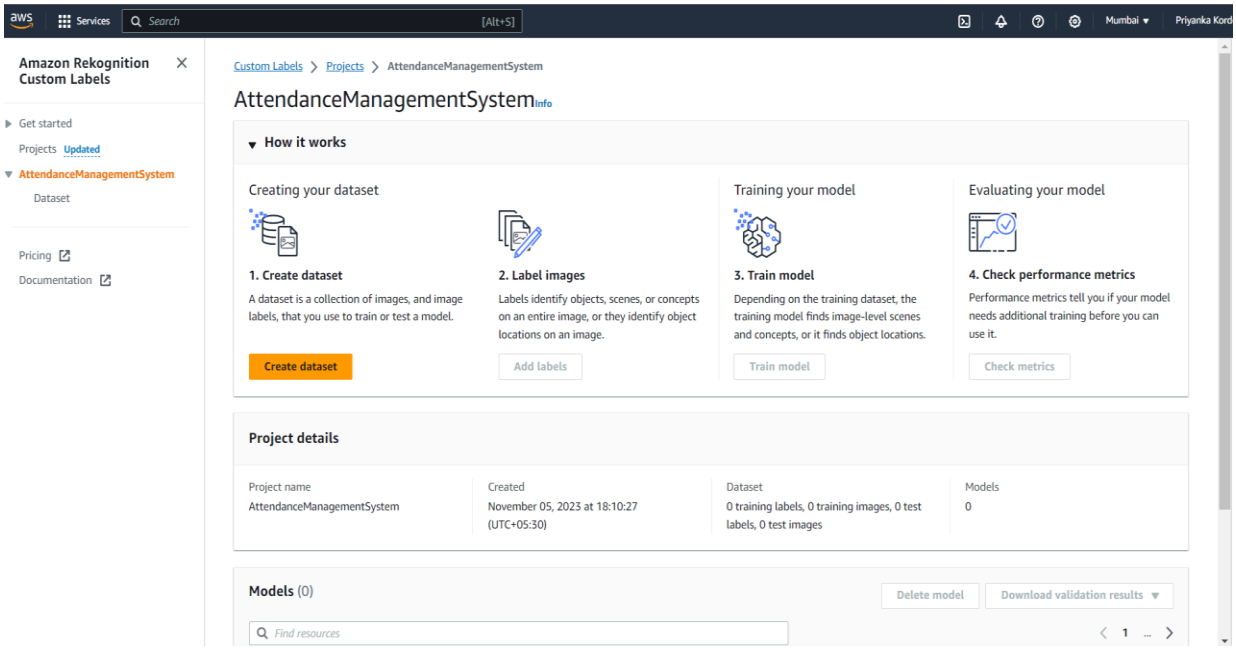
Go to Use Custom Labels → click on Get Started, for first time it will ask you to create S3 bucket with name e.g custom-labels-console-us-east-1-73d5fda5c8.

Without this S3 bucket you can't use recognition. Click on Create S3 bucket.

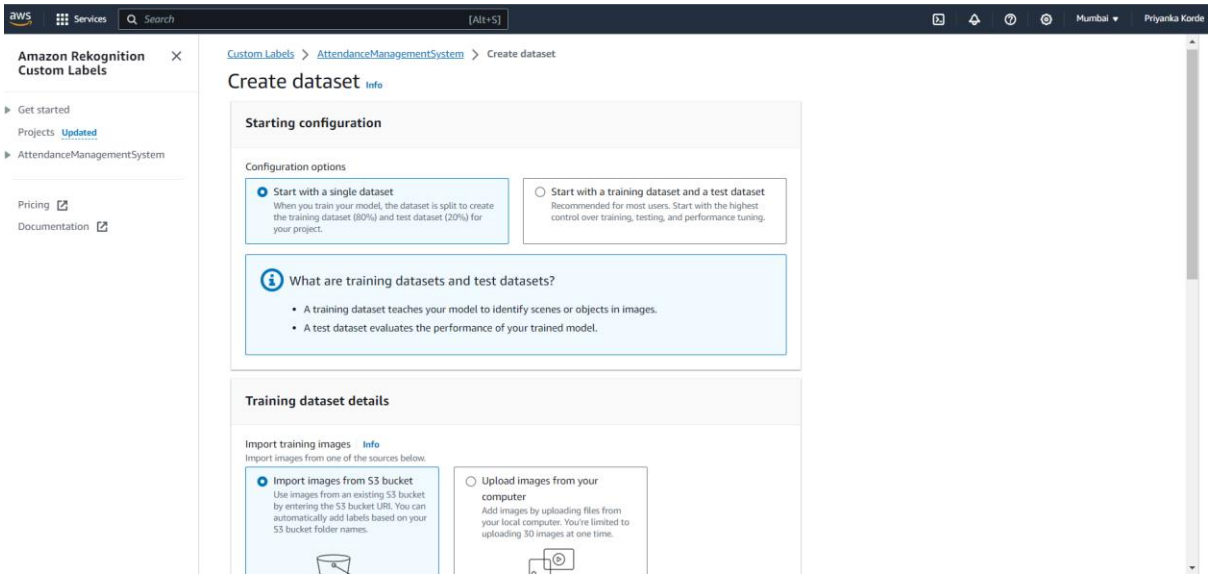


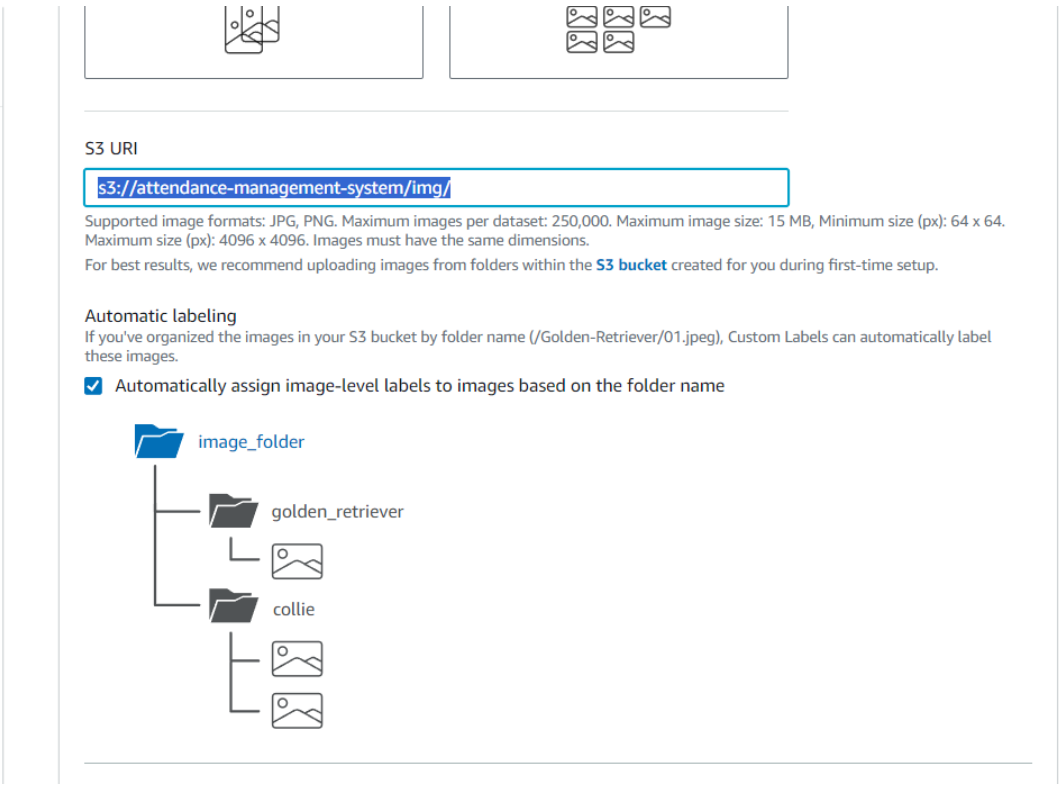


5) In Custom Labels → Project → Create Project → give Project Name "AttendanceManagementSystem".

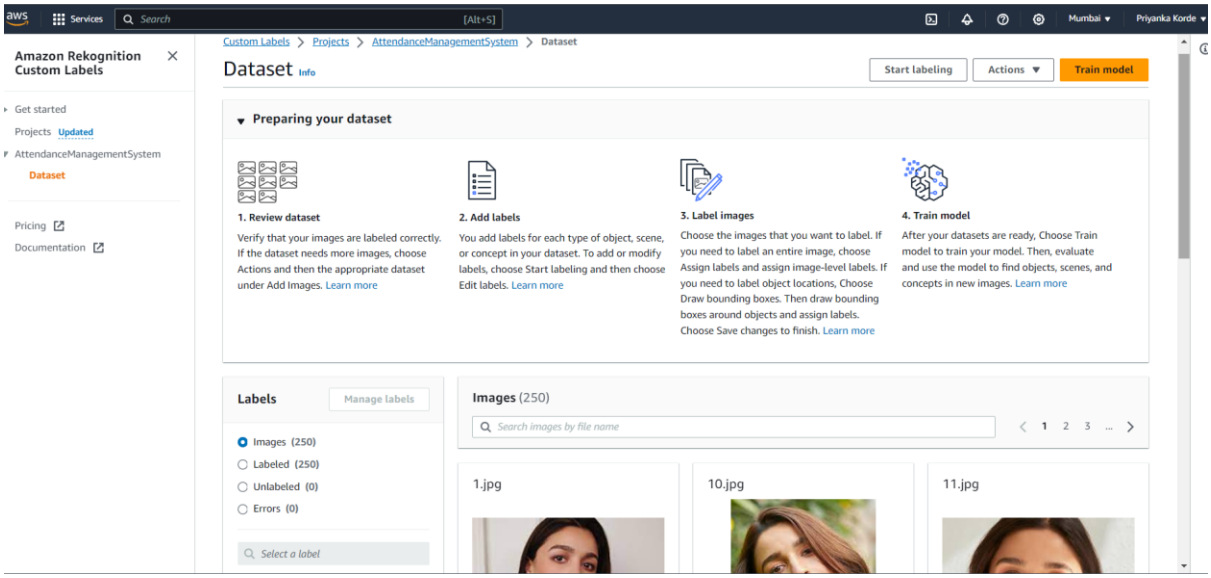


6) Click on Create Dataset → choose Start with Single Dataset → select Import images from S3 → give URI of S3 bucket (attendance-management-system) → check the box of Automatic labeling → click on Create Dataset.

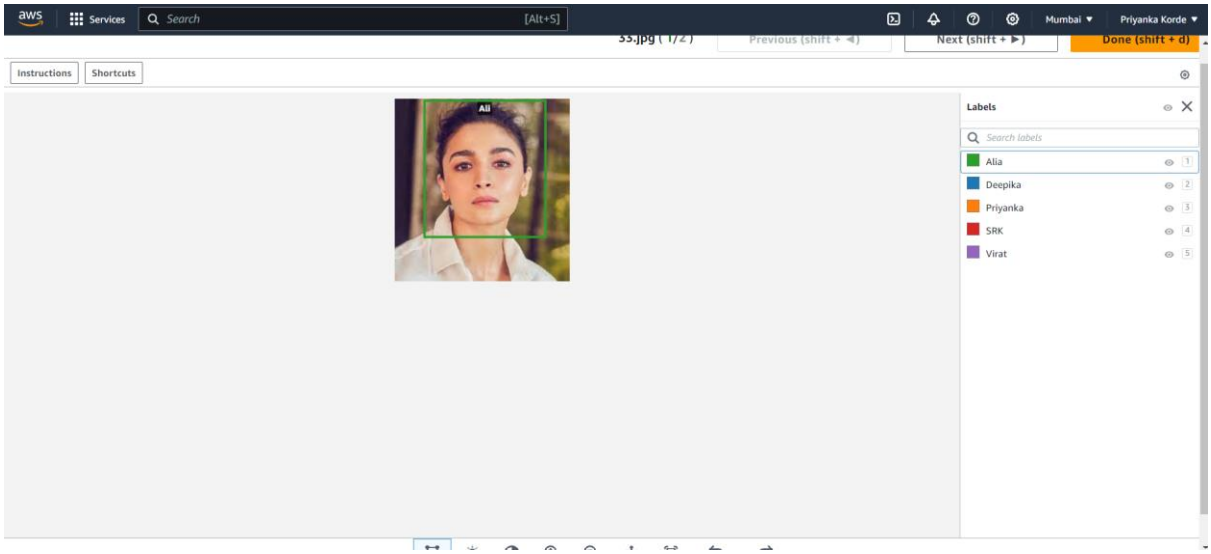


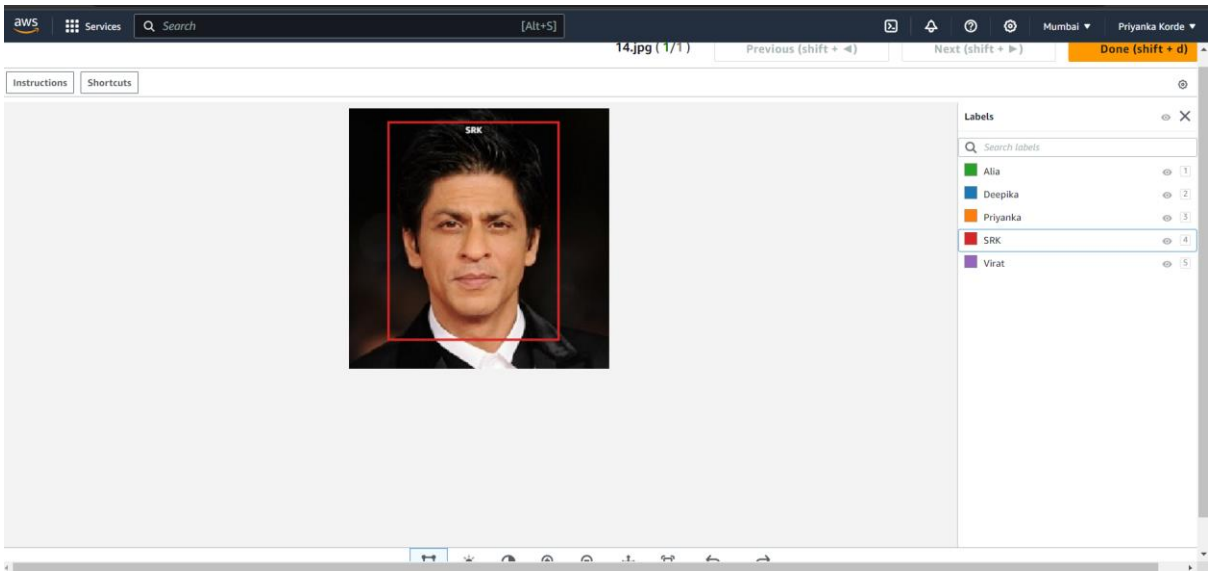


Dataset created successfully.



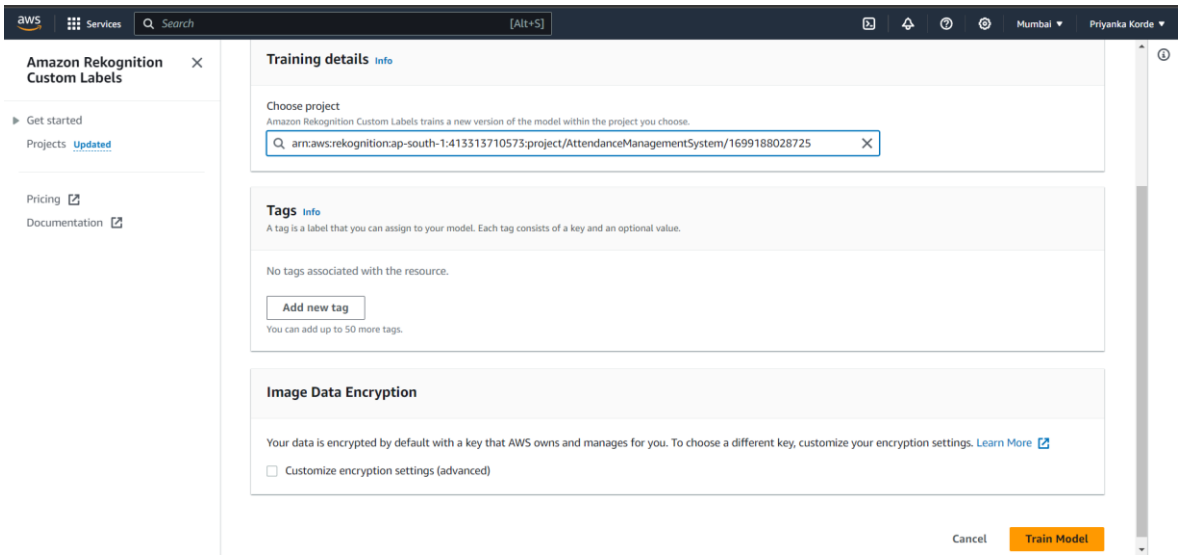
7) Click on Start Labeling → choose the photos to label → select Draw Bounding Boxes method → click on Done.

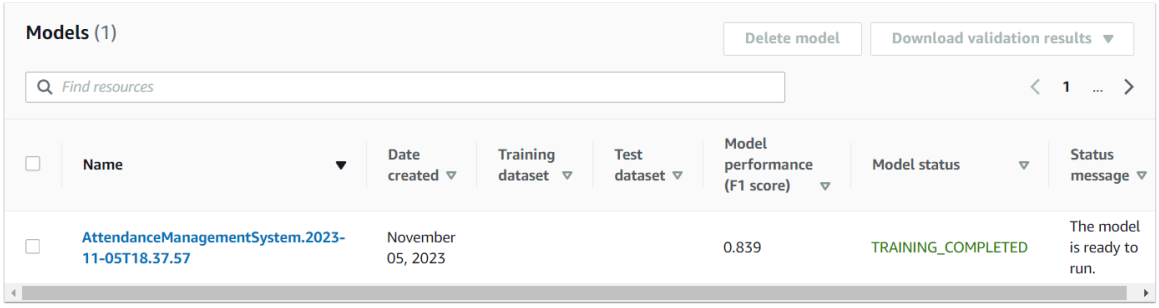




Finish labeling.

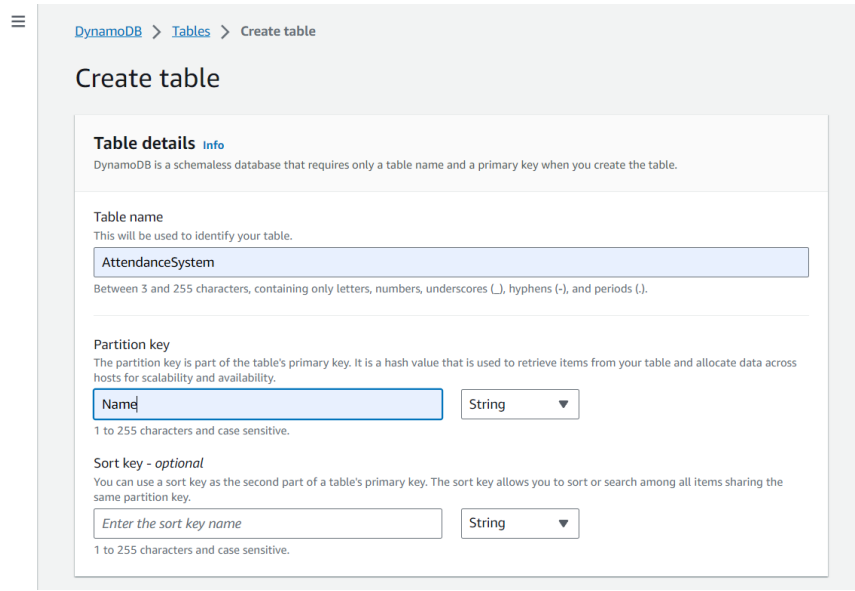
8) Click on Train Model, it will take 30 mins to 24 hrs to complete the process according to number of images. Now wait for completing the train model process.

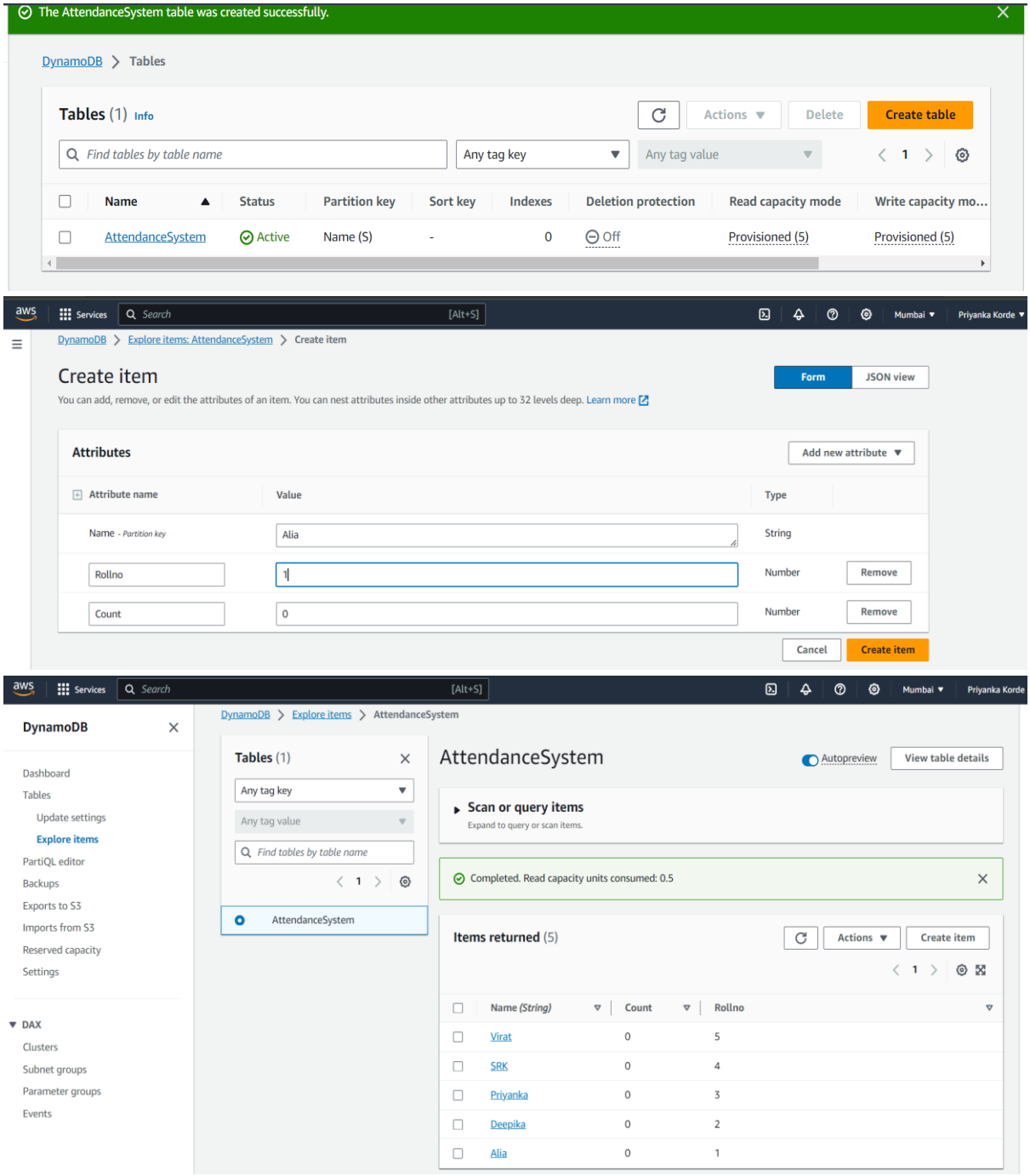




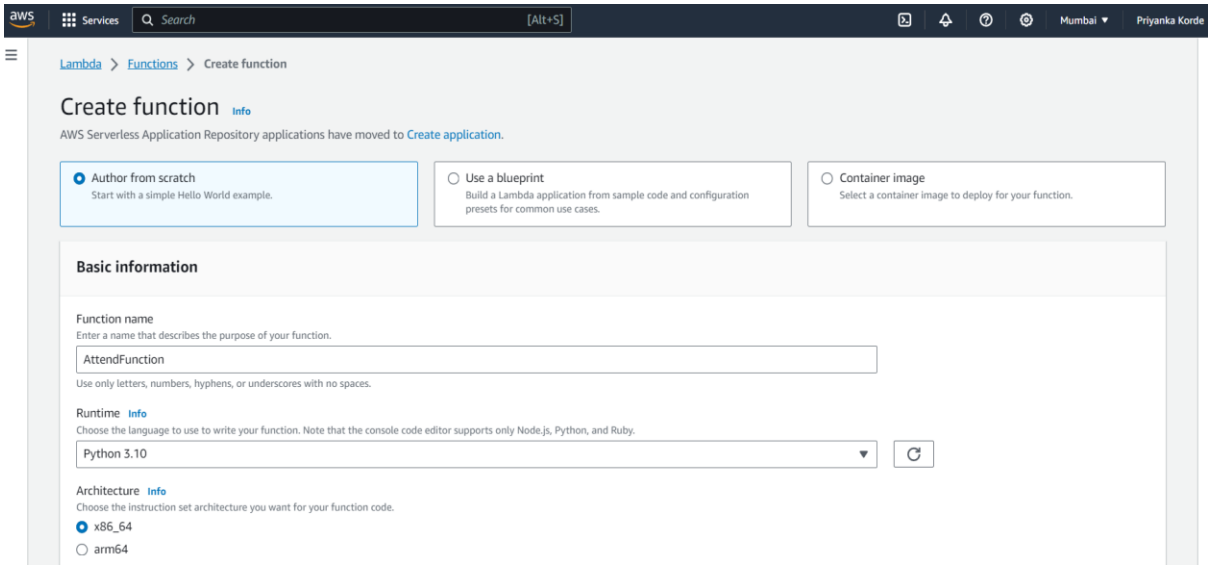
9) Create DynamoDB table.

Give name to table → Give Partition Key as Name → other attributes ‘Rollno’ and ‘Count’. And add the name and roll no. in table. Keep Count as 0.

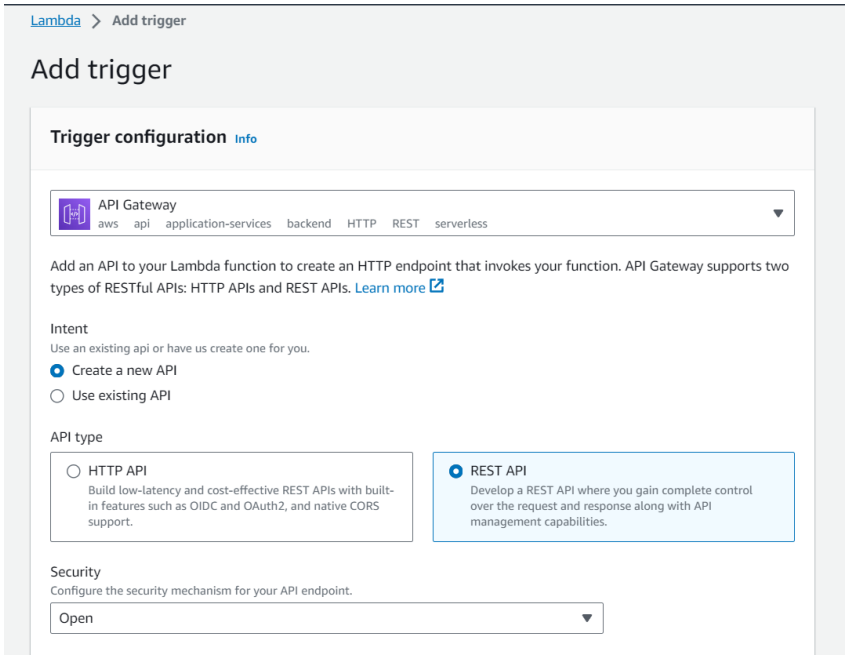




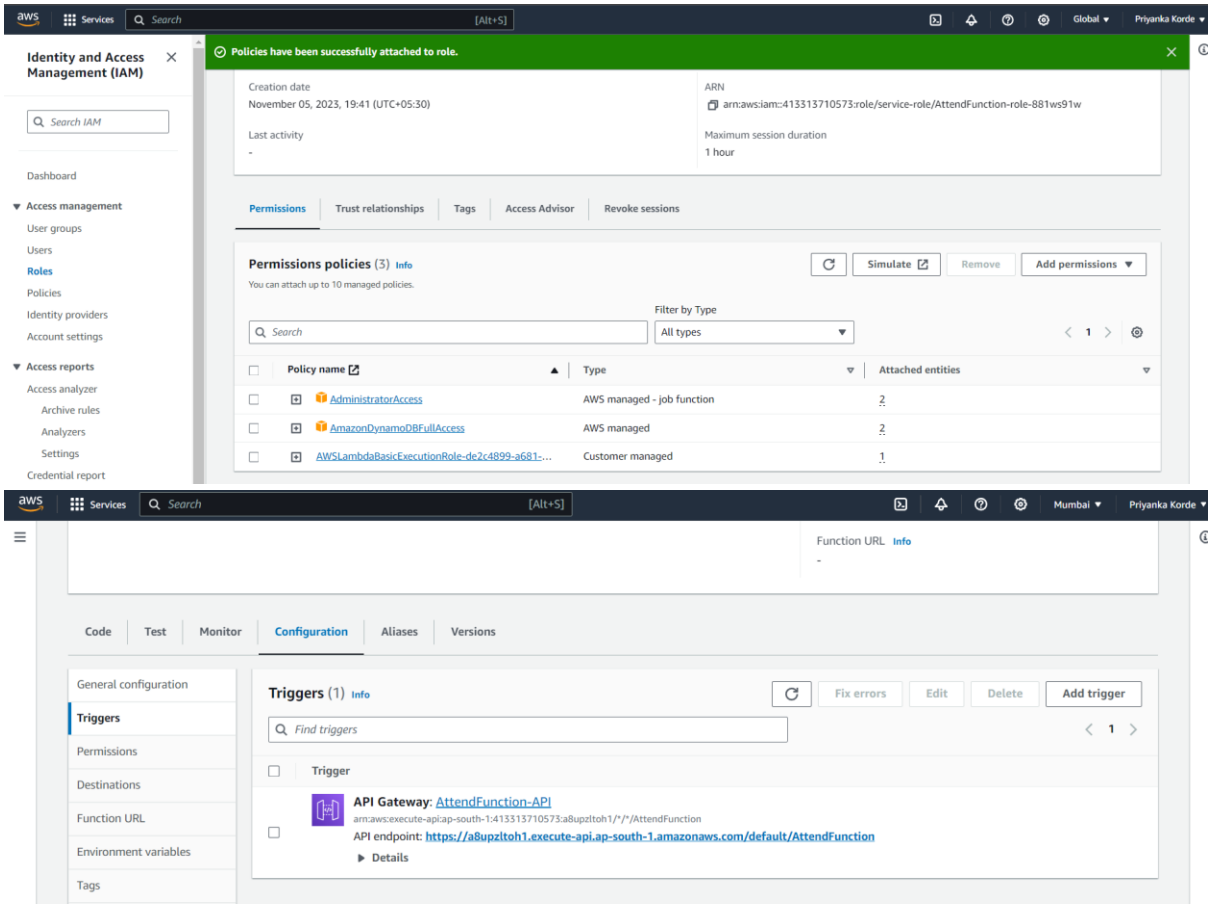
10) Now we will create Lambda function and API Gateway in same region.
For getting attendance from captured img and adding into dynamo db.
Go to Lambda → create Function → Author from Scratch → give Function Name → select Runtime Python 3.10 and all other settings will be default → click on create function.



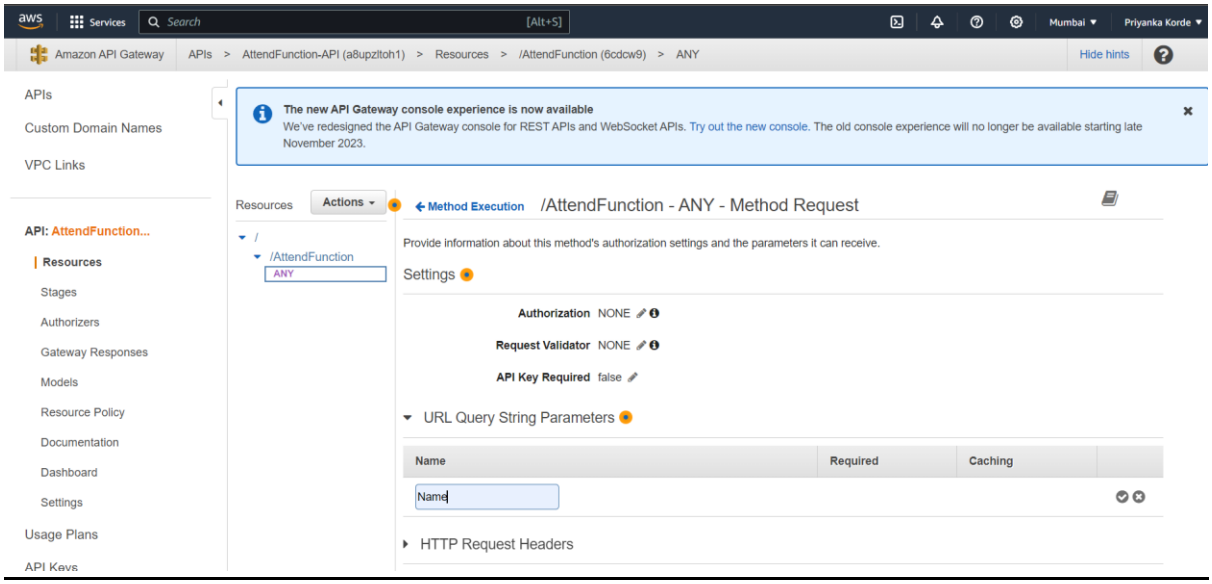
11) To integrate this Lambda Function with API Gateway :
In Lambda Function Overview → Add Trigger → select a source – API Gateway → choose new API → choose a Rest API → security type Open → click on Add.
Your API will be created and automatically attached with lambda.

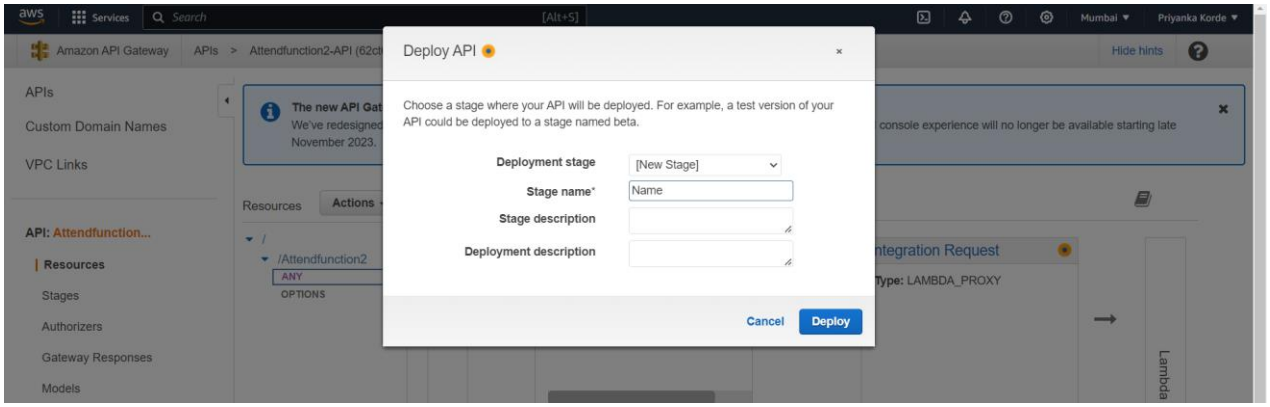


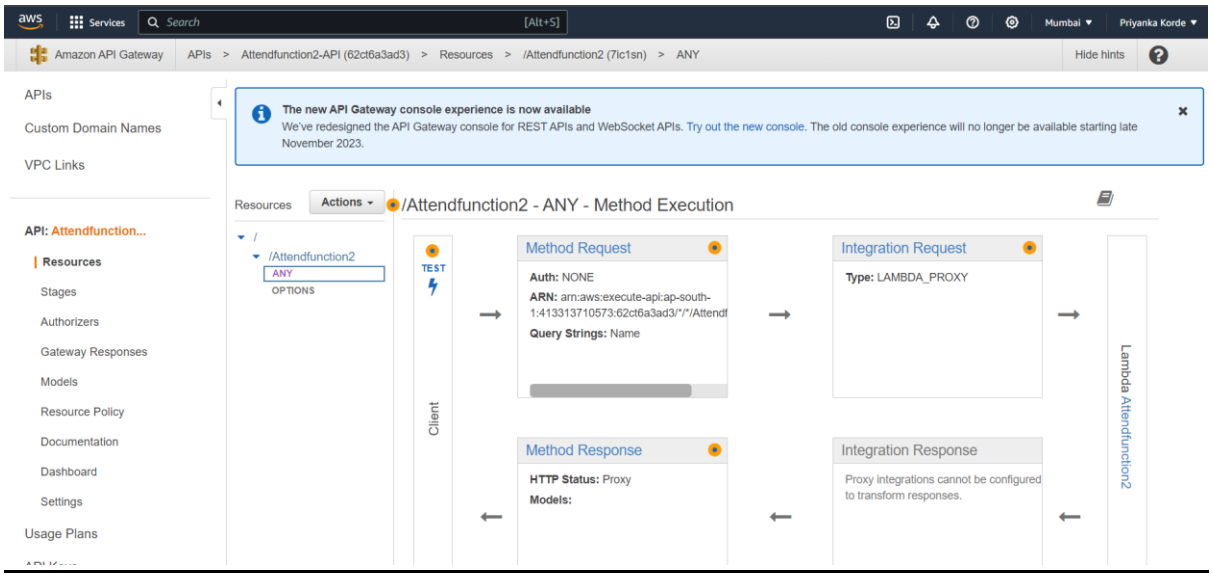
Now in Lambda go to Configuration tab in that → Permissions
There role name is given.
Click on that role name → it will open in IAM Services → Add permissions → Give permission for full access of Dynamodb or All administrator access.
In the configuration tab, the endpoint of your API gateway is given.



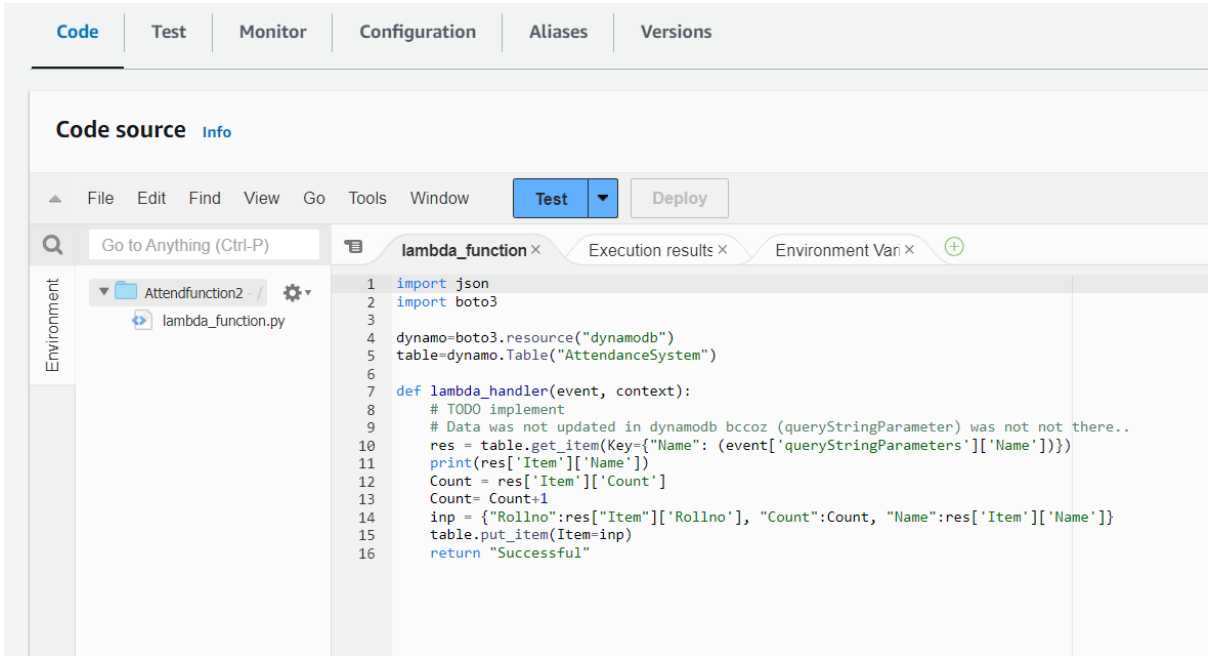
12) Click on AttendFunction-API → Method Execution → Add URL Query String Parameters as “Name” → Deploy API by creating new stage

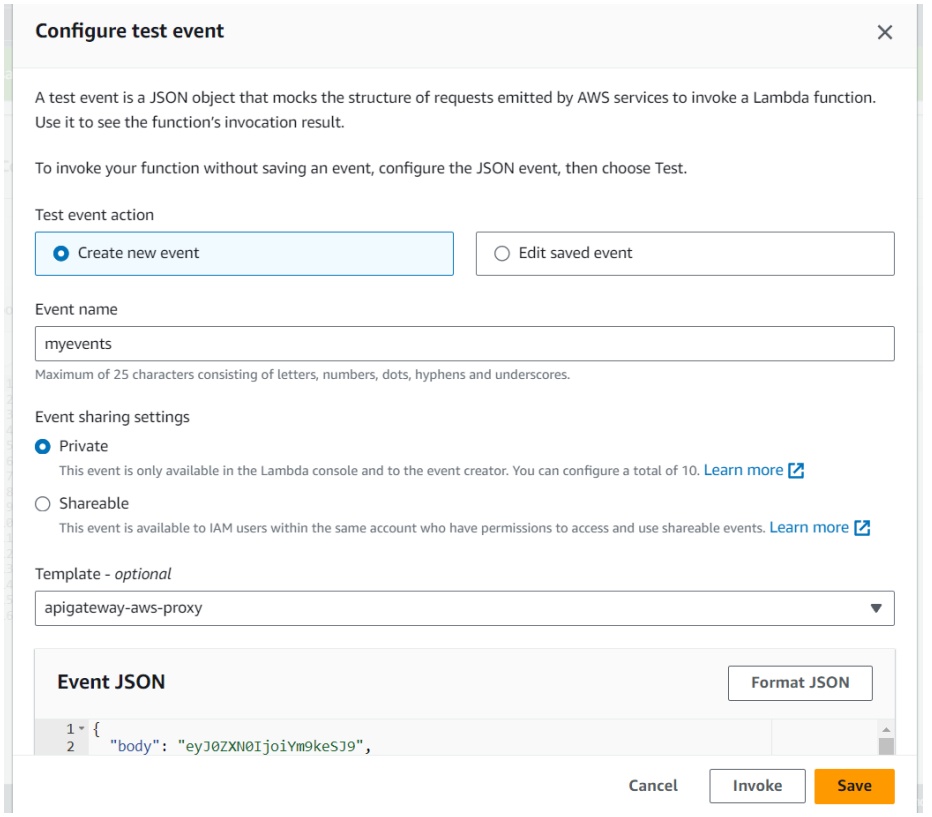


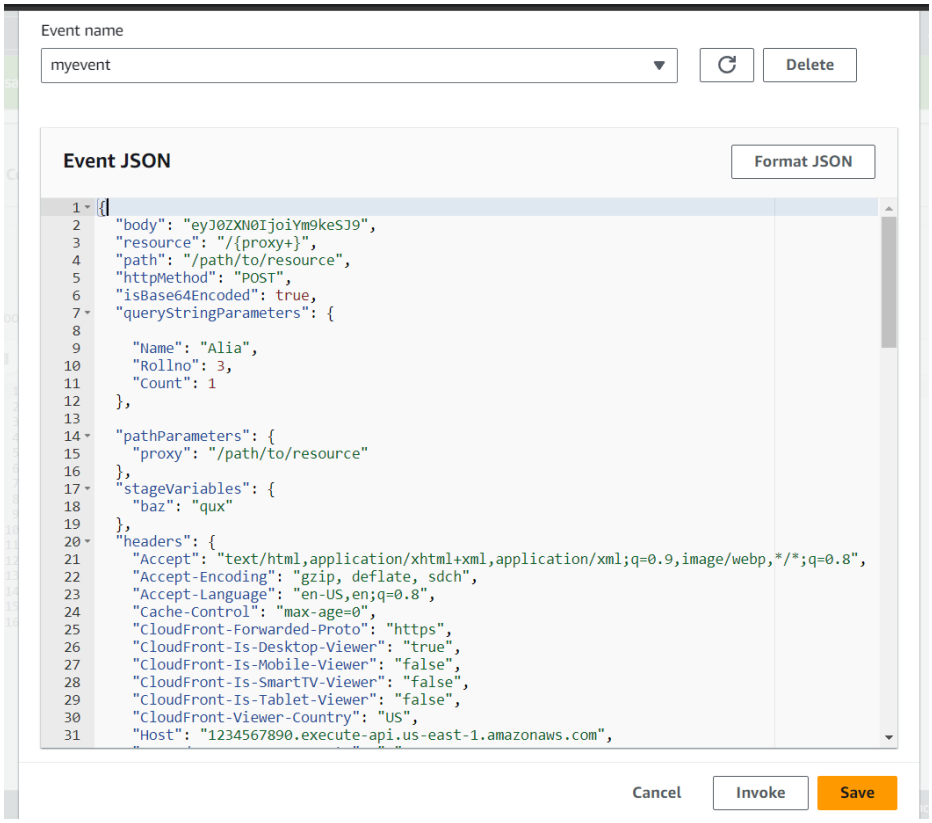




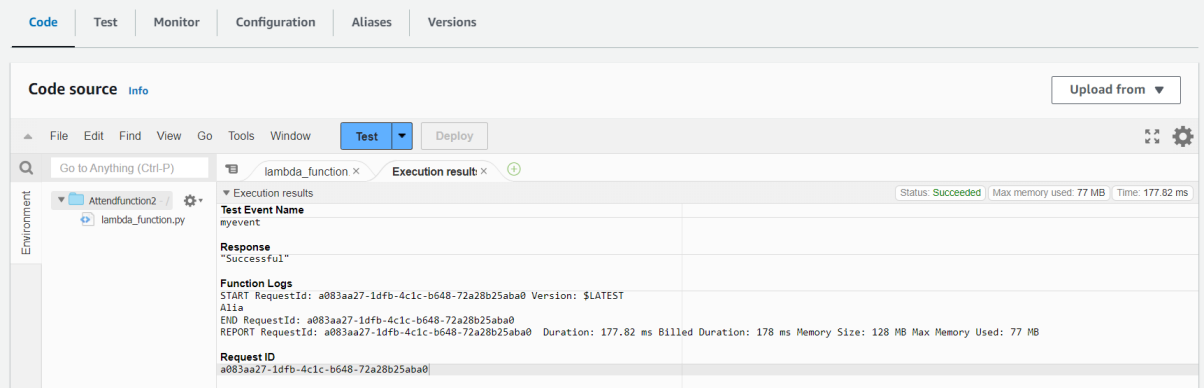
Deploy the AttendFunction code and test it by creating the test event





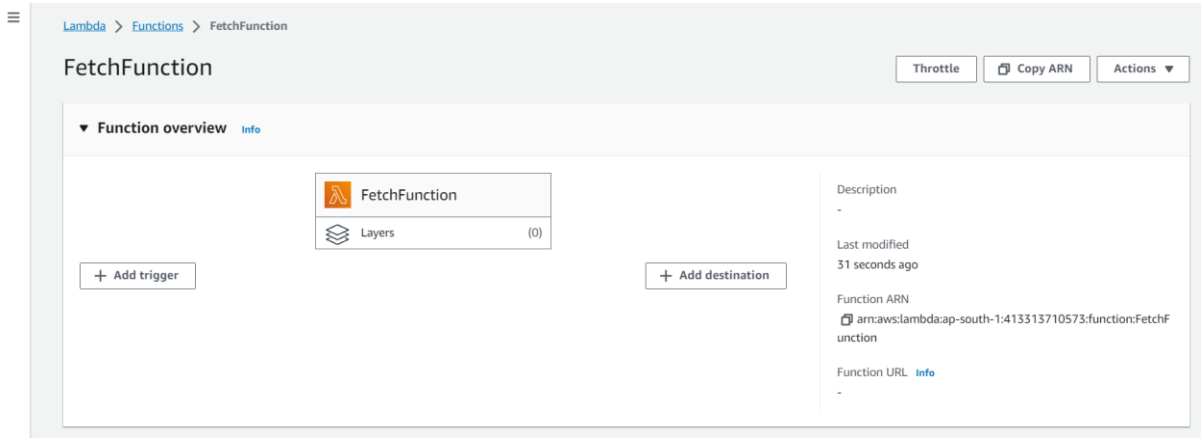


Click on save and test the code

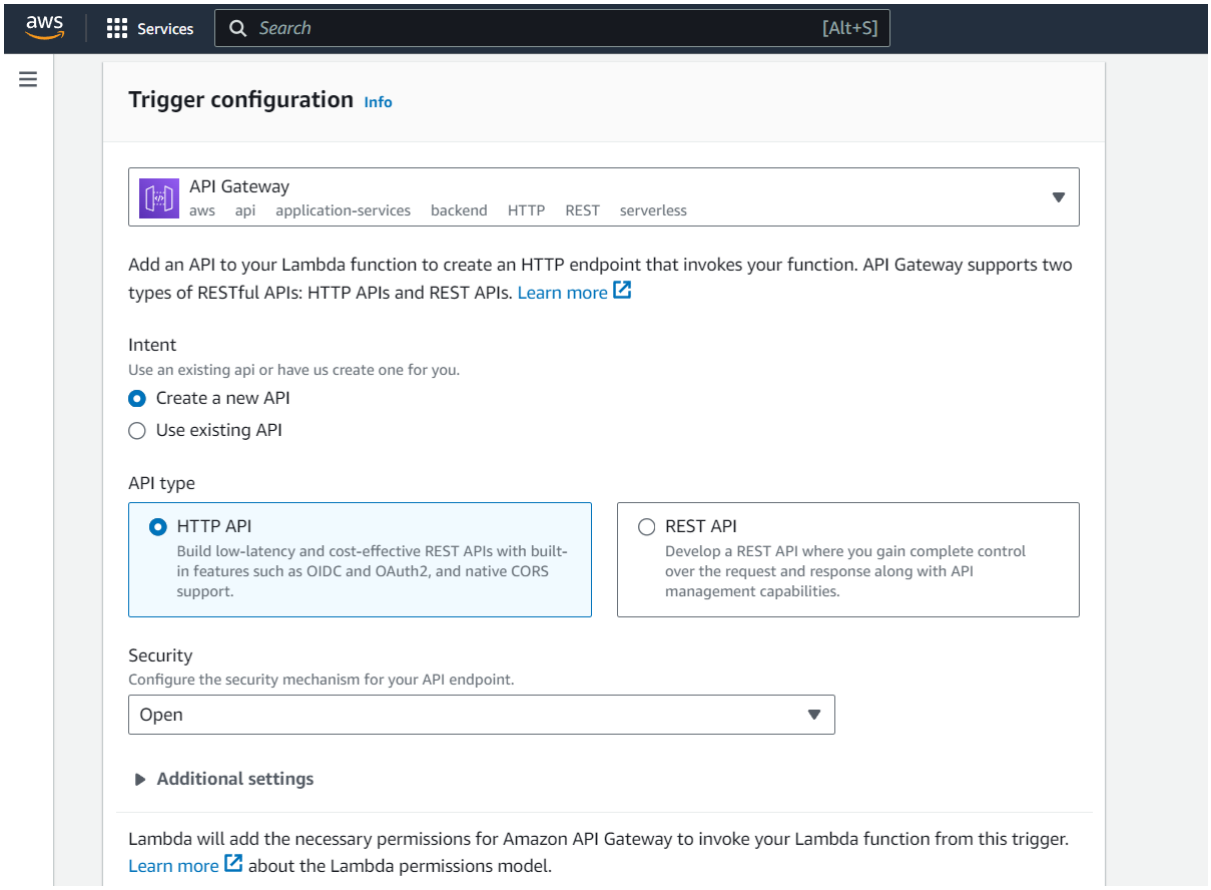


13) For fetching data on created web application we will use API gateway. This API gateway will take the data from dynamodb using Lambda Function. So first we will create Lambda Function to get data from dynamodb.

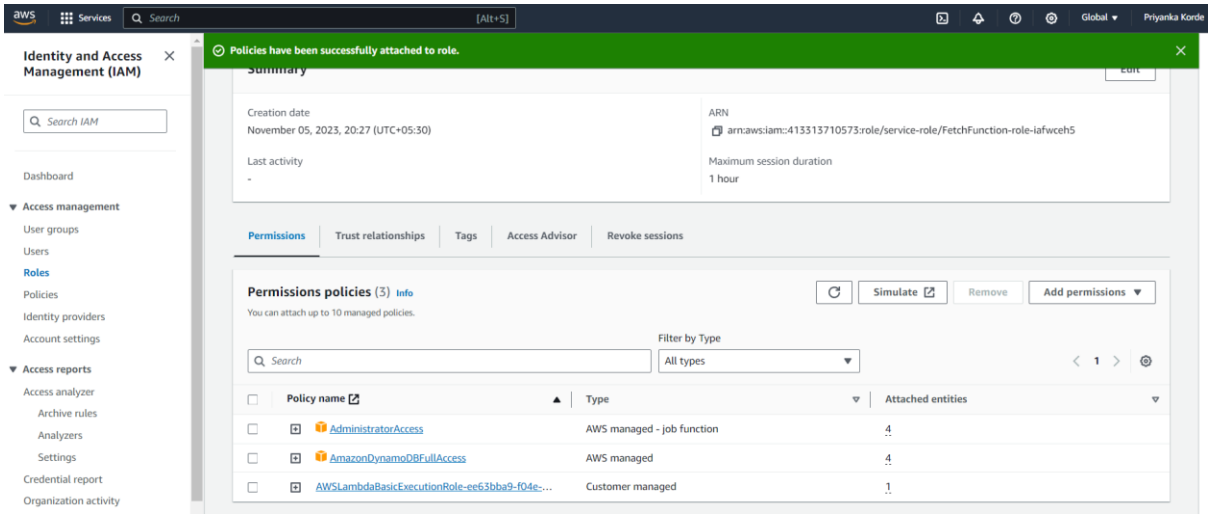
Go to Lambda → create Function → Author from Scratch → give Function Name → select
Runtime Python 3.10 and all other settings will be default → click on create function.



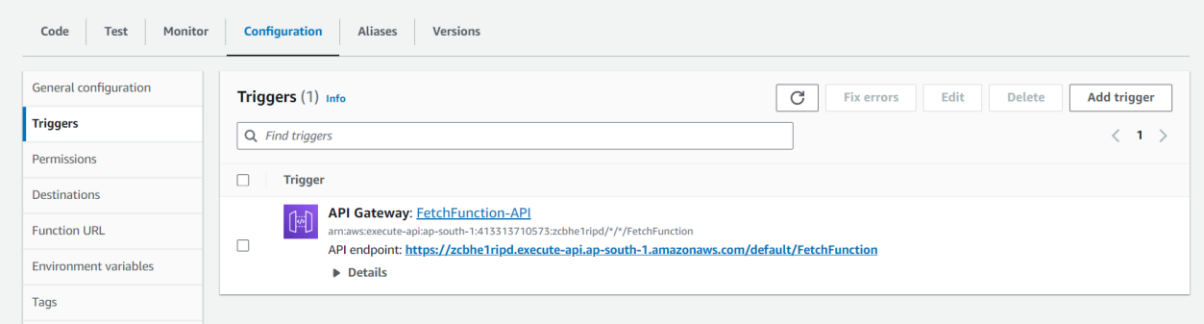
14) To integrate this Lambda Function with API Gateway :
In Lambda Function Overview → Add Trigger → select a source – API Gateway → choose new API → choose a HTTP API → security type Open → click on Add.
Your API will be created and automatically attached with lambda.



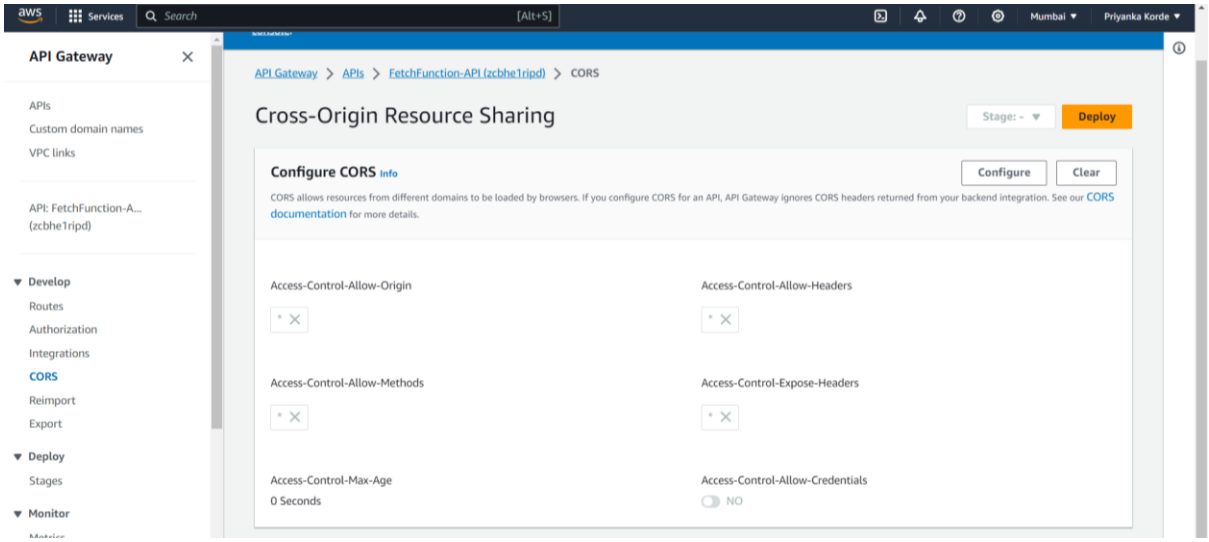
Now in Lambda go to Configuration tab in that → Permissions
There role name is given.
Click on that role name → it will open in IAM Services → Add permissions →
Give permission for full access of Dynamodb or All administrator access.

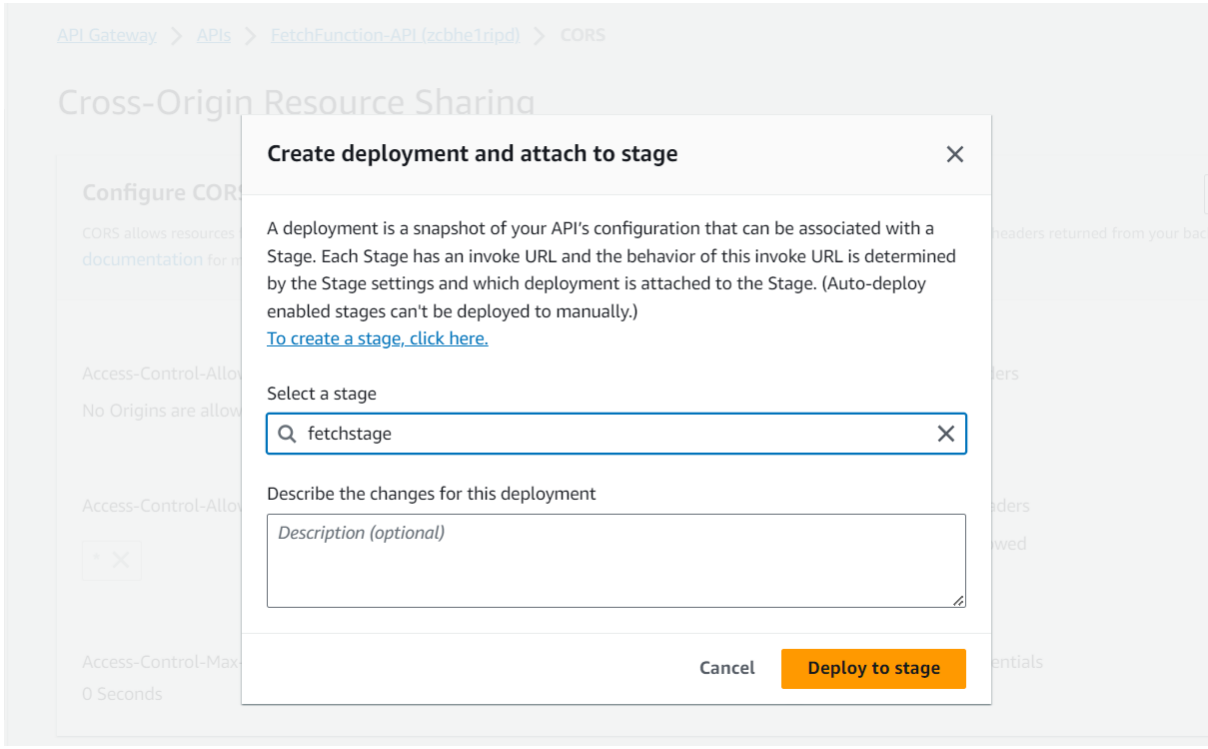


In the configuration tab, the endpoint of your API gateway is given.



Now go to the API Services → click on your API → here go to the CORS option.





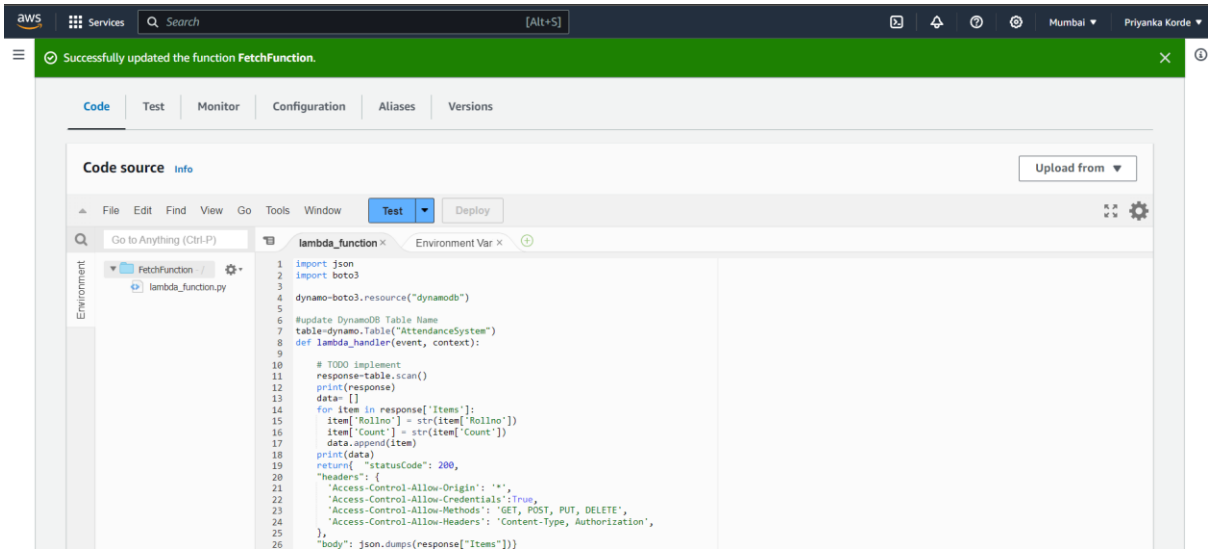
NOTE :

- If while fetching data on web application if data is not shown on web page, then in the

above CORS Configuration you can configure * everywhere then save.

- Sometime it need to clear then also data will be fetch.

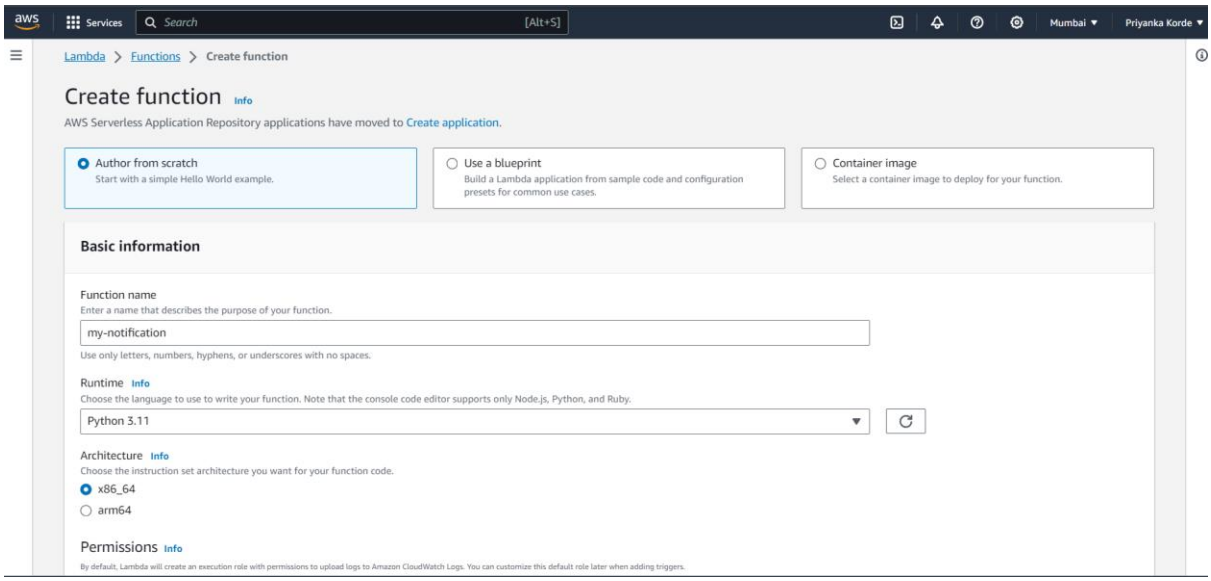
Add code of the Lambda Function → Deploy → Test



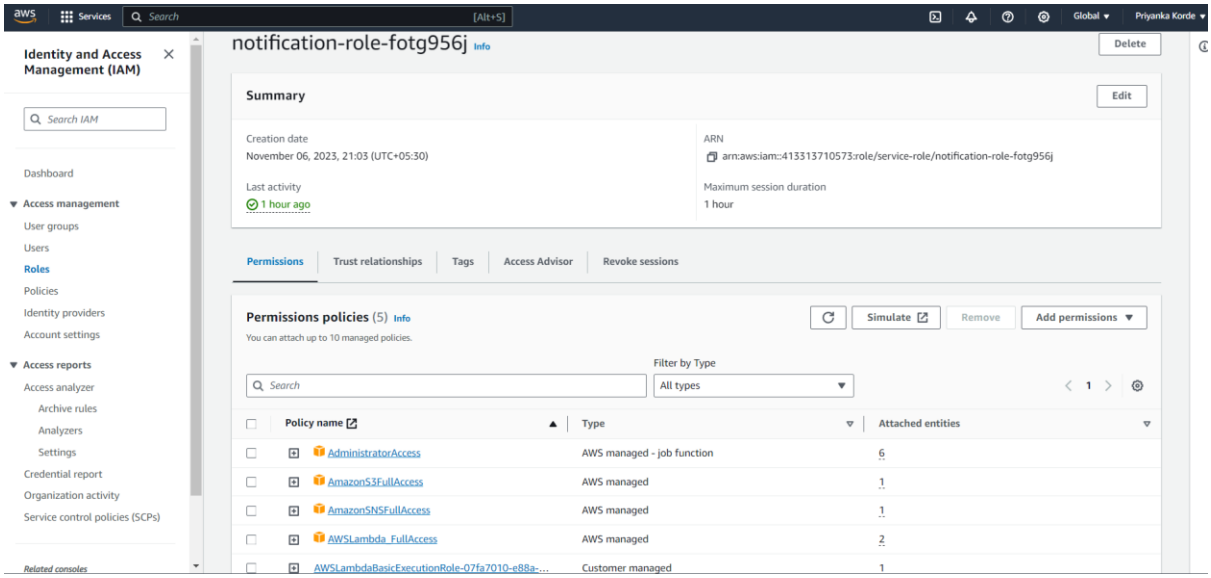
15) Sending Automate Notifications Using Amazon SNS and AWS Lambda

Step 1: Create the Lambda Function

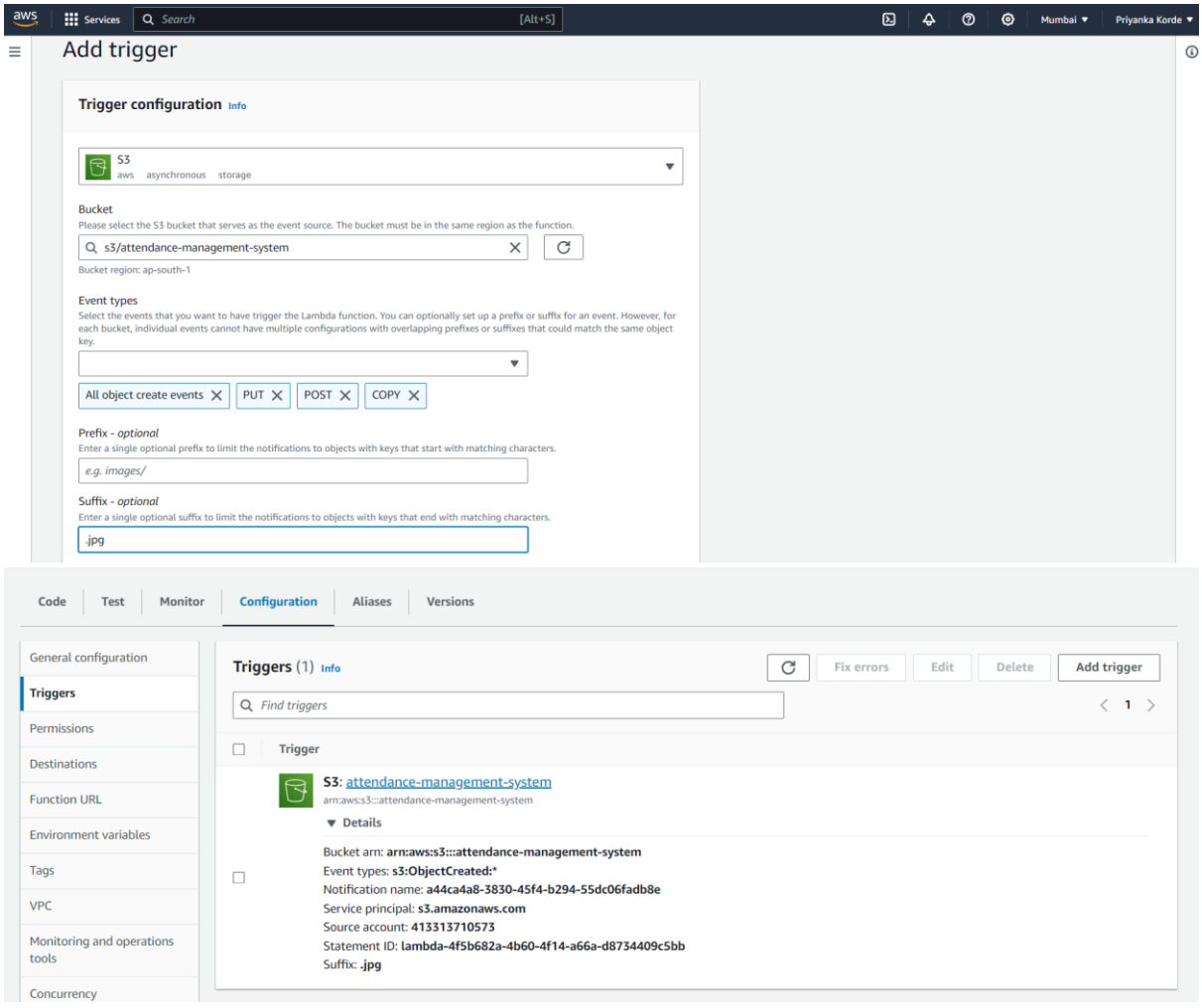
Go to Lambda → create Function → Author from Scratch → give Function Name → select Runtime Python 3.10 and all other settings will be default → click on create function.



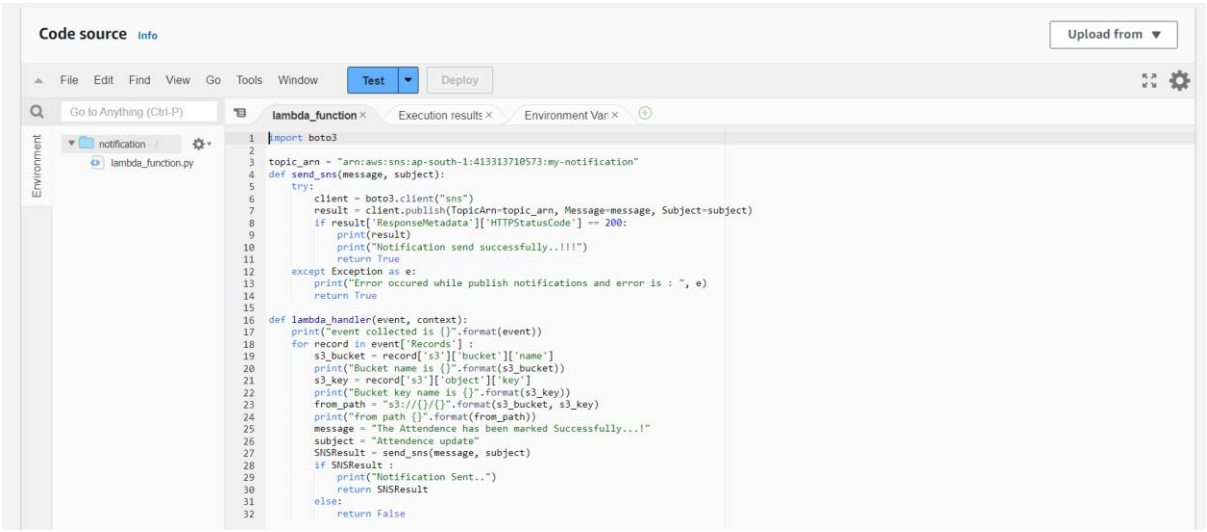
Step 2: Give the role permissions



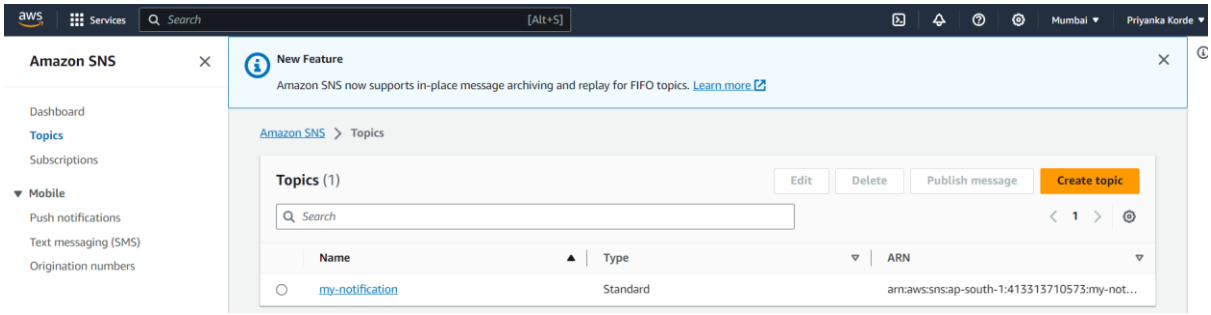
Step 3: Add trigger to lambda function



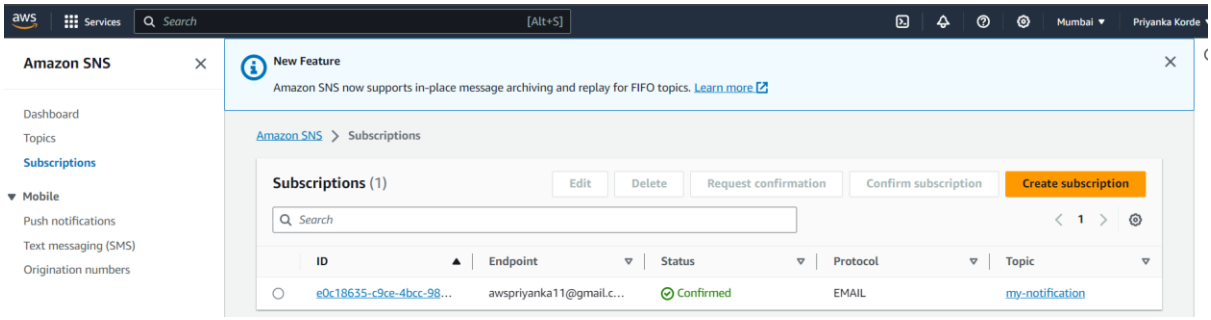
Step 4: Add python code to the function and deploy



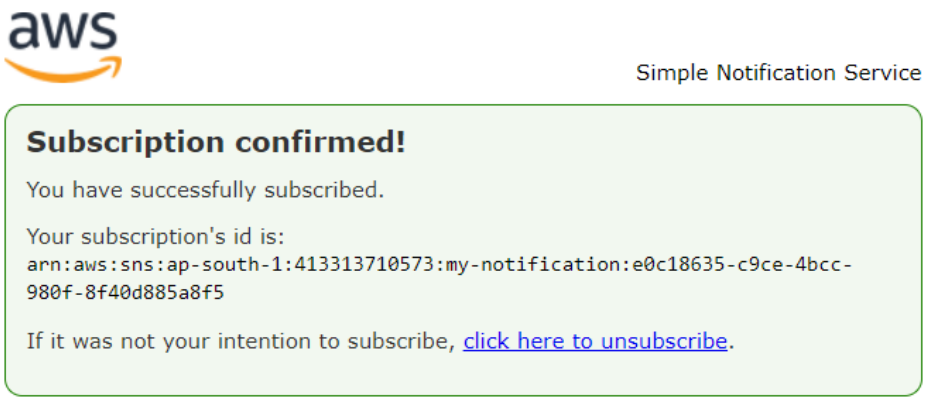
Step 5: Create the SNS Topic



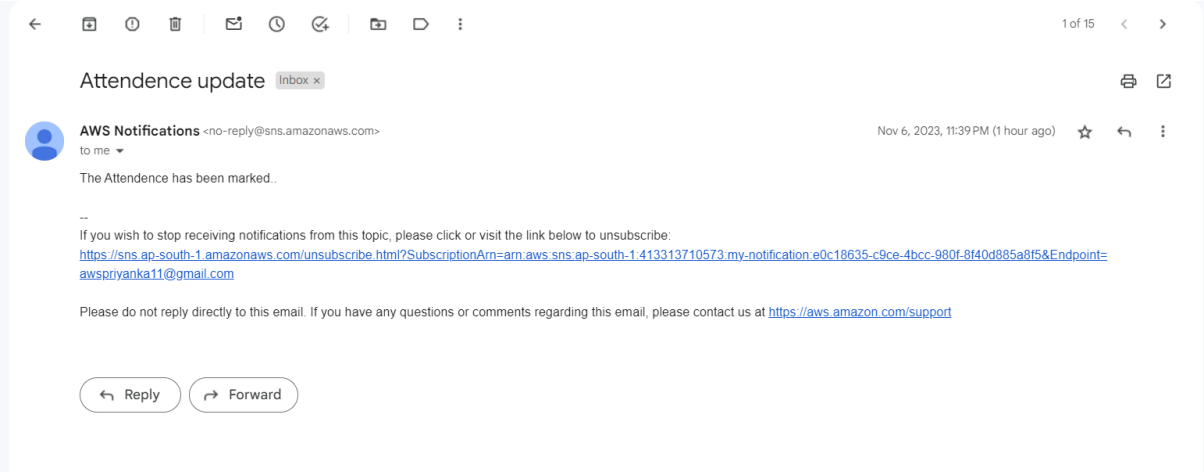
Step 6: Create the subscriptions



Step 7: Confirm the subscription you created



Step 8: When the captured image of student is uploaded to S3 bucket the Lambda Function will get triggered and the notification will be sent to the subscribed person through email.

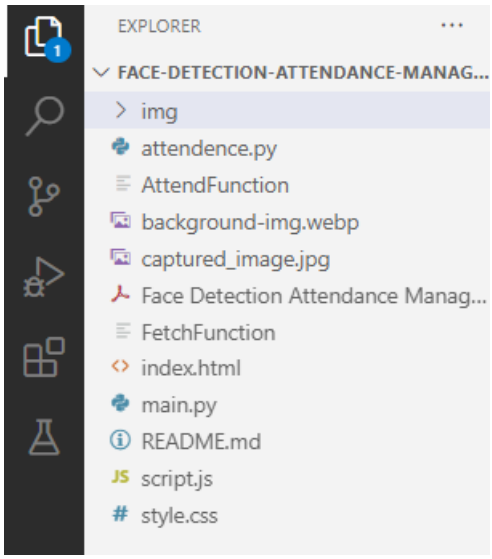


Here all steps are completed for Project.

3.2) Code :

Now to capture a photo of person and add there's attendance in in Dynamodb via API Gateway using Lambda function, we have written one Python Code StudentAttendance.py, main.py and lambda function (AttendFunction, FetchFuntion, my-notification) code.

Project Folder Directory, you also have to create one folder for image i.e. img.



1. StudentAttendance.py :

```
import boto3
import requests
import datetime
import time
import cv2
#Credentials-------------------------------------------------------------------
```

```
client = boto3.client('rekognition',
                      aws_access_key_id="AKIAWAO3KXHWQAXPFSQZ",
                      aws_secret_access_key="yh08nzmewJM+oqZAT0LLl9pQ8A5STtoZLpnVRme
E",
                      #aws_session_token="FwoGZXIvYXdzECEaDIYUDFobtJnX2oDJVCLBAYop3p
7T11dVZL/rjE4nmQQNBQEgYMSXua9XMWaiSGT1v1Giv0j4Txt0883Mrmz4zAlN2RAfXPk4QK6MxHEuEamQ
3U4AgrZ4qA4AVv+uvMuGLWmVPwqUk/uK61R/kE3cNN5Bs3qzWYOzZ22z1RB8IT8YDxS81Wz5tZT/rRBXEG
ODdV6oIR8LIYixYoyBfl3hPWxTpqS/IrOzTcFnFbuoLYZQvLH2IGzf087tsV2bL56CoX62V9eAbv8VORF1
RlGowgIouvqB/AUyLXdmJoVk+HOLePDbLDlYvDU3e7po7lEVq9DW+Aa3vDoqnjqqCEy3WjQtPj8N5Q==",
                       region_name='ap-south-1')


#Capture images for every 1 hour and store the image with current date and time --
-------------------------------------------------------------------------------
for j in range(0, 6):
    current_time = datetime.datetime.now().strftime("%d-%m-%y  %H-%M-%S ")
    print(current_time)
    camera = cv2.VideoCapture(0)

    while True:
        # Capture the video frame by frame
        ret, frame = camera.read()

        # Display the resulting frame
        cv2.imshow('frame', frame)
        # Check if the image needs to be captured
        if cv2.waitKey(1) & 0xFF == ord(' '):
            # Save the captured frame as an image
            cv2.imwrite('img/' + current_time + '.jpg', frame)
            print("Image captured!")
            # Reset the flag
            break

        # Check if the 'q' button is pressed to quit
        if cv2.waitKey(1) & 0xFF == ord('q'):
            exit()


    del (camera)

#Send the captured image to AWS S3 Bucket----------------------------------------
-----------------------------------------
    clients3 = boto3.client('s3', aws_access_key_id="AKIAWAO3KXHWQAXPFSQZ",
                      aws_secret_access_key="yh08nzmewJM+oqZAT0LLl9pQ8A5STtoZLpnVR
meE", region_name='ap-south-1')
    # clients3.upload_file("Hourly Class Images/"+current_time+'.jpg', 'add your S3
bucket name', current_time+'.jpg')

     clients3.upload_file("img/" + current_time + '.jpg', 'attendance-management-
system', current_time + '.jpg')
```

```
    #Recoginze students in captured image ---------------------------------------
-----------------------------------------------
    image_path = 'img/' + current_time + '.jpg'
    with open(image_path,'rb') as source_image:
        source_bytes = source_image.read()
    print(type(source_bytes))

    print("Recognition Service")
    response = client.detect_custom_labels(

    #Update the Rokognition ARN with yours

                                    ProjectVersionArn='arn:aws:rekognition:ap-south-
1:413313710573:project/AttendanceManagementSystem/version/AttendanceManagementSyst
em.2023-11-05T18.37.57/1699189678901',

        Image={
            'Bytes': source_bytes
        },

    )
    print(response)
    if not len(response['CustomLabels']):
         print('Not identified')

    else:
        str = response['CustomLabels'][0]['Name']
        print(str)

         # Update the attendance of recognized student in DynamoDB by calling the
API

                           url      =      "https://a8upzltoh1.execute-api.ap-south-
1.amazonaws.com/default/AttendFunction" + str

        resp = requests.get(url)
        print("Attendence Mark Sucesssful")
        if resp.status_code==200:
            print("Success")

    time.sleep(3600)
```

NOTE :
- In the above code you have to edit your Rekognition Project ARN and API gateway URL.
- Change your IAM user credentials and region name.
- Change S3 bucket name and path of images folder.

2. main.py :

```
# import the opencv library
```

```
import cv2

# define a video capture object
vid = cv2.VideoCapture(0)

# flag to indicate whether to capture an image
capture_image = False

while True:
    # Capture the video frame by frame
    ret, frame = vid.read()

    # Display the resulting frame
    cv2.imshow('frame', frame)
    # Check if the image needs to be captured
    if cv2.waitKey(1) & 0xFF == ord(' '):
        # Save the captured frame as an image
        cv2.imwrite('captured_image.jpg', frame)
        print("Image captured!")
        # Reset the flag
        capture_image = False

    # Check if the 'q' button is pressed to quit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Release the video capture object
vid.release()
# Destroy all the windows
cv2.destroyAllWindows()
```

3. Lambda Function (AttendFunction):

```
import json
import boto3

dynamo=boto3.resource("dynamodb")

#update DynamoDB table name
table=dynamo.Table("AttendanceSystem")
def lambda_handler(event, context):

    # TODO implement

    # Data was not updated in dynamodb bccoz (queryStringParameter) was not not
there..

    res = table.get_item(Key={"Name": event['queryStringParameters']['Name']})
    print(res['Item']['Name'])
    Count = res['Item']['Count']
    Count = Count + 1
```

```
                inp       =       {"Rollno":res["Item"]['Rollno'],"Count":Count,
"Name":res['Item']['Name']}
    table.put_item(Item=inp)
    return "Successful"
```

NOTE :

- Change dynamodb table name.
- Change your dynamo db attributes name.
- And add that queryStringParameter as name in REST API also.

(If we remove this from REST API and Lambda function then the data will not added in dynamodb table.)

4. Lambda Function (FetchFunction)

```
import json
import boto3

dynamo=boto3.resource("dynamodb")

#update DynamoDB Table Name
table=dynamo.Table("AttendanceSystem")
def lambda_handler(event, context):

    # TODO implement
    response=table.scan()
    print(response)
    data= []
    for item in response['Items']:
      item['Rollno'] = str(item['Rollno'])
      item['Count'] = str(item['Count'])
      data.append(item)
    print(data)
    return{  "statusCode": 200,
    "headers": {
      'Access-Control-Allow-Origin': '*',
      'Access-Control-Allow-Credentials':True,
      'Access-Control-Allow-Methods': 'GET, POST, PUT, DELETE',
      'Access-Control-Allow-Headers': 'Content-Type, Authorization',
    },
    "body": json.dumps(response["Items"])}
```

NOTE :

- Change your dynamodb table name and attributes.

5. Lambda Function (my-notification)

```
import boto3

topic_arn = "arn:aws:sns:ap-south-1:413313710573:my-notification"
def send_sns(message, subject):
    try:
        client = boto3.client("sns")
```

```
                result   =   client.publish(TopicArn=topic_arn,   Message=message,
Subject=subject)
        if result['ResponseMetadata']['HTTPStatusCode'] == 200:
            print(result)
            print("Notification send successfully..!!!")
            return True
    except Exception as e:
        print("Error occured while publish notifications and error is : ", e)
        return True

def lambda_handler(event, context):
    print("event collected is {}".format(event))
    for record in event['Records'] :
        s3_bucket = record['s3']['bucket']['name']
        print("Bucket name is {}".format(s3_bucket))
        s3_key = record['s3']['object']['key']
        print("Bucket key name is {}".format(s3_key))
        message = "The Attendence has been marled Successfully...!"
        subject = "Attendence update"
        SNSResult = send_sns(message, subject)
        if SNSResult :
            print("Notification Sent..")
            return SNSResult
        else:
            return False
```

For Fetching Data from Dynamodb on Web Application :
Create on Web Application using Html, CSS and JavaScript
index.html :

```
<!DOCTYPE html>
<html>
<head>
  <title>Attendance Table</title>
                                <link                                rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css">
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="container mt-5">
    <H1>Attendance Report</H1>
    <table id="attendanceTable" class="table styled-table ">
      <thead>
        <tr>
          <th>Name</th>
          <th>Rollno</th>
          <th>Attendance Count</th>
        </tr>
      </thead>
      <tbody></tbody>
```

```
    </table>
  </div>

  <script src="script.js"></script>
</body>
</html>
```

style.css :

```
*{
    padding: 0;
    margin: 0;
}
body {
    background: url("/backgroung.jpg") no-repeat center center fixed;
    background-size: cover;

  }
h1 {
    color: #000000;
    font-style: italic;
    font-weight: bold;
    font-family: 'Times New Roman';
    text-align: center;
}
.styled-table {
    border-collapse: collapse;
    margin: 25px 0;
    font-size: 0.9em;
    font-family: sans-serif;
    min-width: 400px;
    box-shadow: 0 0 20px rgba(247, 246, 246, 0.15);
}

.styled-table thead tr {
    background-color: #08ae92;
    color: #050505;
    text-align: left;
}

.styled-table th,
.styled-table td {
    padding: 12px 15px;
}

.styled-table tbody tr:nth-of-type(even) {
    background-color: #FFFFFF;
}

.styled-table tbody tr:nth-of-type(odd) {
    background-color: #d9d8d8;
}
```

```
.styled-table tbody tr.active-row {
    font-weight: bold;
    background-color: #009879;
}
```

script.js :

```
document.addEventListener('DOMContentLoaded', () => {
                                    fetch('https://zcbhe1ripd.execute-api.ap-south-
1.amazonaws.com/default/FetchFunction')
      .then(response => response.json())
      .then(data => {
        const tableBody = document.querySelector('#attendanceTable tbody');
        data.forEach(student => {
          const row = document.createElement('tr');
          const rollNoCell = document.createElement('td');
          const nameCell = document.createElement('td');
          const attendanceCell = document.createElement('td');

// Change Cridential like Name, Rollno and Count here....
          rollNoCell.textContent = student.Rollno;
          nameCell.textContent = student.Name;
          attendanceCell.textContent = student.Count;

          row.appendChild(rollNoCell);
          row.appendChild(nameCell);
          row.appendChild(attendanceCell);
          tableBody.appendChild(row);
        });
      })
      .catch(error => {
        console.error('Error:', error);
      });
  });
```

NOTE :

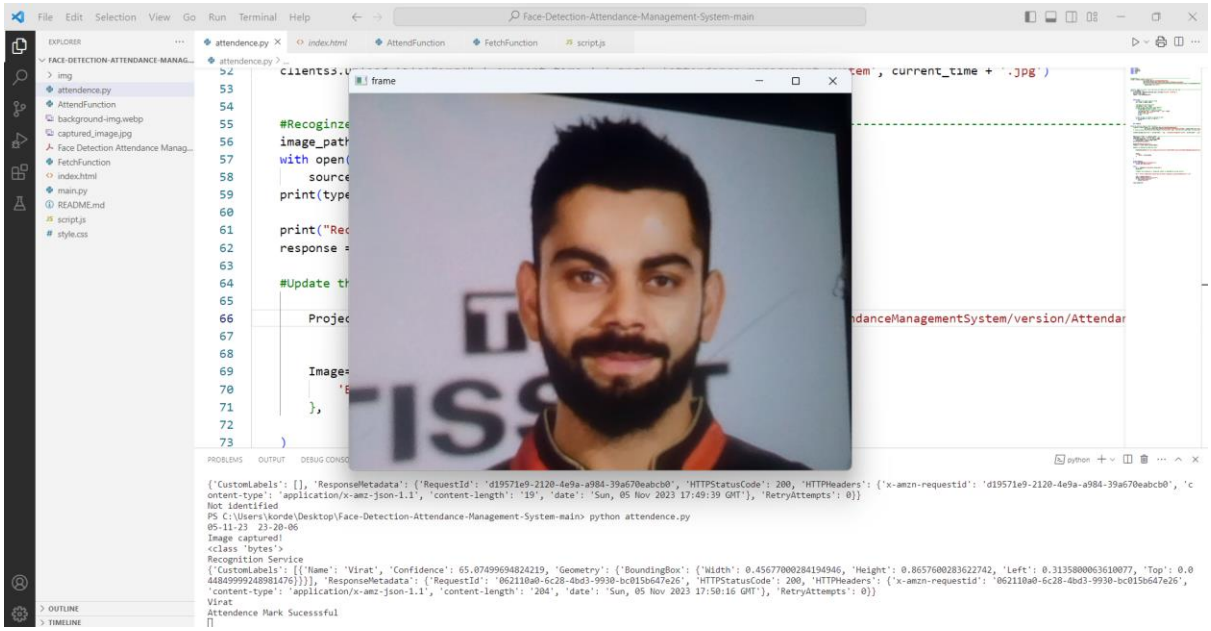
- In script.js code you have to add second API URL. i.e. FetchFunction
- This API we will create in next further steps.

RESULTS

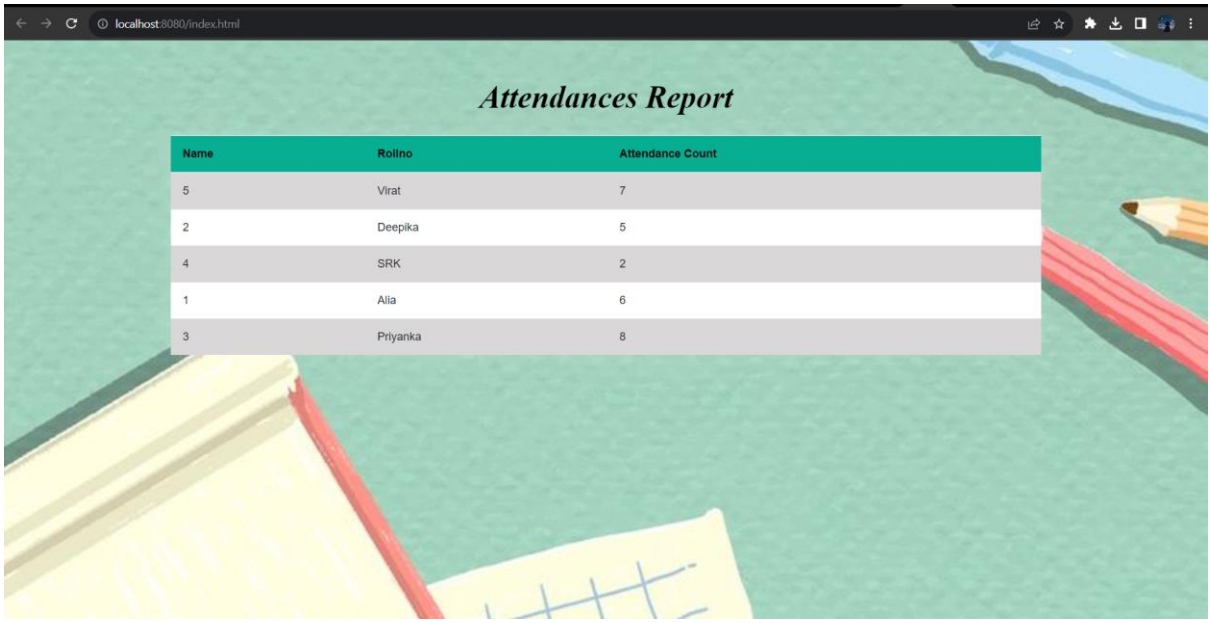
4.1) Output :

Before running the code make sure you start your model in Amazon Rekognition Custom Labels

Next will run Python code, then camera will on → capture image by clicking space button → Person's name will be shown in terminal.



Front End :



4.2) Conclusion:

The implementation of a face recognition attendance system that leverages AWS services represents a significant advancement in attendance management, providing organizations with a seamless, efficient, and secure solution.

The system has demonstrated the potential to revolutionize traditional attendance tracking methods by utilizing Amazon Rekognition for facial recognition capabilities, AWS Lambda for serverless computing, and AWS S3 for image storage.

As a result, our project primarily focused on tracking attendance for various use cases such as the office, events, schools, and colleges, among others.
Thus we created an easy-to-use web page and ended the project via email to validate attendance.

4.3) Future Scope:

Face Recognition system using AWS services holds numerous possibilities for advancements and enhancements. We can use much more innovative ideas to improve the current project. This could be accomplished by simply automating the model train with the labels of the Amazon Rekognition service and looping the codes using the Amazon Sagemaker service. Making an easy-to-use User Interface for end users to make the process easier. Also, for various departments and perspectives of the project, we can use other services in novel ways to achieve the desired results.

As technology evolves and AWS continues to innovate, the scope for enhancing face recognition attendance systems within the AWS ecosystem will expand, offering more robust, efficient, and secure solutions for organizations across various industries.

REFERENCES

[1] https://docs.aws.amazon.com/index.html

[2] https://github.com/CriMenio/Face-Detection-Attendance-Management-System/tree/main

[3] https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-lambda-proxy-integrations.html#apigateway-multivalue-headers-and-parameters

[4] https://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-integration-settings.html

[5] https://github.com/RekhuGopal/PythonHacks/blob/main/AWS_SNS_Lambda_Notification/publishnotification.py

[6] https://www.youtube.com/watch?v=zLL4ZTmDHSI