

# CSE 546 — Project Report

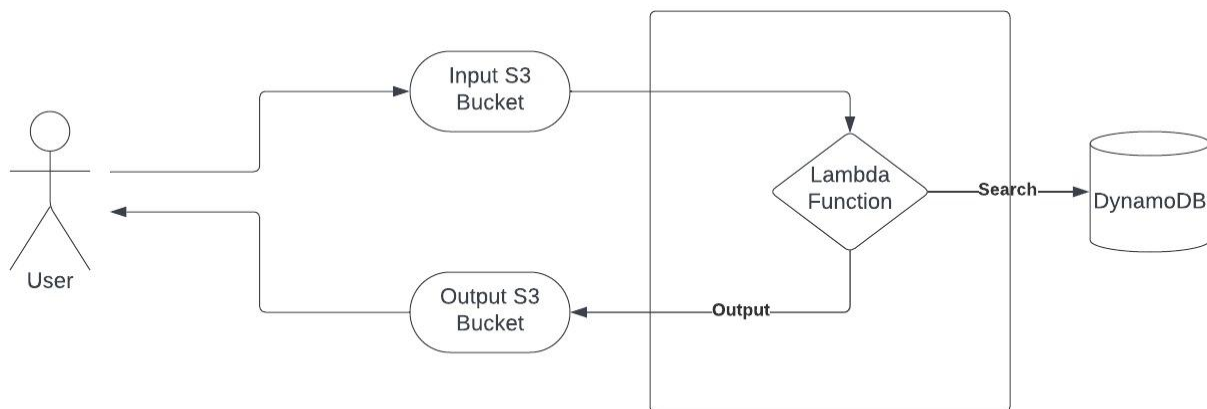
*Kaustubh Manoj Harapanahalli, Shravya Suresh, Sreekar Gadasu*

## 1. Problem statement

The project aims to create a cloud application using the AWS Services, for a smart classroom assistant used by educators. The project has to autoscale catering to the needs of the user. Users will upload videos directly to the S3 bucket (Storage service provided by AWS). With each upload, the project has to trigger a lambda function that will perform face recognition on this video with the help of a Python face recognition library. It will further look for information about the person in the video on DynamoDB and based on the mapped results, it will store the output containing academic details in the S3 bucket, in the form of CSV files.

## 2. Design and implementation

### 2.1 Architecture



Architecture

#### 1. S3 Buckets:

- **Input Bucket:** This is a simple S3 bucket that holds all the input videos of the classroom uploaded by the users. When a video is uploaded, an event is created in this bucket, which ensures that a lambda function is triggered.
- **Output Bucket:** This is a simple S3 bucket created to hold the results. Once the lambda function is executed, it stores the output in this bucket as a CSV file.

2. **Lambda Function:** This is the core component of the project. This component performs facial image recognition on the videos sent into the input bucket and sends the output to the output bucket. When a video is uploaded to the S3 bucket, it triggers the execution of the lambda function on that video. The handler is coded in Python, which makes use of the ffmpeg framework to extract the frames from the video and further uses the Python face\_recognition library to recognize the first face that was detected in the video. We have a predefined mapping that helps us with the name of the student. Based on this name, we then run a search in the

dynamoDB table to retrieve the student information. Once we have the student information, it stores the CSV results into the output S3 bucket.

3. **DynamoDB:** Once the face recognition code runs on the video, we get the person's name from a predefined list. Based on the name, the lambda function then retrieves students' information such as the student's name, major, and year from the dynamoDB tables. This information was already fed into the noSQL tables based on a JSON file provided as a part of the project.

## 2.2 Autoscaling

The main idea behind AWS Lambda auto-scaling is to dynamically change the function's concurrency parameters according to its demands and usage patterns. Lambda functions can effectively manage spikes and dips in invocation rates thanks to this functionality. Here's a summary of how it functions:

### Scaling Up:

- **Concurrency Increases:** When the volume of incoming requests rises, lambda functions automatically expand by running more concurrently.
- **Limits on Burst Concurrency:** AWS Lambda offers a brief burst of concurrency, enabling quick scaling to accommodate increases in request rates.
- **Provisioned concurrency** is a feature that helps applications that are latency-sensitive by allowing users to specify the amount of pre-initialized function instances that are available to react promptly to events.
- **Scaling Out:** Rather of scaling up, which involves expanding resources for a single instance, the function scales out, or increases the number of instances.

For our use-case, as the amount of uploaded videos continues to grow, the Lambda function is triggered and AWS Lambda proficiently adjusts the scaling by generating more instances of the Lambda function. As events flow at a faster rate (such as a surge in video uploads), Lambda swiftly expands to effectively manage the amplified workload. With the aid of Docker, the Lambda function spins up extra containers (instances) of the Docker image to efficiently handle the incoming videos simultaneously.

### Scaling Down:

- **Automatic Adjustment:** Lambda automatically scales down resources by reducing the number of concurrent executions as the request rate drops.
- **Idle Instances:** Depending on the settings, instances may be automatically scaled down if provisioned concurrency is consumed and they stay idle.

Once the videos have been processed, Lambda seamlessly adjusts to the decreasing workload. To avoid excessive resource usage and maximize efficiency, Lambda dynamically reduces the amount of instances or containers responsible for executing the function. This means that as the workload decreases, Lambda intelligently scales down to the base configuration.

### Scaling Configuration:

- **Concurrency Limits:** Users can set a maximum concurrency limit for each function, which defines the maximum number of instances that can run simultaneously.

- **Reserved Concurrency:** This feature allows users to allocate a portion of their account's total concurrency limit to a specific function. This ensures that the function has the necessary resources available when needed.

### Integration with AWS Services:

- **Triggers and Scaling:** AWS services such as API Gateway, S3, and DynamoDB Streams can all act as triggers for Lambdas. The trigger's setup and nature may have an impact on the scaling behavior.
- **CloudWatch Metrics:** Users can monitor function metrics and set up alarms by integrating AWS Lambda with CloudWatch. Scaling actions can be initiated by these alarms.

### Performance Considerations:

- **Cold Starts:** A cold start, which adds extra delay, can happen when a function is called after it has been idle. Provisioned concurrency aids in reducing this problem.
- **Throttling:** More invocations of the function are throttled if it surpasses its concurrent limit. It is necessary to take this behavior into account when building high-throughput systems.

## 2.3 Member Tasks

**1. Kaustubh Manoj Harapanahalli:** I looked at the implementation of multiple parts of the ML aspects of the project, starting with

- **Converting video to images:** This was carried out as mentioned in the project description using ffmpeg and the output images were stored in a temporary folder.
- **Face Recognition and Post Processing:** For the face recognition, I used the face recognition library and the provided encoding file to generate a CSV file. This CSV file was pushed to the output S3 bucket that was available. To check for the mapping of face to person parameters, I used the implementation related to DynamoDB created by Sreekar to create the mentioned CSV file.
- **Dockerization:** We modified the provided Dockerfile to include the encoding file in the docker image, and the docker image was built to work on a linux machine using the parameter: `--platform=linux/amd64`.
- **ECR storage:** The build docker image was pushed to ECR and the lambda function that was created pulled this docker image and ran the process.

**2. Shravya Suresh:** I took on a number of significant responsibilities for the project in order to develop a smart classroom assistant on the AWS cloud. Using the built docker image I created a new AWS Lambda function. I then set up the necessary access permissions and defined the runtime environment for my Lambda function. In order to start the Lambda function whenever new video files are uploaded to a particular S3 bucket, I also built up a trigger mechanism. As soon as a new video became available, this automation made sure that processes like face recognition and video processing continued to run smoothly. In addition, I created and oversaw two S3 buckets: an input bucket and an output bucket. Videos that required processing were placed in the input bucket, while the final CSV files containing

academic data were kept in the output bucket. I also contributed to the testing of the application to ensure that it works as expected.

**3. Sreekar Gadasu:** I have taken a careful and meticulous approach to oversee the DynamoDB aspect of our project, guiding it through various crucial stages. Furthermore, I created a dedicated function specifically tailored for effortlessly uploading student data into the DynamoDB table. This function was carefully adjusted to handle data formatting and insertion, resulting in a systematic and structured method for inputting data. Afterwards, I smoothly incorporated these DynamoDB functionalities into the Lambda code. This involved configuring the Lambda function to respond to S3 uploads, allowing it to efficiently process video content and extract student names. With a seamlessly integrated combination of these components, our Lambda function is able to confidently and effectively carry out a multitude of tasks, such as extracting student details from uploaded videos and retrieving pertinent information from our DynamoDB table. This sophisticated system provides a streamlined and highly efficient means of managing student data based on the content uploaded to our S3 bucket.

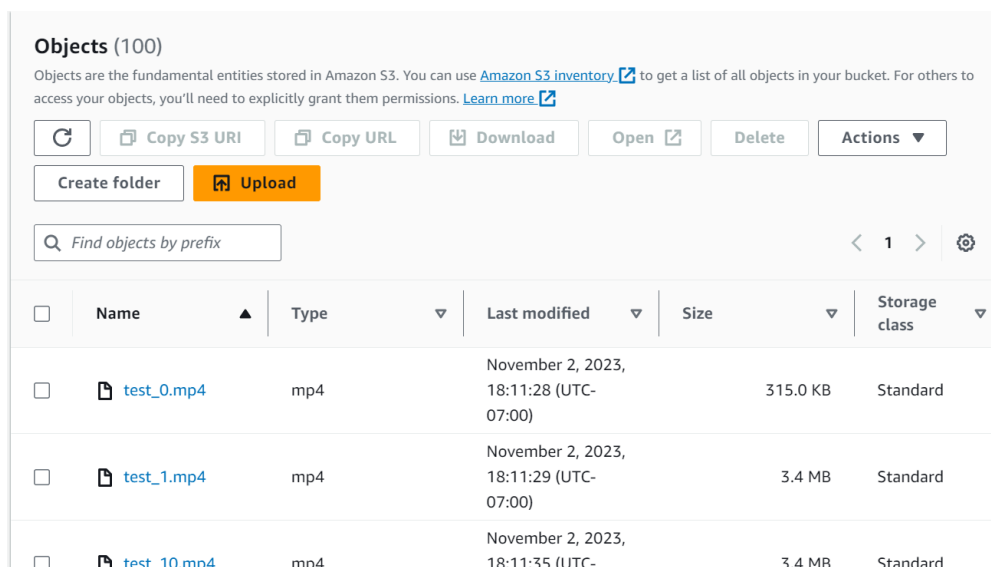
### 3. Testing and evaluation

Elaborate Testing was performed throughout the project.

#### Unit Testing performed:

The following units were tested.

- Uploading videos to the S3 input bucket.






**Objects (100)**

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Refresh Copy S3 URI Copy URL Download Open Delete Actions

Create folder Upload

Find objects by prefix

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	 <a href="#">test_0.mp4</a>	mp4	November 2, 2023, 18:11:28 (UTC-07:00)	315.0 KB	Standard
<input type="checkbox"/>	 <a href="#">test_1.mp4</a>	mp4	November 2, 2023, 18:11:29 (UTC-07:00)	3.4 MB	Standard
<input type="checkbox"/>	 <a href="#">test_10.mp4</a>	mp4	November 2, 2023, 18:11:35 (UTC-	3.4 MB	Standard

- Triggering the Lambda function when a new video is available.

Log streams
Tags
Metric filters
Subscription filters
Contributor Insights
Data protection

Log streams (3)
Delete
Create log stream
Search all log streams

Filter log streams or try prefix search
Exact match
Show expired
Info
1

<input type="checkbox"/>	Log stream	Last event time
<input type="checkbox"/>	2023/11/03/[\$LATEST]56e154392cea4bdea1c0b25ace622...	2023-11-02 18:15:28 (UTC-07:00)
<input type="checkbox"/>	2023/11/03/[\$LATEST]715568a274db45809d314a5dc487...	2023-11-02 18:15:28 (UTC-07:00)
<input type="checkbox"/>	2023/11/03/[\$LATEST]ec341b9cf8964c4ea814c05a849faa08	2023-11-02 18:11:19 (UTC-07:00)

Function overview
Info

testName

S3

+ Add trigger

+ Add destination

- Extracting frames from the video using ffmpeg. [PASS]
- Performing face recognition on the extracted frames. [PASS]
- Searching DynamoDB for academic information. [PASS]
- Storing academic information in the S3 output bucket.

Objects (100)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Refresh
Copy S3 URI
Copy URL
Download
Open
Delete
Actions

Create folder
Upload

Find objects by prefix
1

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
--------------------------	------	------	---------------	------	---------------

	A	B	C	D	E	F	G	H
1	test_30	floki	history	junior				
2								
3								
4								

### Integration Testing Performed:

Verifying the proper communication and interoperability of the AWS Lambda functions with external services like S3 and DynamoDB is the main goal of integration testing in this project. It ensures that parts function well together by examining data flow, communication protocols, and data consistency. Real-world interactions with external dependencies are a part of integration tests, which assess how well Lambda functions manage different circumstances while interacting with these services. This stage of testing is essential for identifying potential problems that could occur when various parts interact and guarantees that the system operates as intended when used in an actual setting.

## 4. Code

### Code Snippets from the code

- **Face Recognition Code:**

```
def face_recognition_handler(event, context):
    # Get the bucket name and object key from the event
    bucket_name = event["Records"][0]["s3"]["bucket"]["name"]
    object_key = event["Records"][0]["s3"]["object"]["key"]

    # Download the object from S3 to the Lambda execution environment
    local_filename = "/tmp/" + object_key
    s3.download_file(bucket_name, object_key, local_filename)
    video_file_path = local_filename
    output_path = "output/"

    # Load the known face encodings and names from files

    encodings_dict = open_encoding("encoding")

    # Convert the dictionary to separate lists of known_faces and known_names

    known_faces = list(encodings_dict["encoding"])
    known_names = list(encodings_dict["name"])

    # print(known_faces)
    print(type(known_names))
```

- Loading Data to the output bucket

```
# If there is a match, use the known face's name
if True in matches:
    first_match_index = matches.index(True)
    name = known_names[first_match_index]
    result = search_database_table(name)
    output_file_name = object_key.split(".")[0]
    output = [
        output_file_name,
        result["name"],
        result["major"],
        result["year"],
    ]
    print(output)

    output_path = output_file_name + ".csv"
    with open("/tmp/" + output_path, mode="w") as file:
        writer = csv.writer(file)
        writer.writerow(output)
    # Upload CSV file to S3 bucket
    s3.upload_file(
        "/tmp/" + output_path, "paas-output-bucket", output_path
    )
    os.remove("/tmp/" + output_path)
    break
```

#### Docker File:

```
63 # Copy function code
64 COPY handler.py ${FUNCTION_DIR}
65 RUN chmod 755 /entry.sh
66 COPY encoding ${FUNCTION_DIR}
```

To ensure that the application runs smoothly, we copied the encoding file into the docker image which was the only change made in comparison to the provided dockerfile.

#### Handler.py:

This is a Python script that is intended to be used as an AWS Lambda function for face recognition in a smart classroom assistant project. It interacts with Amazon S3, DynamoDB, and the face\_recognition library. Here's an explanation of the functionalities of the code:

- Imports: Numerous Python modules, such as pickle, csv, os, boto3, face\_recognition, and boto3\_client, are imported by the script. These modules handle facial recognition, file operations, AWS services, and CSV files.
- search\_database\_table Function: Using a supplied attribute value (probably a student's name), this function looks for academic data in a DynamoDB table. Using the boto3 library, it establishes a connection to DynamoDB and launches a scan with a filter condition.
- open\_encoding Function: This function uses the Python pickle module to read a binary file that probably contains known face encodings. Data that has been encoded is loaded and returned.
- face\_recognition\_handler Function: This is the primary function that responds to the AWS Lambda event that arises from the upload of a new movie to the designated S3 input bucket. The data, including the object key and source S3 bucket name, is extracted from the event. It enables the Lambda execution environment to download and process the video file from the S3 input bucket. It uses ffmpeg to extract the video's frames, which are then saved in the /tmp/ directory. It loads names and recognized face encodings from a file that contains encoding information of the students. Iteratively goes over the retrieved frames, applies facial recognition to each one, then compares the results with faces that are already known. If a match is discovered, the academic data is extracted from DynamoDB using the student's name, compiled into a CSV file, and uploaded to the designated S3 output bucket.



# Portfolio Reports

## Individual Contributions:

**Shravya Suresh (1225488290)**

### Individual Role

The following were my individual contributions to the project.

- **Docker File:** To create a unique container image for the Lambda function, I customized the Dockerfile. Changed the file to set up the necessary dependencies and environment to analyze videos and recognize faces in the AWS cloud effectively.
- **Lambda Handler File:** I contributed to creating a custom AWS Lambda function using Python, ensuring it had essential tools and libraries like ffmpeg and face\_recognition for video processing and face recognition.
- **Lambda Function Configurations:** In order to interface with other AWS services, I set up access rights and specified the runtime environment for the Lambda function.
- **Trigger Mechanism:** I developed a trigger mechanism to start the Lambda function automatically when uploading fresh video files to a specified S3 bucket.
- **S3 Buckets:** Created two S3 buckets, one for input and one for output, were created and handled. Videos that needed to be processed were stored in the input bucket. The generated CSV files including academic data were kept in the output bucket. When fresh content was uploaded, face recognition and video processing activities could be automatically automated thanks to the trigger and Lambda function interaction with S3 buckets.
- **Testing:** I wrote unit tests to verify the accuracy and dependability of the application's separate components, like face recognition and video processing, and actively participated in the testing phase of the project.

### Skills/Knowledge Acquired through the project.

- **AWS Cloud Services:** I developed a thorough understanding of AWS services like Lambda, S3, and DynamoDB, as well as how to configure and integrate them for the creation of serverless applications.
- **Containerization with Docker:** I gained knowledge on how to build and manage Docker containers to bundle dependencies and apps for faster deployment.
- **Video Processing and Face Recognition:** I used libraries like face\_recognition to apply face recognition algorithms and ffmpeg to gain expertise in video processing.
- **Serverless Architecture:** The project improved my knowledge of serverless architecture, which made it possible to develop scalable and reasonably priced cloud solutions.
- **Testing and Quality Assurance:** I helped with the testing stage, which helped me become more proficient at writing unit tests for distinct parts and guaranteeing better code quality.
- **Automation and Troubleshooting:** I improved my ability to automate procedures for the timely completion of tasks and sharpened my problem-solving abilities, especially with regard to debugging and fixing problems with cloud-based apps.

## **Individual Contributions:**

### **Sreekar Gadasu (1227891563)**

I was a key player in the project, with a specific focus on integrating DynamoDB. My responsibilities and contributions were organized into three main categories:

#### **Implementation:**

##### **DynamoDB Implementation:**

- To set up the database, extensive efforts went into designing DynamoDB tables that could effectively handle video metadata and processing results. Significant considerations were made to optimize the storage setup.
- To facilitate data upload, I created a Lambda function that was specially designed to fetch relevant information from DynamoDB using video names as references. This streamlined approach allowed for efficient and effective data management.
- The success of this process hinged on seamlessly linking the Lambda function with DynamoDB tables. By ensuring a smooth and cohesive connection, data storage and retrieval became a seamless and efficient process.

##### **DynamoDB Integration:**

- As we worked to incorporate new technologies, the fusion of Docker and Lambda required us to establish a unified and seamless setting for conducting video processing operations. Our goal was to smoothly integrate the Docker container, which housed the video recognition software, into the Lambda infrastructure to optimize video processing capabilities.
- The connection between DynamoDB and Lambda was established to effectively store and retrieve video metadata and processing results. This seamless integration allowed for a seamless flow of data between our video processing modules and the database, guaranteeing proper storage and easy accessibility of data in DynamoDB. This has greatly improved our data management and streamlined our subsequent data analysis.

#### **Testing:**

During the functional testing process, DynamoDB's performance within the Lambda-Docker setup was thoroughly evaluated. A key focus was placed on the efficiency of three critical tasks: data upload, storage, and retrieval. This involved carefully assessing the system's ability to successfully upload processed video data to DynamoDB, verifying the accurate storage and organization of data, and double-checking the smooth retrieval of video metadata and processing outcomes as needed. Through this testing, our goal was to guarantee the efficient and dependable operation of DynamoDB within the broader system.

To sum up, my duties involved supervising the architectural design process and guaranteeing the smooth integration of Lambda, Docker, and DynamoDB to optimize video processing. Additionally, I conducted thorough tests to meticulously validate the system's functionality.