

PROJECT ENTITLED

**Face Recognition Attendance Management**  
**System With AWS**

**Group Members:**

Priyanka Korde

Aniket Singh

Kiran Watole

**Project Guide:**

Dheerendra Solanki

## Table of Contents

Chapters	Contents	Pg. No.
1.	Introduction	3
	1.1) Introduction	3
	1.2) Problem Statement	3
2.	Proposed System	4
	2.1) System Architecture	4
	2.2) Datasets	4
	2.3) Requirements	5
	2.4) AWS Services Used	6
3.	Implementation	6
	3.1) Steps	6
	3.2) Code	23
4.	Results	28
	4.1) Output	28
	4.2) Conclusion	31
	4.3) Future Scope	31
5.	References	33

# 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.

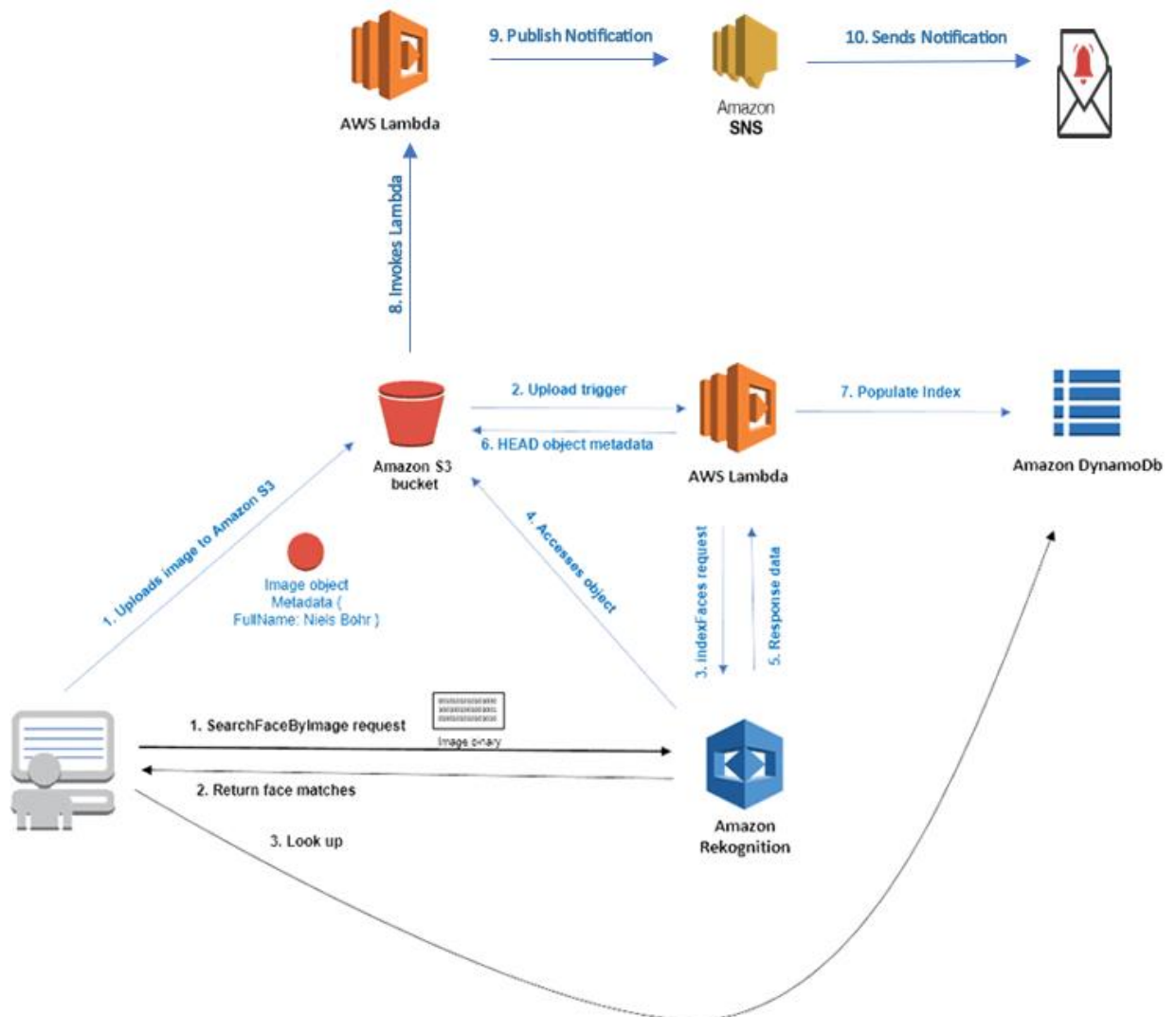
## **1.2) Problem Statement:**

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.

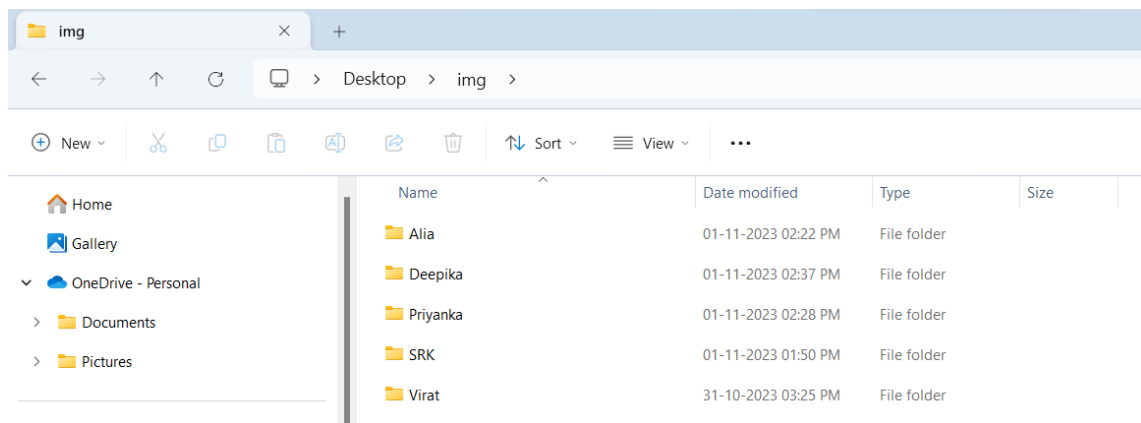
## PROPOSED SYSTEM

### 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.

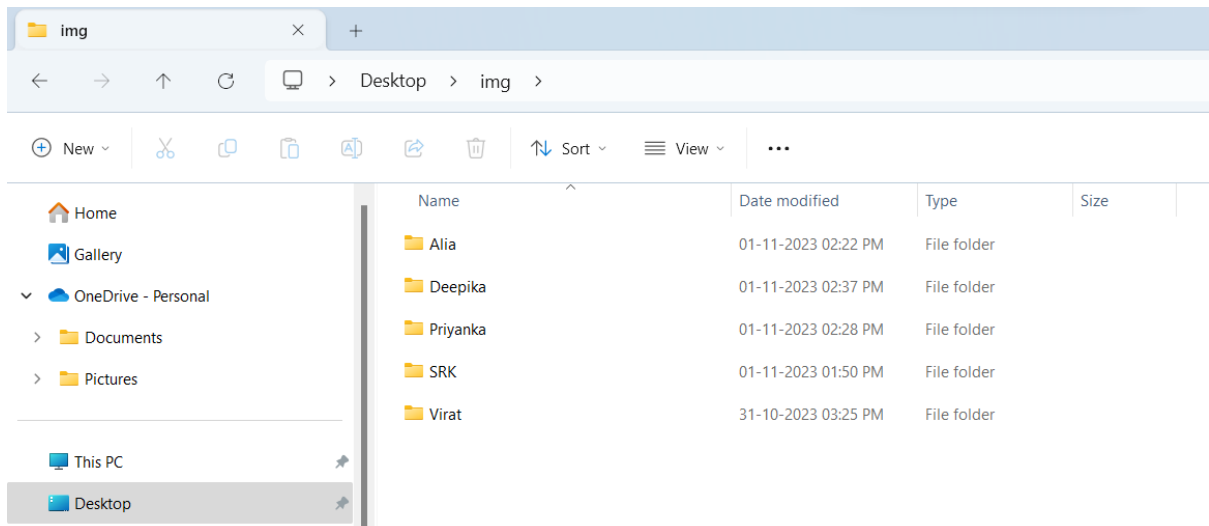
## 6. Simple Notification Service (SNS)

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.

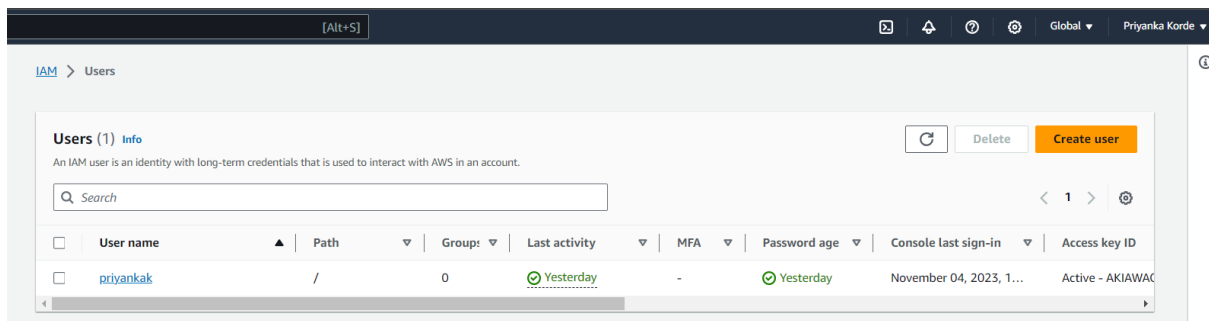
# IMPLEMENTATION

## 3.1) Steps:

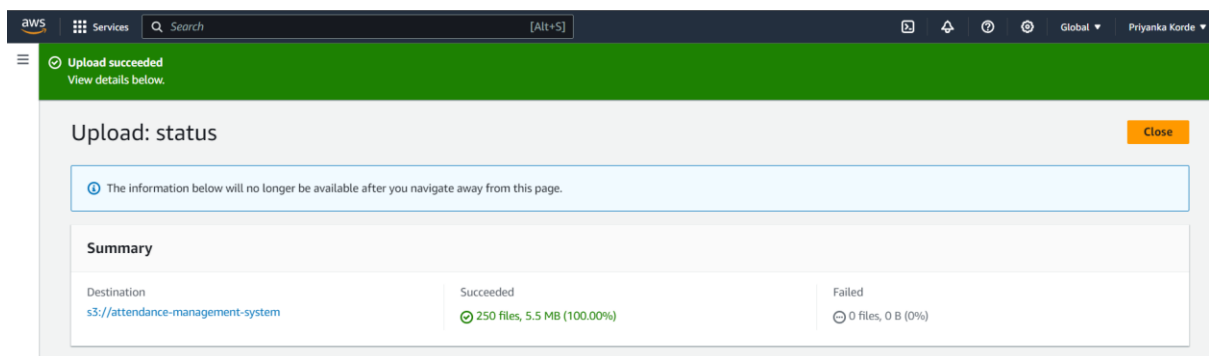
1) First we will create a training dataset in our local system. We will take a min 50-60 images of faces for each person.

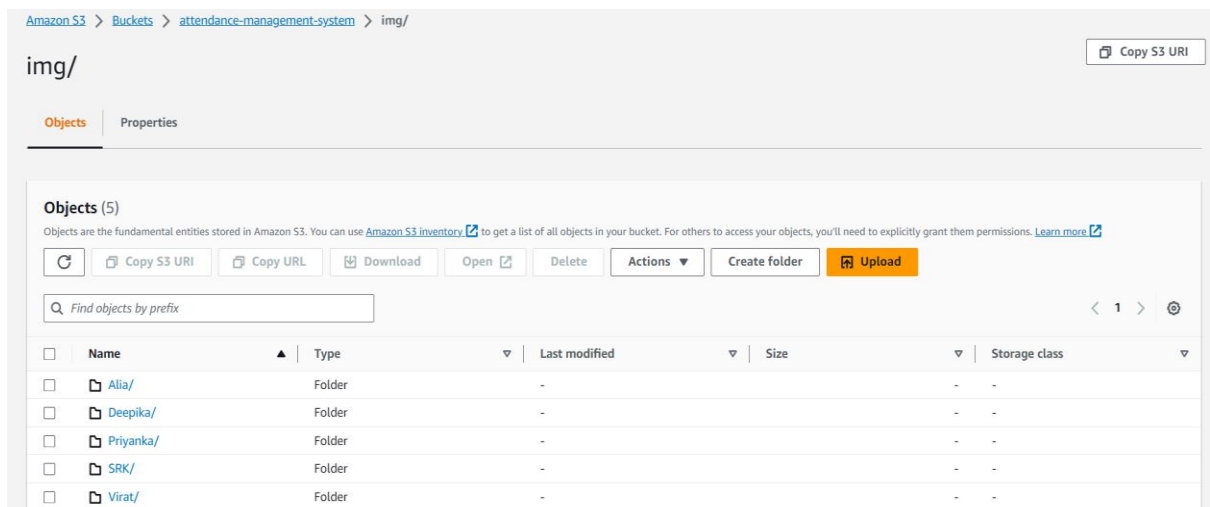
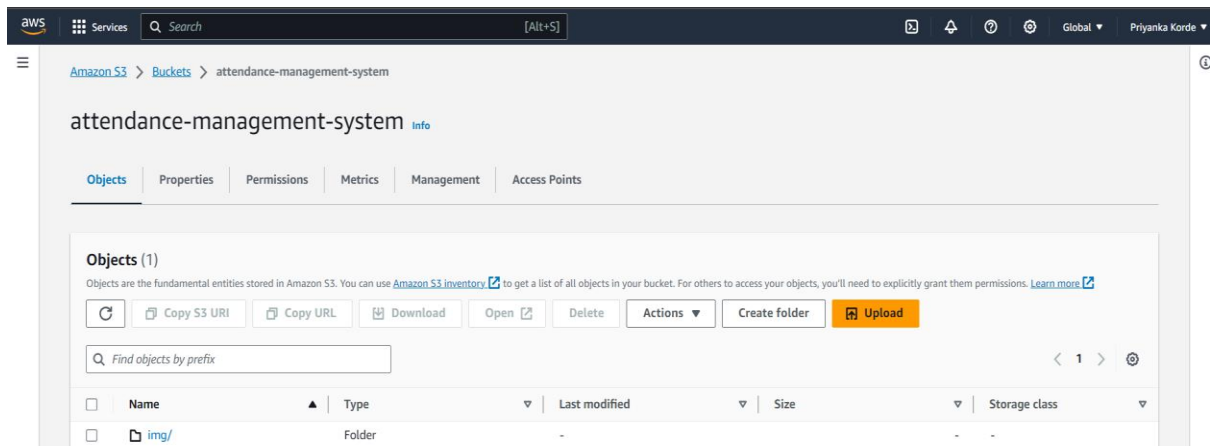


2) In AWS Account we will create one IAM User. And define policy → Do AWS Login using IAM User.



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

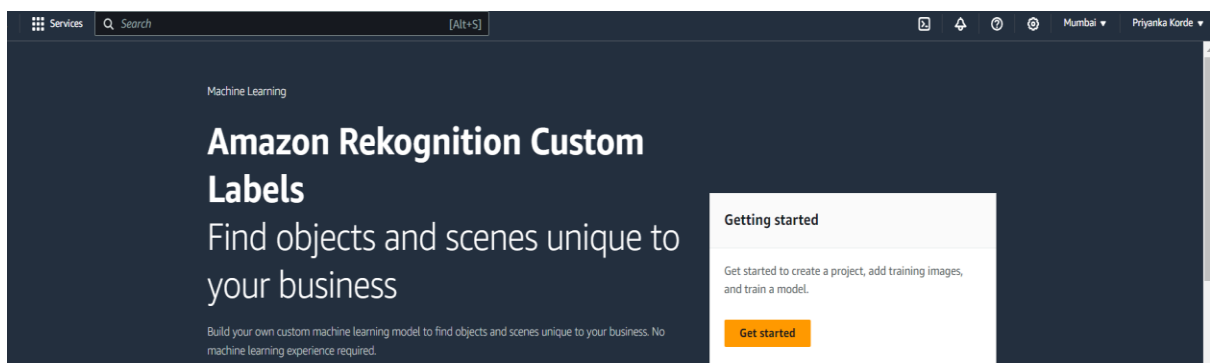




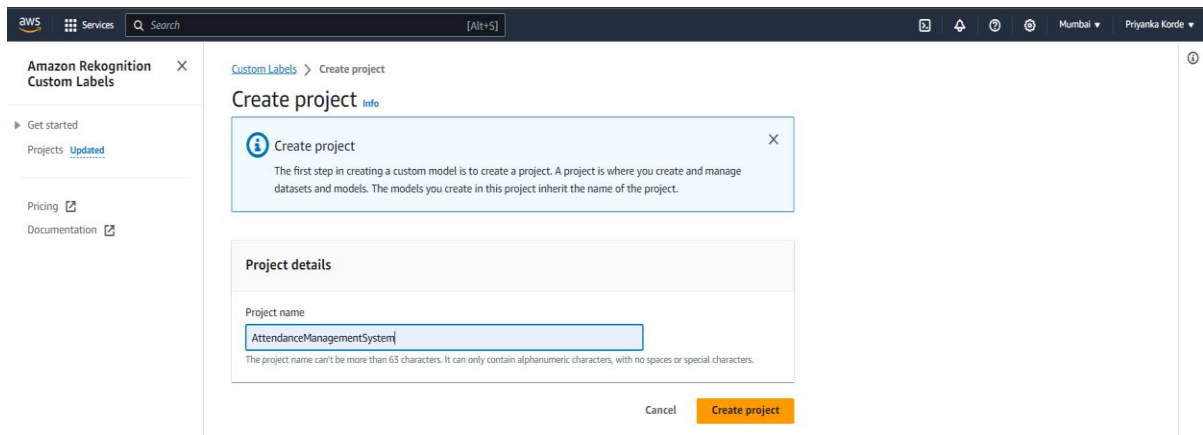
4) Go to AWS Rekognition Service in the same region where you have created S3 bucket.

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.

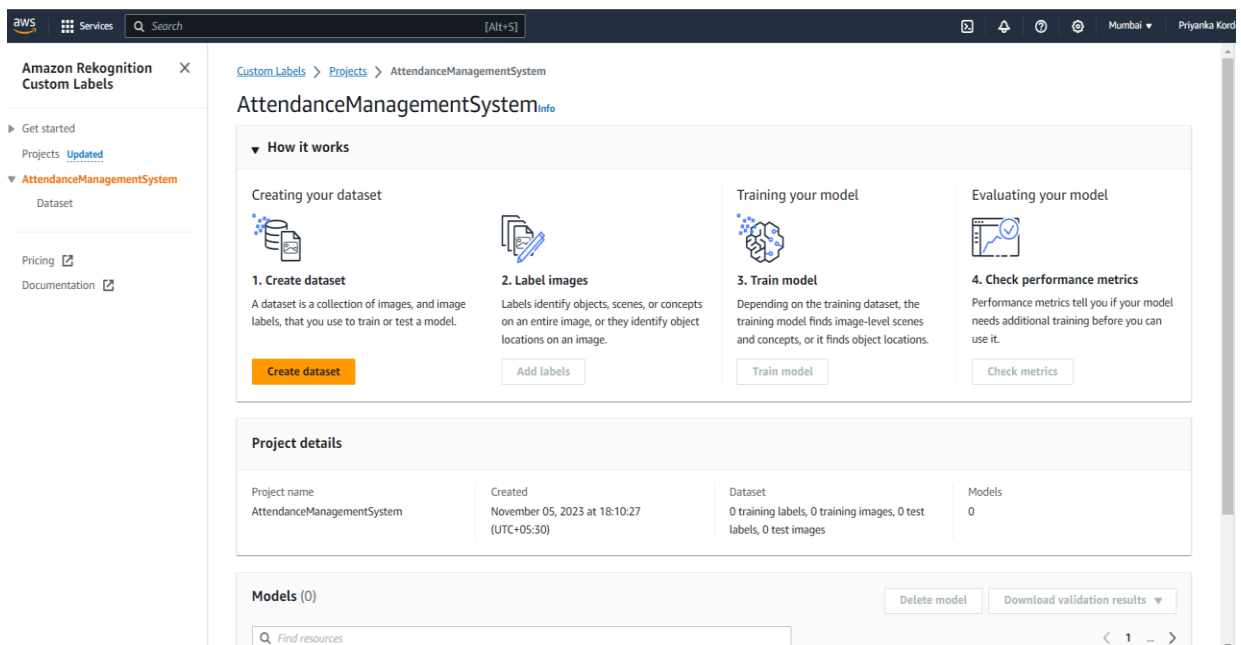
Without this S3 bucket you can't use recognition. Click on Create S3 bucket.



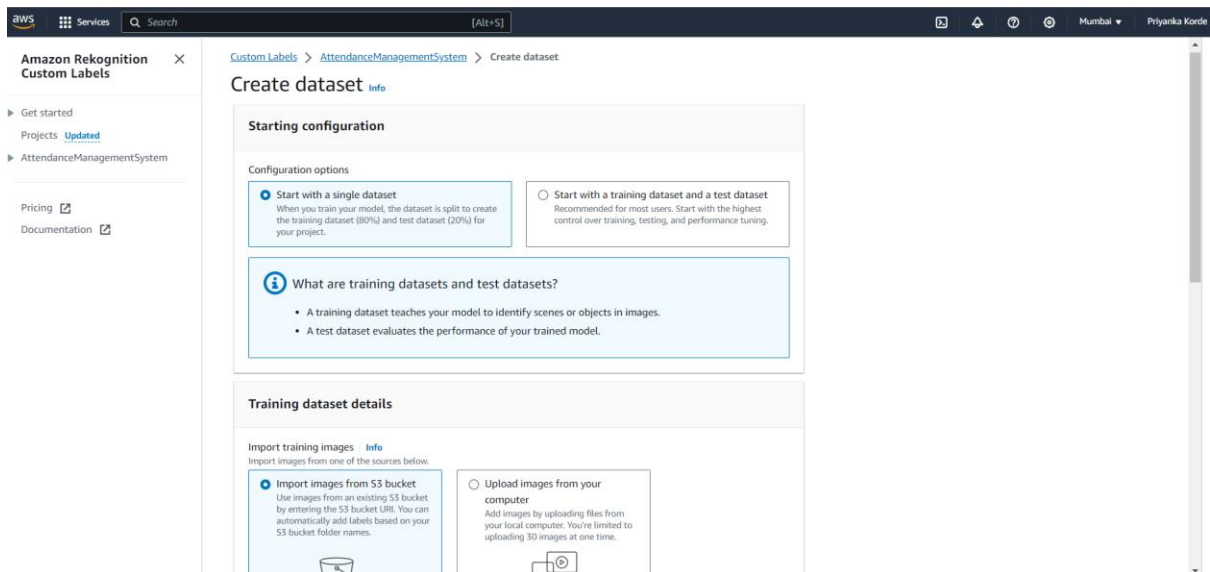




5) In Custom Labels → Project → Create Project → give Project Name “AttendanceManagementSystem”.



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.



#### S3 URI

`s3://attendance-management-system/img/`

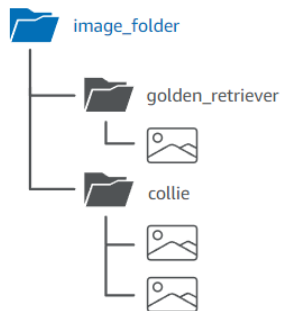
Supported image formats: JPG, PNG. Maximum images per dataset: 250,000. Maximum image size: 15 MB, Minimum size (px): 64 x 64. Maximum size (px): 4096 x 4096. Images must have the same dimensions.

For best results, we recommend uploading images from folders within the **S3 bucket** created for you during first-time setup.

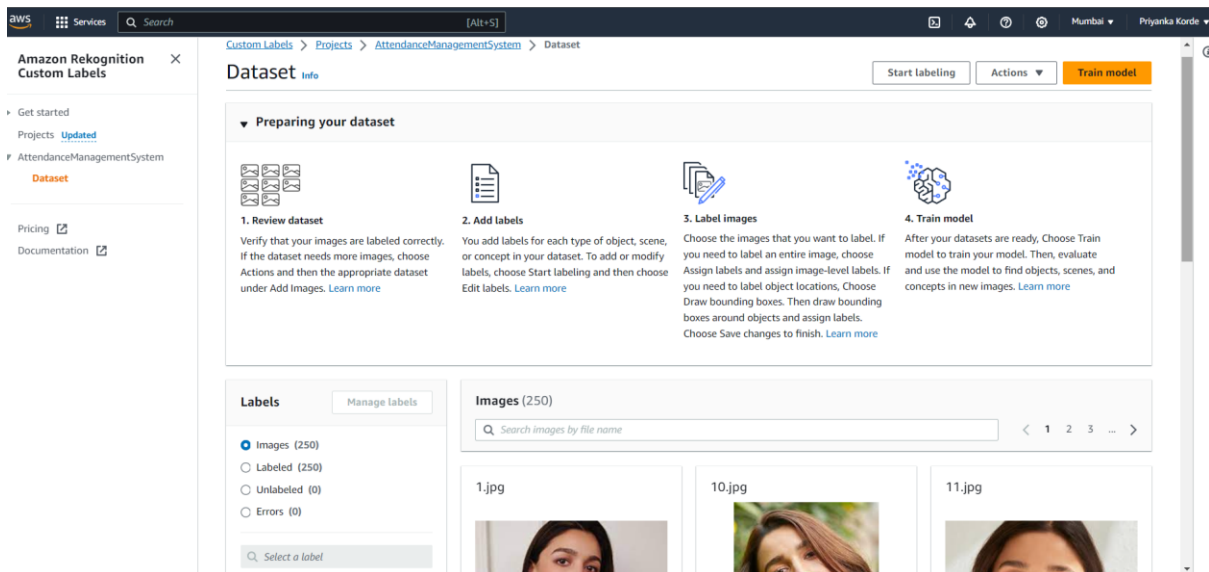
#### Automatic labeling

If you've organized the images in your S3 bucket by folder name (/Golden-Retriever/01.jpeg), Custom Labels can automatically label these images.

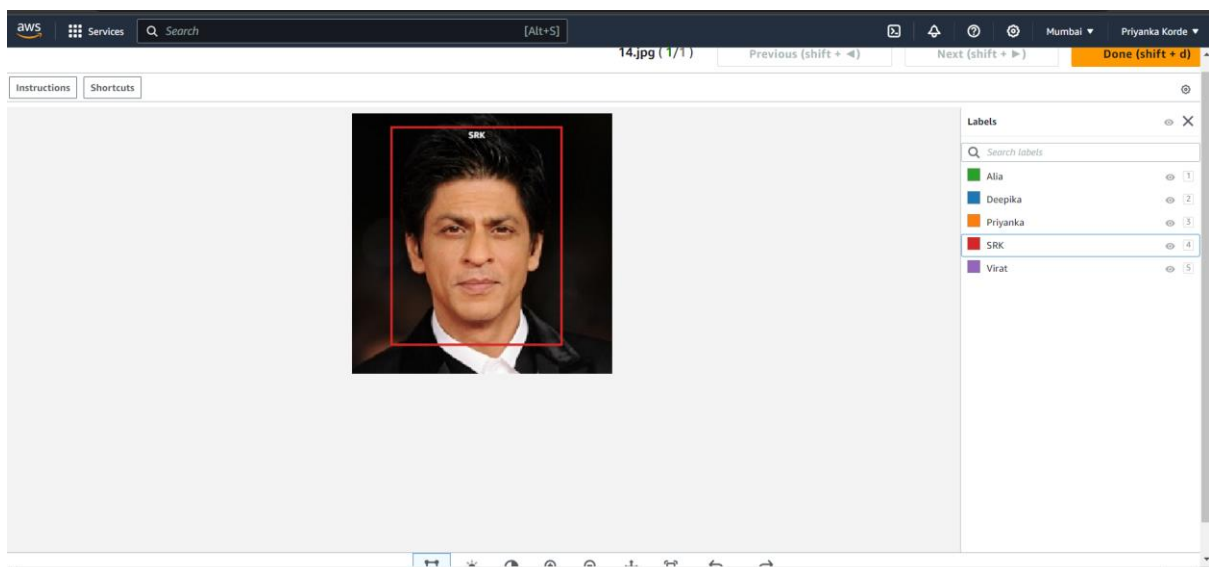
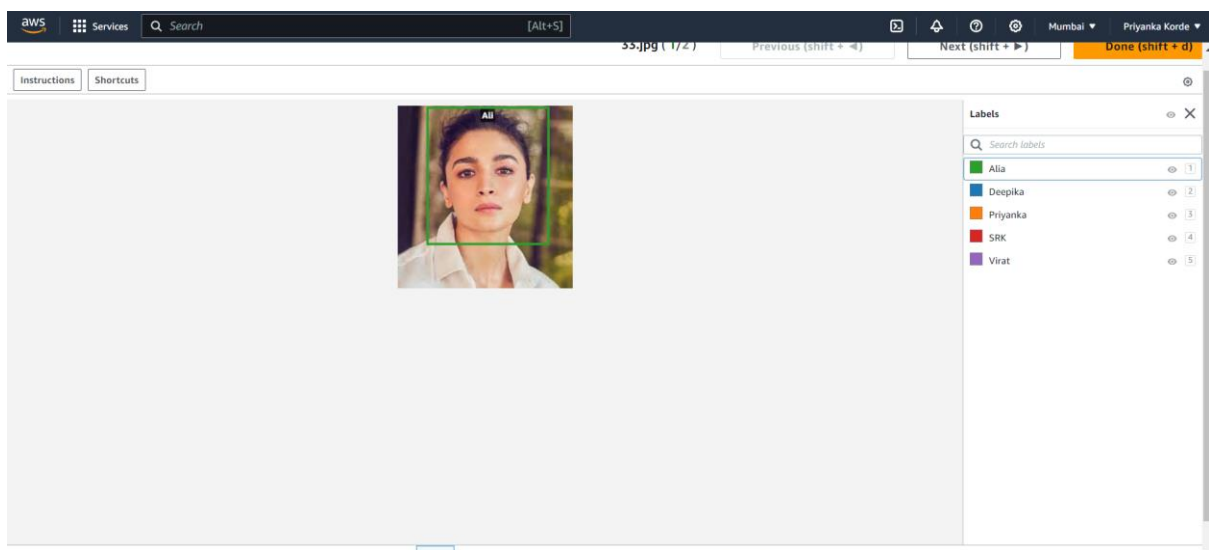
- ☒ Automatically assign image-level labels to images based on the folder name



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.

**Training details** [Info](#)

Choose project  
Amazon Rekognition Custom Labels trains a new version of the model within the project you choose.

**Tags** [Info](#)  
A tag is a label that you can assign to your model. Each tag consists of a key and an optional value.

No tags associated with the resource.

You can add up to 50 more tags.

**Image Data Encryption**

Your data is encrypted by default with a key that AWS owns and manages for you. To choose a different key, customize your encryption settings. [Learn More](#)

☐ Customize encryption settings (advanced)

**Models (1)**

<input type="checkbox"/>	Name	Date created	Training dataset	Test dataset	Model performance (F1 score)	Model status	Status message
<input type="checkbox"/>	AttendanceManagementSystem.2023-11-05T18.37.57	November 05, 2023			0.839	TRAINING_COMPLETED	The model is ready to run.

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.

[DynamoDB](#) > [Tables](#) > Create table

**Create table**

**Table details** [Info](#)  
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

**Table name**  
This will be used to identify your table.

Between 3 and 255 characters, containing only letters, numbers, underscores (\_), hyphens (-), and periods (.).

**Partition key**  
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

1 to 255 characters and case sensitive.

**Sort key - optional**  
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

1 to 255 characters and case sensitive.

✓ The AttendanceSystem table was created successfully.

DynamoDB > Tables

Tables (1) Info

Find tables by table name

Any tag key

Any tag value

< 1 >

Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mo...
AttendanceSystem	Active	Name (S)	-	0	Off	Provisioned (5)	Provisioned (5)

Form JSON view

Create item

You can add, remove, or edit the attributes of an item. You can nest attributes inside other attributes up to 32 levels deep. [Learn more](#)

Attributes

Add new attribute

Attribute name	Value	Type
Name - Partition key	Alia	String
Rollno	1	Number
Count	0	Number

Cancel Create item

DynamoDB > Explore items > AttendanceSystem

AttendanceSystem

Autopreview View table details

Scan or query items

Expand to query or scan items.

Completed. Read capacity units consumed: 0.5

Items returned (5)

< 1 >

Name (String)	Count	Rollno
Virat	0	5
SRK	0	4
Priyanka	0	3
Deepika	0	2
Alia	0	1

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.

**Create function** [Info](#)

AWS Serverless Application Repository applications have moved to [Create application](#).

☒ **Author from scratch**  
 Start with a simple Hello World example.

☐ **Use a blueprint**  
 Build a Lambda application from sample code and configuration presets for common use cases.

☐ **Container image**  
 Select a container image to deploy for your function.

---

**Basic information**

**Function name**  
 Enter a name that describes the purpose of your function.  
 AttendFunction  
 Use only letters, numbers, hyphens, or underscores with no spaces.

**Runtime** [Info](#)  
 Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.  
 Python 3.10

**Architecture** [Info](#)  
 Choose the instruction set architecture you want for your function code.  
☒ **x86\_64**  
☐ arm64

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.

**Add trigger**

**Trigger configuration** [Info](#)

API Gateway  
 aws api application-services backend HTTP REST serverless

Add an API to your Lambda function to create an HTTP endpoint that invokes your function. API Gateway supports two types of RESTful APIs: HTTP APIs and REST APIs. [Learn more](#)

**Intent**  
 Use an existing api or have us create one for you.  
☒ **Create a new API**  
☐ Use existing API

**API type**

☐ **HTTP API**  
 Build low-latency and cost-effective REST APIs with built-in features such as OIDC and OAuth2, and native CORS support.

☒ **REST API**  
 Develop a REST API where you gain complete control over the request and response along with API management capabilities.

**Security**  
 Configure the security mechanism for your API endpoint.  
 Open

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.

The screenshot shows the AWS IAM console with a green notification banner: "Policies have been successfully attached to role." Below this, the "Permissions" tab is active, displaying a table of attached policies:

Policy name	Type	Attached entities
AdministratorAccess	AWS managed - job function	2
AmazonDynamoDBFullAccess	AWS managed	2
AWSLambdaBasicExecutionRole-dc2c4899-a681...	Customer managed	1

Below the IAM console, the AWS API Gateway console is shown. The "Configuration" tab is active, displaying the "Triggers (1)" section. A single trigger is listed:

- Trigger:** API Gateway: AttendFunction-API
- API endpoint:** <https://a8upz1toh1.execute-api.ap-south-1.amazonaws.com/default/AttendFunction>

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

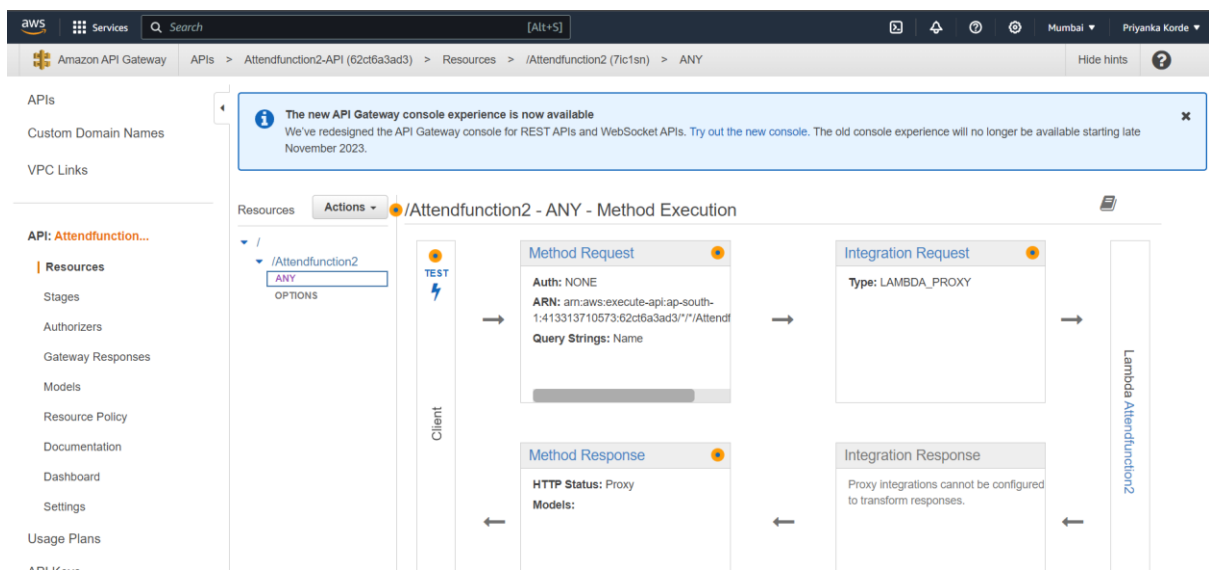
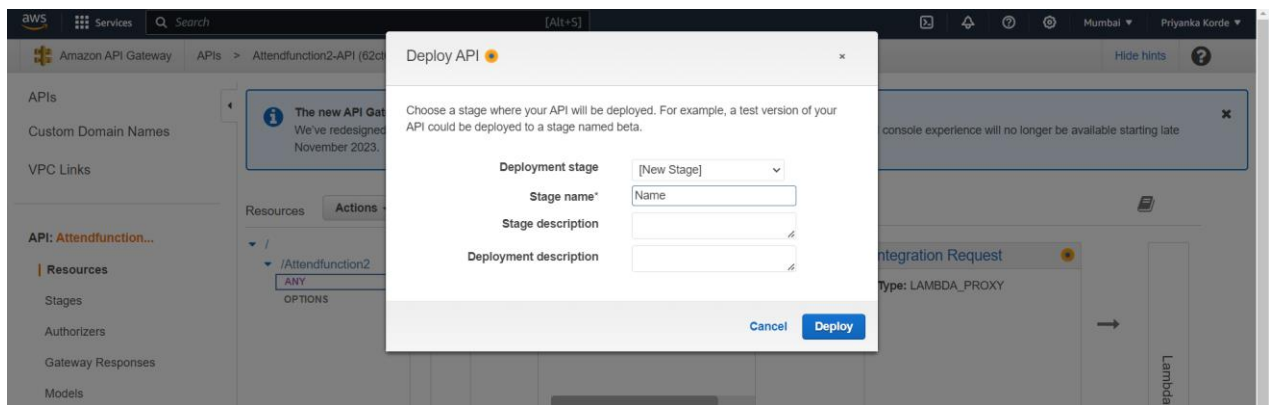
The screenshot shows the AWS API Gateway console. A notification banner states: "The new API Gateway console experience is now available. We've redesigned the API Gateway console for REST APIs and WebSocket APIs. Try out the new console. The old console experience will no longer be available starting late November 2023."

The breadcrumb trail is: Amazon API Gateway > APIs > AttendFunction-API (a8upz1toh1) > Resources > /AttendFunction (6cdow9) > ANY.

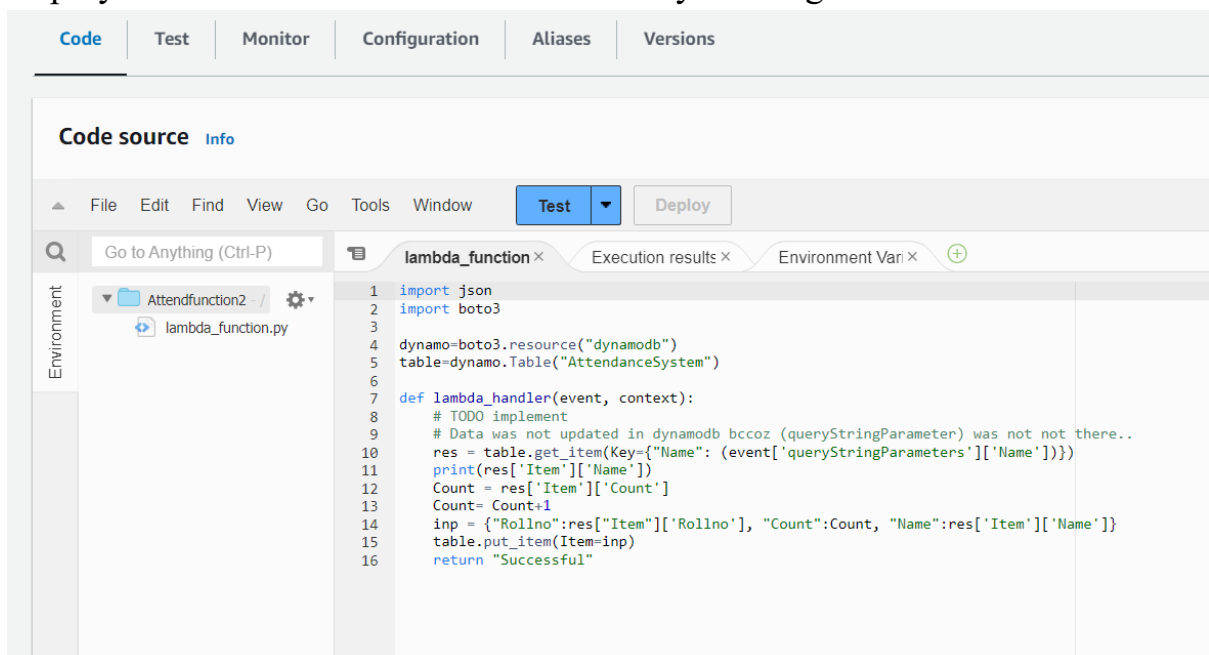
The "Method Execution" tab is active, showing the "Settings" section for the "/AttendFunction - ANY - Method Request". The settings are:

- Authorization:** NONE
- Request Validator:** NONE
- API Key Required:** false
- URL Query String Parameters:**

Name	Required	Caching
Name		
- HTTP Request Headers:** (collapsed)



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





Configure test event

A test event is a JSON object that mocks the structure of requests emitted by AWS services to invoke a Lambda function. Use it to see the function's invocation result.

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

☒ Create new event

☐ Edit saved event

Event name

myevents

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

☒ Private

This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [Learn more](#)

☐ Shareable

This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)

Template - optional

apigateway-aws-proxy

Event JSON

Format JSON

1 - {

2   "body": "eyJ0ZXN0Ijo1Ym9keS99",

Cancel

Invoke

Save

Event name

myevent

Event JSON

Format JSON

1 - {

2   "body": "eyJ0ZXN0Ijo1Ym9keS99",

3   "resource": "/{proxy+}",

4   "path": "/path/to/resource",

5   "httpMethod": "POST",

6   "isBase64Encoded": true,

7   "queryStringParameters": {

8     "Name": "Alia",

9     "Rollno": 3,

10    "Count": 1

11   },

12   "pathParameters": {

13     "proxy": "/path/to/resource"

14   },

15   "stageVariables": {

16     "baz": "qux"

17   },

18   "headers": {

19     "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8",

20     "Accept-Encoding": "gzip, deflate, sdch",

21     "Accept-Language": "en-US,en;q=0.8",

22     "Cache-Control": "max-age=0",

23     "CloudFront-Forwarded-Proto": "https",

24     "CloudFront-Is-Desktop-Viewer": "true",

25     "CloudFront-Is-Mobile-Viewer": "false",

26     "CloudFront-Is-SmartTV-Viewer": "false",

27     "CloudFront-Is-Tablet-Viewer": "false",

28     "CloudFront-Viewer-Country": "US",

29     "Host": "1234567890.execute-api.us-east-1.amazonaws.com",

30   }

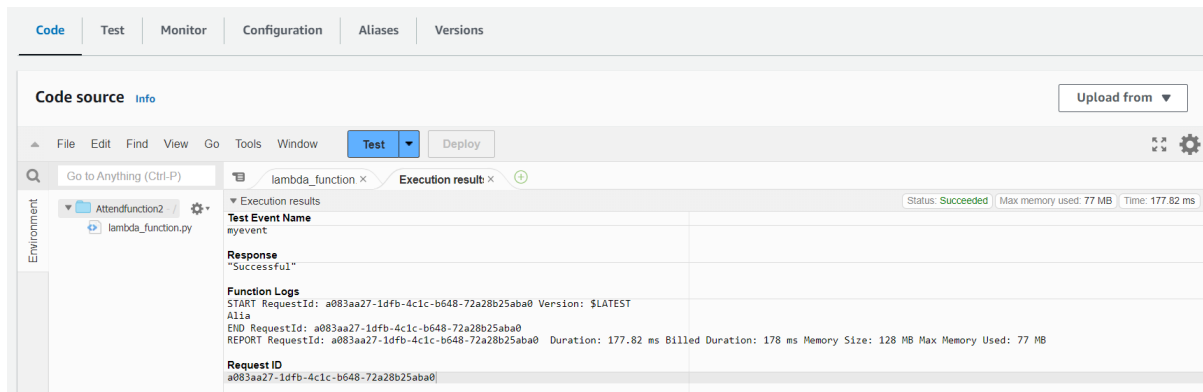
31   }

Cancel

Invoke

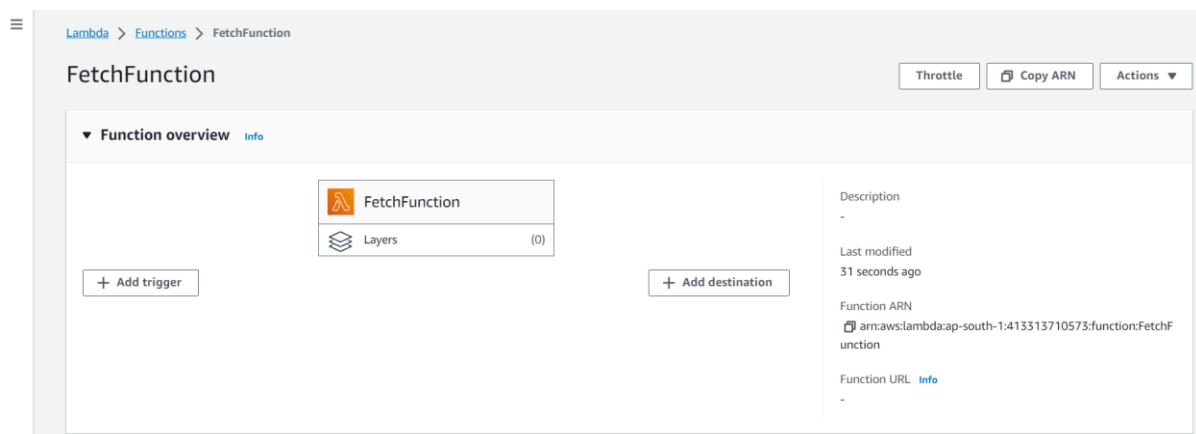
Save

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.

**Trigger configuration** [Info](#)

**API Gateway**  
aws api application-services backend HTTP REST serverless

Add an API to your Lambda function to create an HTTP endpoint that invokes your function. API Gateway supports two types of RESTful APIs: HTTP APIs and REST APIs. [Learn more](#)

**Intent**  
Use an existing api or have us create one for you.

☒ Create a new API  
☐ Use existing API

**API type**

☒ **HTTP API**  
Build low-latency and cost-effective REST APIs with built-in features such as OIDC and OAuth2, and native CORS support.

☐ **REST API**  
Develop a REST API where you gain complete control over the request and response along with API management capabilities.

**Security**  
Configure the security mechanism for your API endpoint.

Open

► **Additional settings**

Lambda will add the necessary permissions for Amazon API Gateway to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

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.

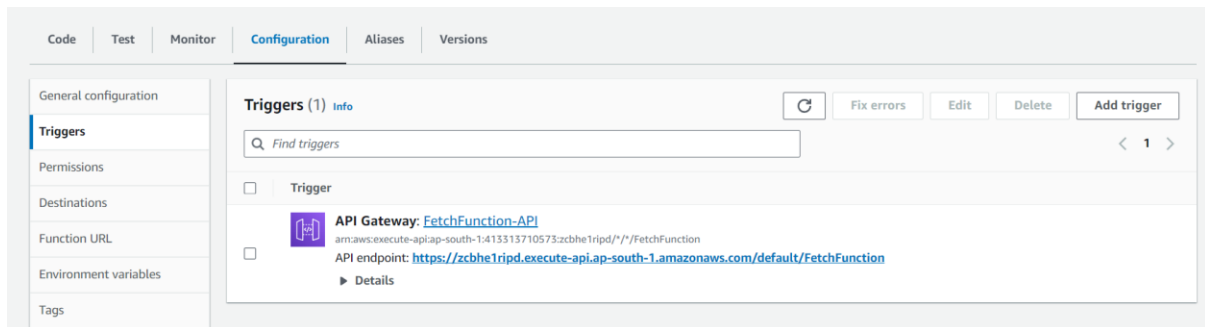
**Summary**

Creation date: November 05, 2023, 20:27 (UTC+05:30)  
Last activity: -  
ARN: arn:aws:iam::413313710573:role/service-role/FetchFunction-role-lafwceh5  
Maximum session duration: 1 hour

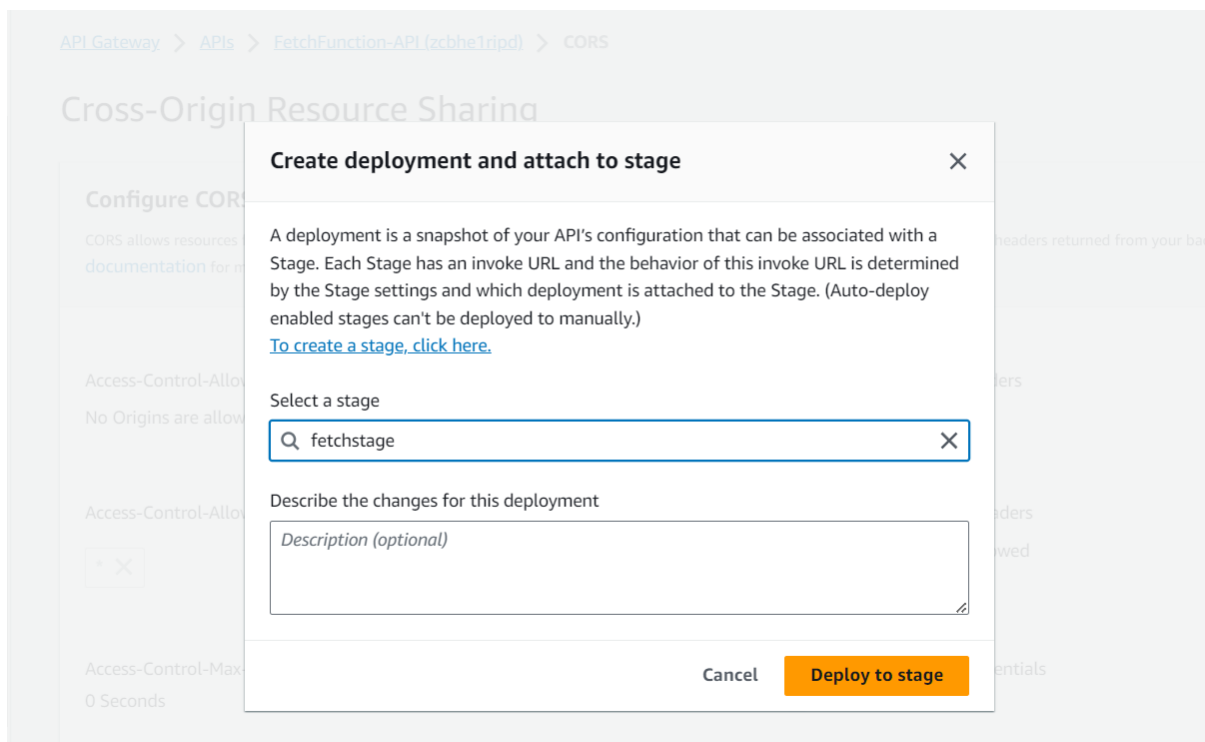
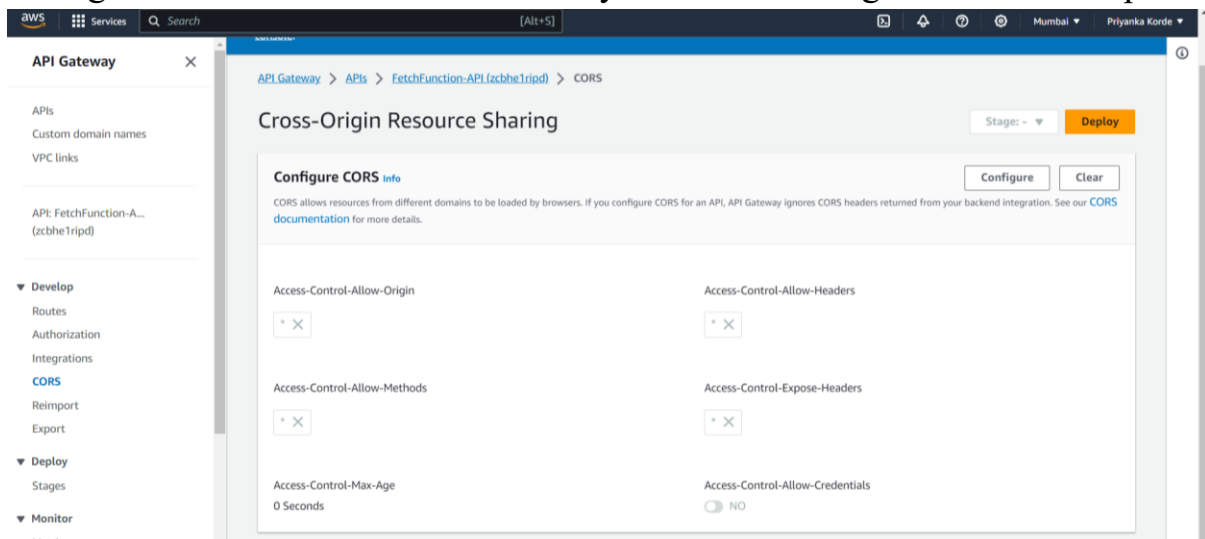
**Permissions policies (3)** [Info](#)  
You can attach up to 10 managed policies.

Policy name	Type	Attached entities
AdministratorAccess	AWS managed - job function	4
AmazonDynamoDBFullAccess	AWS managed	4
AWSLambdaBasicExecutionRole-ee63bba9-f04e-...	Customer managed	1

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.



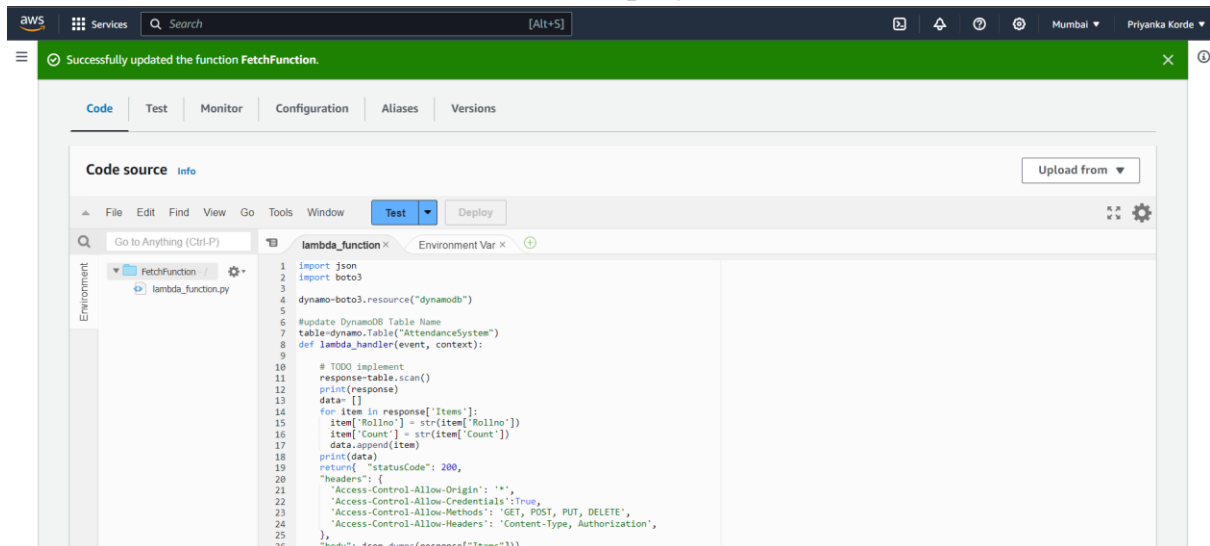
## NOTE :

- 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.

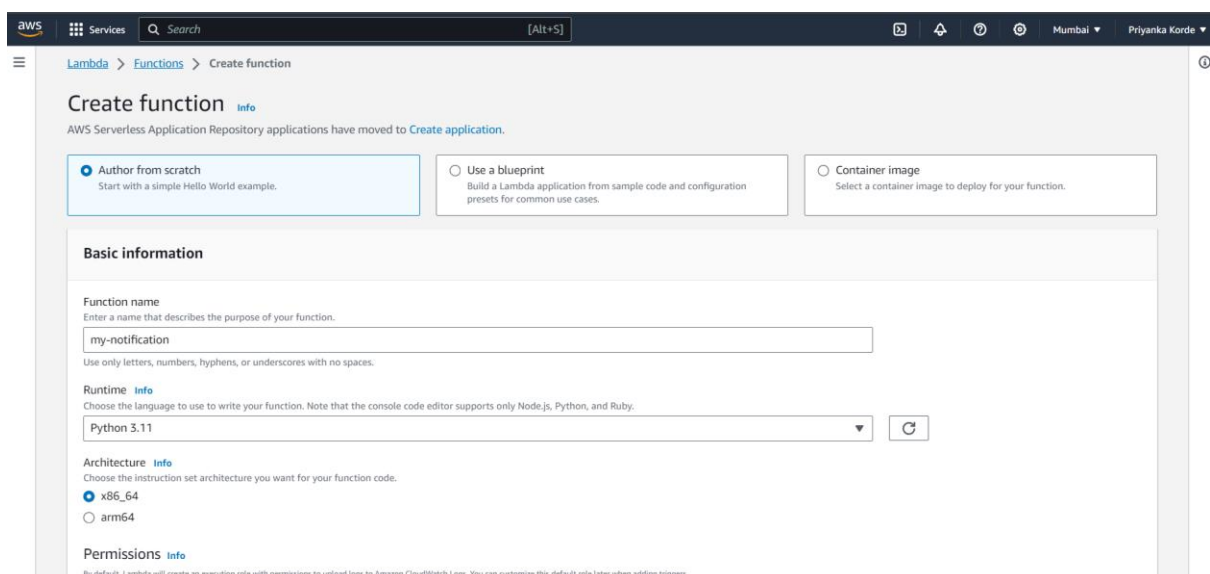
Add code of the Lambda Function → Deploy → Test



## 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

The screenshot shows the AWS IAM console for the role 'notification-role-fotg956j'. The left sidebar shows the 'Identity and Access Management (IAM)' menu with options like Dashboard, Access management, Access reports, and Related consoles. The main content area shows the role's summary and a list of permissions policies.

**Summary**

- Creation date: November 06, 2023, 21:03 (UTC+05:30)
- ARN: `arn:aws:iam::413313710573:role/service-role/notification-role-fotg956j`
- Last activity: 1 hour ago
- Maximum session duration: 1 hour

**Permissions policies (5)**

You can attach up to 10 managed policies.

Policy name	Type	Attached entities
<a href="#">AdministratorAccess</a>	AWS managed - job function	6
<a href="#">AmazonS3FullAccess</a>	AWS managed	1
<a href="#">AmazonSNSFullAccess</a>	AWS managed	1
<a href="#">AWSLambda_FullAccess</a>	AWS managed	2
<a href="#">AWSLambdaBasicExecutionRole-07fa7010-e88a-...</a>	Customer managed	1

## Step 3: Add trigger to lambda function

The screenshot shows the AWS Lambda console for a function named 'attendance-management-system'. The 'Add trigger' configuration is shown, with the trigger type set to 'S3' and the bucket set to 's3/attendance-management-system'. The event types are set to 'All object create events'. The prefix is set to 'images/' and the suffix is set to '.jpg'.

**Trigger configuration**

- Trigger type: S3
- Bucket: s3/attendance-management-system
- Event types: All object create events
- Prefix: images/
- Suffix: .jpg

**Triggers (1)**

Trigger
<b>S3: attendance-management-system</b> arn:aws:s3::attendance-management-system

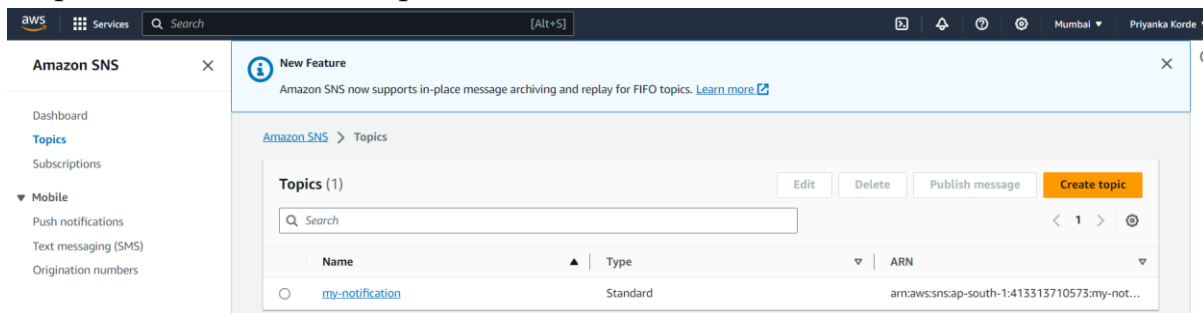
**Details**

- Bucket arn: `arn:aws:s3::attendance-management-system`
- Event types: `s3:ObjectCreated:*`
- Notification name: `a44ca4a8-3830-45f4-b294-55dc06fad8be`
- Service principal: `s3.amazonaws.com`
- Source account: `413313710573`
- Statement ID: `lambda-4f5b682a-4b60-4f14-a66a-d8734409c5bb`
- Suffix: `.jpg`

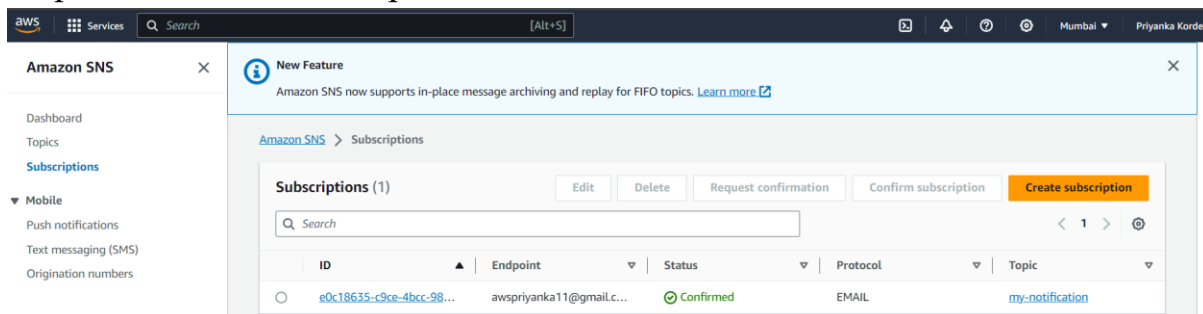
## Step 4: Add python code to the function and deploy

```
1 import boto3
2
3 topic_arn = "arn:aws:sns:ap-south-1:413313710573:my-notification"
4 def send_sns(message, subject):
5     try:
6         client = boto3.client("sns")
7         result = client.publish(TopicArn=topic_arn, Message=message, Subject=subject)
8         if result['ResponseMetadata']['HTTPStatusCode'] == 200:
9             print(result)
10            print("Notification send successfully..!!!")
11            return True
12    except Exception as e:
13        print("Error occured while publish notifications and error is : ", e)
14        return True
15
16 def lambda_handler(event, context):
17     print("event collected is {}".format(event))
18     for record in event['Records']:
19         s3_bucket = record['s3']['bucket']['name']
20         print("Bucket name is {}".format(s3_bucket))
21         s3_key = record['s3']['object']['key']
22         print("Bucket key name is {}".format(s3_key))
23         from_path = "s3://{}/{}".format(s3_bucket, s3_key)
24         print("from path {}".format(from_path))
25         message = "The Attendance has been marked Successfully..."
26         subject = "Attendance update"
27         SNSResult = send_sns(message, subject)
28         if SNSResult:
29             print("Notification Sent..")
30             return SNSResult
31         else:
32             return False
```

## Step 5: Create the SNS Topic



## Step 6: Create the subscriptions



## Step 7: Confirm the subscription you created



Simple Notification Service

### Subscription confirmed!

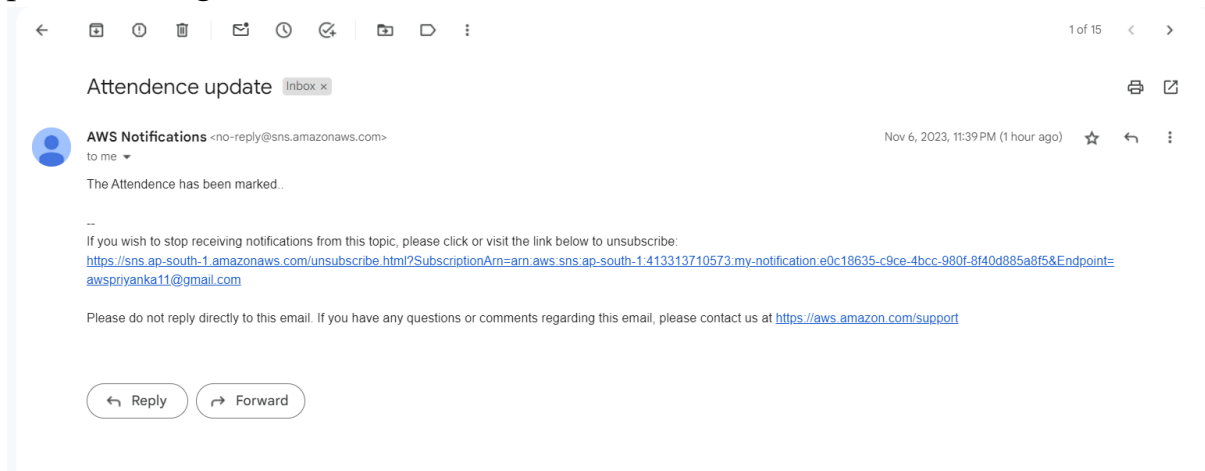
You have successfully subscribed.

Your subscription's id is:

arn:aws:sns:ap-south-1:413313710573:my-notification:e0c18635-c9ce-4bcc-980f-8f40d885a8f5

If it was not your intention to subscribe, [click here to unsubscribe](#).

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.

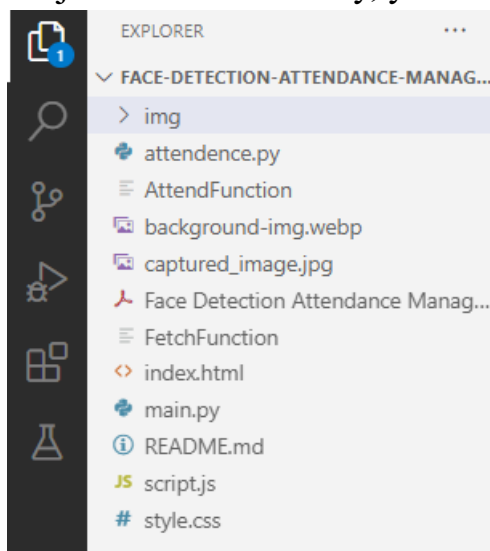


Here all steps are completed for Project.

### 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="AKIAWA03KXHWQAXPFSQZ",
                      aws_secret_access_key="yh08nzmewJM+oqZAT0LL19pQ8A5STtoZLpnVRmeE",
                      #aws_session_token="FwoGZXIvYXdzECEaDIYUDFobtJnX2oDJVCLBAYop3p7T11dVZL/rjE4nmQQNBQEGYMSXua9XMWaiSGT1v1Giv0j4Txt0883Mrmz4zA1N2RAfXPk4QK6MxHEuEamQ3U4AgrZ4qA4AVv+uvMuGLWmVPwqUk/uK61R/kE3cNN5Bs3qzWY0zZ22z1RB8IT8YDxS81Wz5tZT/rRBXEGODdV6oIR8LIYixYoyBf13hPwxTpqS/IrOzTcFnFbuoLYZQvLH2IGzf087tsV2bL56CoX62V9eAbv8VORF1RlGowgIouvqB/AUyLXdmJoVk+H0LePDblDL1YvDU3e7po71EVq9DW+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="AKIAWA03KXHWQAXPFSQZ",
                            aws_secret_access_key="yh08nzmewJM+oqZAT0LL19pQ8A5STtoZLpnVRmeE", 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')

```

```

#Recognize 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://a8upzlt0h1.execute-api.ap-south-
1.amazonaws.com/default/AttendFunction" + str

resp = requests.get(url)
print("Attendance 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 Attendance has been marled Successfully...!"
        subject = "Attendance 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("/background.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 :

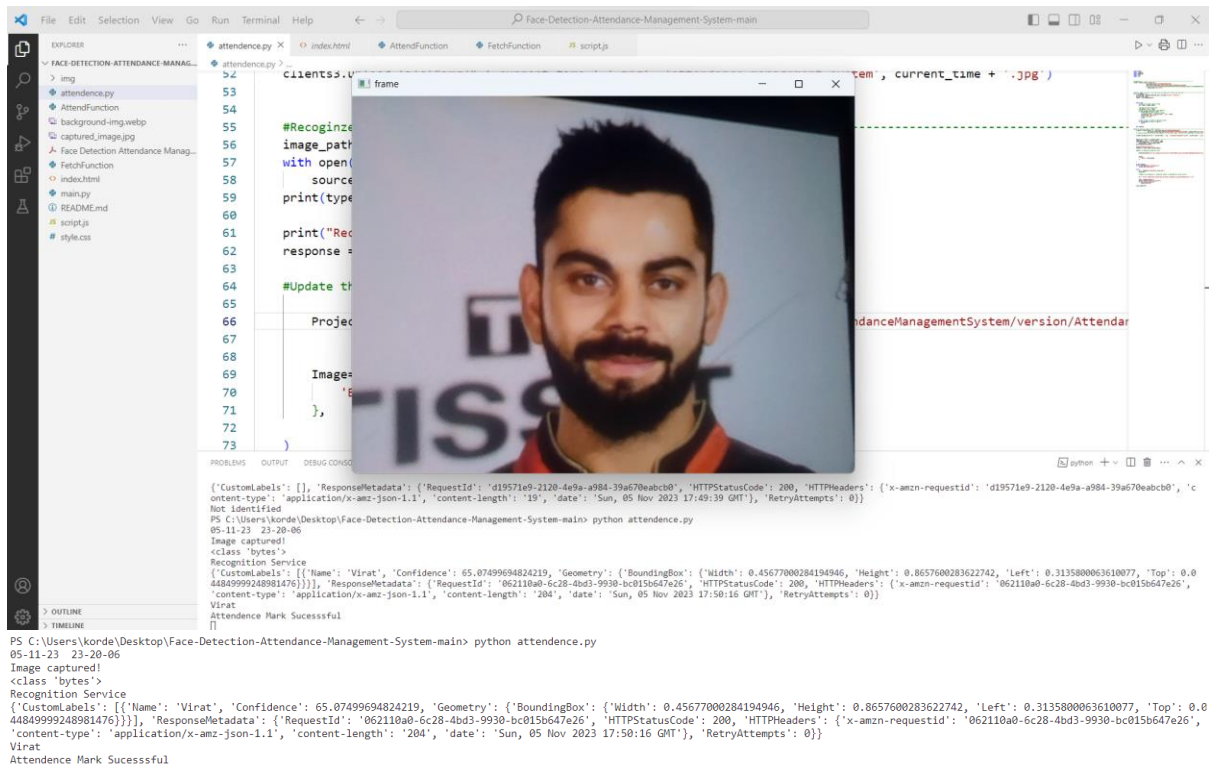
- In script.js code you have to add second API URL. i.e. FetchFunction
- This API we will create in next further steps.

# RESULTS

## 4.1) Output :

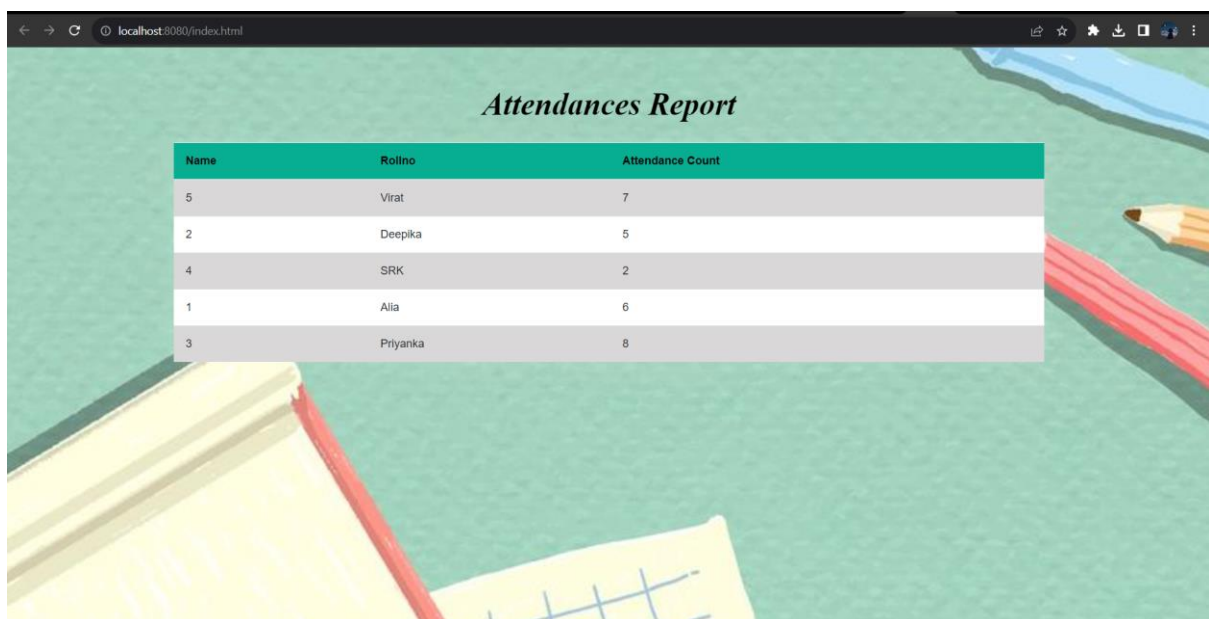
Before running the code make sure you start your model in Amazon Rekognition Custom Labels

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



```
PS C:\Users\korde\Desktop\Face-Detection-Attendance-Management-System-main> python attendance.py
05-11-23 23:20:06
Image captured!
<class 'bytes'>
Recognition Service
{'CustomLabels': [{'Name': 'Virat', 'Confidence': 65.07499694824219, 'Geometry': {'BoundingBox': {'Width': 0.45677000284194946, 'Height': 0.8657600283622742, 'Left': 0.3135800063610077, 'Top': 0.04484999248981476}}}], 'ResponseMetadata': {'RequestId': '062110a0-6c28-4bd3-9930-bc015b647e26', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': '062110a0-6c28-4bd3-9930-bc015b647e26', 'content-type': 'application/x-amz-json-1.1', 'content-length': '204', 'date': 'Sun, 05 Nov 2023 17:50:16 GMT'}, 'RetryAttempts': 0}}
Virat
Attendance Mark Successful
```

## 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](https://github.com/RekhuGopal/PythonHacks/blob/main/AWS_SNS_Lambda_Notification/publishnotification.py)
- [6] <https://www.youtube.com/watch?v=zLL4ZTmDHSI>