CSE 546 — Project Report

Aatish Dineshbhai Chaudhari - 1225418103

Priyal Ritesh Desai - 1225416166

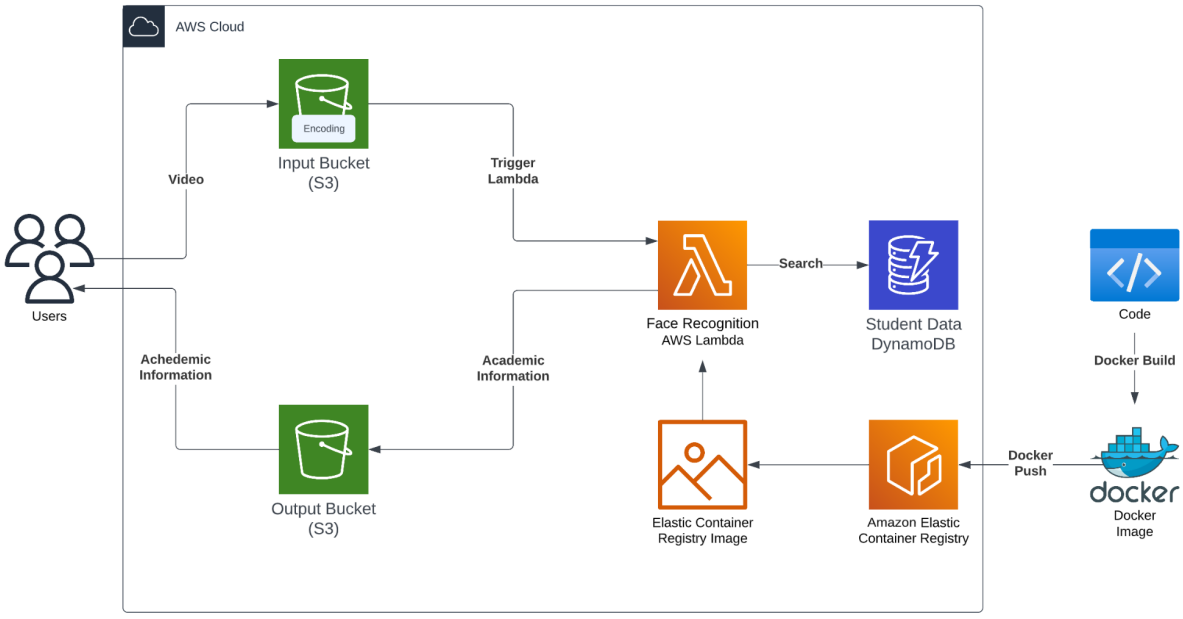
Verik Jagdish Vekaria - 1225674957

1. Problem statement

This project aims to create an elastic application that uses PaaS cloud services, specifically AWS Lambda and other supporting services from AWS, to automatically scale out and in on-demand and cost-effectively. The program is a clever classroom assistant that uses classroom videos to perform face recognition and provides recognized students with pertinent academic information. The project will use the video's extracted frames to perform face recognition using the Python face recognition library and the ffmpeg multimedia framework.

2. Design and implementation

2.1 Architecture



Our architecture consists of four key AWS components - S3, Lambda, Dynamodb, ECR. In addition, Docker is used to build an image and push it to Elastic Container Registry (ECR).

S3

Amazon Web Services (AWS) offers the Amazon Simple Storage Service (S3), a web service that offers highly scalable and reliable cloud-based storage for data of any form and size. We have used two S3 buckets, one to store video uploaded by the user and the second to store the output of video in CSV format.

Lambda

AWS Lambda is a serverless computing service that enables programmers to run their code without worrying about server administration. Without having to provision or manage servers, it enables developers to build and run applications in response to events and triggers. AWS Lambda is a well-liked option for creating serverless applications and is widely used by developers and businesses worldwide. We have used it to process video and it is triggered by S3 when the user uploads a video in S3.

ECR (Elastic Container Registry)

Amazon Web Services (AWS) offers the fully-managed container registry service known as Elastic Container Registry (ECR) (AWS). It is essential for the development and deployment of container-based applications because it makes it simple for users to store, manage, and deploy Docker container images. We have used ECR to store our docker Image and use that image to deploy in AWS Lambda.

Dynamodb

AWS's Amazon DynamoDB is a fully-managed NoSQL database service with low latency, high performance, and scalability. Any amount of data can be stored and retrieved using DynamoDB, which can handle high-volume workloads by automatically scaling and provisioning throughput capacity. Additionally, it offers features for disaster recovery and data replication, including global tables, streams, and backups. It also supports ACID transactions. Applications that require scale-out, high throughput, and low latency data access for web, mobile, gaming, and IoT typically use DynamoDB. We have used Dynamodb to store student data which contains Id, name, major, and year.

2.2 Autoscaling

AWS Lambda is automatically scalable by default and doesn't need any additional configuration. This is so because AWS Lambda is a serverless computing service, which means it manages resource scaling based on incoming workloads automatically. AWS Lambda automatically adds more resources to handle incoming requests as they come in and removes those resources as the workload decreases. As a result, developers can concentrate on writing code and creating features rather than worrying about manually scaling their applications or managing servers.

An event-driven architecture is used by AWS Lambda, which monitors for events and starts a function when one happens, in our case, it is S3 create object event. The service is set up to run several parallel instances of a function, each of which will handle a different request. The incoming workload, which is automatically managed by AWS Lambda, determines the number of instances running at any given time.

2.3 Member Tasks

Aatish Dineshbhai Chaudhari - 1225418103

I was in charge of designing the application's end-to-end flow as well as building up the main AWS architecture and its services. I set up DynamoDB, and input/output S3 buckets, built and deployed the docker image on ECR, and worked with my colleagues to comprehend and learn various methods of accessing the components. I was the primary tester and debugger for the entire application. I used the workload generator provided to us to transmit a large number of requests (both Test case 1 and Test case 2) to test the complete application.

Priyal Ritesh Desai - 1225416166

I created an "uploader.py" script to upload a JSON file containing student information to DynamoDB. This involved reading a JSON file, parsing it, and inserting the data into a DynamoDB table using the boto3 library. I also wrote the script to retrieve videos from the "cse546project2input" S3 bucket and virtually store them in the "/tmp" folder. I also implemented a feature to extract images from the videos using ffmpeg. I accomplished this by reading in the video file, extracted 4 frames using ffmpeg, and then storing those images in the /tmp folder as well for further recognition.

Verik Jagdish Vekaria - 1225674957

I contributed three functions to the code: process_image(), get_data_from_dynamodb(), and store_into_s3(). The process_image() function reads images from the /tmp directory, recognizes faces with the face recognition library, and returns the names of the students who were recognized. The function get_data_from_dynamodb() retrieves information about recognized students from a DynamoDB table. The function store_into s3() generates a CSV file containing the student's name, major, and year and uploads it to an S3 bucket. These functions were used in the face_recognition_handler() function to process a video file, recognize students' faces, retrieve their data from DynamoDB, and upload the data to an S3 bucket.

3. Testing and evaluation

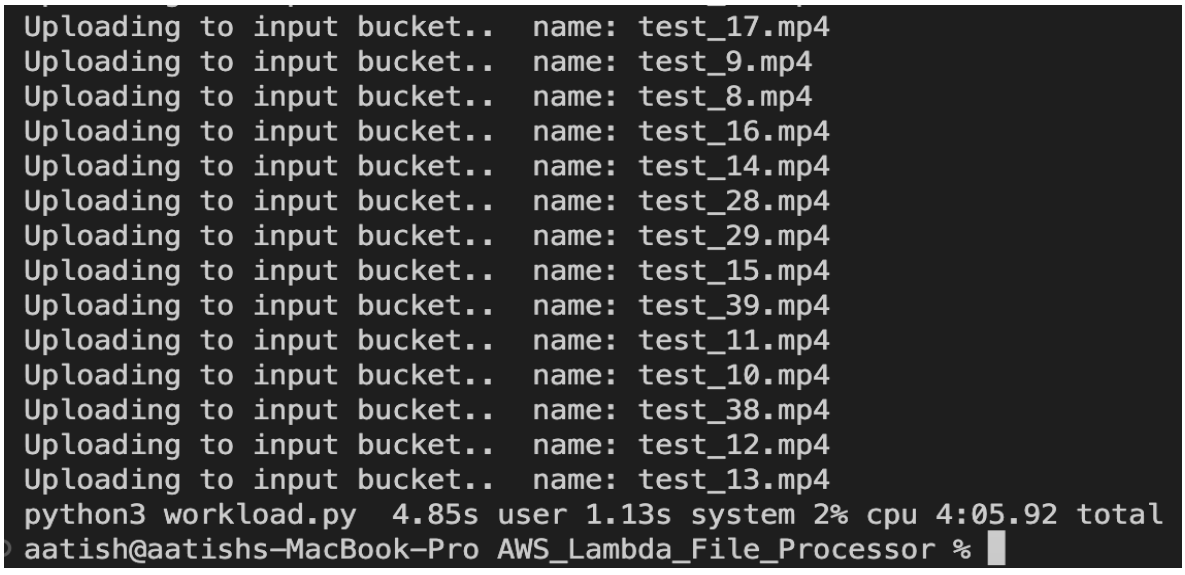
3.1 Testing:

- Tested whether the video is uploaded in the S3 input bucket properly.
- Tested whether the contents of student_data.json are properly uploaded to DynamoDB.
- Tested whether lambda events are triggered when the video is uploaded in S3.
- Tested whether the event triggered is triggered only when the video is uploaded to the S3 bucket.
- Tested whether a new lambda instance runs on every event.
- Tested whether the video is stored in the /tmp folder for processing.
- Tested whether images extracted from the video are present in the /tmp folder.
- Tested whether the face recognition is proper.
- Tested whether the search in dynamoDB is working properly.
- Tested whether the csv file is stored in the S3 output bucket.
- Tested whether the name of the csv file is correct based on the input video.
- Tested whether the content in the csv file matches the expected output.

3.2 Evaluation:

We put the system through rigorous testing to determine its efficiency and accuracy in processing a huge number of videos. We fed 108 videos into the system by running workload.py and timed how long it took to process them. We are glad to inform that the system analyzed all 108 videos in just 4 minutes, which is a remarkable effort given the volume of data handled.

By comparing the recognized faces to the data stored in the DynamoDB database, we were able to assess the system's accuracy. We personally checked the findings and discovered that the system was quite good at recognizing faces and retrieving the associated data from the database.



4. Code

4.1 Program Files

Uploader.py

This Python script is designed to upload JSON data to a DynamoDB “student” table using the AWS SDK for Python (boto3).

handler.py

This Python script demonstrates how to build a face recognition system on AWS using services such as S3, DynamoDB, and Boto3. The script follows a straightforward workflow, first downloading a video object from an S3 bucket, then extracting frames from the video with FFmpeg and processing each frame with a face recognition library to obtain face data. The script then compares the face data to pre-stored encodings associated with student names in a pickle file. When a match is found, the DynamoDB table is searched for student data, which is then saved in a CSV file and uploaded to an S3 output bucket.

The script also includes several helper functions:

```
# 1.Handler Function

def face_recognition_handler(event, context):



# 2.Function to process video and extract frames

def process_video_object(file_name):



# 3.Function to process image using face recognition lib and get face data

def process_image():



# 4.Function to print temp folder contents

def print_temp_folder_contents():



# 5.Function to create CSV of student info and put in output S3 bucket

def store_into_s3(file_name, content):
```

```
# 6.Function to read the 'encoding' file

def open_encoding(filename):



# 7.Function to search student name in dynamoDB and get info

def get_data_from_dynamodb(names):
```

requirements.txt :

This is a list of packages and their versions needed to run our application. When building a new web tier, it should be installed exactly as is.

4.2 Installation And Execution

4.2.1 Installation

1. **Push Docker Image to ECR**
    a. Retrieve an authentication token and authenticate your Docker client to your registry.
        i. aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 450187694173.dkr.ecr.us-east-1.amazonaws.com
    b. Build your Docker image using the following command. For information on building a Docker file from scratch see the instructions here . You can skip this step if your image is already built:
        i. docker build -t repo_project2
    c. After the build completes, tag your image so you can push the image to this repository
        i. docker tag repo_project2:latest 450187694173.dkr.ecr.us-east-1.amazonaws.com/repo_project2:latest
    d. Run the following command to push this image to your newly created AWS repository:
        i. docker push 450187694173.dkr.ecr.us-east-1.amazonaws.com/repo_project2:latest
2. **Lambda function**
    a. Create AWS Lambda function using the ECR Image.
    b. Increase ram size.
    c. Add permission to access S3 and dynamoDB.

d. Make sure lambda processor architecture is same as the one your build your docker image like x86,arm

3. **dynamoDB**

    a. Create Table to store student data
    b. Make name as partition key
    c. Export student.json data to table

4. **S3**

    a. Create two buckets one for input and another for output

4.2.2 Execution

From the local terminal execute the following command to run the workload generator runs 2 test cases one with 10 videos and another with 100 videos.

```
python3 workload.py
```

5. Individual Contribution Report For Portfolio

5.1 Verik Jagdish Vekaria (1225674957)

Design:

As part of my contribution to the project, I focused on integrating the face recognition library with AWS services like S3 and DynamoDB. I started by familiarizing myself with the face recognition library and its capabilities, which allowed me to understand how to read faces from a video and compare them to a predefined encoding.

Based on this understanding, I designed the process flow for the application, which involved reading images from the /tmp directory, recognizing faces with the face recognition library, querying DynamoDB for student data, and storing the data in a CSV file i.e. the student's name, major, and year. This CSV file was then uploaded to S3 bucket, making it accessible for further analysis.

Implementation:

After finalizing the design, I started the implementation phase and began coding the function that would make the core of the application.

- The first function, process_image(), read images from the /tmp directory and recognized faces with the face recognition library. It then returned the names of the recognized students.
- The next function I developed, get_data_from_dynamodb(), was responsible for querying DynamoDB for student data based on the names returned by the process_image() function. This function retrieved the data for each recognized student, including their major and year, and returned it to the main program.
- Finally, I implemented the store_into_s3() function, which generated a CSV file containing the student data and uploaded it to an output S3 bucket. The file was named based on the input video file, ensuring easy identification and tracking of data.

Testing:

- Tested whether the face recognition is proper.
- Tested whether the search in dynamoDB is working properly.
- Tested whether the csv file is stored in the S3 output bucket.
- Tested whether the name of the csv file is correct based on the input video.
- Tested whether the content in the csv file matches the expected output.

5.2 Priyal Ritesh Desai (1225416166)

Design:

I worked with my teammates to design the flow of the project. I primarily helped with writing a script for uploading json data to DynamoDB, retrieving the videos from the S3 input bucket, storing the video and then extracting the images and storing them in the /tmp folder. Overall I designed and implemented a robust and scalable solution for extracting images from videos and stored them temporarily for further use for face_recognition functionality.

Implementation:

- I created a file uploader.py for storing the json data to DynamoDB. For that, I first established a connection to DynamoDB using the boto3 library. The method reads the student_data.json file, parses the file and then iterates over the JSON object to insert each item into the 'student' table.
- In handler.py, I implemented the following functions:
    - Face_recognition_handler: It is to be executed as soon as the video is uploaded in the S3 bucket and the event is triggered. It retrieves the file_name from the event object and calls the consequent functions for further processing.
    - Process_video_object: It downloads the video file from the S3 bucket and stores the video in the /tmp folder of the currently running lambda instance virtually. Then it extracts frames from the video. For extracting the frames from the video and storing them to the /tmp folder ffmpeg is used.

Testing:

- Tested whether the video is uploaded in the S3 input bucket properly.
- Tested whether the contents of student_data.json are properly uploaded to DynamoDB.
- Tested whether lambda events are triggered when the video is uploaded in S3.
- Tested whether the event triggered is triggered only when the video is uploaded to the S3 bucket.
- Tested whether a new lambda instance runs on every event.
- Tested whether the video is stored in the /tmp folder for processing.
- Tested whether images extracted from the video are present in the /tmp folder.