CSE 546 — Project Report2

S.No.	Name	Mail ID	*Student ID*
1	Venkata Divya Sai Gorijala	vgorijal@asu.edu	1225658473
2	Chandrika Kondapi	ckondapi@asu.edu	1223212406
3	Priyadarshini Ramakrishnan	pramak10@asu.edu	1225407339

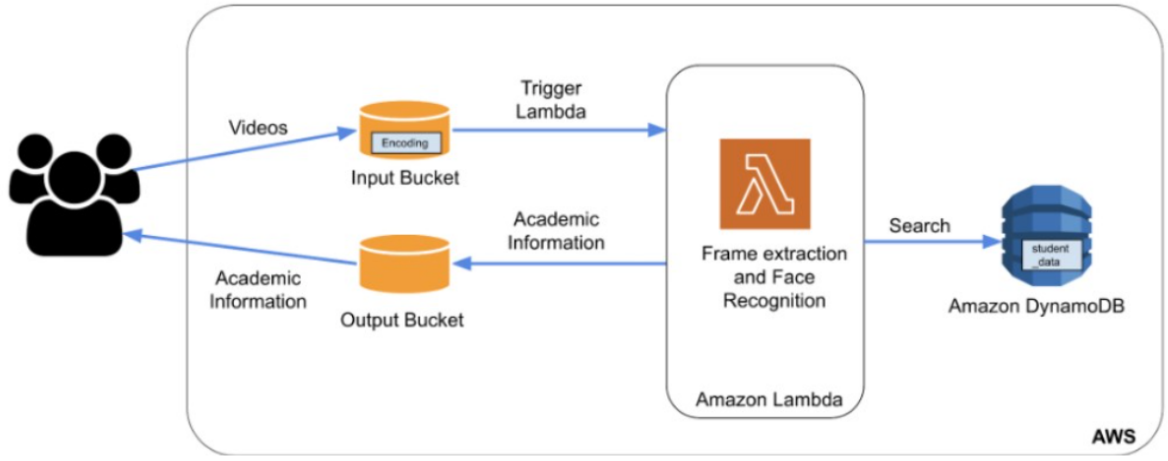
1. Problem statement

The goal is to leverage cloud resources for a facial recognition service that can extract faces from videos uploaded by users. This service is significant because it enables users to conduct searches based on the faces captured in the videos, yielding pertinent results. For example, users can quickly locate lectures featuring a particular instructor or watch documentaries about a specific individual. In this project particularly, we are focusing on detecting faces of students from the input video uploaded by users in S3 bucket and using an AWS lambda function we are retrieving details of the detected student from a DynamoDB table and storing the result as a CSV file in another S3 bucket.

2. Design and implementation

2.1 Architecture

The below architecture is followed in the project



1. Input Bucket: S3 bucket created in project receives the input videos after running workload generator and stores them inside the bucket.

2. Output Bucket: S3 bucket created receives the input from the lambda function after the face recognition algorithm has been performed and stores the result in the bucket.

3. AWS Lambda: In AWS Lambda, a custom image-based function is created that performs face recognition and is triggered by any new object created in S3 input bucket. It is also programmed to automatically send recognition results to the output bucket upon receiving input.

4. DynamoDB: A NoSql database containing student data. The given JSON data was uploaded to a table created in DynamoDB. The lambda function queries this table to obtain relevant results, which are then formatted into CSV and sent to the output bucket.

2.2 Autoscaling

Since we are utilizing AWS Lambda for the purpose of this project autoscaling is automatically taken care of. By default, Lambda provides your account with a total concurrency limit of 1,000 across all functions in a region. Hence, AWS automatically scales the function to handle increasing loads without the need for additional implementation.

2.3 Member Tasks

Venkata Divya Sai Gorijala:

- Created the docker container image.
- Created AWS ECR repository, pushed the image to ECR
- Deployed the image to lambda and set up the S3 trigger for lambda.

Chandrika Kondapi:

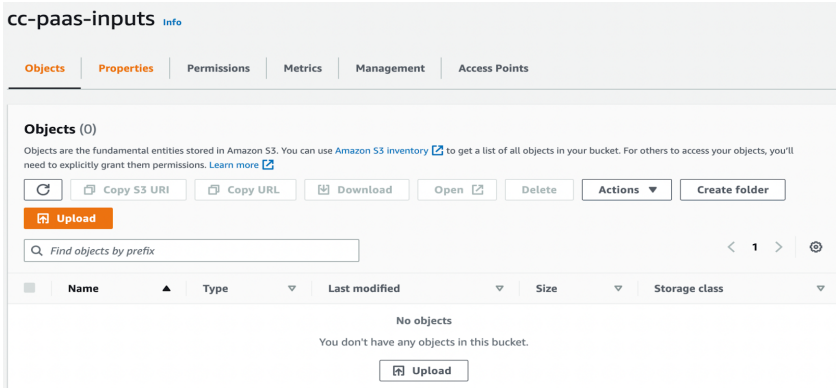
- Created S3 input and output buckets.
- Written the code for downloading the video to local from input S3 bucket and extracting frames from the given MP4 video.
- Sorted the frames in ascending order and returned the name of the first image detected.

Priyadarshini Ramakrishnan:

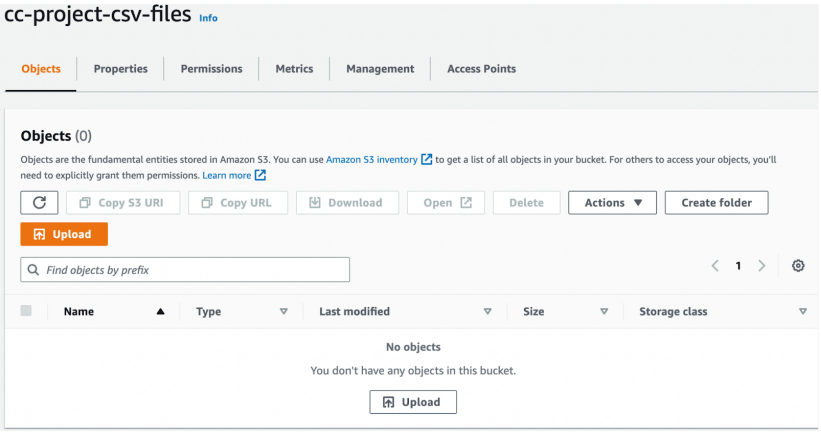
- Created DynamoDB table, loaded the student data to the table.
- For the detected faces, the image names from the dynamo db and write the content into a csv file and upload it to the output S3 bucket.

3. Testing and evaluation

Empty input S3 bucket:

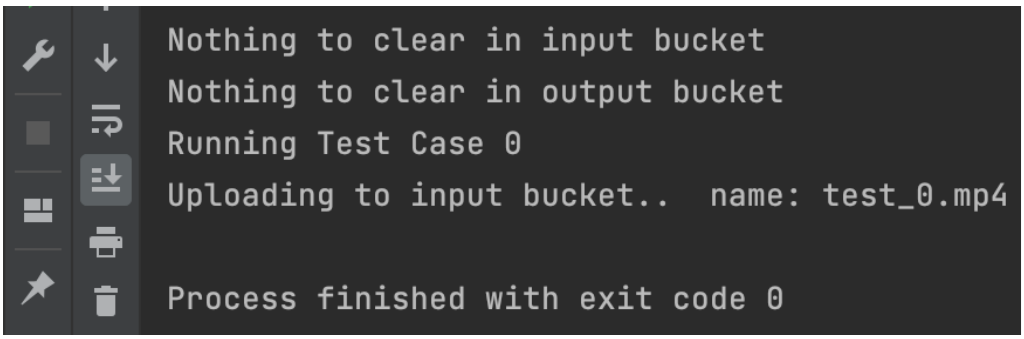


Empty Output S3 bucket:

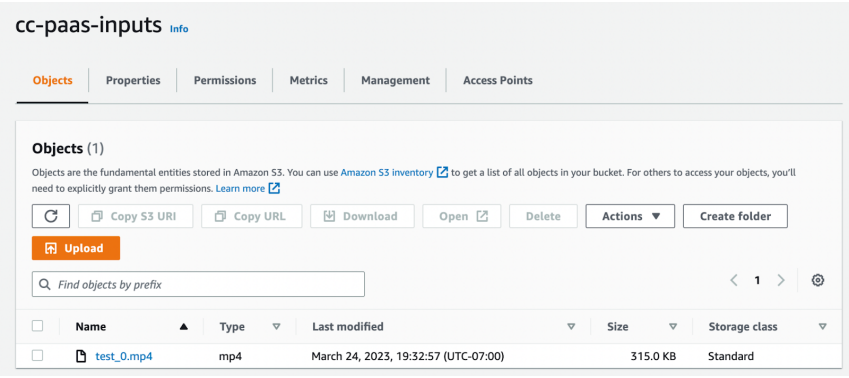


Running test case 1:

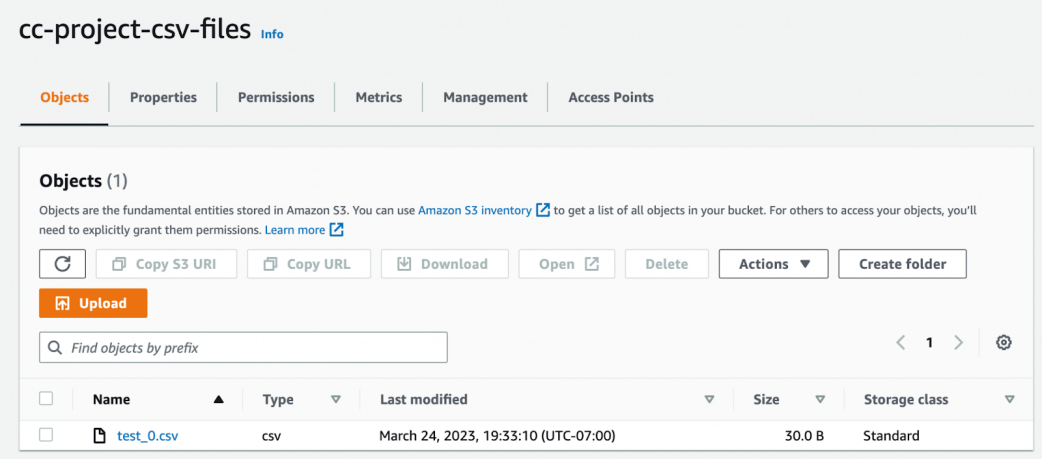
Console output:



Input S3 bucket:

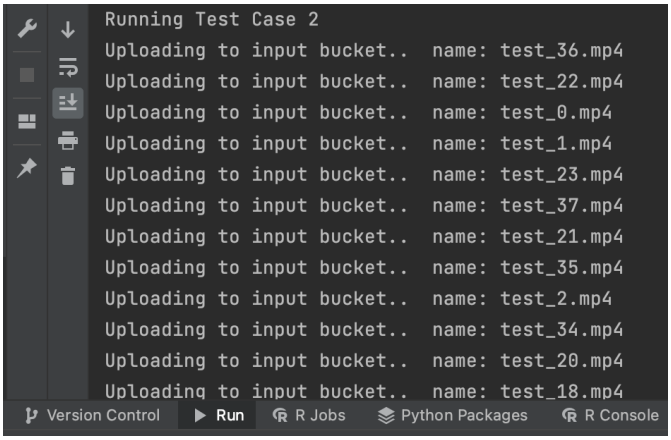


Output S3 bucket

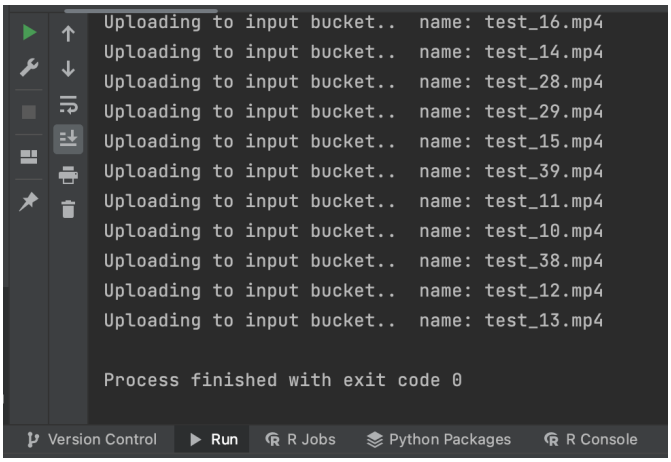


Running test case 2:

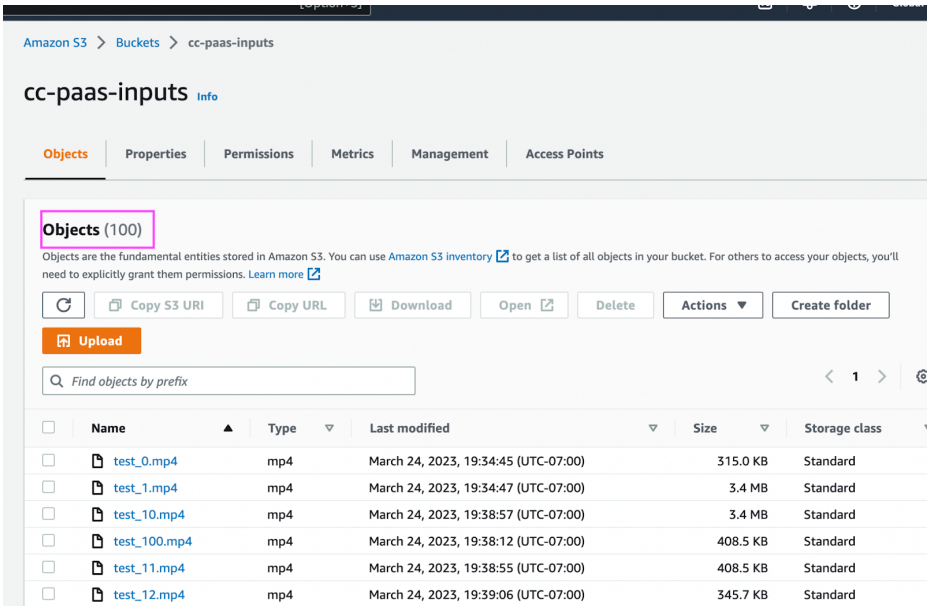
Terminal output:



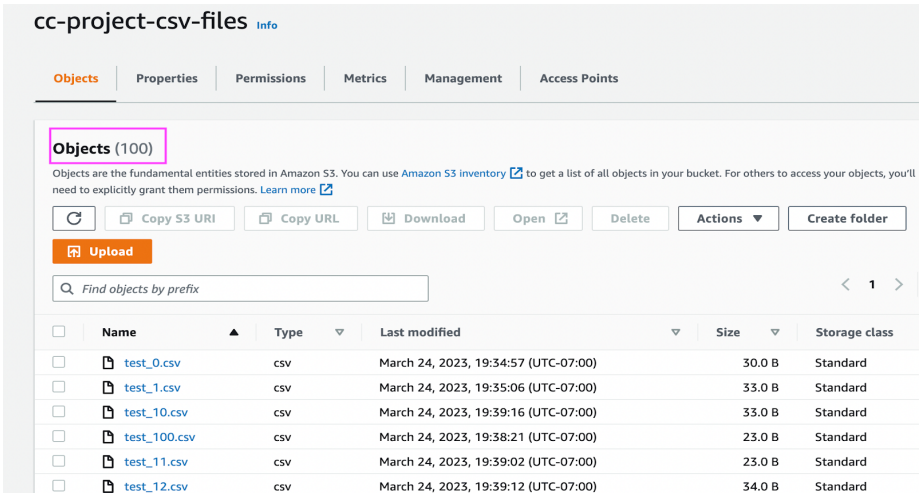
.....



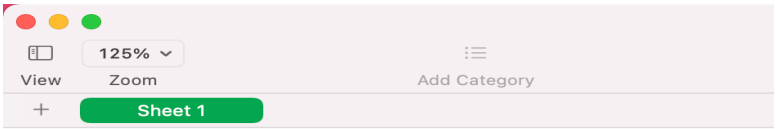
Input S3 bucket:



Output S3 bucket:



CSV file output:



test_18

president_biden	history	sophomore

4. Code

The Dockerfile creates a container image with all the dependencies required to run a Python-based AWS Lambda function, making it easier to deploy and run the function in various environments. A fresh copy of the Python runtime image is obtained, and some additional dependencies are installed, such as cmake, ca-certificates, and libgl1-mesa-glx.AWS Lambda Runtime Interface Client for Python is installed, and function's dependencies are installed using pip. The built dependencies and function code are then copied into the final runtime image. It installs the ffmpeg library, sets the working directory to the function directory, and copies the built dependencies and function code into the image. Finally, an entry point script is added to run the function, and the CMD is set to the function handler.

handler.py

face_recognition_handler() - This function downloads the video file from the input S3 bucket to local, extracts frames from the video using ffmpeg, and then uses the face_recognition library to detect faces in the first frame of the video. It then compares the detected face with the faces in an encoding file to identify the person in the video. Finally, it retrieves the person's information from a DynamoDB table, generates a CSV file with the person's name, major, and year, and stores the CSV file in the output S3 bucket.

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. Docker Desktop is installed.
4. Image is created by using the command “docker build.”
5. After the image is created, by using command “docker images” get the latest image and tag the image using “docker tag 353e3d63ff4c 067895197847.dkr.ecr.us-east-1.amazonaws.com/cc_project_2“command. By using the command “docker push 067895197847.dkr.ecr.us-east-1.amazonaws.com/cc_project_2” push the image to ECR repository
6. Lambda function is created with the S3_input_bucket event trigger and then deployed the image from the ECR Repository.
7. In the workload generator set the input and output bucket names accordingly and run the workload generator for both the test cases.