CSE 546 Project II: PaaS Report

Prem Kumar Amanchi
pamanchi@asu.edu
ASUID: 1224421289

Manohar Veeravalli
mveerava@asu.edu
ASUID: 1225551522

Sudhanva Moudgalya
srmoudga@asu.edu
ASUID: 1219654267

1 Problem Statement

The architecture of our system is designed to leverage AWS Lambda for image recognition within video files. The user initiates the process by uploading a video file to the designated S3 bucket, "cse546-paas-input-bucket-videos," which serves as the input repository. Upon successful upload, our Lambda function is triggered, orchestrating the subsequent actions. This Lambda function is containerized within a Docker image, ensuring a portable and consistent environment. Within the Dockerized Lambda, the user's uploaded video is fetched, and the video is processed to extract individual frames.

2 Design and Implementation

2.1 Architecture

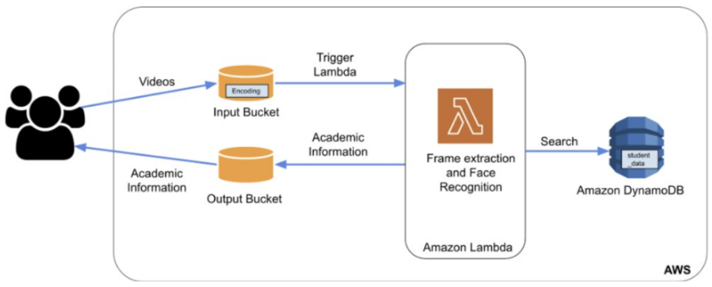

Figure 1: Architecture.

Image recognition is performed on the first frame using a provided encoding file for image comparison. The identified image is then cross-referenced with student data stored in Dynamo DB, specifically the "student" table. The result of this comparison yields the student's name, and subsequently, their academic details including name, major, and year are extracted. These details are consolidated into a CSV file and stored in an output S3 bucket. Finally, the results are returned to the users, providing educators with invaluable insights into their classrooms. This architecture ensures efficient image recognition and data retrieval from a user's uploaded video, streamlining the educational process.

2.2 AutoScaling

Our application leverages the capabilities of Platform as a Service (PaaS) in conjunction with AWS Lambda to seamlessly adapt to fluctuations in demand and system load. This dynamic scaling, managed by AWS, enables us to respond to real-time demands efficiently and cost-effectively.

2.3 Member Tasks

1. **Tasks for Prem:**

   (a) Deployed the container image in Lambda by creating a Lambda function, configuring the container image as the execution environment, setting up necessary environment variables and permissions for the Lambda function to execute. Additionally, monitoring and logging were set up to ensure proper tracking of the Lambda function's performance and execution.

   (b) Created the face recognition part in *handler.py*. It first categorizes the videos into images and look for the face encodings in the image.

   (c) Assigned a role to the AWS Lambda function, granting it specific permissions for various AWS services. The role was configured with the necessary permissions for interacting with S3 (Simple Storage Service) to read and write data, CloudWatch logs to allow logging and monitoring of the Lambda function's activities, and DynamoDB for accessing and modifying data in the database. The permissions were carefully set up using AWS IAM (Identity and Access Management) to ensure the Lambda function had the required access to the specified AWS resources while adhering to the principle of least privilege.

2. **Tasks for Manohar:**

   (a) Built docker image in the local environment and uploaded the built Docker image to the AWS ECR (Elastic Container Registry) repository, ensuring the necessary permissions and authentication are set up for the upload process.

   (b) Uploaded the dataset into the DynamoDB, configuring the necessary tables and attributes as per the data schema requirements, and leveraging the AWS SDK or AWS Management Console for the upload process.

   (c) The code implementation involved in the extraction of data from DynamoDB has been successfully executed within the *handler.py* file.

3. **Tasks for Sudhanva:**

   (a) Created S3 buckets in AWS to facilitate the storage and management of data. This involved navigating to the AWS Management Console, accessing the S3 service, and creating new S3 buckets with specific configurations for storing input data and output files. Permissions and policies were set to ensure secure and controlled access to the S3 buckets.

   (b) Assisted in implementation of a code segment to enable the retrieval of the video from the designated S3 bucket and the subsequent uploading of the output CSV file to another designated S3 bucket. This process involved using the AWS SDK or libraries compatible with the Lambda environment to manage S3 interactions securely.

(c) Created a table in DynamoDB to store relevant data for the application. This step included defining the table schema, specifying primary keys, and configuring attribute properties.

3 Testing and Evaluation

Initially, our testing phase involved the upload of a single video file to the S3 bucket, coupled with the strategic placement of custom logs to meticulously monitor the application's workflow. To monitor these logs, we effectively utilized AWS's in-house CloudWatch application. This method enabled us to successfully upload the video file, subsequently triggering AWS Lambda for image recognition on the video's initial frame extraction. Following the receipt of image recognition results, we executed a search in DynamoDB for matching data, which was then stored as an csv file in the S3 output bucket. To assess and validate our system's performance, we extended our testing to include the use of a workload generator, responsible for uploading test video files to the Input S3 bucket. This, in turn, triggered the Lambda function once the video data became available in the S3 bucket.

Number of Videos Uploaded	**Amount of Time Taken for Processing the Video's**
1	10s
5	20s
10	29s
50	1 min 1s
100	2 mins 5 s

Table 1: Number of videos processed with respect to time.

4 Code

Functions present in the code (handler.py) :

4.1 download_video_from_s3(bucket_name, object_key, destination_path)

This function downloads a video from an S3 bucket specified by `bucket_name` and with the given `object_key`. The video is stored in the `destination_path`.

4.2 extract_images_from_video(video_path, image_output_path)

This function extracts images from a video located at `video_path` and saves them in the `image_output_path`. It uses the FFmpeg library for this task.

4.3 process_image(img_path)

This function processes an image for face recognition. It takes the path to an image (`img_path`) and returns the name of the recognized person based on face recognition.

4.4 get_target_from_dynamodb(name)

This function retrieves data from DynamoDB. It takes a `name` as an input and queries the DynamoDB table for matching records.

4.5 create_csv_file(object_key, record)

This function creates a CSV file with academic records. It takes an `object_key` (used to name the file) and a `record` containing name, major, and year, and saves this data to a CSV file. The CSV file is then uploaded to an S3 bucket.

4.6 face_recognition_handler(event, context)

This is the main Lambda function for processing facial recognition. It's triggered by an S3 event and coordinates the download of a video, extraction of images, face recognition, and database query. It then creates a CSV file with academic records for the recognized person and uploads it to an S3 bucket.

5 Setup

5.1 Cloning the Repository

To get started, clone the GitHub repository for the project to your local machine using the following terminal command:

```
git clone https://github.com/PremAmanchi/CSE546-PaaS
```

5.2 Building and Pushing Docker Image

Navigate to the "docker" directory within the cloned repository using the `cd` command:

```
cd CSE546-PaaS/docker
```

Build and push the Docker image to Amazon Elastic Container Registry (ECR) by following these commands:

1. aws ecr get-login-password –region us-east-1 — docker login –username AWS –password-stdin 548832462032.dkr.ecr.us-east-1.amazonaws.com
2. docker build -t cse546-paas-docker-image .
3. docker tag cse546-paas-docker-image:latest 548832462032.dkr.ecr.us-east-1.amazonaws.com/cse546-paas-docker-image:latest
4. docker push 548832462032.dkr.ecr.us-east-1.amazonaws.com/cse546-paas-docker-image:latest

This will build the Docker image and upload it to AWS ECR.

5.3 Setting Up AWS Resources

To complete the setup, create the necessary AWS resources:

- Create an S3 input bucket named "cse546-paas-input-bucket-videos."
- Create an S3 output bucket named "cse546-paas-output-bucket-results."
- Create a DynamoDB table named "student-academic-records" and load the $student_data.json$ file into this table.

5.4 Creating an AWS Lambda Function

Create an AWS Lambda function with the following configuration:

- Use the Docker image from ECR.
- Select the appropriate processor (e.g., arm or x86_64) based on your local system.
- Set the trigger point to the S3 input bucket, "cse546-paas-input-bucket-videos."
- Increase the Lambda function's memory to 512 MB and the timeout to 10 minutes.

5.5 Executing the Workload Generator

Use the workload generator tool to upload video files to the S3 input bucket, "cse546-paas-input-bucket-videos." Once a video is available in the S3 input bucket, the AWS Lambda function will be triggered.

5.6 AWS Lambda Function Processing

The AWS Lambda function will process the image, run the face-recognition module, and store student details in a CSV file. The CSV file will be uploaded to the S3 output bucket, "cse546-paas-output-bucket-results."

By following these setup instructions, you can successfully configure the system, build and deploy the Docker image, and run the code to process videos and recognize students' faces.