

## CSE 546 — Project 2 Report

### Group members:

S.NO	Name	Mail ID	Student ID
1	Avinash Kodali	akodali5@asu.edu	1224993120
2	Manogna Pagadala	mpagada1@asu.edu	1224798575
3	Manaswi Kommini	mkommini@asu.edu	1226342288

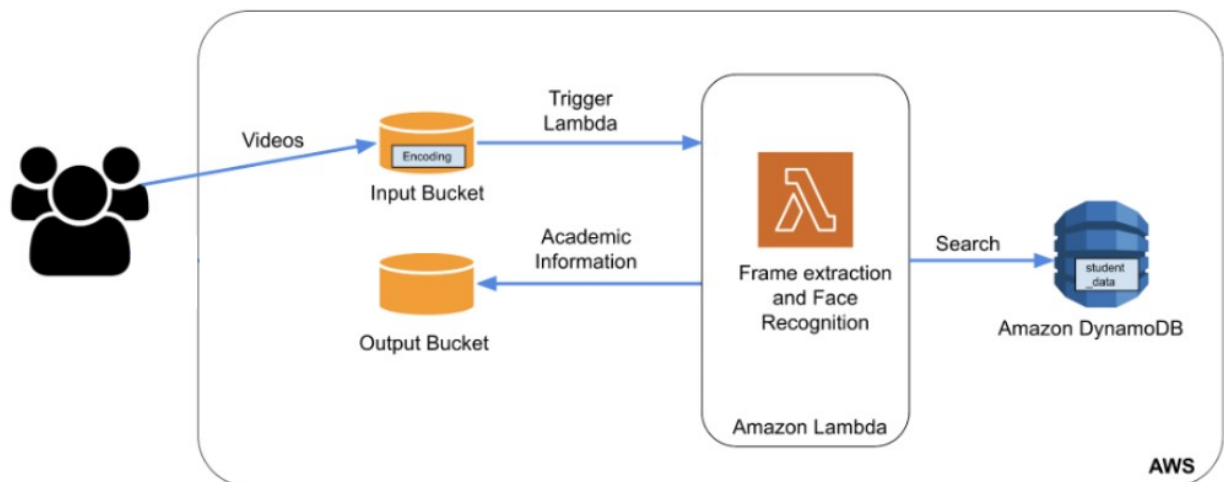
### 1. Problem statement

Using cloud resources to create a facial recognition service that can extract faces from user-uploaded films is the aim. This service is important since it allows users to search for certain faces in the videos and get relevant results. For instance, viewers may easily find courses with a given lecturer or watch films on a particular person. In this specific project, our main goal is to identify student faces from input videos that users have uploaded to an S3 bucket. We then use an AWS lambda function to retrieve the identified student's data from a DynamoDB table, which we then save as a CSV file in another S3 bucket.

### 2. Design and implementation

#### 2.1 Architecture

The below architecture is followed in the project.



1. The S3 bucket, designated as the Input Bucket within our project infrastructure, serves as the repository for video content. Following the execution of the workload generator, these videos are deposited into this bucket for further processing.
2. On the other end, the Output Bucket, also hosted on Amazon S3, is established to receive the results generated by our face recognition algorithm. Once the AWS Lambda function has performed its tasks, the outcomes are saved within this bucket for subsequent access.
3. AWS Lambda plays a pivotal role in our system. It hosts a custom image processing function specialized in face recognition. This Lambda function is designed to be triggered whenever new objects are added to the S3 Input Bucket. Following recognition, it automatically deposits the results into the Output Bucket as part of its programmed functionality.
4. DynamoDB serves as our NoSQL database, housing a collection of student data in the form of JSON records. These JSON data entries are organized within a DynamoDB table created for this purpose. Our Lambda function utilizes this database to retrieve pertinent information, subsequently converting the acquired data into a CSV format before dispatching it to the Output Bucket.

## **2.2 Autoscaling**

Given that our project relies on AWS Lambda, the built-in auto scaling feature is seamlessly managed. Lambda, as a default provision, allocates a total concurrency limit of 1,000 across all functions within a specific region. Consequently, AWS naturally adjusts the function's capacity to accommodate rising workloads, eliminating the necessity for any supplementary configuration.

## **2.3 Member Tasks**

Manogna Pagadala:

- In charge of putting together the Docker container image.
- Created an Elastic Container Registry (ECR) repository on AWS and uploaded the Docker image to it with success.
- Setup the S3 trigger for AWS Lambda and deployed the Docker image to make sure it reacts to new object creations.

Avinash Kodali:

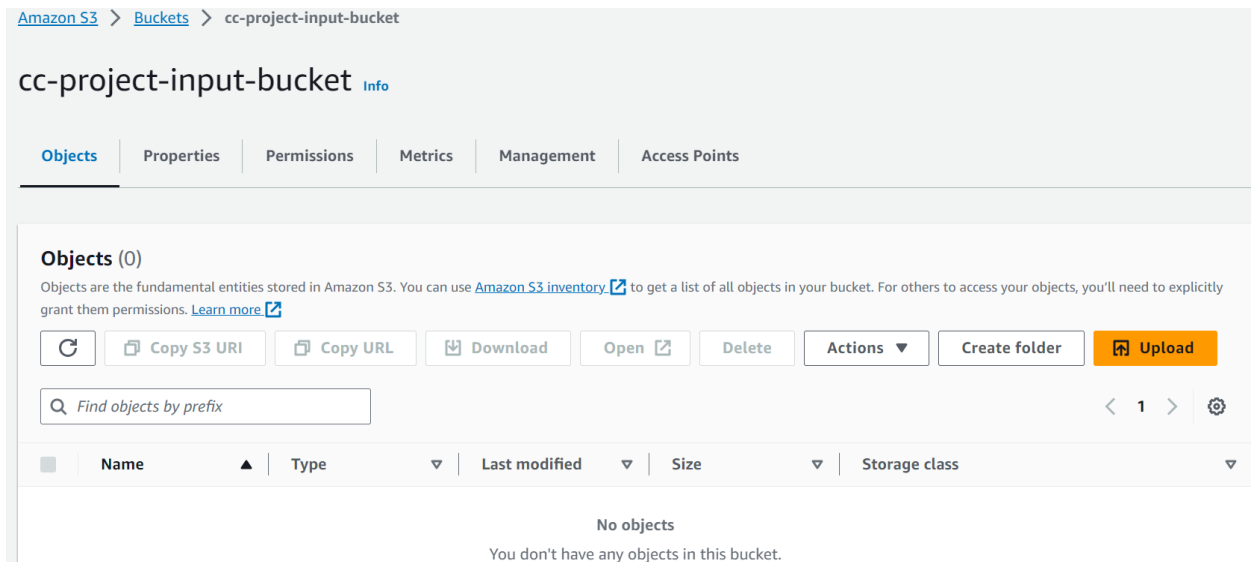
- Designed and oversaw the input and output buckets on S3.
- Written the code to extract frames from the supplied MP4 video and download video content to the local environment from the input S3 bucket.
- Provided the name of the first image with an identified face and arranged the retrieved frames in ascending order.

Manaswi Kommini:

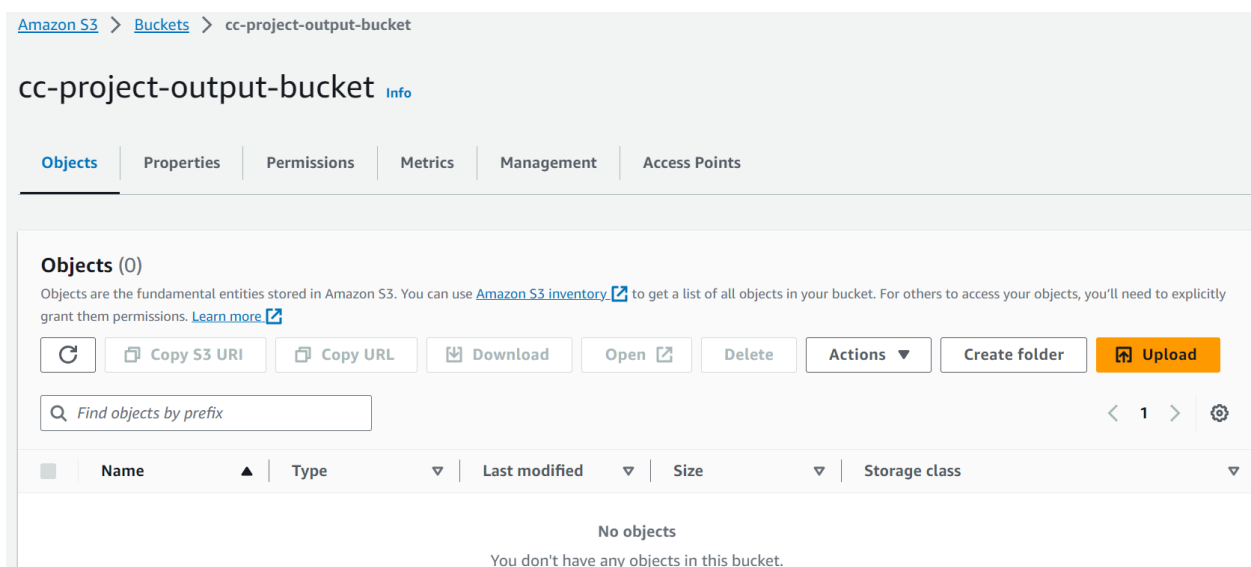
- Skillfully created a DynamoDB table and imported the student information into it.
- Maintained the focus on creating a CSV file by associating the identified faces with the matching picture names from the DynamoDB.
- Uploaded this CSV file to the output S3 bucket to finish the operation.

### 3. Testing and evaluation

Empty input S3 bucket:



Empty output S3 bucket:



Test case 1:

Terminal output:

```
C:\Users\Manogna\AppData\Local\Programs\Python\Python310\python.exe C:\Users\Manogna\Downloads\cse546-project-lambda-master\workload.py
Nothing to clear in input bucket
Nothing to clear in output bucket
Running Test Case 1
Uploading to input bucket.. name: test_0.mp4
Uploading to input bucket.. name: test_1.mp4
Uploading to input bucket.. name: test_2.mp4
Uploading to input bucket.. name: test_4.mp4
Uploading to input bucket.. name: test_5.mp4
Uploading to input bucket.. name: test_6.mp4
Uploading to input bucket.. name: test_7.mp4
Uploading to input bucket.. name: test_8.mp4

Process finished with exit code 0
```

Input S3 bucket:





Amazon S3 > Buckets > cc-project-input-bucket

### cc-project-input-bucket [Info](#)

[Objects](#) | [Properties](#) | [Permissions](#) | [Metrics](#) | [Management](#) | [Access Points](#)

**Objects (8)**  
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](#)

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	 <a href="#">test_0.mp4</a>	mp4	November 3, 2023, 17:22:46 (UTC-07:00)	315.0 KB	Standard
<input type="checkbox"/>	 <a href="#">test_1.mp4</a>	mp4	November 3, 2023, 17:22:48 (UTC-07:00)	3.4 MB	Standard
<input type="checkbox"/>	 <a href="#">test_2.mp4</a>	mp4	November 3, 2023, 17:22:55 (UTC-07:00)	408.5 KB	Standard
<input type="checkbox"/>	 <a href="#">test_4.mp4</a>	mp4	November 3, 2023, 17:22:57 (UTC-07:00)	345.7 KB	Standard

Output S3 bucket:

Amazon S3 > Buckets > cc-project-output-bucket

## cc-project-output-bucket [Info](#)

[Objects](#) | [Properties](#) | [Permissions](#) | [Metrics](#) | [Management](#) | [Access Points](#)

**Objects (8)**  
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](#)

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	<a href="#">test_0.csv</a>	csv	November 3, 2023, 17:22:54 (UTC-07:00)	49.0 B	Standard
<input type="checkbox"/>	<a href="#">test_1.csv</a>	csv	November 3, 2023, 17:23:08 (UTC-07:00)	52.0 B	Standard
<input type="checkbox"/>	<a href="#">test_2.csv</a>	csv	November 3, 2023, 17:23:04 (UTC-07:00)	42.0 B	Standard
<input type="checkbox"/>	<a href="#">test_4.csv</a>	csv	November 3, 2023, 17:23:02 (UTC-07:00)	53.0 B	Standard

Test\_0.csv file:

	A	B	C
1	name	major	year
2	president_trump	physics	junior
3			
4			

Test case 2:

Terminal output:

```
C:\Users\Manogna\AppData\Local\Programs\Python\Python310\python.exe C:\Users\Manogna\Downloads\cse546-project-lambda-master\workload.py
Nothing to clear in input bucket
Nothing to clear in output bucket
Running Test Case 2
Uploading to input bucket.. name: test_0.mp4
Uploading to input bucket.. name: test_1.mp4
Uploading to input bucket.. name: test_10.mp4
Uploading to input bucket.. name: test_100.mp4
Uploading to input bucket.. name: test_11.mp4
Uploading to input bucket.. name: test_12.mp4
Uploading to input bucket.. name: test_13.mp4
Uploading to input bucket.. name: test_14.mp4
Uploading to input bucket.. name: test_15.mp4
Uploading to input bucket.. name: test_16.mp4
Uploading to input bucket.. name: test_17.mp4
Uploading to input bucket.. name: test_18.mp4
```

```
Run workload ×

Uploading to input bucket.. name: test_83.mp4
Uploading to input bucket.. name: test_84.mp4
Uploading to input bucket.. name: test_85.mp4
Uploading to input bucket.. name: test_86.mp4
Uploading to input bucket.. name: test_87.mp4
Uploading to input bucket.. name: test_88.mp4
Uploading to input bucket.. name: test_89.mp4
Uploading to input bucket.. name: test_9.mp4
Uploading to input bucket.. name: test_90.mp4
Uploading to input bucket.. name: test_91.mp4
Uploading to input bucket.. name: test_92.mp4
Uploading to input bucket.. name: test_93.mp4
Uploading to input bucket.. name: test_94.mp4
Uploading to input bucket.. name: test_95.mp4
Uploading to input bucket.. name: test_96.mp4
Uploading to input bucket.. name: test_97.mp4
Uploading to input bucket.. name: test_98.mp4
Uploading to input bucket.. name: test_99.mp4

Process finished with exit code 0
```

Input S3 bucket:

Amazon S3 > Buckets > cc-project-input-bucket

### cc-project-input-bucket [Info](#)

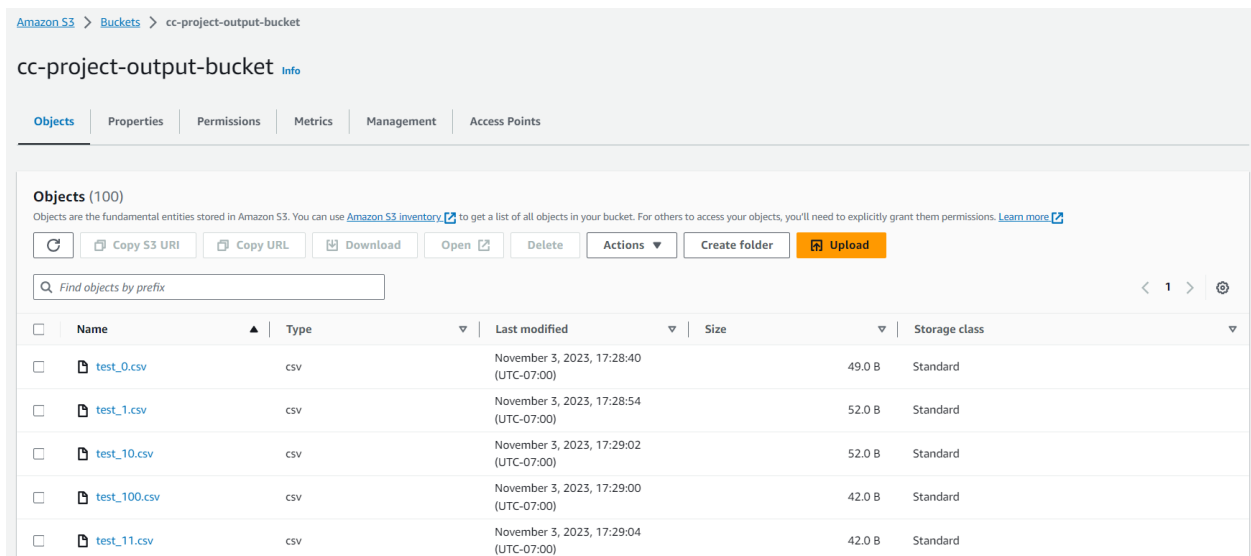
[Objects](#) | [Properties](#) | [Permissions](#) | [Metrics](#) | [Management](#) | [Access Points](#)

**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](#)

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	<a href="#">test_0.mp4</a>	mp4	November 3, 2023, 17:28:31 (UTC-07:00)	315.0 KB	Standard
<input type="checkbox"/>	<a href="#">test_1.mp4</a>	mp4	November 3, 2023, 17:28:33 (UTC-07:00)	3.4 MB	Standard
<input type="checkbox"/>	<a href="#">test_10.mp4</a>	mp4	November 3, 2023, 17:28:40 (UTC-07:00)	3.4 MB	Standard
<input type="checkbox"/>	<a href="#">test_100.mp4</a>	mp4	November 3, 2023, 17:28:48 (UTC-07:00)	408.5 KB	Standard

Output S3 bucket:



Test\_1.csv file:

name	major	year
president_biden	history	sophomore

## 4. Code

### Dockerfile

A Dockerfile is used to generate a container image that can be deployed and used in a wide range of environments. This image contains all the dependencies needed to run the Python based AWS lambda function. It is structured into multiple stages to efficiently manage the facial recognition task. In the initial stage we set up the environment by defining arguments for function directories, runtime versions, and operating system distributions. Alpine Linux is used as the base image and installs necessary dependencies such as Python, AWS Lambda Runtime Interface Client, and additional tools like boto3, face\_recognition, ffmpeg. The file also copies the function code and related files into the container image and sets up an entry script to initialize the Lambda function. We have modified the given docker file to copy the encoding file to Lambda function.

### handler.py

face\_recognition\_handler:

1. This is the main function that gets triggered by an S3 event, specifically the addition of an object in an S3 bucket.
2. Retrieves the name of the bucket and the uploaded video file's key and then downloads the video from S3 to a temporary location.
3. It uses ffmpeg to extract frames from the video and then iterates through the extracted frames performing facial recognition using face\_recognition library by comparing with a set of known face encodings.
4. Writes the recognized face's information to a CSV file and uploads it to the output S3 bucket.

Query\_write\_to\_s3:

1. Accesses the DynamoDB table named 'student1' and retrieves information based on a provided 'name' key and writes fetched data to a CSV file (name, major, year).
2. Uploads the created CSV file to the output S3 bucket.

### **Upload\_data.py**

This python code uploads the data in student\_data.json file to 'student1' dynamoDB table. The json file has an array of objects, where each object represents a student record with 'name', 'id', 'major', 'year' attributes. It accesses each record in the json file and uploads the data to dynamoDB table using put\_item request.

### **Installation and Running:**

1. Created input and output S3 buckets.
2. Created the dynamo\_db table and loaded the data from the "student.json" to the table.
3. Install the Docker Desktop and create a docker image by using the command "docker build."
4. After the image is created, get the latest image and tag the image using the "docker tag" command. By using the command "docker push " push the image to the ECR repository.
5. Lambda function is created from the deployed image from the ECR Repository and an S3 input bucket event trigger is created.
6. In the workload generator set the input and output bucket names accordingly and run the workload generator for both the test cases.