# CSE 546 — Project Report

*Hari Sai Charan Challa(1225461861)*

*Sathwik Katakam(1225445585)*

*Hari Murugan Ravindran(1222324688)*

## 1. Problem statement

In this project, we are building an elastic cloud application which will act as a smart classroom assistant for educators. It will effectively make use of the Paas Cloud to automatically scale out and scale in based on demand while being cost-efficient. In this application, it takes videos from the user's classroom as input then performs face recognition on these videos, identifies the student's name, retrieves the data from the database for the recognised students and sends the appropriate academic information of each student to the user. This application will utilize AWS Lambda which is the first and most widely used function-based serverless-computing service and also other supporting AWS Services.

## 2. Design and implementation

### 2.1 Architecture

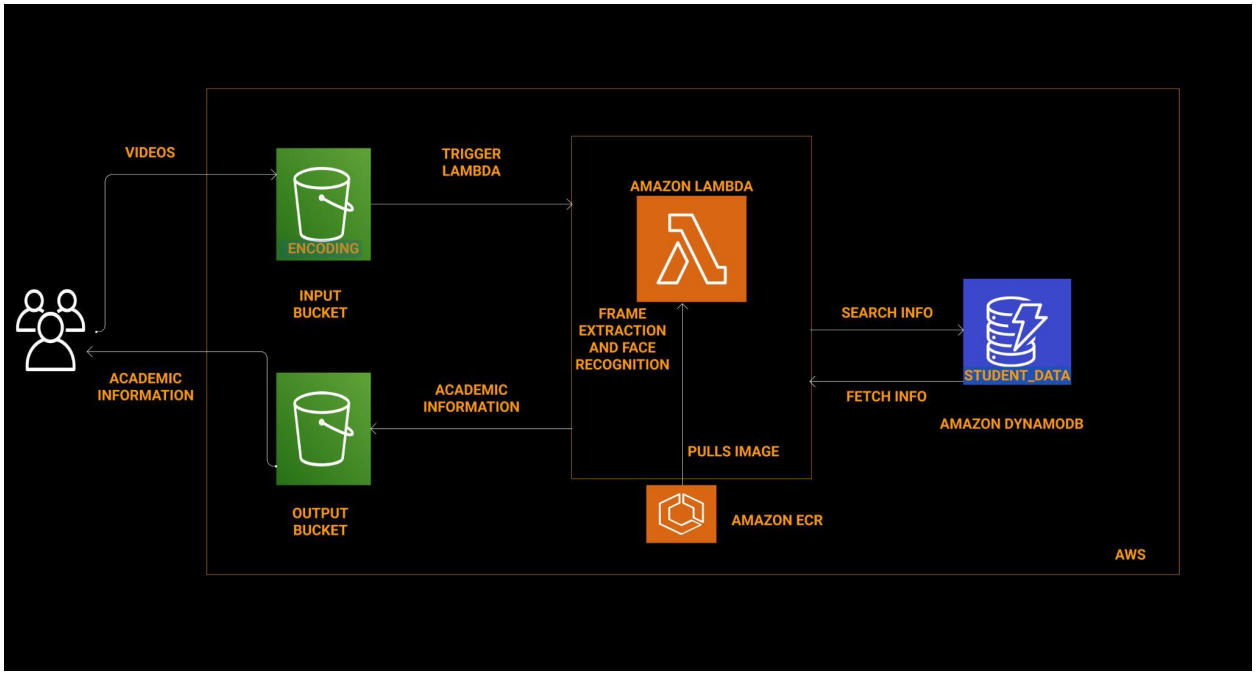


**Figure 1**

**Description:**

We first need to create the initial setup that we need for the smooth execution of our application. Initially, we will be creating two buckets in S3 one acts as the input bucket which receives the input video and the other acts as the output bucket which is used to store the output of the lambda function which is the student's academic information in CSV format. We are also creating our lambda function using the customized container image pre-installed with ffmpeg and face_recognition library. This part of the application is responsible for frame extraction and face recognition. The application's rear end is made up of Amazon DynamoDB which is a NoSQL database service which is used for storing the student's academic records. First, we set up our table which contains students' names along with their respective academic details. It is used by the lambda function for looking up students' records who are identified in the videos and sending all of their academic records to the output bucket which is followed by sending it to the user from which the videos are pushed to the input bucket as output.

Our application execution can be divided into four major phases:

- Input and Lambda Function Trigger.
- Frame Extraction and Face Recognition.
- Searching in DynamoDB.
- Providing Results to the User.

**Input and Lambda Function Trigger:**

In this first phase initially, the input videos from the user's classroom are uploaded into the input bucket. We have already created a trigger on the input S3 storage thus once the videos are uploaded into the bucket the trigger of our lambda function is invoked.

**Frame Extraction and Face Recognition:**

In this phase, we are making use of the Function as a service(FaaS) provided by AWS called AWS Lambda. We have created our customized Lambda function using the customized container image pre-installed with ffmpeg and face_recognition library. In this part after the trigger is invoked the Lambda function acts and makes use of the libraries installed with the help of the Docker Container to process the input provided in the input bucket. We are doing two major processing in these input videos. First is to extract a frame from these videos and then to recognize the face of the individual in that frame.

**Searching in DynamoDB:**

The final part of the Lambda function is after the name of the individual is identified in the video. We have already uploaded the student's name and their respective academic records to DynamoDB.We try to find the respective student academic records from DynamoDB and use them for generating our results.

**Providing Results to the User:**

After we get the academic records from the DynamoDB for the given student name we need to generate a CSV file which contains the video file name along with the student's academic information and stored it in the output S3 bucket.

**Usage of AWS Services:**

- **AWS S3:**
  - We use two buckets in our Paas Application. First is the input bucket into which the input videos from the users are pushed. This bucket also acts as the trigger for our Lambda Function.
  - The Next one is our Output Bucket which is used to store the identified students' academic information fetched from the DynamoDB in a Separate CSV file.
- **AWS ECR:**
  - Amazon ECR is the fully managed Container Registry which offers us high-performance hosting so that we can deploy our application images and artifacts anywhere.
  - We are building our lambda function on a customized container image pre-installed with ffmpeg and face_recognition library.
- **AWS Lambda:**
  - It is one of the most important services in implementing the Paas Application. We can run our code thinking about setting up servers, etc thus it is called Serverless computing. It is also very useful for the elasticity of our application as it automatically scale out and scale in.
  - In our application first, this function is triggered by the input Bucket where the computation occurs for frame extraction and face recognition.
  - It is also responsible for fetching the student academic information from the Amazon DynamoDB, tabulating it into the CSV file and pushing it into the output.
- **AWS DynamoDB:**
  - It is the NoSQL database used for storing the data in key-value stores as well as the document data structure.
  - In our application we are storing the students academic information on the database which is then retrieved for the specific identified students which we use for the Output report generation.

## 2.2 Autoscaling

We have built this application keeping in mind to reduce the waiting time of users to get their expected student's academic records. Our application is used by many numbers of users simultaneously so accommodating all these requests with minimal time for processing is necessary. Thus we make use of AutoScaling to meet the increase in demand for users. So our application will Scale out when there is a sudden surge in the number of users and Scale in when there are fewer users to serve. We provide Autoscaling in our application by leveraging the feature provided in the Lambda Function. As we have

built a PaaS application we make use of Lambda Function for processing the videos uploaded by the users as input in the input bucket. We make use of the Auto Scaling feature provided by the Lambda function named "Concurrency".We have provided a concurrency limit of 10 in our application but by default, we are provided with a concurrency limit of 1000 across all regions.

**2.3 Member Tasks**

**Hari Sai charan challa (1225461861)**

My work in this project spanned mainly across Docker, ECR, S3 buckets and Lambda Triggers. I have first set up a git account and a new AWS account for collaborative development. Then, created docker images after understanding how docker and containers work. I also learnt about AWS ECR and have pushed created docker image into ECR account. I have also created S3 buckets 'input-video-hsh' and 'output-video-hsh', lambda function 'image-function'. Finally, linked all 3 parts docker-image, input bucket and lambda function via lambda triggers. In the end, I have tested all these parts without involving facial recognition code. Once all the parts have been developed we have also done a few rounds of end to end testing.

**Sathwik Katakam (1225445585)**

I was involved in the following three modules of the project: S3 client, Frame Extractor, and Face Detection. The S3 client interacts with input and output S3 buckets and downloads input videos to a local temp path. It also uploads the resultant data obtained after querying the DynamoDB to the output S3 bucket in a CSV format. The Frame Extractor module extracts frames from the input videos using the FFmpeg library and saves them in a local directory. The Face Detection module uses the face_recognition library to load and compare encodings of the saved frames to detect faces. Once a match is found, the detected face is passed on to the next module to query the database for student-related data.

**Hari Murugan Ravindran(1222324688)**

Responsible for the setting up of the student's academic information and retrieval on Amazon DynamoDB and also the part where the identified students' records are tabulated into a CSV file and Pushed into the Output Bucket as Output. Initially, I would set up the Amazon DynamoDB then I used Amazon Cloud9 which is the cloud-based integrated Development Environment(IDE) and created the table used for storing the student academic info, then I used the student_data.json for populating the database which will be helpful in getting the students academic information for the identified student name. Now I'm also responsible for the part where the data is fetched by the lambda function from the student data for the identified student name. Then the data is stored in the CSV files which are then pushed into the Output Bucket. The output file consists of the video file name which the users have uploaded as input on which the frame extraction and face recognition happened, the name of the identified student as well as their academic record. After all of these different kinds of testing are done. I have also manually cross-verified whether the students' names are identified correctly and also whether their respective academic record is stored as the output in the Bucket as the CSV file.

## 3. Testing and evaluation

We tested the project functionalities in a modular fashion ensuring each individual module is working as expected. To ensure the working of lambda function, we first tested it against a custom event by uploading a single random test file to the S3 bucket. This triggers an event change and successfully invokes the client. We then created an image which contains the code and dependencies using docker and uploaded it to the ECR.

The following steps were followed to test and evaluate the system:

1. There are two test case folders that were given to us. One which contains 8 test videos and the second which contains 100 of them. Workload generator was run against these test cases.
2. We ensured that input videos are being uploaded to S3 bucket.
3. Verify that the lambda functions get triggered and invoke the client by automatically scaling based on the demand.
4. Ensure that student json file is being uploaded to dynamo DB and is queryable based on the given partition key (student name)
5. Verify that the output files written to s3 bucket match the count of input bucket and verify the contents of csv file generated for each test case against the expected output.
6. With the 100 test cases being uploaded sequentially to the S3 bucket, it took 2 minutes to process the requests and upload results to the output S3 bucket.

## 4. Code

**handler.py**

Various methods present in the handler file are described below in detail.

- **face_recognition_handler**

  This method is the entry point to the lambda function. Once an object has been pushed into S3 bucket, this function gets triggered. It basically collects videos from the input bucket, processes it and stores the output into different S3 bucket. In the processing step, it basically recognizes the person in the input video using various python libraries like dlib, face-recognition etc.

- **identify_person**

  This function takes an encoding as an input and returns the name of the person associated with the encoding. It uses face-recognition library in python to achieve this functionality.
  compare-faces in

- **query_db**

  This person takes in the person's name as input and returns all the metadata corresponding to that person by querying the DynamoDB. It is a NoSQL database used commonly by various popular organizations to achieve their business goals. It uses the boto3 python library for accessing dynamoDB.

- **write_results_s3**

This function takes the result of face-recognition as an input and uploads this data to S3 bucket as a CSV file.

- **generate_id**

  This function takes in frame number as an input and converts that into three digit decimal representation. For example:- If we send 10 as input then the function returns 010 as output.

- **Remove_file**

  This method removes the file from the hard disk of the server if it exists.

- **open_encoding**

  This function converts the byte stream of the given encoding file into an object notation using the load function of the python pickle library.

**Dockerfile**

This file contains various commands which will build the execution platform required by our application.

First, It installs python and other basic dependencies like ffmpeg etc. Then it installs various python libraries like dlib, face-recognition, numpy etc. Finally, it sets up an entry point to our application.

**requirements.txt**

This file includes all the python libraries like face-recognition, dlib, numpy etc required to successfully implement the project.

Explain in detail how to install your programs and how to run them.

1. Clone the git repository from the given link (https://github.com/hchalla2/Cloud_Project_2)
2. Build a docker image and then push it into AWS ECR service using below commands.
   a. docker build -t docker-image .
   b. docker tag <tag>
   c. docker push <src_account>
3. Create a lambda function and two S3 buckets for input and output storage.
4. Link lambda function to input bucket via triggers setup.
5. Then, deploy docker image present in ECR into lambda function.
6. Now execute the workload generator (python workload.py)

**Individual contributions:**

**Hari sai charan challa (1225461861)**

**Requirements analysis and Tasks division:**

We have spent a lot of time to clearly understand the project requirements to avoid back and forth. Then, the entire project was divided into three tasks and I was majorly involved in building and deploying docker images, setting up lambda triggers, setting up GIT and AWS accounts.

**Implementation:**

**Docker:**

First, I have studied about docker tool and containers using youtube tutorials. Then, prepared docker commands like 'docker build', 'docker push' etc needed for the project using AWS documentation. Also studied about various issues like multi-threading, race conditions etc in the container environments which could happen in our project.

**Lambda setup:**

I have studied about AWS lambda functionality, its advantages through various internet tutorials. I have created a lambda function called 'image-function' to deploy our application without any servers.

**ECR:**

Elastic container registry stores all the images required for the execution of lambda functions. I have studied its functionality and usage from the AWS documentation. We have stored our image named as 'docker-image' in our ECR account.

**Setup S3 based triggers:** I have created 2 buckets 'input-video-hsh' and 'output-video-hsh' to store input and output of our project respectively. After this, I have linked the input bucket to the created lambda function 'image-function' as a trigger.

**Setup GIT repository and AWS account:**

As we have exceeded the S3 requirements for our earlier AWS account, I have created a new AWS account for the 2nd project. Along with this I have also created 3 IAM users corresponding to 3 people on the team on the root account for shared development of the project. I have created a GIT repository which aids in the collaborative development of the team.

**Testing:**

**Integrated-testing:** After setting up docker images, S3 triggers and lambda functions I have done a few rounds of testing without using actual face-recognition code. Few issues like different line breaks (LF, CRLF), missing files were found during this phase.

**End-to-end testing:** Once the face-recognition module and dynamoDB module have been implemented, we have integrated all the required modules and performed a few rounds of testing to make sure everything works satisfactorily. During this phase no major issues were found. This might be due to extensive integrated testing.

**Sathwik Katakam (1225445585)**

**Design:**

I was majorly involved in the design and implementation of frame extraction and face detector modules for the test videos. I have also collaborated with my teammates in the design phase to come up with an architecture for this project. Dividing the projects into modules, we have identified three key parts and split it among ourselves. I also took up the responsibility of handling the S3 client module which interacts with the input and output S3 bucket.

**Implementation:**

1. S3 client: This part is to do with interaction with input and output S3 buckets. Once input videos are uploaded to the input S3 bucket, the lambda function gets triggered and invokes the underlying client instance. There is a method in handler.py that gets invoked for this event-driven change in the S3 bucket. We pass the input S3 bucket and the name of the video file which downloads the data to a given temp local path. There is a second module in the handler.py which helps in uploading the resultant data obtained after querying the dynamo DB. This method takes in the bucket name, the local file path and the resultant output file to be written. Output is written in a csv format with proper headers for each test file in the input bucket.
2. Frame Extractor:
    a. This module is responsible for extracting frames from the video. We create a temporary directory in the local machine and save these frames extracted from the videos from the input S3 bucket. FFmpeg library was used to extract JPEG frames for these videos which are saved in the local directory that is passed to the library. We take these saved frames as the input to the next module which detects faces for the given test case
3. Face Detection:
    a. The next step involved in the project after extracting frames from videos is face detection. We leverage the face_recognition library to load the frames and compare encodings. The list of frames created in the previous step are passed as input here and are iterated over to detect faces. Encodings are generated for each of the frames and compared with face encodings given to us as part of the project. Once there is a match in encoding, we return the detected face and ignore all the other frames. This is then being passed on to the next module which queries the database to get student related data.

**Testing:**

Each individual module above was tested using unit tests. The face detection module's output was detected against the given expected output file. The S3 client's functionality was tested by verifying that files were being uploaded correctly to the S3 bucket. I also performed end-end testing after integrating these modules with the rest of the code. We verified that lambda correctly invokes the S3 client and correct data is being sent to dynamo DB for querying the json file. The entire setup was also tested against the multi-threaded workload generator to handle concurrent requests.

**Hari Murugan Ravindran(1222324688)**

**Design:**

We have discussed how to implement the PaaS Application for detecting individual students' academic information by extracting a single frame from their video and recognizing their faces. It is where we have concluded that Data fetching from DynamoDB is one of the crucial steps for the successful execution of this project. I was responsible for the DynamoDB setup, Creating the Table, Uploading the necessary data and the final Student Academic Information fetching part in the Lambda Function. I have gone through all the docs, blogs and materials related to AWS Services like DynamoDB, Cloud9, S3, ECR and Lambda before working on this project. I can say that I was in charge of the Student Academic Information Data Storage on DynamoDB and the Data Retrieval of the Recognized Students from DynamoDB.

**Implementation:**

- The Base task is the setting up of the DynamoDB for storage and retrieval. I have used AWS Cloud9 which is the cloud-based integrated development environment (IDE) for handling the DynamoDB.
- By using AWS Cloud9 I first set up the DynamoDB table and loaded the data from the given JSON file to our created table.
- Then the Data is said to be verified with the JSON file whether the populated database is the same or not.
- Then once the name of the student is identified using the face recognition part the student's details are fetched from the DynamoDB.
- Then the Details are Stored in the CSV format as a file for each of the recognized students and saved in the Output Bucket.

**Testing:**

There were various phases of testing involved in integrating DynamoDB for the application.

1. Unit-testing: First all of the individual components of the System are checked.DynamoDB creation is checked, and Fetching of the Academic Record from DynamoDb as well as the Pushing of the Academic Record as CSV for the Output Bucket is tested for the smooth functioning of the application.
2. Integration-testing: The two major integrations are the Endpoint of the Lambda which searches the DynamoDB and the link to the Output bucket. They are checked and verified whether there are no anomalies in their behavior.
3. End-end testing: During this phase, the entire application is in perspective of the endpoint of the application that is tested. After the student's name is identified the Lambda function searches the DynamoDB, Gets the student's Academic Information and Generates it into the CSV and stores it in the Output Bucket.