CSE 546 — Project - 2 Report

Sai Varun Reddy, Mullangi
Siva Nagi Reddy, Munaganuru
Pavan Krishna Reddy, Madireddy

1. Problem statement

Our project aims to develop a cloud-based application that integrates advanced face-recognition techniques using the Python face-recognition library, it processes the collected classroom videos, extracts students' faces, and recognizes them using the face-recognition model. Once the students are identified, the assistant fetches their corresponding academic records from the DynamoDB database, providing educators with a comprehensive view of each student's performance. This integration of face recognition and database management allows for an efficient, accurate, and cost-effective solution to manage student recognition and academic record retrieval. Our project is implemented in the AWS Lambda Platform as a Service (PaaS) environment, which allows for seamless auto-scaling to meet the varying demands of the users.

2. Design and implementation

2.1 Architecture

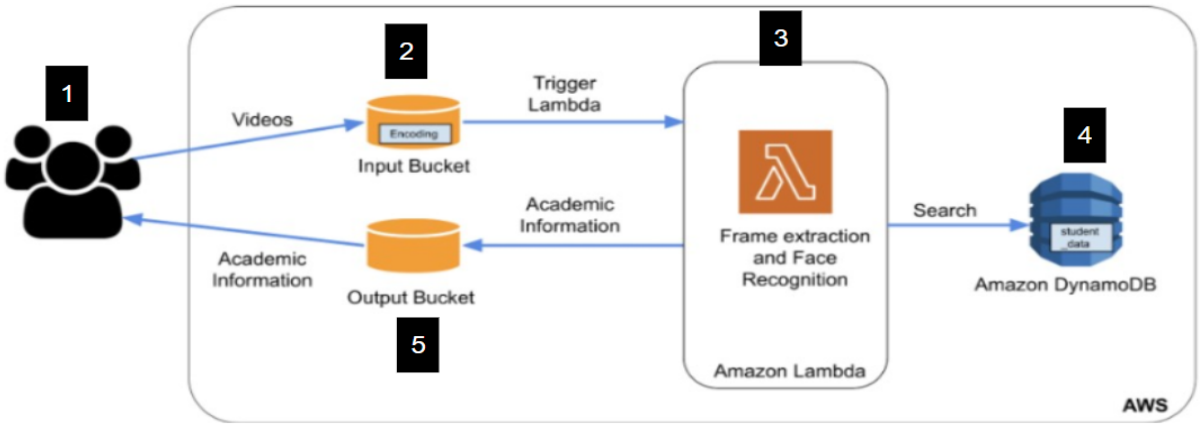

Fig 1- Architecture of the smart classroom assistant for educators

The above picture is the overall higher level with all major components of our implementation of this project. Each component's functionalities are as follows.

Component 1 - Workload generator:

To test the speed and accuracy of the *smart classroom assistant cloud* service, we use a concurrent workload generator that enables us to make multiple requests simultaneously.

Component 2 - Amazon S3: Input Bucket

The Videos taken from the classroom or for testing or uploaded into this input S3 bucket and whenever a new video is uploaded it triggers component 3(lambda function)

Component 3 - Lambda Function

The AWS Lambda function in this project is responsible for handling the main face recognition and student record retrieval process. Upon receiving an event, the Lambda function first retrieves the video file from the component2 S3 input bucket and downloads it to a temporary location. Then, it extracts frames from the video using the FFmpeg tool, which allows for efficient video processing. The face recognition process begins by loading the known face encodings (stored in a file) and iterating through the extracted frames to identify student faces. It compares the face encodings in the frames to the known encodings using the Python face-recognition library and identifies the best match for each detected face.

Once the students are recognized, the Lambda function queries the DynamoDB table using the student's name as the partition key. It retrieves the major and year information for each identified student and creates a DataFrame to store this data. This DataFrame is then converted to a CSV file, which is uploaded to the component 5 S3 output bucket. By implementing the face recognition and record retrieval processes in the Lambda function, this project creates a seamless and efficient workflow that successfully meets the goals of the smart classroom assistant.

Component 4 - Amzon Dynamo DB

Dynamo Db is loaded with the academic records of the students. And lambda function queries these records to find a student record.

Component 5 - Amazon S3: Output Bucket

This output bucket contains CSV files with the name, major, and year of the recognized students.

2.2 Autoscaling

Autoscaling is a key feature of AWS Lambda that allows our smart classroom assistant application to automatically scale in and out to meet the demand. Whenever a video is uploaded in the input bucket, the Lambda function is invoked, and AWS will automatically provision the necessary compute resources to run code based on the incoming requests. As the number of requests increases, AWS will scale out the compute resources by launching additional instances of our function. Similarly, as the number of requests decreases, AWS will scale in the compute resources by terminating the unnecessary instances. This allows the application to handle sudden spikes in traffic without any manual intervention. In our project, we have leveraged this

autoscaling capability by creating a Docker image and using it to create an AWS Lambda function. This function will automatically scale in and out based on the incoming requests, ensuring that our application can handle varying workloads efficiently

2.3 Member Tasks

Munaganuru Siva Nagi Reddy - Component (2,4,5)

As part of my project responsibilities, I took charge of configuring the Input and out S3 buckets and attached for lambda attached a trigger to the input bucket(component 2). So that whenever a user uploads a video file in the form of mp4 into it a trigger is sent to invoke the lambda function. And also updated the dynamo DB with student academic records and have provided permissions for the lambda function so that it can fetch the student records and then these outputs are stored in the output bucket in the form of CSV.

Sai Varun Reddy Mullangi - Component (3)

As part of my project responsibilities, I worked on Component 3 which involves frame extraction and recognition. To facilitate debugging, I set up logging to capture and analyze any errors encountered during the project setup on AWS.

I focused on the face recognition event handler, which triggers the processing of videos when there is an object create event, such as a video uploaded to S3. Using ffmpeg, I divided the video into frames and stored them in a temporary folder as Lambda has write access only to the tmp folder.

Next, I used the face recognition library to detect faces among the frames, and optimized the process by implementing a loop-breaking logic that stops the processing once the first match is found, thereby avoiding processing of the remaining frames. Finally, based on the name of the identified face, I queried the DynamoDB to retrieve the academic information of the person and wrote the information into a CSV for storage in the S3 bucket.

Pavan Krishna Reddy Madireddy - Component (1, 3)

As part of the project, I worked on two parts, the first one being the Dockerfile where I had to go through the given file and understanding all the details within the file. To complete the project I had to make some customizations like adding some extra files using COPY command. Then I built the docker image locally using the docker client which was saved in my macbook. Then I assigned a tag and uploaded it to Amazon Elastic Cloud Registry private repository which can be used in further steps of the projects to create a lambda function from the image.

I also worked on the workload generator part where I had run various tests and verified the sanity of the output produced by the lambda functions. The workload generator uploads videos into

input S3 bucket which inturn triggers the lambda functions as configured and generated the output csv files into output buckets. Verified the output csv files data with the expected output.

3. Testing and evaluation

From the test we conducted, we can see that the service is upscaling (Lambda inbuilt functionality) and handling the workload without any breakdowns. The whole process of the reading the video, splitting into frames, using the face recognition library to detect faces, querying the Dynamodb for information and finally storing the results in output bucket takes less than a minute.

For the workload of 100 concurrent requests below is the graph for the number of Lambda invocations

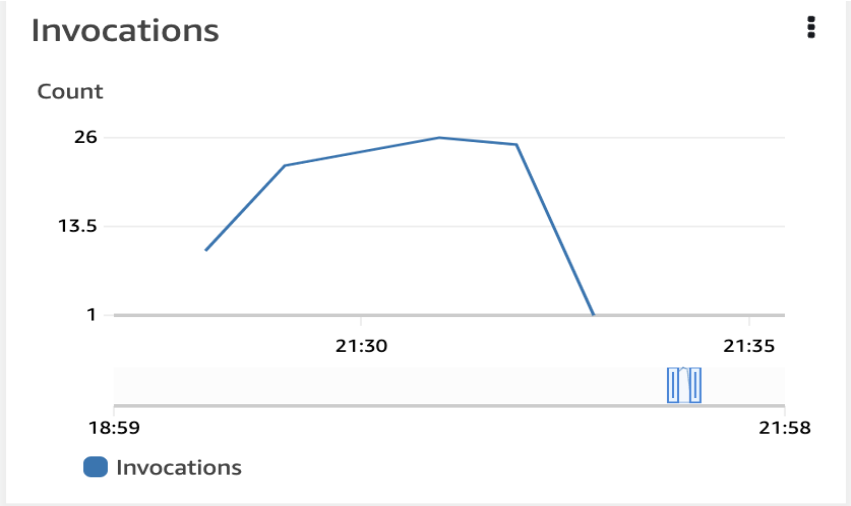

Fig: Lambda Invocations

Next we checked for the workload of 100, there are no errors or missing data datapoints, which is evident in the below graph

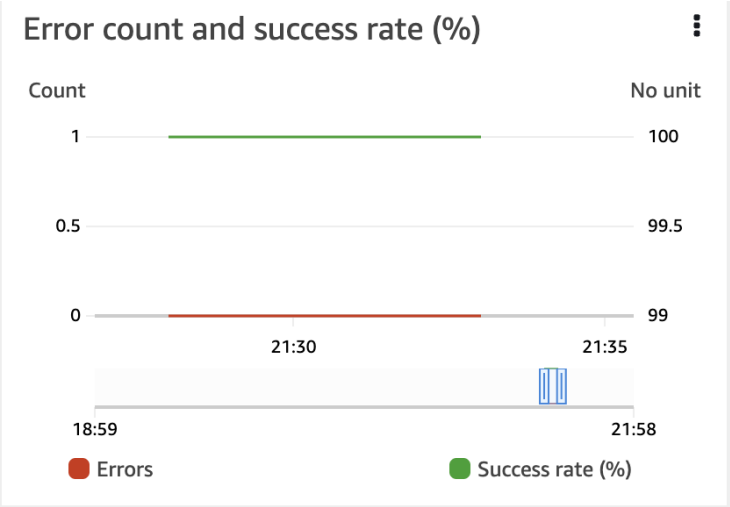

Fig: Success rate

Finally we did a sanity check of the processed videos by random downloading the csv files from output bucket.

4. Code

handler.py

In our handler.py file, we have a function named face_recognition_handler that is triggered whenever an object is added to a specified S3 bucket. First, we create two variables input_bucket and output_bucket, which represent the input and output S3 buckets, respectively. Then, we set up logging to provide insight into the code's execution. We also create S3 and DynamoDB clients to interact with the respective services. We define a function called open_encoding to read the 'encoding' file.

Next, the face_recognition_handler function is called with an event and context parameter. The event contains information about the S3 object, such as the bucket name and key. We download the video from the input bucket, store it in the /tmp directory, and extract the frames from the video, which are saved in the /tmp/frames directory. We use the face_recognition library to recognize the face in each frame, and then we query the DynamoDB table using the face's name to retrieve additional information about the person, such as their major and year. Finally, we write the data to a CSV file and upload it to the output bucket. We created the "tmp" folder to store the frames because Lambda does not have write access to any folder other than /tmp.

Dynamo.py

In this file, we are importing the boto3 and json modules. We then open a JSON file named "student_data.json" in read mode and parse the JSON data using the json.load() method. We then

iterate over the parsed data and create a new dictionary where we convert the id to a string and store it in a dictionary with key "N" and the other fields in a dictionary with key "S". We append this new dictionary to a list named "new_data". We then create a DynamoDB client using boto3 and specify the table name. We upload the data to the table using a for loop that iterates over each item in the "new_data" list and uses the put_item() method of the DynamoDB client to upload each item to the table. Finally, we print the response received from the put_item() method.

Docker File :

For the AWS lambda function we are using a custom-built Image and the Docker File in the code is used to build this image. The image is based on Alpine Linux, a lightweight Linux distribution, and uses Python 3.8 as the runtime environment. The Dockerfile is split into three stages. The first stage installs GCC and updates pip. The second stage builds the function and its dependencies and installs the Lambda Runtime Interface Client for Python. The third stage creates the final runtime image, copies the built dependencies, and installs libraries present in the requirements.txt. The entry point script /entry.sh is used to run the handler function handler.face_recognition_handler when the container is started.

Overall, this Dockerfile sets up a container environment that can be used to run a Python function on AWS Lambda. The use of Alpine Linux keeps the image small and lightweight, while the three-stage build process ensures that only the necessary dependencies are included in the final image. The installation of ffmpeg allows the function to process video data, and the entrypoint script simplifies the process of running the function in a container.