CSE 546 - Project Report by Group Alpha

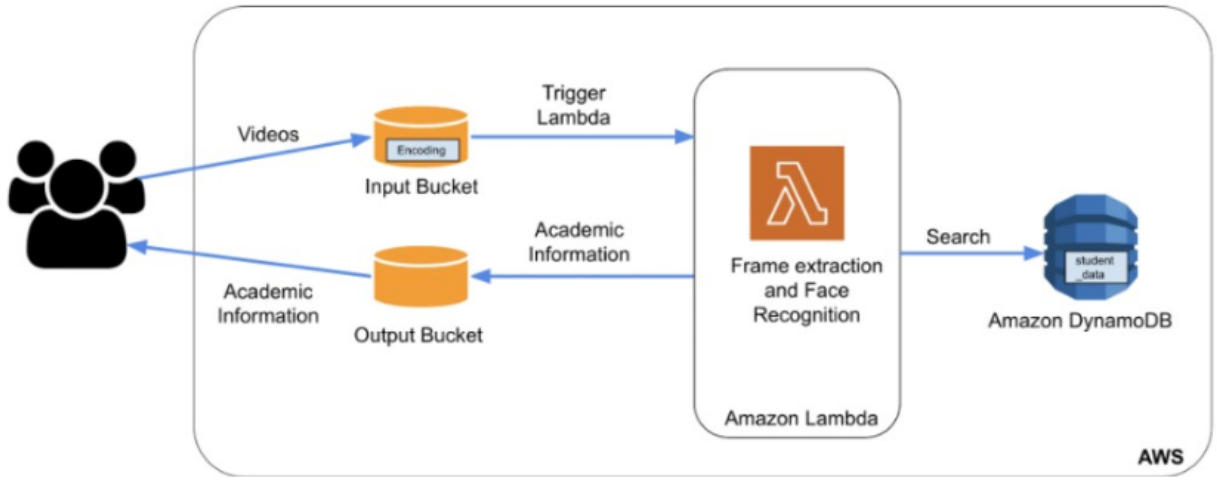
Abhiram Gulanikar *(1222295360)*
Vinay Pandhariwal *(1219501465)*
Prathmesh Sambrekar *(1222154895)*

1. Problem statement

In this project, we are required to build an elastic application that can automatically scale out and in on demand and cost-effectively by using the PaaS cloud. Specifically, we will build this application using AWS Lambda and other supporting services from AWS. The cloud app will implement a smart classroom assistant for educators. This assistant should take videos from the user's classroom, perform face recognition on the collected videos, look up the recognized students in the database, and return the relevant academic information of each student back to the user. Furthermore, talking about the project's importance, the project deals with a very recent trend in cloud computing of Function as a service (FAAS) where a piece of code/function is only executed in a PAAS-like auto-scalable environment.

2. Design and Implementation

2.1 Architecture



2.1.1 Components

- **AWS S3**: The Amazon S3 consists of an input and the output bucket. Users upload videos to the input bucket stored in S3. When a new video becomes available in the input bucket, it triggers the Lambda function to process the video. Finally, the output from the lambda function consisting of the student's academic information is stored as a file in the output bucket in S3.
- **AWS LAMBDA**: After the video has been uploaded to S3 using the PUT Triggering event, the lambda code will be invoked. A container image serving as the endpoint for a lambda function will contain a single handler.py file that will handle all events sent by the lambda function.
  One face recognition handler function, which accepts an event as an argument, will be executed by the handler.py file. Prior to collecting movies from the S3 bucket, the face recognition handler function will attempt to retrieve S3 Information from the event, such as the bucket name and key name. The video from S3 must then be downloaded to a local directory called tmp before it can be used to extract frames with the ffmpeg library. The face recognition results are utilized to retrieve student data from DynamoDB. The student data will then be saved to a csv file and placed in the output bucket on S3.
- **DynamoDB**: The program's data is stored in the DynamoDB component. This is where the comprehensive student data is kept. The following information concerning students is kept on file: NAME, MAJOR, YEAR. To get the requested student's data, the lambda sends a request to DynamoDB.

2.2 Autoscaling

We make use of AWS Lambda to handle the auto-scaling functionality. Every time a video is uploaded in the S3 input bucket, the Lambda function is invoked, and AWS Lambda creates an instance of the function and processes the event. Once the event returns a response the handler code will process it and keep the instance active and wait to process new events. If the function is invoked when the existing instances are handling existing events, then Lambda initializes another instance to handle the new event. As multiple events occur lambda route them to available instances or creates new instances as required.
When the number of events decrease (in this case S3 PUT operation) lambda stops unused instances to free up space for other functions. In this manner auto scaling is handled in our project.

2.3 Member Tasks

Abhiram Gulanikar:

- I was involved in developing/deploying end-to-end cloud-based architecture of the project. The cloud architecture which involves services like AWS S3, AWS Lambda, AWS DynamoDB.
- **Trigger Lambda form S3:** There are numerous ways to implement the design phase of integrating S3 with Lambda functions, including manually passing a video key name to the lambda function, which will fetch the video, or having the lambda function itself be triggered whenever a video is uploaded to S3. I have used the design that AWS Lambda gives us for event triggering.
- **Implementation:** I implemented the functionality to extract frames from the videos and compare their encoding with the database encodings, retrieve the fields from DynamoDB and write the fetched information to csv file and put the file in the output bucket in S3.
- I tested the application using the workload generator. I verified the accuracy of the application by comparing the results with the expected results provided in the mappings file and confirmed all inputs are processed through the CloudWatch records and logs.

Vinay Pandhariwal

- I was involved in the developing/deploying end-to-end cloud-based architecture of the project i.e., designing program on lambda function and to containerize it.
- **Face recognition in Lambda:** This functionality was created with the assumption that the facial recognition function would be used with the picture path from the lambda and would return the name of the student in the image as the result.
- **Docker image creation for the lambda function:** I must create the Docker Image for the lambda, handling other services and face recognition into a single docker image, to incorporate the face recognition model into lambda function. I used a Docker file including all the necessary commands to produce a docker image that is ready to be run on a lambda in the cloud and has all the necessary environment established. Then, for a new lambda to be formed utilizing it, I uploaded the image to AWS Elastic Container Registry.
- **Fetching information from DynamoDB:** This capability was created with the idea that it would be given the student's name for which it would retrieve data from the DynamoDB students table. I have created a student table in DynamoDB with a given schema.
- I have also validated that the DynamoDB service is fetching the correct record for each query with a given student name from the DynamoDB.

Prathmesh Sambrekar

- I was involved in developing/deploying end-to-end cloud-based architecture of the project. The cloud architecture which involves services like AWS S3, AWS Lambda, AWS DynamoDB. The lambda consists of following services: Fetching videos from S3, fetching face-recognition results from face recognition model, fetching information of the student recognized in previous step, putting student information in S3.
- **Design Lambda Function:** The primary goal of the project was to utilize lambda functions. AWS offers a variety of implementation options for these functions, and I chose to execute function via container image as a design choice to handle videos and run lambda function directly based on image from AWS ECR (Elastic Container Registry).
- I wrote a functionality to fetch videos from S3 using boto3 in python and store it in the tmp folder in the container.
- I tested the application using the given workload generator and tested the services using both unit testing and integration testing. I verified the accuracy of the facial recognition model that is now operating in a Docker container by logging its output and reviewing CloudWatch logs.

3. Testing and Evaluation

Test Cases	Evaluation
Frame Extraction from Video	Frame from the video is extracted after 0.4 sec
Video Upload	Video Upload in S3 confirmed
Retrieving result for each video sent at every 0.5-sec interval	Correct result for each video with latency around 0.5 seconds
Each video that is posted to S3 in Lambda's storage should be downloaded locally.	Printing list of files present in the local folder confirms the video download.
Result of face recognition	Printed the name of the student as output
Fetching information of the name (received from face recognition model) from DynamoDB	Fetched data is recorded to the console for verification.
Put information fetched from DynamoDB to Output bucket in S3	Verify that there is a csv file with results generated in the output bucket in S3

4. Code

- **handler.py**: The python file contains the following functionalities:
    1. A function to read the encoding file
    2. A function to fetch video from S3
    3. A function to put result in S3

4. A function to fetch data from DynamoDB
5. A main function which is the entry point to perform the above functionalities including frame extraction and finally put the results in output bucket in S3.

- **requirements.txt:** Contains all the packages to run the code successfully.
- **entry.sh**
- **Dockerfile:** Contains steps to create the docker image.
- **workload.py:** Contains functionality to create requests to the app.
- **encoding:** Stores the names of the known faces (given by the professor)
- **mapping:** Workload generator uses this mapping to check correctness of the app's output.

Steps to run the application:

- Copy code to be run on lambda function into your desktop/laptop.
- Install docker on your desktop/laptop.
- Install aws cli on your desktop/laptop.
- Open terminal in the code folder directory.
- Run command - "aws configure" and enter Access Key ID, Secret Access Key and Default region name (Fetch this information from AWS).
- Run command "docker build -t <image-name> ." to build the docker image of the lambda code.
- Run command to create ECR repository on AWS.
  "aws ecr create-repository --repository-name <image-name> --image-scanning-configuration scanOnPush=true --region <region-name>"
- Run command to tag the image to match the repository name "docker tag <image-name>: latest <Account ID>.dkr.ecr.us-east-1.amazonaws.com/<image-name>:latest"
- Run command to register docker to ECR "aws --region us-east-1 ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin <Account ID>.dkr.ecr.us-east-1.amazonaws.com"
- Run command to push the image to ECR "docker push <Account ID>.dkr.ecr.us-east-1.amazonaws.com/<image-name>: latest"
- Open lambda console on AWS portal. Create a new lambda function using the "Container Image" option and select the image uploaded in the previous step.
- Add S3 trigger on the lambda and select the correct S3 bucket from the dropdown.
- Modify the general configuration of the lambda - Memory: 2048 MB Timeout: 30s
- Upload the video to the S3 input bucket.
- Output will be visible in S3 output bucket.