Cloud-Based Face Recognition Attendance System

Abstract

The Cloud-Based Face Recognition Attendance System is an AI-driven solution designed to modernize and automate student attendance tracking using facial recognition technology. Built entirely on Amazon Web Services (AWS), the system ensures scalability, security, and high availability. Students can register their facial data and mark attendance via a web portal, with all image processing and identity verification handled by AWS Rekognition. Serverless AWS Lambda functions manage the backend logic, while Amazon S3, DynamoDB, and SNS are used for storage, database management, and real-time notifications, respectively. The solution eliminates manual attendance procedures, minimizes errors, and enhances data security. With a user-friendly interface and real-time processing capabilities, this system offers an efficient and reliable alternative for educational institutions aiming to implement smart attendance management.

1. Introduction

1.1 Project Overview

The Cloud-Based Face Recognition Attendance System is an intelligent, AI-powered solution designed to automate and modernize the process of taking student attendance. Built entirely using scalable and secure AWS cloud services, this system enables students to register their face and mark their attendance in real time through a web portal. By leveraging advanced facial recognition technology and serverless computing, the system ensures efficient processing, high accuracy, and cost-effectiveness for educational institutions.

1.2 Objectives

- Enable students to register their faces through a user-friendly web interface.
- Allow students to mark attendance by uploading a live face image for verification.
- Ensure secure and efficient facial recognition using AWS Rekognition.
- Store student profiles and attendance logs using a scalable NoSQL database.
- Automate the backend processing using AWS Lambda functions.
- Use AWS SNS to notify admins about attendance updates and issues.
- Maintain a highly available and responsive system using AWS services.

Detailed Description of the Modules:

I. Amazon S3 (Simple Storage Service)

Purpose:
Used for storing all face images securely and reliably.

Functions:

- Stores registered student face images and attendance-time face images in separate folders.
- Acts as a trigger source for AWS Lambda functions upon image upload.
- Provides durable and highly available storage for facial image data.

Interaction:
Lambda functions automatically access images from S3 for registration and attendance comparison. S3 is also used as the storage point for EC2-hosted front-end uploads.

II. AWS Rekognition

Purpose:
Provides facial recognition capabilities for registration and attendance matching.

Functions:

- Extracts facial features and indexes them into a Rekognition collection during registration.
- Compares new face images against the collection to verify student identity.
- Returns confidence scores to determine match accuracy.

Interaction:
Triggered by Lambda during both registration and attendance processes. Directly interacts with stored Rekognition collection and returns face match results to Lambda.

III. AWS Lambda

Purpose:
Provides backend processing using a serverless compute model.

Functions:

- Handles student face registration: triggers on S3 image upload, calls Rekognition, updates DynamoDB.
- Handles attendance marking: compares new face with existing collection, logs attendance.
- Sends notification via SNS based on success/failure.

Interaction:
Triggered by S3 events, accesses Rekognition for matching, reads/writes to DynamoDB, and sends alerts via SNS.

IV. Amazon Dynamic DB

Purpose:
Stores structured student and attendance data in a scalable NoSQL format.

Functions:

- Stores student information such as registration number and image reference.
- Records each attendance event with a timestamp for future reference.
- Supports real-time updates with low latency.

Interaction:
Accessed by Lambda functions for both student registration and attendance marking. Each match result is logged here.

V. AWS SNS (Simple Notification Service)

Purpose:
Sends real-time alerts and notifications to admins.

Functions:
- Sends email/SMS to the administrator upon successful or failed attendance/registration.
- Notifies system stakeholders of anomalies or issues instantly.

Interaction:
Invoked by Lambda based on success/failure of face verification. Admin is subscribed to SNS topic for real-time updates.

VI. AWS IAM (Identity and Access Management)

Purpose:
Manages secure access and permissions across AWS services.

Functions:
- Defines roles and policies for Lambda, EC2, Rekognition, etc.
- Ensures services follow least privilege principle.
- Protects sensitive operations and controls user access.

Interaction:
IAM roles are assigned to Lambda, EC2, and other AWS services to ensure secure operations and restricted data access.

VII. Amazon EC2 (Elastic Compute Cloud)

Purpose:
Hosts the web interface used by students for system interaction.

Functions:
- Runs a lightweight web server (Apache/Nginx) to serve HTML/CSS/JS portal.
- Enables students to register and mark attendance through browser upload.
- Acts as the entry point to the entire system.

Interaction:
Uploads face images to S3, where they trigger backend Lambda functions. Can communicate with backend APIs if extended.

VIII. AWS CloudWatch

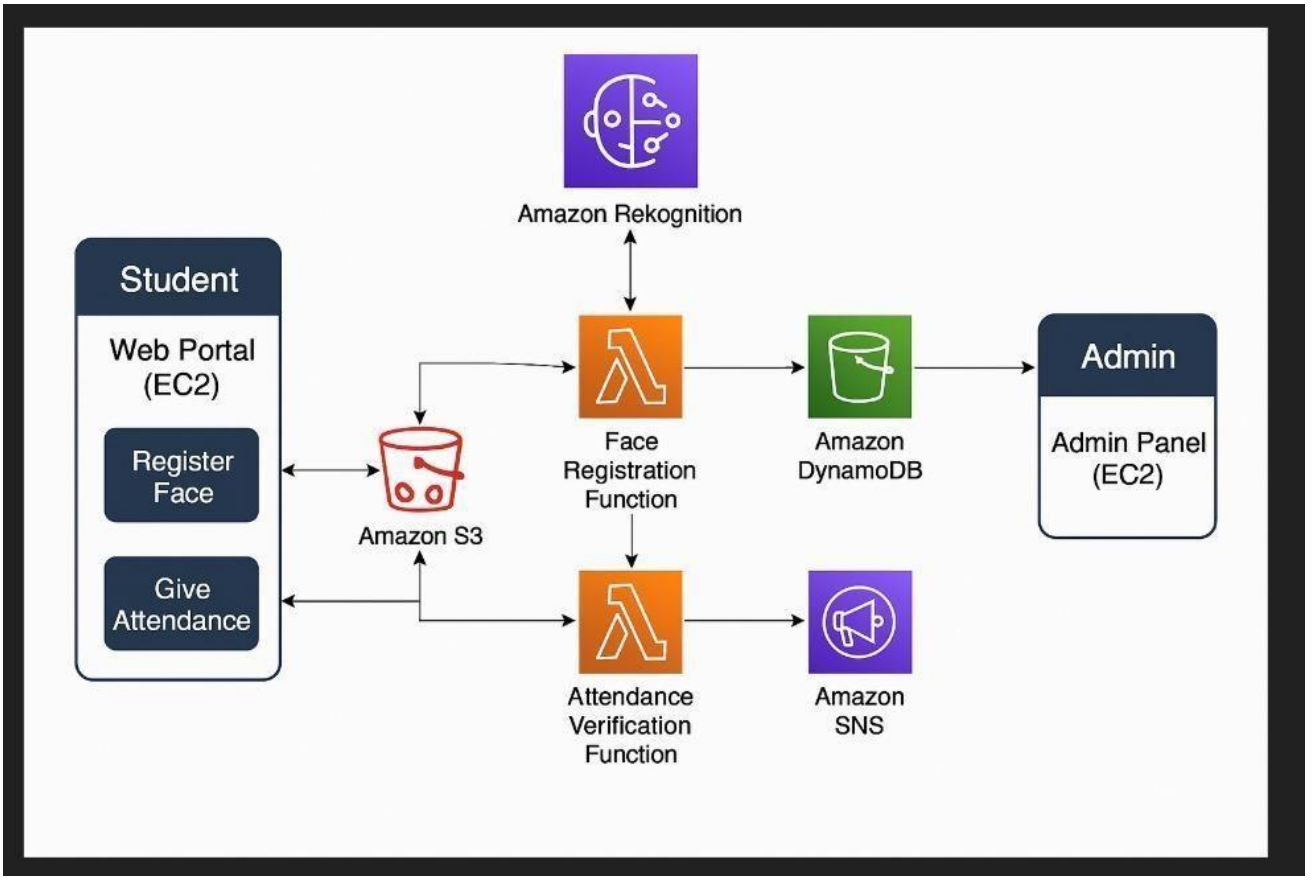
Purpose:
Monitors and logs system activity and performance.

Functions:

- Logs execution details of Lambda functions.
- Tracks error rates and processing times.
- Provides alerts for unexpected behavior or failures.

Interaction:
Integrates with Lambda, EC2, and other services. Enables real-time system monitoring and debugging through logs and alarms.

2. Architecture & Workflow



2.1 Workflow Steps

1. Student Face Registration via Web Portal

2. Image Upload to Amazon S3
3. Trigger Lambda for Face Indexing
4. Face Indexed in AWS Rekognition
5. Student Details Saved to DynamoDB
6. Attendance Image Upload via Web Portal
7. Lambda Invoked for Face Comparison
8. Attendance Logged and Admin Notified via SNS.

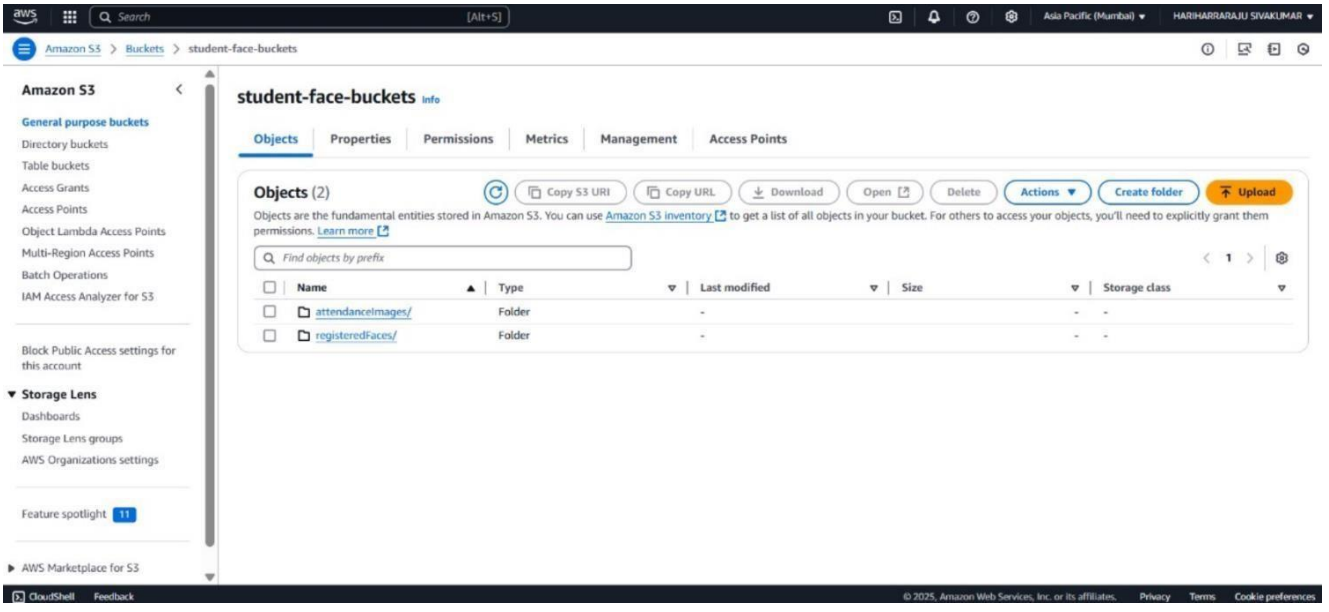
3. Implementation

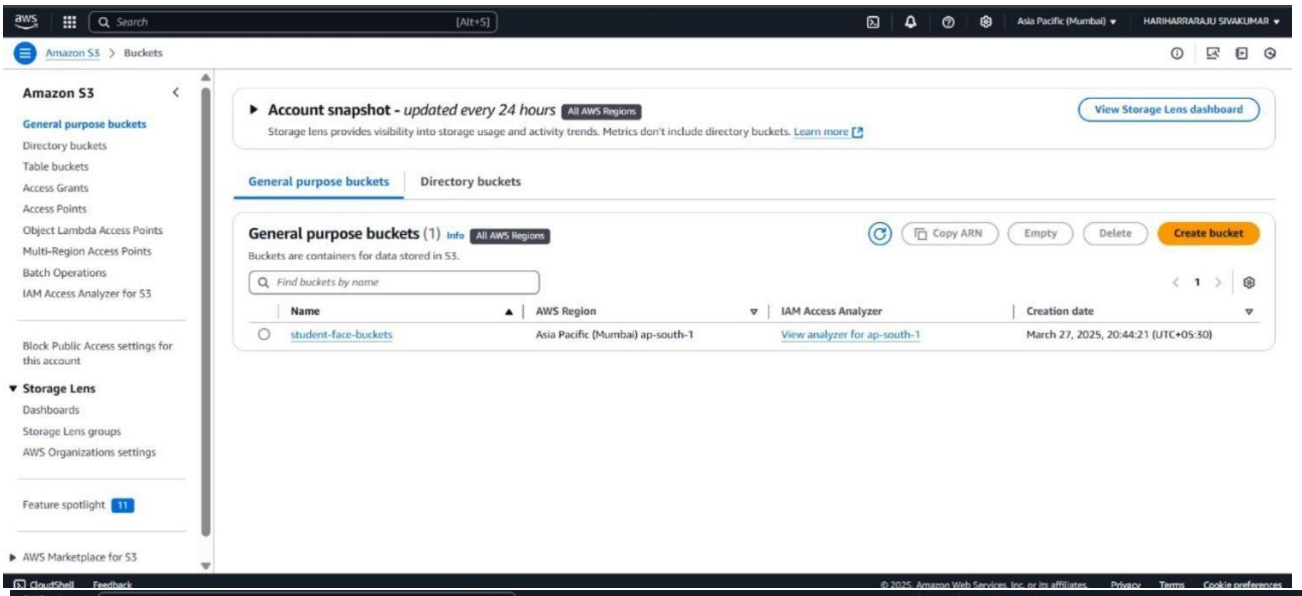
AWS Services Used

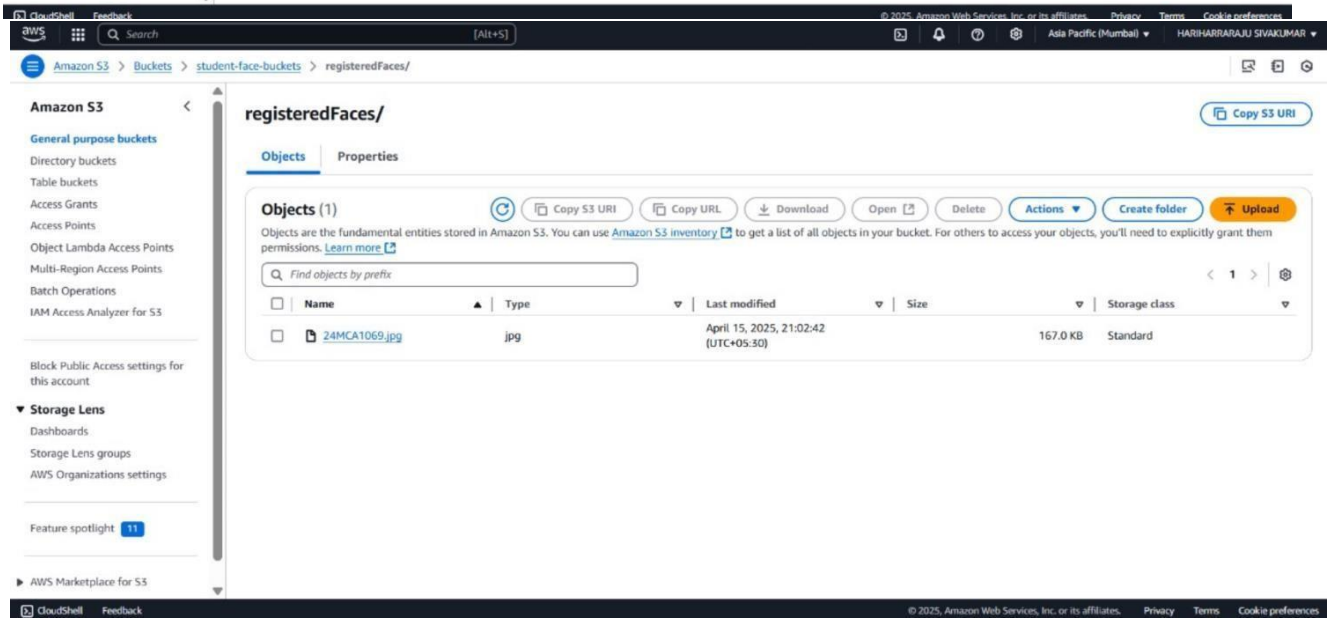
Amazon S3

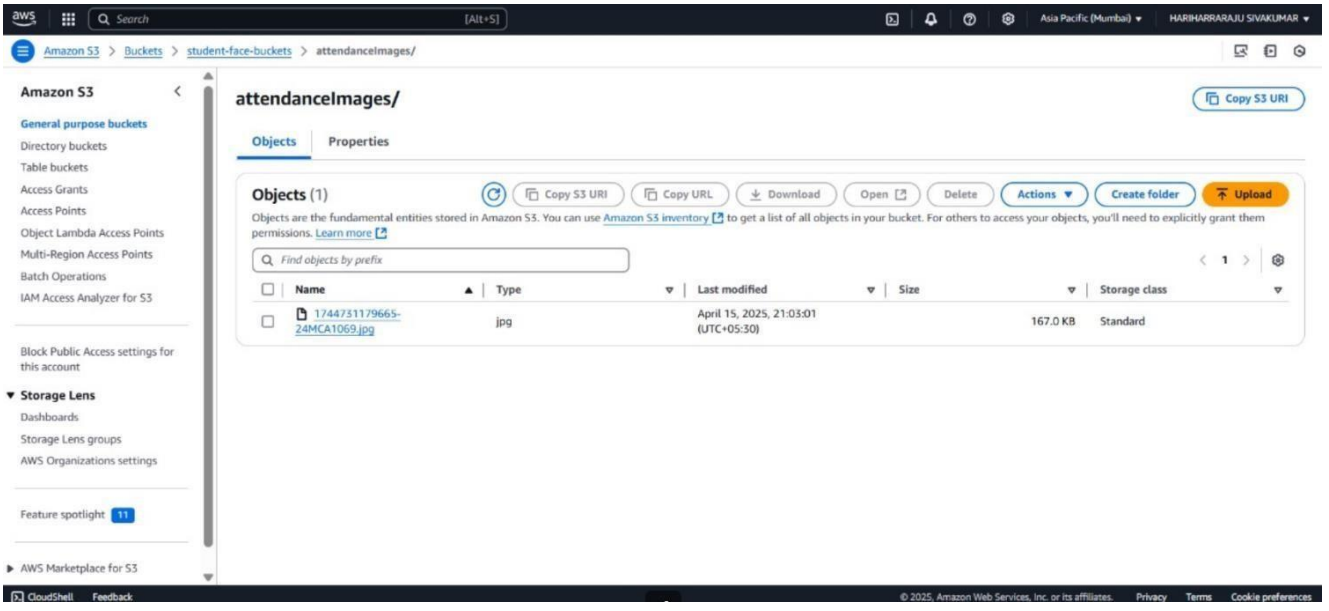
- Open the **AWS Console**, go to **S3**, and click **Create Bucket**. ○ Enter a **unique bucket name**, choose the **region**, uncheck **Block all public access**, and create the bucket.
- Inside the bucket, click **Create Folder** and name it **registeredFaces/**. Repeat the step to create another folder named **attendanceImages/**.

Screenshot:




Search [Alt+S]
Asia Pacific (Mumbai)
HARIHARRARAJU SIVAKUMAR
Amazon S3 > Buckets
Amazon S3
General purpose buckets
Directory buckets
Table buckets
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings
Feature spotlight 11
AWS Marketplace for S3
Account snapshot - updated every 24 hours
All AWS Regions
Storage lens provides visibility into storage usage and activity trends. Metrics don't include directory buckets. Learn more
View Storage Lens dashboard
General purpose buckets
Directory buckets
General purpose buckets (1) Info
All AWS Regions
Buckets are containers for data stored in S3.
Copy ARN
Empty
Delete
Create bucket
Find buckets by name
Name
AWS Region
IAM Access Analyzer
Creation date
student-face-buckets
Asia Pacific (Mumbai) ap-south-1
View analyzer for ap-south-1
March 27, 2025, 20:44:21 (UTC+05:30)
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Search [Alt+S]
Asia Pacific (Mumbai)
HARIHARRARAJU SIVAKUMAR
Amazon S3 > Buckets > student-face-buckets > registeredFaces/
Amazon S3
General purpose buckets
Directory buckets
Table buckets
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings
Feature spotlight 11
AWS Marketplace for S3
registeredFaces/
Copy S3 URI
Objects
Properties
Objects (1)
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Find objects by prefix
Name
Type
Last modified
Size
Storage class
24MCA1069.jpg
jpg
April 15, 2025, 21:02:42 (UTC+05:30)
167.0 KB
Standard
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Search [Alt+S]
Asia Pacific (Mumbai)
HARIHARRARAJU SIVAKUMAR
Amazon S3 > Buckets > student-face-buckets > attendanceImages/
Amazon S3
General purpose buckets
Directory buckets
Table buckets
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings
Feature spotlight 11
AWS Marketplace for S3
attendanceImages/
Copy S3 URI
Objects
Properties
Objects (1)
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Find objects by prefix
Name
Type
Last modified
Size
Storage class
1744731179665-24MCA1069.jpg
jpg
April 15, 2025, 21:03:01 (UTC+05:30)
167.0 KB
Standard
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

AWS Rekognition

- Open the **AWS CLI** or launch **CloudShell** from the AWS Console.
- Run the commands to list collections and view indexed faces

aws rekognition list-collections
aws rekognition list-faces --collection-id "face-attendance" --region ap-south-1

- Confirm collection is created with a success message.

Screenshot:

```
C:\Users\harir_4vapqty>aws rekognition list-collections
{
    "CollectionIds": [
        "face-attendance"
    ],
    "FaceModelVersions": [
        "7.0"
    ]
}
```

```
C:\Users\harir_4vapqty>aws rekognition list-faces --collection-id "face-attendance" --region ap-south-1
{
    "Faces": [
        {
            "FaceId": "dc70dab7-66c6-4498-ab4c-567cd4959334",
            "BoundingBox": {
                "Width": 0.38072699308395386,
                "Height": 0.3533079922199249,
                "Left": 0.30300799012184143,
                "Top": 0.17239199578762054
            },
            "ImageId": "3d8bc3fa-b05c-3a4a-a622-9e0f5b14ee5d",
            "ExternalImageId": "24MCA1069",
            "Confidence": 99.9939956665039,
            "IndexFacesModelVersion": "7.0"
        }
    ],
    "FaceModelVersion": "7.0"
}
```

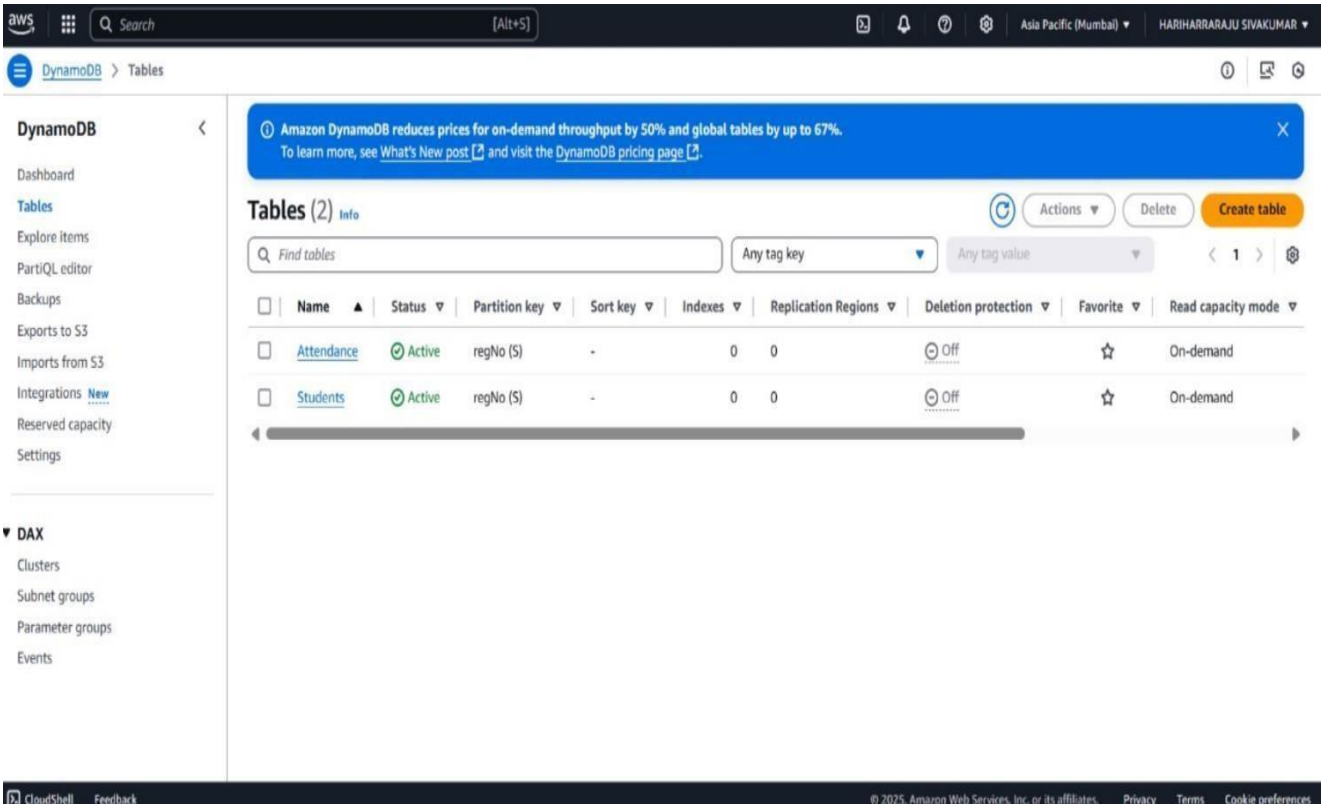
Dynamo DB:

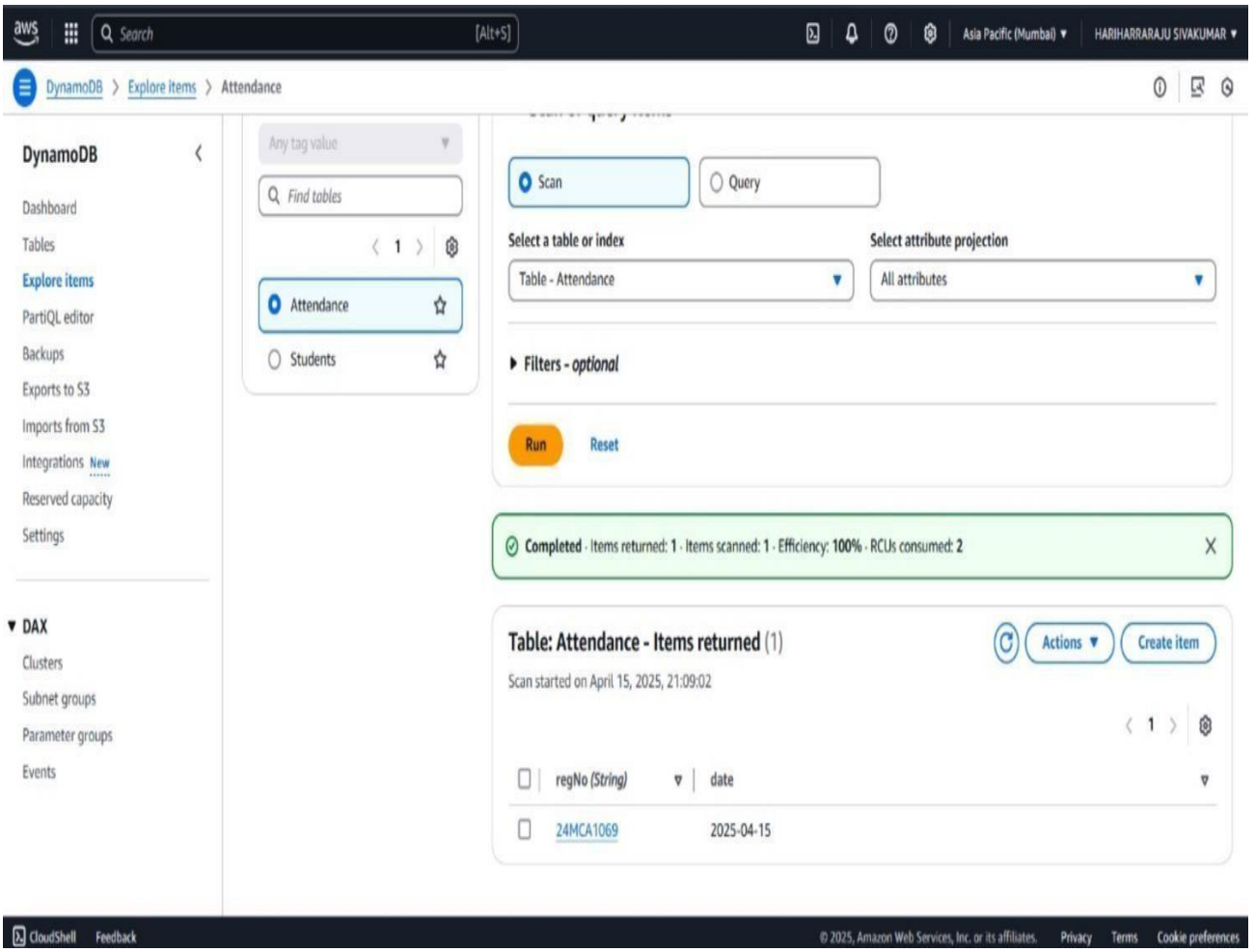
- Open the **AWS Console**, go to **DynamoDB**, and click **Create table.**

- For the Students table, set the table name to Students and use student_id (String) as the partition key. You can later add attributes like name, programme, etc., while inserting items.

- Repeat the process to create the Attendance table with regNo (String) as the partition key. Add attributes like timestamp, status, and image_url.

Screenshot:



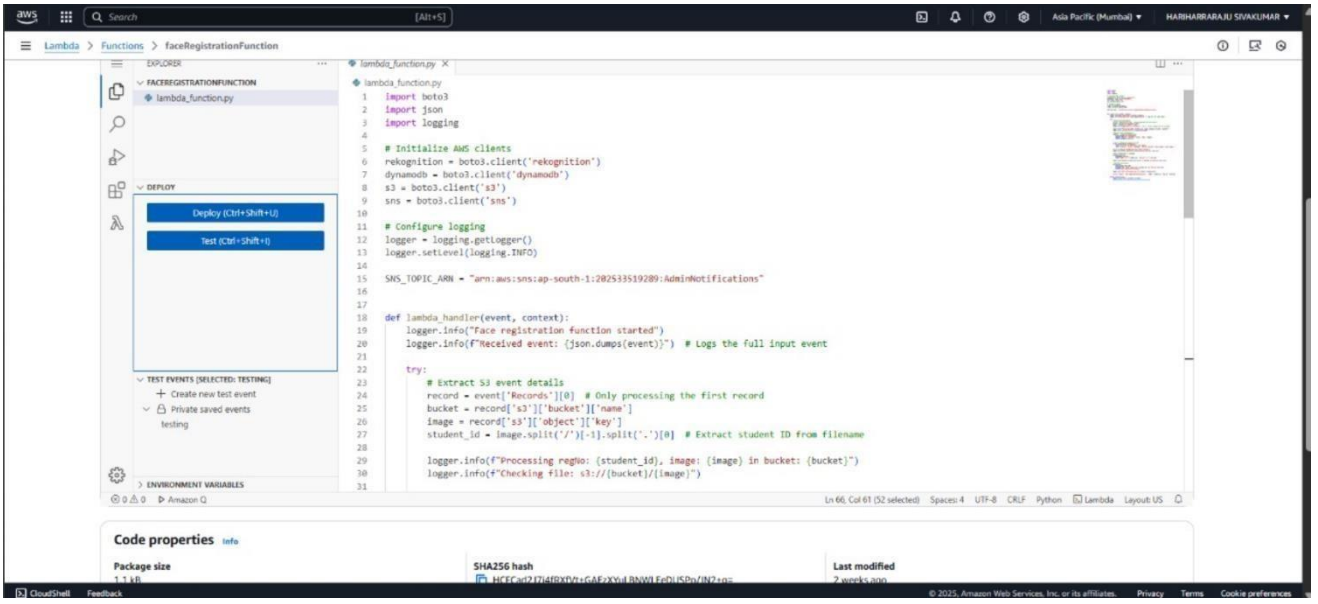


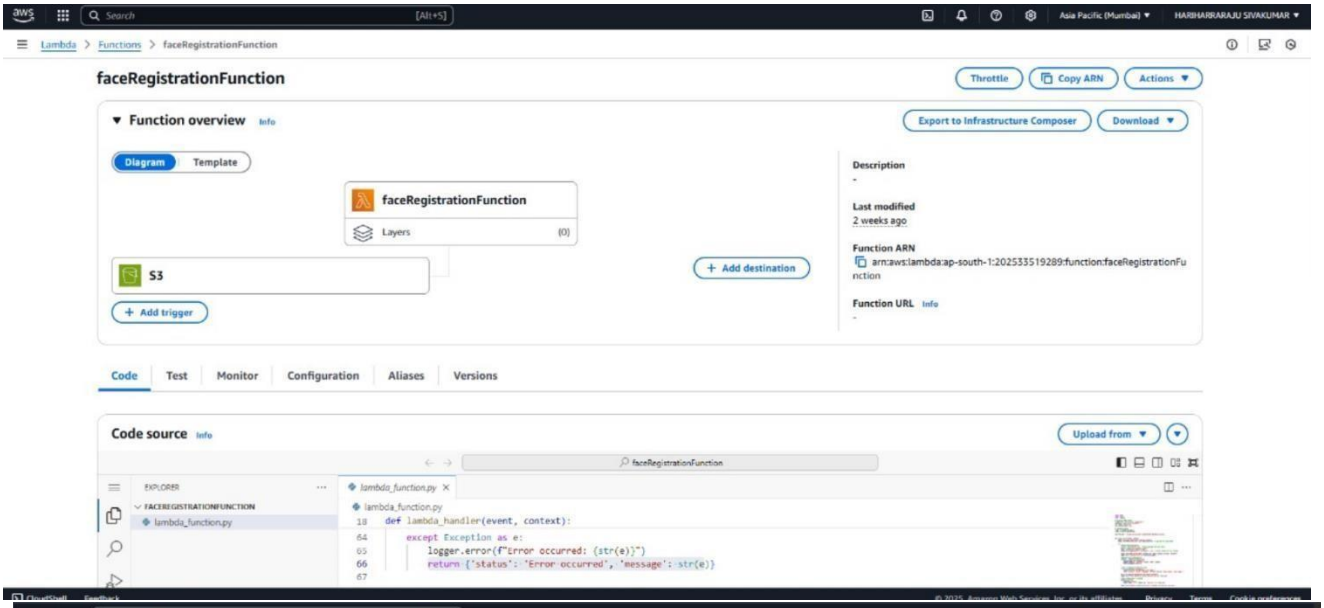
AWS Lambda

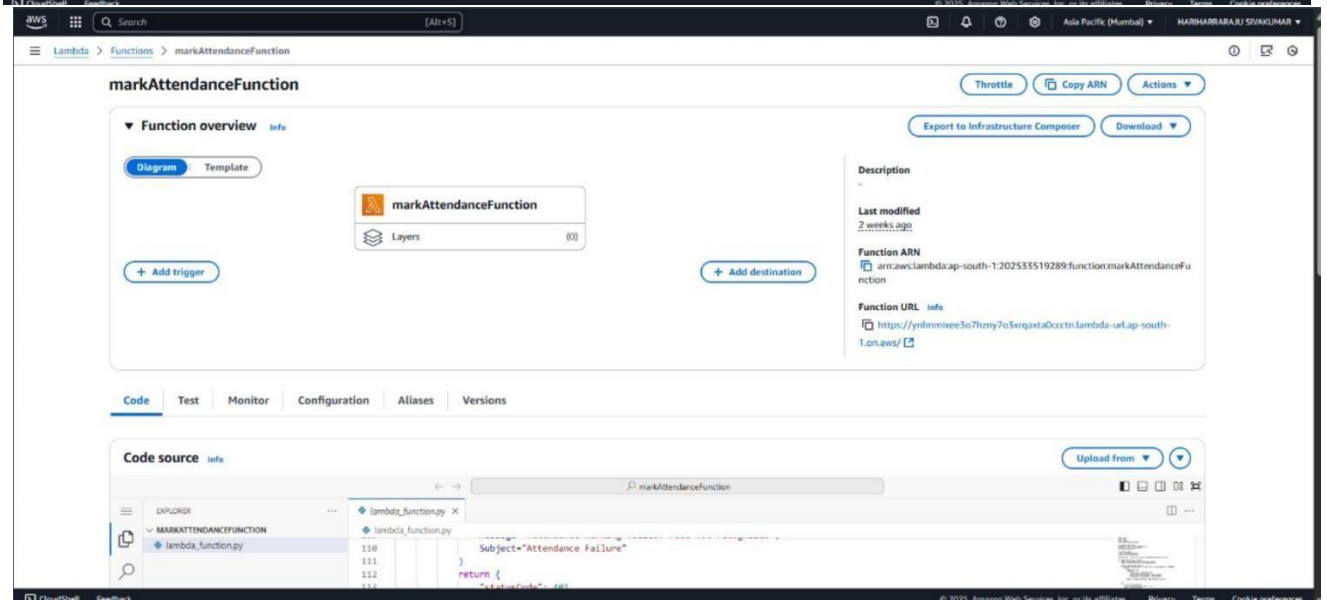
- Lambda function is triggered automatically when an image is uploaded to the registeredFaces/ folder in S3.

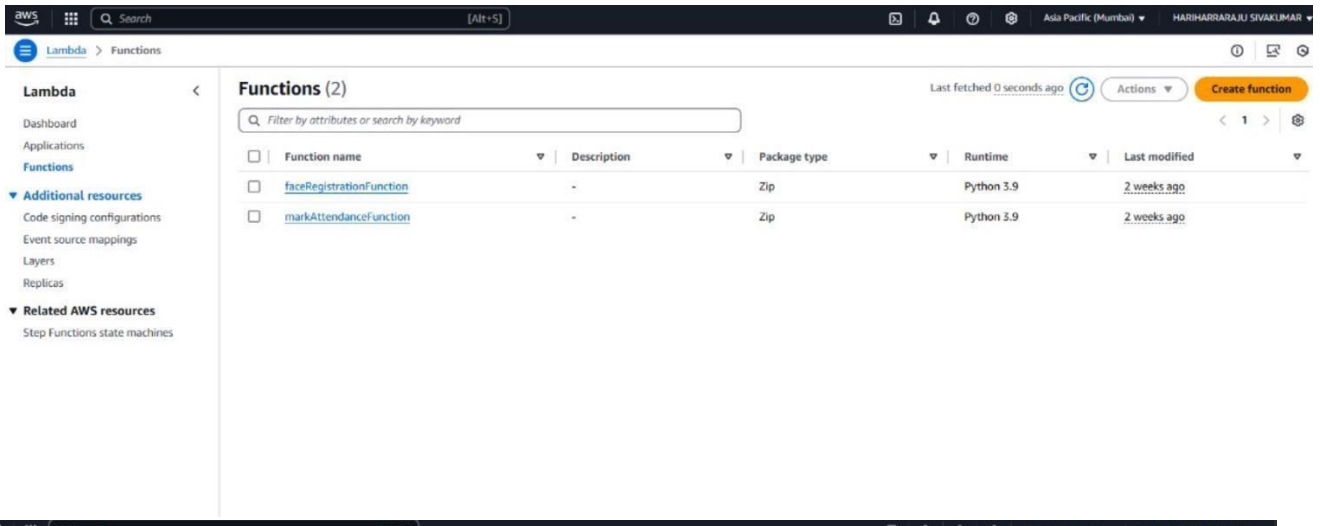
- Uses AWS Rekognition's IndexFaces API to extract facial features from the uploaded image.
- Stores metadata in DynamoDB, including student_id, faceId, and image details for future reference.

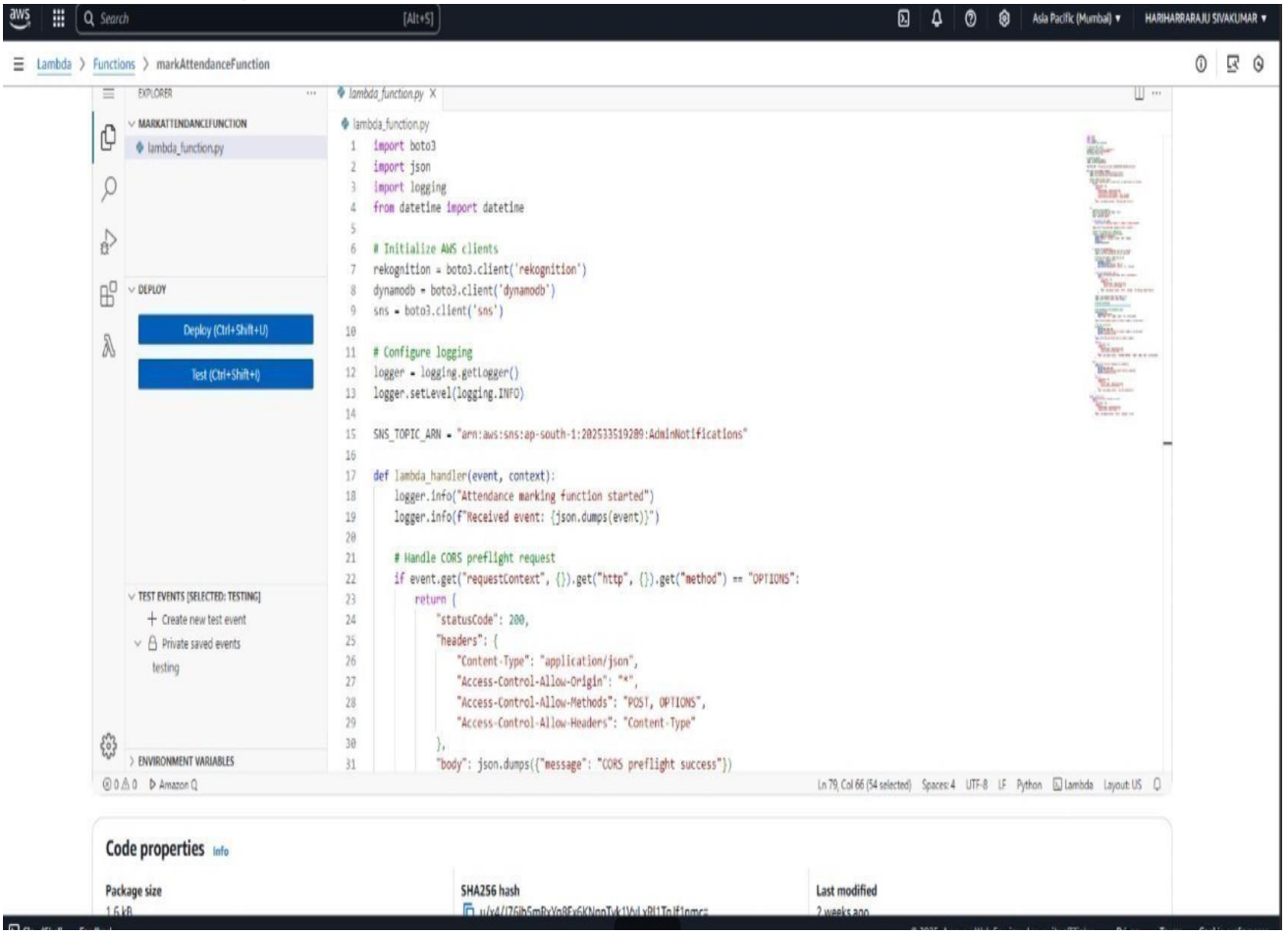
Screenshot:


Lambda > Functions > faceRegistrationFunction
lambda_function.py
import boto3
import json
import logging
# Initialize AWS clients
rekognition = boto3.client('rekognition')
dynamodb = boto3.client('dynamodb')
s3 = boto3.client('s3')
sns = boto3.client('sns')
# Configure logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)
SNS_TOPIC_ARN = "arn:aws:sns:ap-south-1:202533519289:AdminNotifications"
def lambda_handler(event, context):
logger.info("Face registration function started")
logger.info(f"Received event: {json.dumps(event)}") # Logs the full input event
try:
# Extract S3 event details
record = event['Records'][0] # Only processing the first record
bucket = record['s3']['bucket']['name']
image = record['s3']['object']['key']
student_id = image.split('/')[-1].split('.')[0] # Extract student ID from filename
logger.info(f"Processing regNo: {student_id}, image: {image} in bucket: {bucket}")
logger.info(f"Checking file: s3://{bucket}/{image}")
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)
Code properties Info
Package size
SHA256 hash
Last modified
2 weeks ago


faceRegistrationFunction
Throttle
Copy ARN
Actions
Function overview Info
Export to Infrastructure Composer
Download
Diagram
Template
faceRegistrationFunction
Layers
(0)
S3
+ Add trigger
+ Add destination
Description
Last modified
2 weeks ago
Function ARN
arn:aws:lambda:ap-south-1:202533519289:function:faceRegistrationFunction
Function URL Info
Code
Test
Monitor
Configuration
Aliases
Versions
Code source Info
Upload from
except Exception as e:
logger.error(f"Error occurred: {str(e)}")
return {'status': 'Error occurred', 'message': str(e)}


markAttendanceFunction
Throttle
Copy ARN
Actions
Function overview Info
Export to Infrastructure Composer
Download
Diagram
Template
markAttendanceFunction
Layers
(0)
+ Add trigger
+ Add destination
Description
Last modified
2 weeks ago
Function ARN
arn:aws:lambda:ap-south-1:202533519289:function:markAttendanceFunction
Function URL Info
https://ynlmmixee3o7hzny7o3xrqaxta0ccctn.lambda-url.ap-south-1.on.aws/
Code
Test
Monitor
Configuration
Aliases
Versions
Code source Info
Upload from
Subject="Attendance Failure"
return {





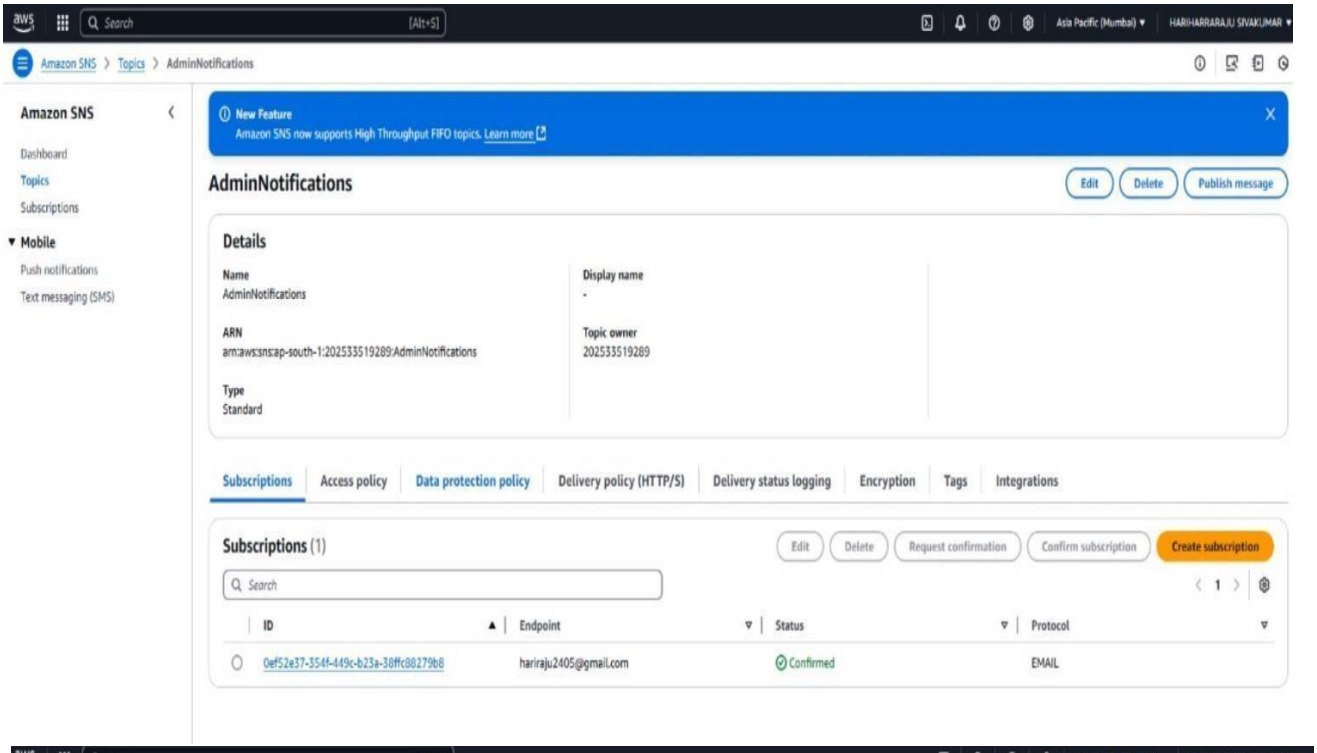
Amazon SNS

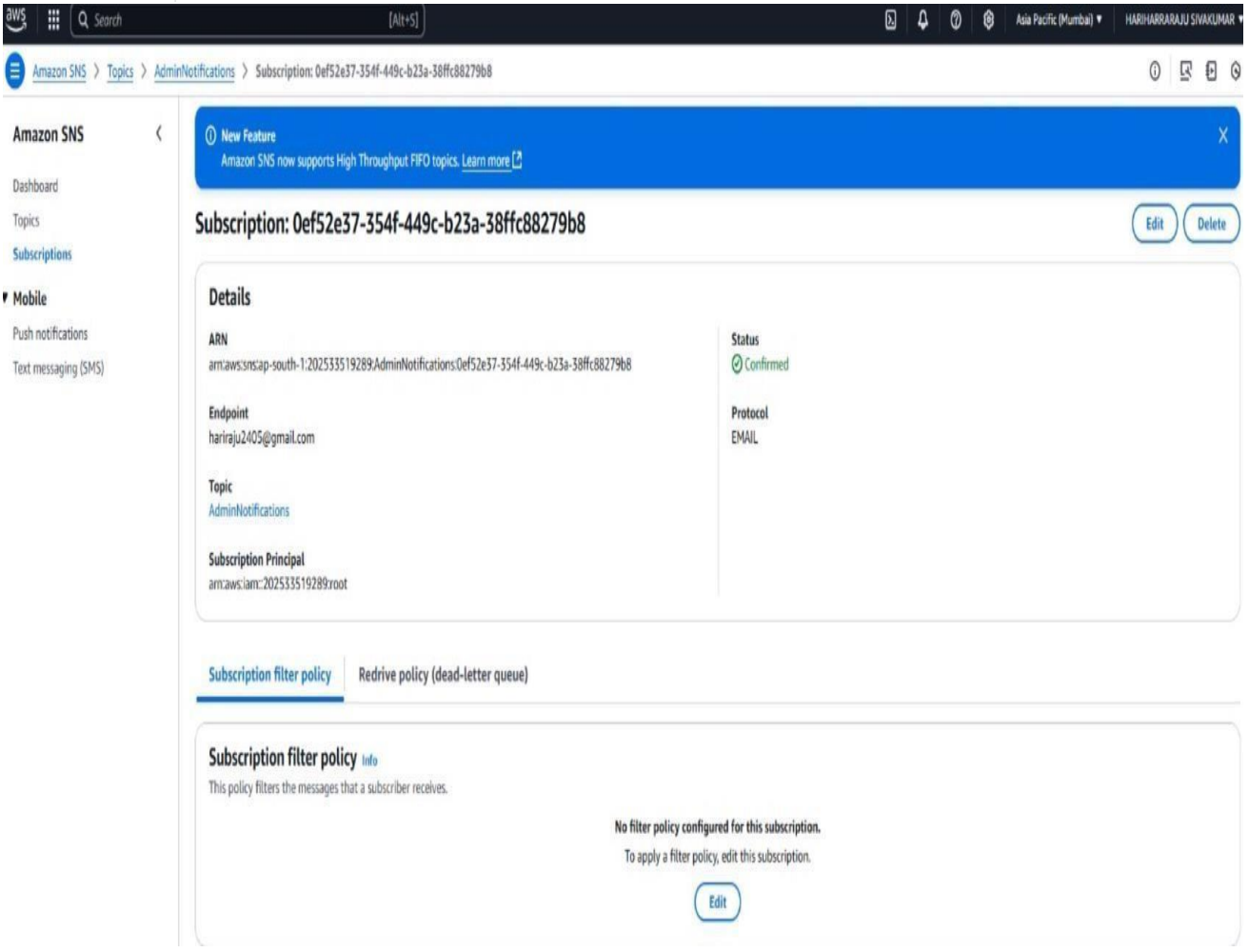
- Go to the **AWS Console**, navigate **to SNS > Topics**, and click **Create topic**. Choose Standard as the type and name it attendanceNotifications.

- After creating the topic, click **Create subscription**. Choose **Email** as the protocol and enter the destination.

- Confirm the subscription via the inbox or OTP. In your Lambda function, use **boto3** to publish messages to the attendanceNotifications topic when a face is registered.

Screenshot:

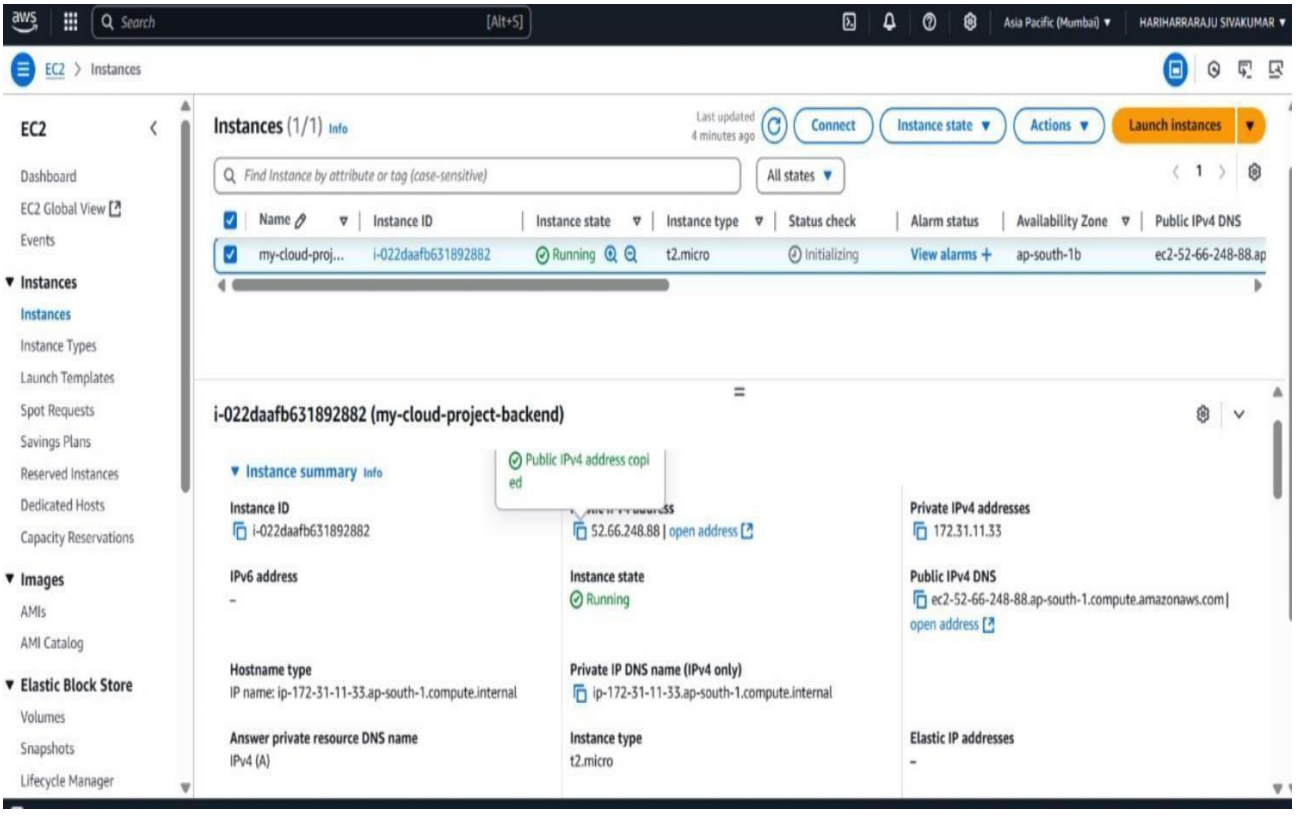

Amazon SNS
Topics
AdminNotifications
New Feature
Amazon SNS now supports High Throughput FIFO topics. Learn more
AdminNotifications
Edit
Delete
Publish message
Details
Name
AdminNotifications
Display name
-
ARN
arn:aws:sns:ap-south-1:202533519289:AdminNotifications
Topic owner
202533519289
Type
Standard
Subscriptions
Access policy
Data protection policy
Delivery policy (HTTP/S)
Delivery status logging
Encryption
Tags
Integrations
Subscriptions (1)
Edit
Delete
Request confirmation
Confirm subscription
Create subscription
Search
ID
Endpoint
Status
Protocol
0ef52e37-354f-449c-b23a-38ffc88279b8
hariraju2405@gmail.com
Confirmed
EMAIL


Amazon SNS
Topics
AdminNotifications
Subscription: 0ef52e37-354f-449c-b23a-38ffc88279b8
New Feature
Amazon SNS now supports High Throughput FIFO topics. Learn more
Subscription: 0ef52e37-354f-449c-b23a-38ffc88279b8
Edit
Delete
Details
ARN
arn:aws:sns:ap-south-1:202533519289:AdminNotifications:0ef52e37-354f-449c-b23a-38ffc88279b8
Status
Confirmed
Endpoint
hariraju2405@gmail.com
Protocol
EMAIL
Topic
AdminNotifications
Subscription Principal
arn:aws:iam::202533519289:root
Subscription filter policy
Redrive policy (dead-letter queue)
Subscription filter policy Info
This policy filters the messages that a subscriber receives.
No filter policy configured for this subscription.
To apply a filter policy, edit this subscription.
Edit

Amazon EC2

- Go to **EC2 > Launch Instance**, choose your preferred instance type.
- Design a modern UI using **HTML, CSS, and JS**. Create two forms:
  - Register Face Form
  - Mark Attendance Form
- Use **Flask** to create a lightweight server to handle image uploads and interact with AWS S3 for storage. Host the server on your EC2 instance.

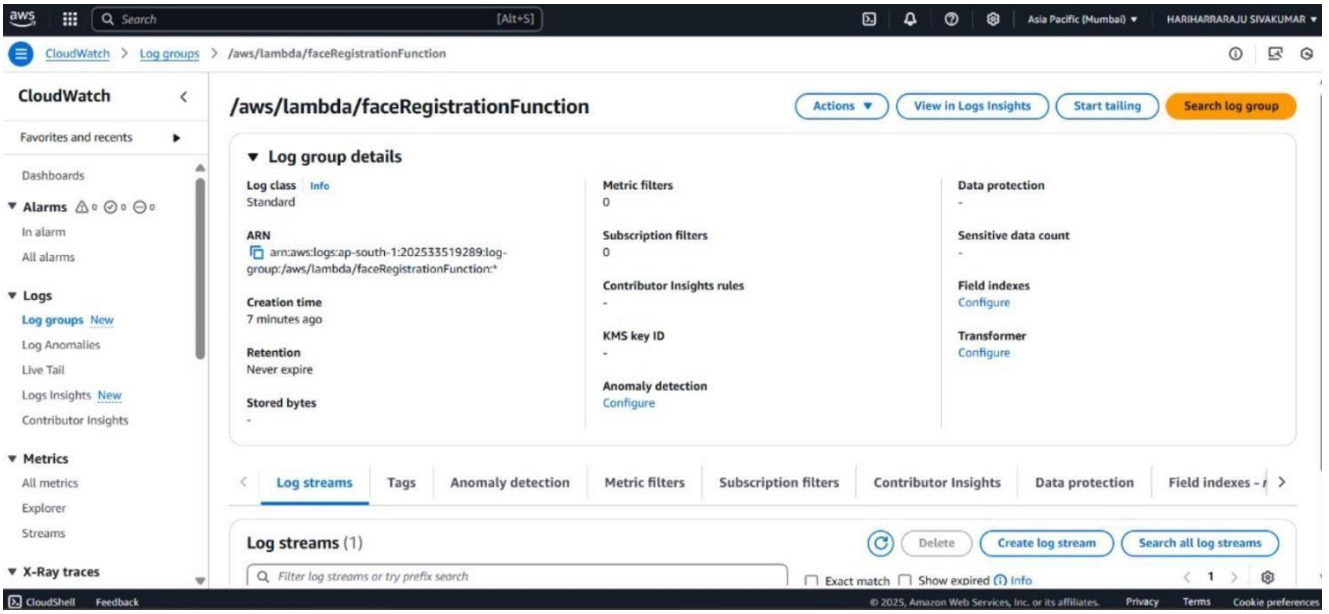
Screenshot:

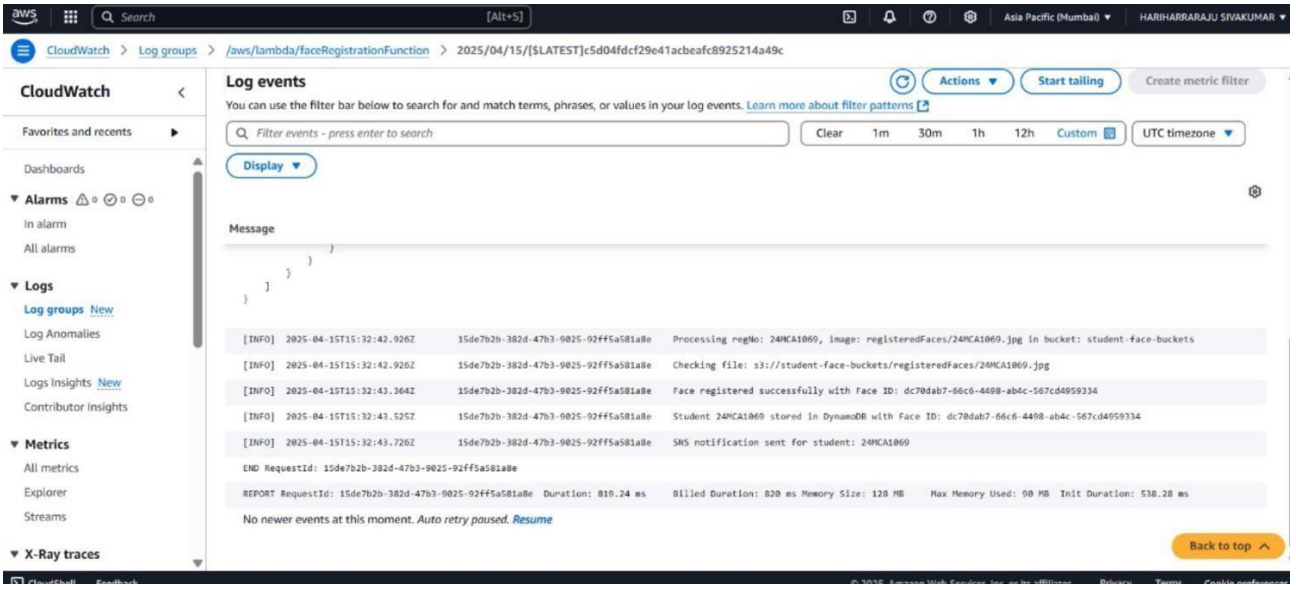


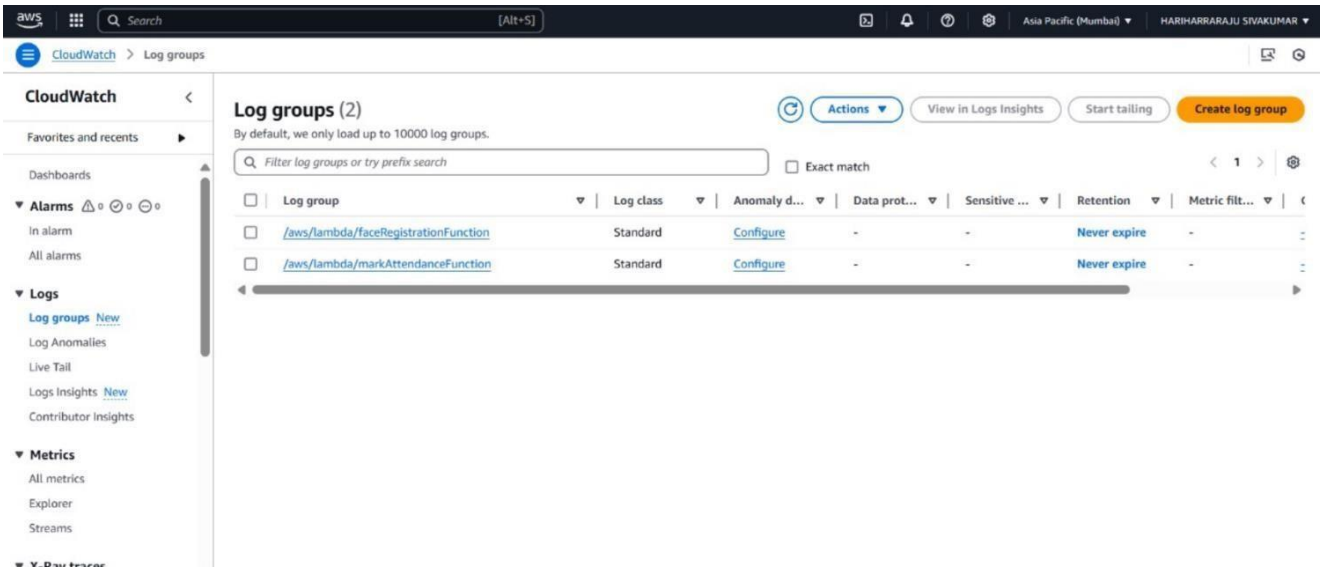
AWS Cloudwatch

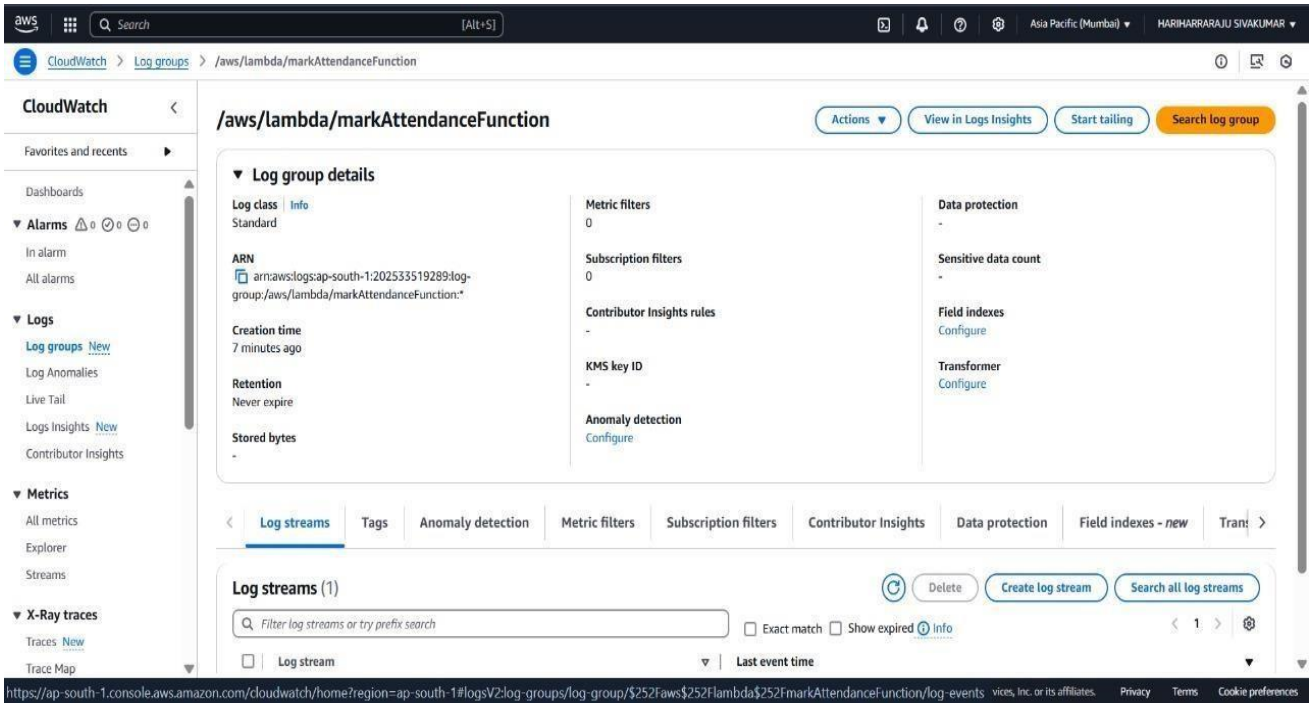
- Go to **CloudWatch > Logs** to view logs for both your Lambda functions and debug any issues.
- Navigate to **Alarms > Create alarm**, select a relevant metric like **Lambda > Errors**, and set a threshold.
- Link the alarm to your existing **SNS topic** to receive email/SMS alerts whenever the threshold is breached.

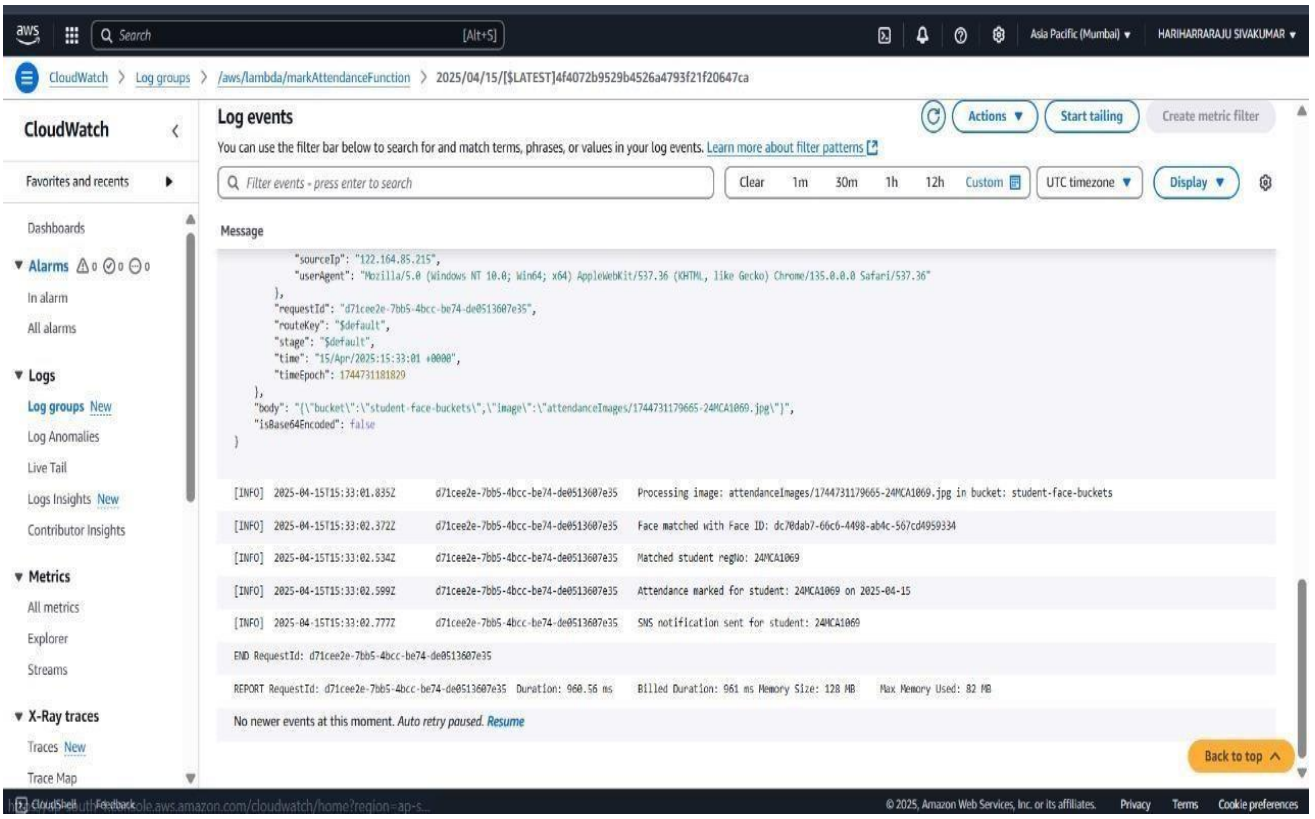
Screenshot:


CloudWatch
Log groups
/aws/lambda/faceRegistrationFunction
Actions
View in Logs Insights
Start tailing
Search log group
Log group details
Log class Info
Standard
ARN
arn:aws:logs:ap-south-1:202533519289:log-group:/aws/lambda/faceRegistrationFunction:*
Creation time
7 minutes ago
Retention
Never expire
Stored bytes
-
Metric filters
0
Subscription filters
0
Contributor Insights rules
-
KMS key ID
-
Anomaly detection
Configure
Data protection
-
Sensitive data count
-
Field indexes
Configure
Transformer
Configure
Log streams
Tags
Anomaly detection
Metric filters
Subscription filters
Contributor Insights
Data protection
Field indexes
Log streams (1)
Delete
Create log stream
Search all log streams
Filter log streams or try prefix search
Exact match
Show expired Info
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


CloudWatch
Log groups
/aws/lambda/faceRegistrationFunction
2025/04/15/[$LATEST]c5d04fdcf29e41acbeafc8925214a49c
Log events
Actions
Start tailing
Create metric filter
You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns
Filter events - press enter to search
Clear
1m
30m
1h
12h
Custom
UTC timezone
Display
Message
[INFO] 2025-04-15T15:32:42.926Z 15de7b2b-382d-47b3-9025-92ff5a581a8e Processing regNo: 24MCA1069, image: registeredFaces/24MCA1069.jpg in bucket: student-face-buckets
[INFO] 2025-04-15T15:32:42.926Z 15de7b2b-382d-47b3-9025-92ff5a581a8e Checking file: s3://student-face-buckets/registeredFaces/24MCA1069.jpg
[INFO] 2025-04-15T15:32:43.364Z 15de7b2b-382d-47b3-9025-92ff5a581a8e Face registered successfully with Face ID: dc70dab7-66c6-4498-ab4c-567cd4959334
[INFO] 2025-04-15T15:32:43.525Z 15de7b2b-382d-47b3-9025-92ff5a581a8e Student 24MCA1069 stored in DynamoDB with Face ID: dc70dab7-66c6-4498-ab4c-567cd4959334
[INFO] 2025-04-15T15:32:43.726Z 15de7b2b-382d-47b3-9025-92ff5a581a8e SNS notification sent for student: 24MCA1069
END RequestId: 15de7b2b-382d-47b3-9025-92ff5a581a8e
REPORT RequestId: 15de7b2b-382d-47b3-9025-92ff5a581a8e Duration: 819.24 ms Billed Duration: 820 ms Memory Size: 128 MB Max Memory Used: 90 MB Init Duration: 538.28 ms
No newer events at this moment. Auto retry paused. Resume
Back to top


Search [Alt+S]
Asia Pacific (Mumbai)
HARIHARRAJU SIVAKUMAR
CloudWatch > Log groups
CloudWatch
Favorites and recents
Dashboards
Alarms
In alarm
All alarms
Logs
Log groups New
Log Anomalies
Live Tail
Logs Insights New
Contributor Insights
Metrics
All metrics
Explorer
Streams
Log groups (2)
By default, we only load up to 10000 log groups.
Actions
View in Logs Insights
Start tailing
Create log group
Filter log groups or try prefix search
Exact match
Log group
Log class
Anomaly d...
Data prot...
Sensitive ...
Retention
Metric filt...
/aws/lambda/faceRegistrationFunction
Standard
Configure
Never expire
/aws/lambda/markAttendanceFunction
Standard
Configure
Never expire


Search [Alt+S]
Asia Pacific (Mumbai)
HARIHARRAJU SIVAKUMAR
CloudWatch > Log groups > /aws/lambda/markAttendanceFunction
CloudWatch
Favorites and recents
Dashboards
Alarms
In alarm
All alarms
Logs
Log groups New
Log Anomalies
Live Tail
Logs Insights New
Contributor Insights
Metrics
All metrics
Explorer
Streams
X-Ray traces
Traces New
Trace Map
/aws/lambda/markAttendanceFunction
Actions
View in Logs Insights
Start tailing
Search log group
Log group details
Log class Info
Standard
ARN
arn:aws:logs:ap-south-1:202533519289:log-group:/aws/lambda/markAttendanceFunction:*
Creation time
7 minutes ago
Retention
Never expire
Stored bytes
-
Metric filters
0
Subscription filters
0
Contributor Insights rules
-
KMS key ID
-
Anomaly detection
Configure
Data protection
-
Sensitive data count
-
Field indexes
Configure
Transformer
Configure
Log streams
Tags
Anomaly detection
Metric filters
Subscription filters
Contributor Insights
Data protection
Field indexes - new
Log streams (1)
Delete
Create log stream
Search all log streams
Filter log streams or try prefix search
Exact match
Show expired Info
Log stream
Last event time
https://ap-south-1.console.aws.amazon.com/cloudwatch/home?region=ap-south-1#logsV2:log-groups/log-group/$252Faws$252Flambda$252FmarkAttendanceFunction/log-events
vices, Inc. or its affiliates.
Privacy
Terms
Cookie preferences



4. Coding:

Lambda mark Attendance Function:

```
  Importboto3       importjson
import logging from datetime
import datetime

# Initialize AWS clients rekognition =
boto3.client('rekognition') dynamodb =
boto3.client('dynamodb')       sns       =
boto3.client('sns')

# Configure logging logger =
logging.getLogger()
logger.setLevel(logging.INFO
)

SNS_TOPIC_ARN                    =                    "arn:aws:sns:ap-south-
1:202533519289:AdminNotifications"

def          lambda_handler(event,          context):
   logger.info("Attendance marking function started")
   logger.info(f"Received                    event:
   {json.dumps(event)}")
```

```
    # Handle CORS preflight request if event.get("requestContext", {}).get("http",
    {}).get("method") == "OPTIONS":
        return {
            "statusCode":       200,
            "headers": {
                "Content-Type": "application/json",
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Methods": "POST, OPTIONS",
                "Access-Control-Allow-Headers": "Content-Type"
            },
            "body": json.dumps({"message": "CORS preflight success"})
        }
try:
        # Extract input parameters
        body   =   json.loads(event.get("body",   "{}"))
        bucket = body.get("bucket") image
        = body.get("image")

        if not bucket or not image: raise ValueError("Missing 'bucket' or 'image' in

        request payload") logger.info(f"Processing image: {image} in bucket:

        {bucket}")

        # Search for a matching face in Rekognition response =
        rekognition.search_faces_by_image(
            CollectionId='face-attendance',
            Image={'S3Object': {'Bucket': bucket, 'Name': image}},
            MaxFaces=1,
            FaceMatchThreshold=85
        )
        if response.get('FaceMatches'):
            face_id        =        response['FaceMatches'][0]['Face']['FaceId']
            logger.info(f"Face matched with Face ID: {face_id}")

            # Retrieve the student's regNo using a scan
            scan_response = dynamodb.scan(
                TableName='Students',
                FilterExpression='face_id = :face_id',
                ExpressionAttributeValues={':face_id': {'S': face_id}}
            )

            if   not   scan_response.get('Items'):   logger.error("No   matching
               student found in the database.") return {
                   "statusCode":              404,
                   "headers": {
                       "Content-Type": "application/json",
                       "Access-Control-Allow-Origin": "*"
                   },
                   "body": json.dumps({'status': 'Error', 'message': 'No matching student found'})
               }

            regNo        =        scan_response['Items'][0]['regNo']['S']
            logger.info(f"Matched student regNo: {regNo}")
```

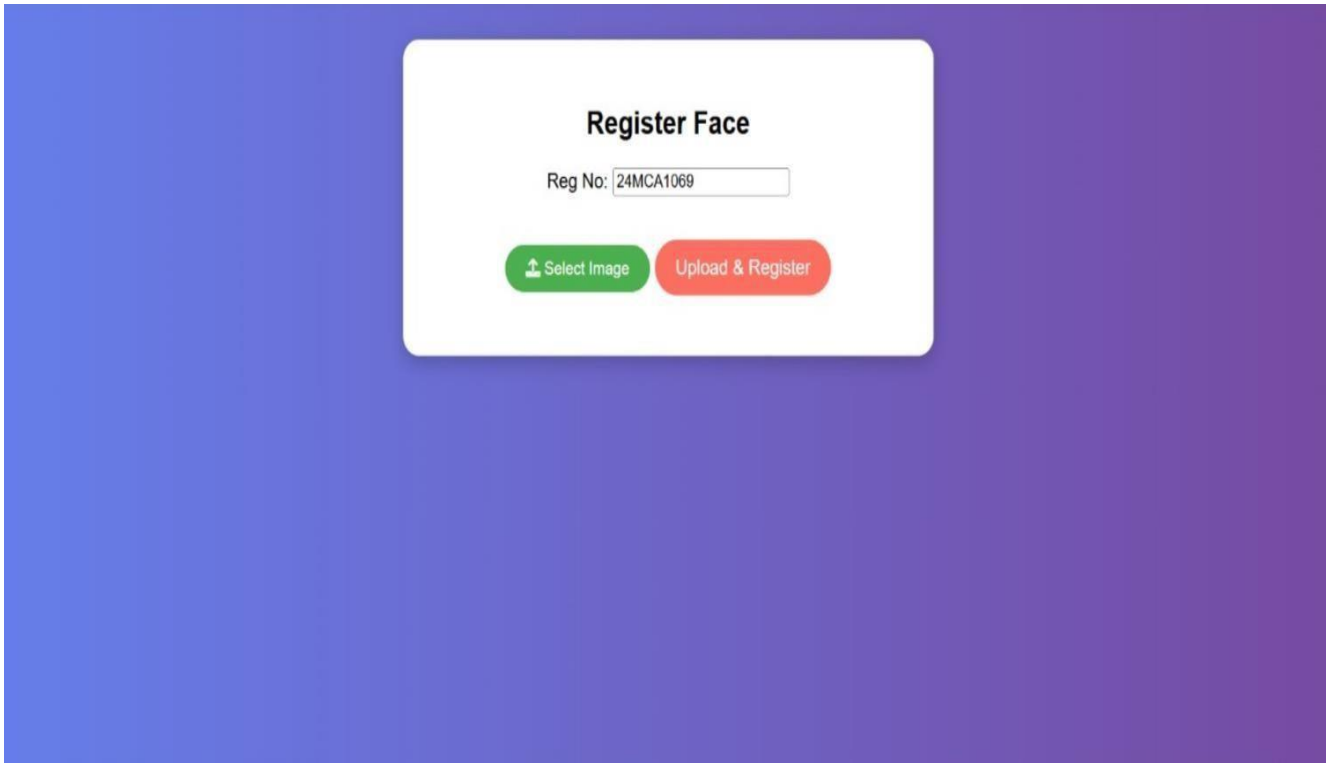
```
        # Get the current date current_date =
        datetime.utcnow().strftime('%Y-%m-%d')

        # Mark attendance in the Attendance table
        dynamodb.put_item(
            TableName='Attendance',
            Item={'regNo': {'S': regNo}, 'date': {'S': current_date}}
        )
        logger.info(f"Attendance marked for student: {regNo} on {current_date}")

        # Send SNS notification
        sns.publish(
            TopicArn=SNS_TOPIC_ARN,
            Message=f"Attendance marked for student: {regNo} on {current_date}",
            Subject="Attendance Success"
        )
        logger.info(f"SNS notification sent for student: {regNo}")

        return {
            "statusCode": 200,
            "headers": {
                "Content-Type": "application/json",
                "Access-Control-Allow-Origin": "*"
            },
            "body": json.dumps({'status': 'Attendance Marked', 'regNo': regNo, 'date': current_date})
        }
    else:
        logger.error("Face not recognized for attendance")
        sns.publish(
            TopicArn=SNS_TOPIC_ARN,
            Message="Attendance marking failed: Face not recognized",
            Subject="Attendance Failure"
        )
        return {
            "statusCode": 401,
            "headers": {
                "Content-Type": "application/json",
                "Access-Control-Allow-Origin": "*"
            },
            "body": json.dumps({'status': 'Face Not Recognized'})
        }

except Exception as e:
    logger.error(f"Error occurred: {str(e)}")
    return {
        "statusCode": 500,
        "headers": {
            "Content-Type": "application/json",
            "Access-Control-Allow-Origin": "*"
        },
        "body": json.dumps({'status': 'Error', 'message': str(e)})
    }
```

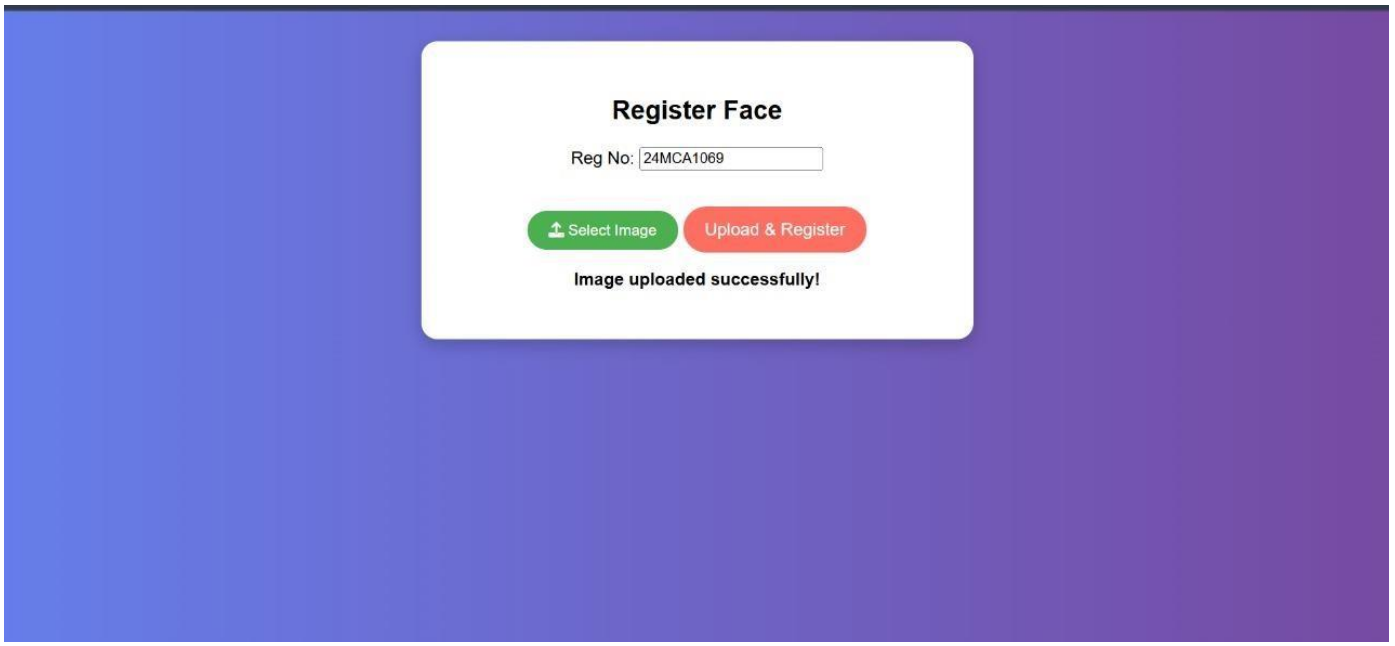
Face Register Function Lambda:

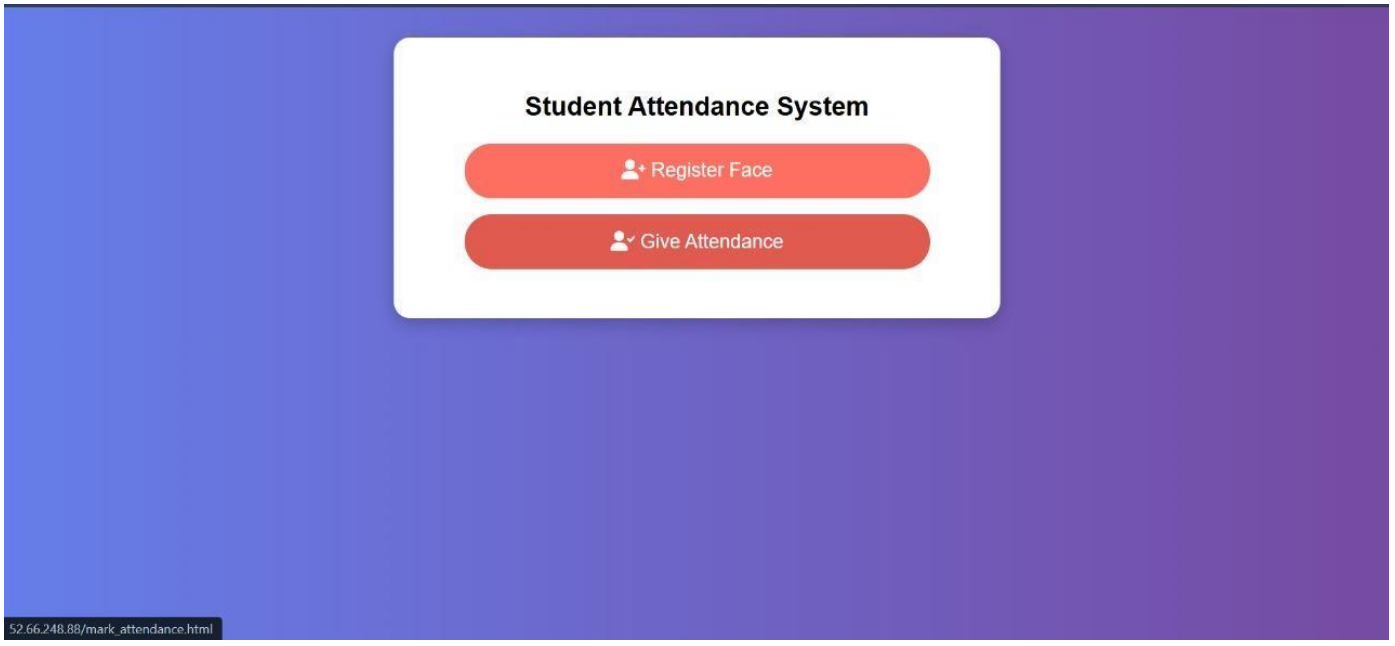
```
import boto3 import json
import logging

# Initialize AWS clients
rekognition = boto3.client('rekognition')
dynamodb = boto3.client('dynamodb')
s3 = boto3.client('s3') sns =
boto3.client('sns')

# Configure logging logger =
   logging.getLogger()
logger.setLevel(logging.INFO)

SNS_TOPIC_ARN = "arn:aws:sns:ap-south-1:202533519289:AdminNotifications"


def lambda_handler(event, context): logger.info("Face
   registration function started")
    logger.info(f"Received event: {json.dumps(event)}") # Logs the full input event
   try:
      # Extract S3 event details
      record = event['Records'][0] # Only processing the first record bucket
      = record['s3']['bucket']['name'] image
      = record['s3']['object']['key']
      student_id = image.split('/')[-1].split('.')[0] # Extract student ID from filename

      logger.info(f"Processing regNo: {student_id}, image: {image} in bucket:
            {bucket}") logger.info(f"Checking file: s3://{bucket}/{image}")

      # Register face in Rekognition response =
      rekognition.index_faces(
         CollectionId='face-attendance',
         Image={'S3Object': {'Bucket': bucket, 'Name': image}},
         ExternalImageId=student_id
      )

      # Check if Rekognition detected a face if
      not response.get('FaceRecords'):
         logger.error("No face detected in the image.")
         return {'status': 'Error', 'message': 'No face detected. Please upload a clear image.'}
```

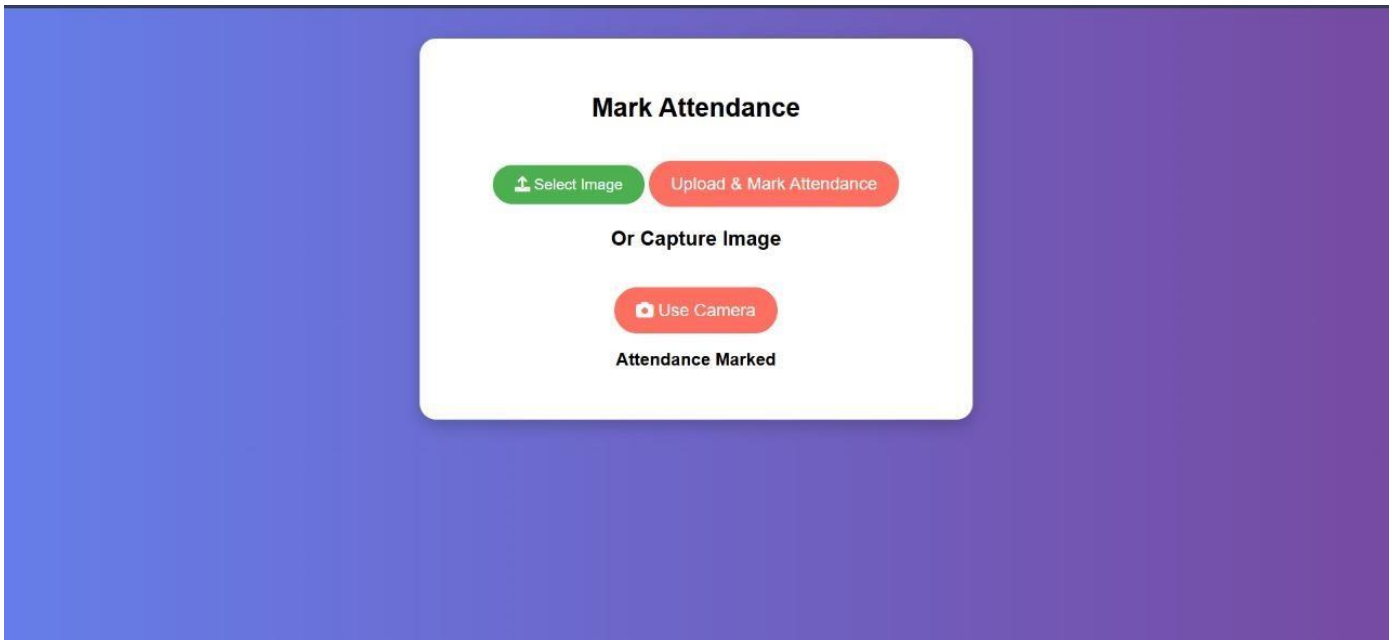
```
        face_id = response['FaceRecords'][0]['Face']['FaceId']
        logger.info(f"Face registered successfully with Face ID: {face_id}")

        # Store student data in DynamoDB dynamodb.put_item(
            TableName='Students',
            Item={'regNo': {'S': student_id}, 'face_id': {'S': face_id}}
        )
        logger.info(f"Student {student_id} stored in DynamoDB with Face ID: {face_id}")

          # Send SNS notification
        sns.publish(
            TopicArn=SNS_TOPIC_ARN,
            Message=f"New student registered: {student_id} with Face ID: {face_id}", Subject="Face Registration Success"
        )
        logger.info(f"SNS notification sent for student: {student_id}")
return {'status': 'Face Registered Successfully!', 'regNo': student_id, 'face_id': face_id}
    except Exception as e: logger.error(f"Error occurred: {str(e)}")
        return {'status': 'Error occurred', 'message': str(e)}
face register function lambda
```

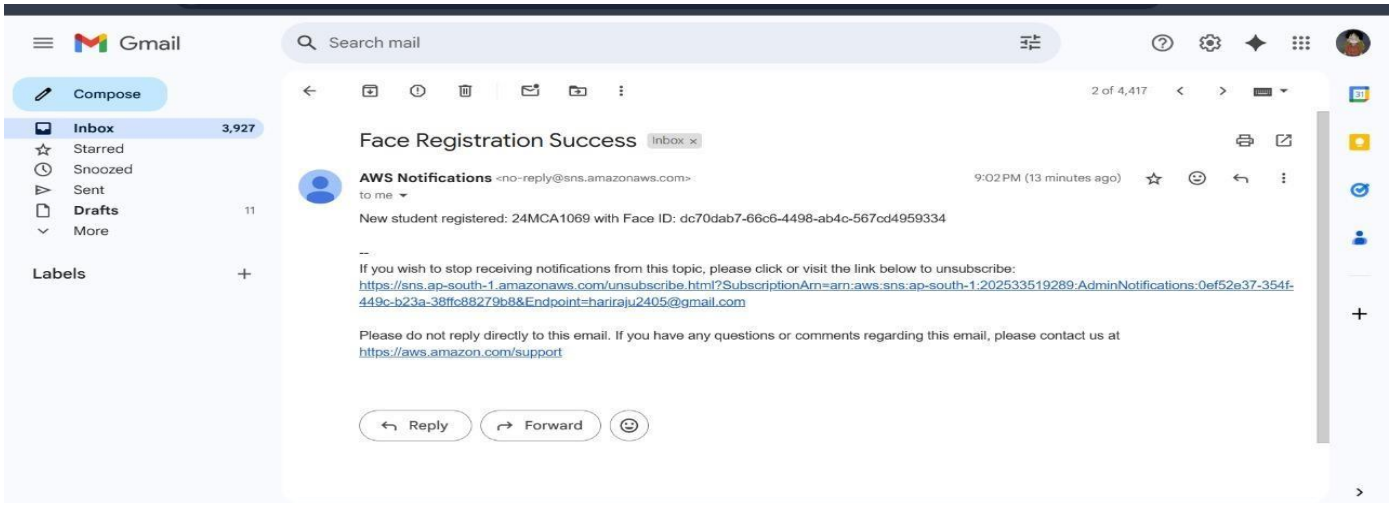
OUTPUT:

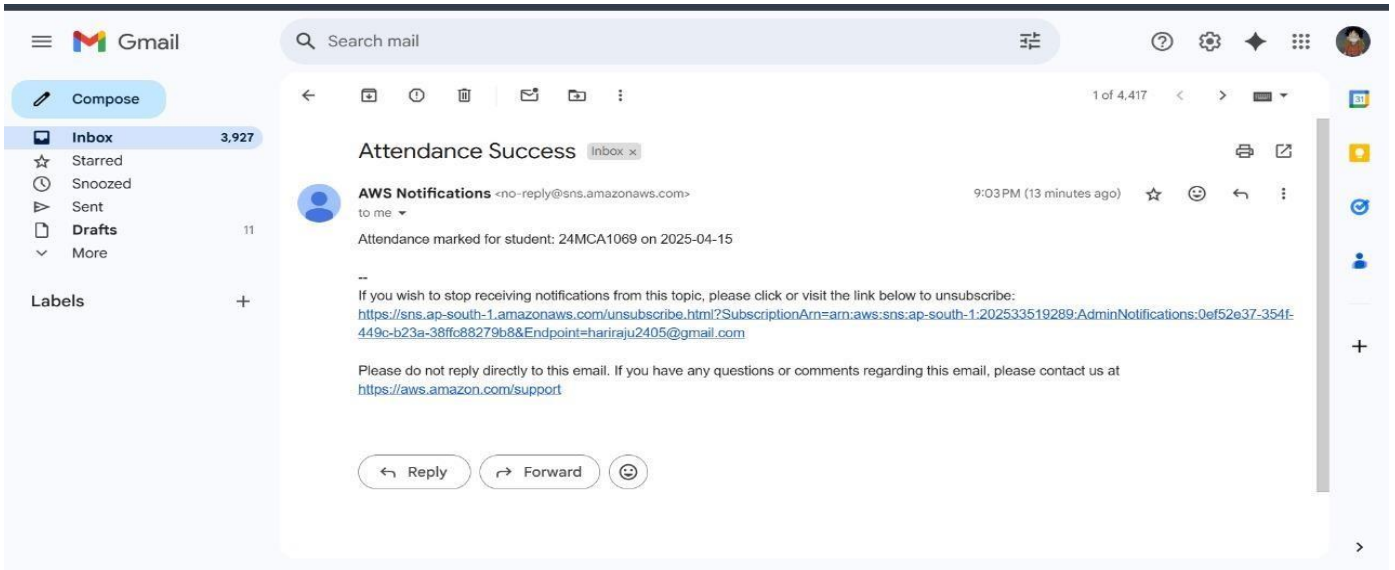



Register Face
Reg No: 24MCA1069
Select Image
Upload & Register
Image uploaded successfully!


Student Attendance System
Register Face
Give Attendance
52.66.248.88/mark_attendance.html


Mark Attendance
Select Image
Upload & Mark Attendance
Or Capture Image
Use Camera
Attendance Marked


Gmail
Search mail
Compose
Inbox 3,927
Starred
Snoozed
Sent
Drafts 11
More
Labels
2 of 4,417
Face Registration Success
Inbox
AWS Notifications <no-reply@sns.amazonaws.com>
to me
9:02 PM (13 minutes ago)
New student registered: 24MCA1069 with Face ID: dc70dab7-66c6-4498-ab4c-567cd4959334
--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.ap-south-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:ap-south-1:202533519289:AdminNotifications:0ef52e37-354f-449c-b23a-38ffc88279b8&Endpoint=hariraju2405@gmail.com
Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at
https://aws.amazon.com/support
Reply
Forward


Gmail
Search mail
Compose
Inbox 3,927
Starred
Snoozed
Sent
Drafts 11
More
Labels
1 of 4,417
Attendance Success
Inbox
AWS Notifications <no-reply@sns.amazonaws.com>
to me
9:03 PM (13 minutes ago)
Attendance marked for student: 24MCA1069 on 2025-04-15
--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.ap-south-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:ap-south-1:202533519289:AdminNotifications:0ef52e37-354f-449c-b23a-38ffc88279b8&Endpoint=hariraju2405@gmail.com
Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at
https://aws.amazon.com/support
Reply
Forward

5. Conclusion

The Cloud-Based Face Recognition Attendance System presents a modern, automated, and efficient solution to traditional attendance tracking. By integrating facial recognition with robust AWS services, the system minimizes human effort, reduces errors, and ensures secure data handling. Its serverless design makes it cost-effective and easy to scale across institutions. Through real-time notifications, logs, and monitoring, the system maintains transparency and reliability in student attendance tracking.

6. Future Enhancements

- Add multi-factor authentication (face + OTP) for enhanced security. ○ Build an admin dashboard for monitoring attendance trends and student activity. ○ Support offline face capture with background sync when internet becomes available. ○ Integrate mobile app for on-the-go attendance marking. ○ Add facial liveness detection to prevent spoofing with static images.
- Include voice notification alerts for visually impaired students.

7. References

- **Cloud Based Smart Attendance System for Educational Institutions:** This paper presents a prototype of a cloud-based end-to-end smart attendance system designed to address the challenges of manual attendance tracking.

- **Cloud Enabled Attendance System":** This project introduces a cloud-enabled attendance system utilizing an Arduino ESP8266 microcontroller and an RFID reader. The system features a local server within the ESP8266, eliminating the need for an external server and simplifying deployment.

- **Online Classroom Attendance System Based on Cloud Computing:** This study proposes an automatic attendance system where RFID readers are installed in classrooms. The system assigns unique identities to each classroom and integrates with cloud computing for data management.

- **Cloud Based Intelligent Attendance System Through Video Streaming:** This paper presents an intelligent attendance system that leverages cloud computing and video streaming technologies. The architecture and algorithms used aim to enhance the efficiency of attendance tracking.

- **Cloud Based Attendance Management and Information System:** This dissertation discusses the development of a cloud-based attendance management system. It combines hardware components, such as RFID readers and microcontrollers, with software subsystems to process and store attendance data efficiently