CSE 546 Project - 2 Report

Arizona State University

Aditya Reddy Mali - 1224165813
Paromita Roy - 1224708511
Tejesh Reddy Sigineni - 1222183339

1. Problem Statement

This project aims to build and implement an elastic application that performs face recognition on videos which are input by the user. This application could be used in a smart classroom setting which would enablxe educators to look up academic information of students recognised from the video, whose details are stored in a database.
It is built using the Platform as a Service functionality provided by AWS, using features such as AWS Lambda, DynamoDB and S3.

2. Design and Implementation

2.1. Architecture

In our implementation, a user is asked to upload an MP4 file. The video is uploaded into an S3 bucket (input-bucket-video). Once the video is uploaded successfully, a Lamba function is triggered. This function is created out of an image containing the face recognition functionality, also known as the Lambda handler function. It extracts a frame from the video and stores it as an image. A facial recognition module is run on this image, which creates an encoding. The encoding is matched with a known encoding file to retrieve the name of the student. Finally, we use this name to query a DynamoDB, which contains all the relevant information of every student. Based on the query, the details retrieved are stored in CSV format and sent to another S3 bucket. This bucket serves as the output bucket (output-bucket-vid).

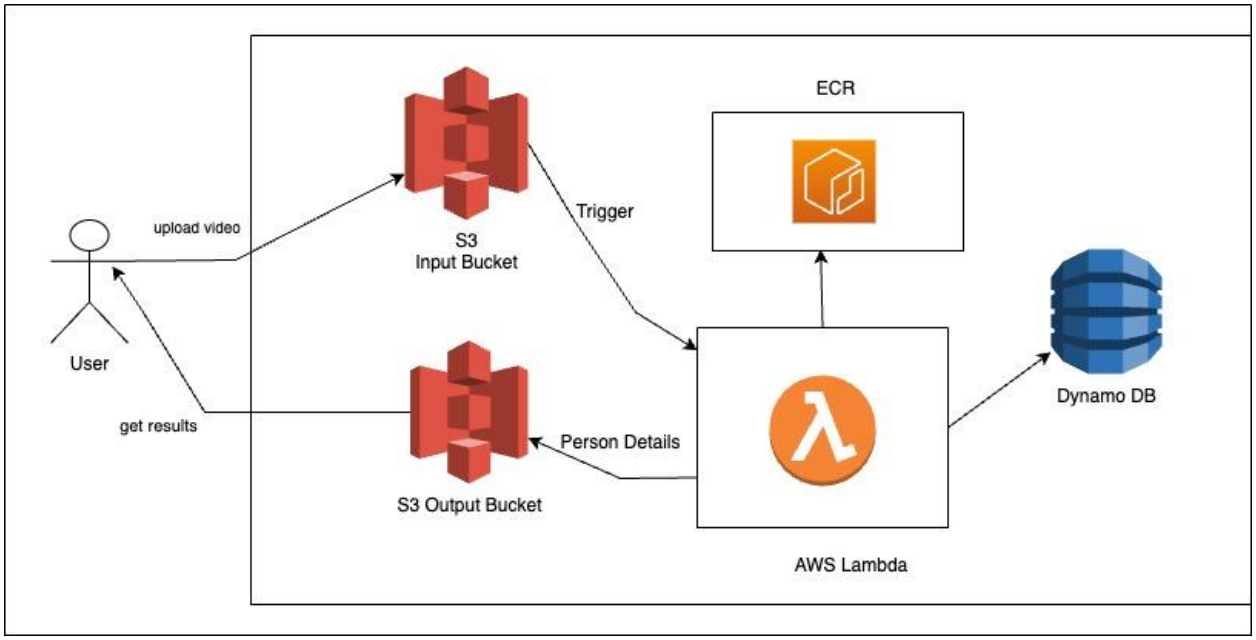

Figure 1. Architecture Diagram

2.2. Autoscaling

AWS Lambda allows you to run code without managing servers. It is a serverless computing solution. One of AWS Lambda's primary features os to automatically modify the number of instances running based on the volume of incoming requests. When autoscaling is set up for a Lambda function, AWS Lambda automatically tracks the volume of requests and modifies the number of instances running, to account for variations in traffic. In other words, as traffic grows, Lambda will automatically add more instances to handle the load, and when it reduces, Lambda will automatically delete instances to conserve resources and cut expenses.

2.3. Member Tasks

1. **Aditya Reddy Mali:** Worked on writing Python code on the Lambda function to get student records from DynamoDB. Processed these results into CSV format and stored it in the output bucket. Configured the IAM users and policies. Worked on end to end testing. Wrote the architecture section in the document.
2. **Paromita Roy:** Loaded data onto the DynamoDB with the data given to us. Set up and configured Docker. Worked on building the Docker image and pushing it to Amazon ECR. Wrote code to query the DynamoDB to compare and retrieve results. Compared the results to check if outputs are appropriate. Worked on the Testing and Evaluation part of the document.
3. **Tejesh Reddy Sigineni:** Worked on the facial recognition service. Used face_recognition library to detect the face in each frame and compared the

detected face's encoding to the known face encodings to determine if a match existed. Used OpenCV library to read the video file and display the output. It took the videos from the input S3 bucket and processed them. Compared it with the encoding file given to us to check results. Wrote on the Code and Outputs section in the document.

3. Testing and Evaluation

To test this application, we used the workload generator provided to us. These are the steps followed:

1. Verify that both S3 buckets are initially empty.
2. Run the following command to upload the MP4 files:

```
python3 workload.py
```

3. Verify that the number of files in the input bucket are equal to the number of videos uploaded by the user.
4. Check the logs to ensure that the Lambda function is being triggered as soon as the input bucket starts getting populated.
5. Check the output S3 bucket to see if results are being populated.
6. Once the execution of the program has been completed, verify that the number of objects in the output bucket is equal to the number of videos initially uploaded by the user.
7. Check each object in the output bucket to confirm that the CSV files contain the right details.

4. Code

The files involved in this application are as follows:

1. **Dockerfile:** The dockerfile given to us required some changes. We included the upload of encoding in this file.
2. **Entry.sh:** This file ensures that the program runs in the appropriate directory.
3. **Handler.py:** This is the entry point when the lambda function kicks in. It contains code responsible for the facial recognition on the file uploaded to the S3 bucket. It also queries the DynamoDB to fetch results. It runs an ffmpeg command which extracts a frame from the video and converts it to jpeg format. Then, an encoding is created and is compared with the encoding file given to us. This returns the name associated with the person in the image. Finally, the output is converted to CSV format and is pushed into the output S3 bucket.
4. **Mapping:** This file contains the mapping for every MP4 file. It gives us information about every tag associated with each video.

5. **Workload.py**: This file has functions which contain code to clear both the input and output buckets when the program initially starts. It also pushes the user uplaoded MP4 files into the input S3 bucket.
6. **ecr_push.sh:** Created a script to run all commands necessary to log into, build and tag the created image, and push it to the ECR repository. This avoids running multiple commands repeatedly.

Steps to create the environment and run the program:

1. Set up docker on your system.
2. Retrieve an authentication token and authenticate your Docker client to your registry.

```
aws ecr get-login-password --region us-east-1 | docker login --username
AWS --password-stdin 704676190155.dkr.ecr.us-east-1.amazonaws.com
```

3. Create an image containing using the Docker file to include files from the directory.

```
docker build -t smart-classroom .

docker tag smart-classroom:latest
704676190155.dkr.ecr.us-east-1.amazonaws.com/smart-classroom:latest
```

4. Push the image to Amazon ECR.

```
docker push
704676190155.dkr.ecr.us-east-1.amazonaws.com/smart-classroom:latest
```

5. Update the image URI on the Lambda function to reflect the uploaded *handler.py*.
6. Run workload generator with the following command:

```
python3 workload.py
```

7. Check the input bucket and verify that the videos are being uploaded successfully. Verify that the number of objects in the bucket is equal to the number of videos being uploaded.
8. Check the output bucket and verify the results are being accurately labelled.

5. Outputs and Screenshots

```
(base) ➜  smart-classroom-face-recognition git:(main) x
'Contents'
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_6.mp4
Uploading to input bucket..  name: test_7.mp4
Uploading to input bucket..  name: test_5.mp4
Uploading to input bucket..  name: test_4.mp4
Uploading to input bucket..  name: test_8.mp4
Running Test Case 2
Uploading to input bucket..  name: test_36.mp4
Uploading to input bucket..  name: test_22.mp4
Uploading to input bucket..  name: test_0.mp4
Uploading to input bucket..  name: test_1.mp4
Uploading to input bucket..  name: test_23.mp4
Uploading to input bucket..  name: test_37.mp4
Uploading to input bucket..  name: test_21.mp4
Uploading to input bucket..  name: test_35.mp4
Uploading to input bucket..  name: test_2.mp4
Uploading to input bucket..  name: test_34.mp4
Uploading to input bucket..  name: test_20.mp4
Uploading to input bucket..  name: test_18.mp4
Uploading to input bucket..  name: test_24.mp4
Uploading to input bucket..  name: test_30.mp4
Uploading to input bucket..  name: test_6.mp4
Uploading to input bucket..  name: test_7.mp4
Uploading to input bucket..  name: test_31.mp4
Uploading to input bucket..  name: test_25.mp4
Uploading to input bucket..  name: test_19.mp4
Uploading to input bucket..  name: test_33.mp4
```

Figure 2. Running the workload generator

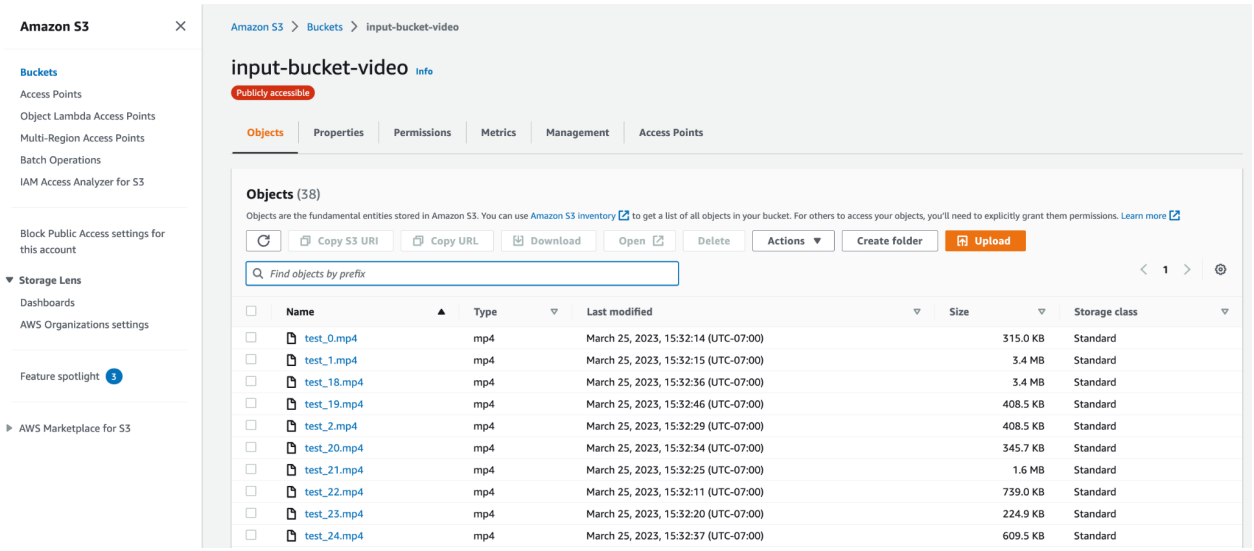

Figure 3. Objects in the input-bucket

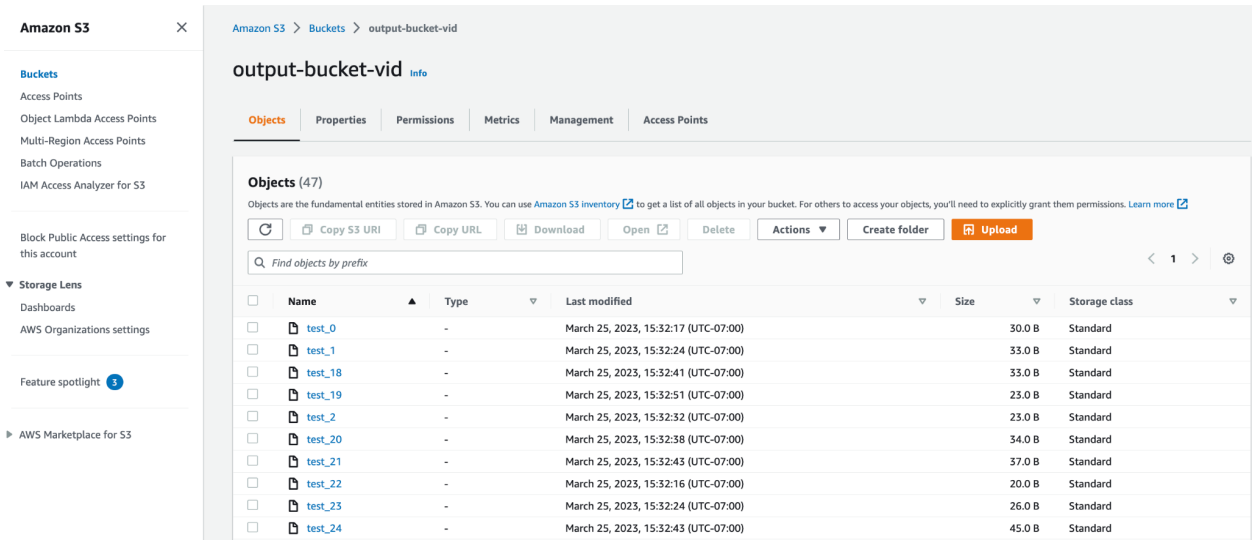

Figure 4. Objects in the output-bucket

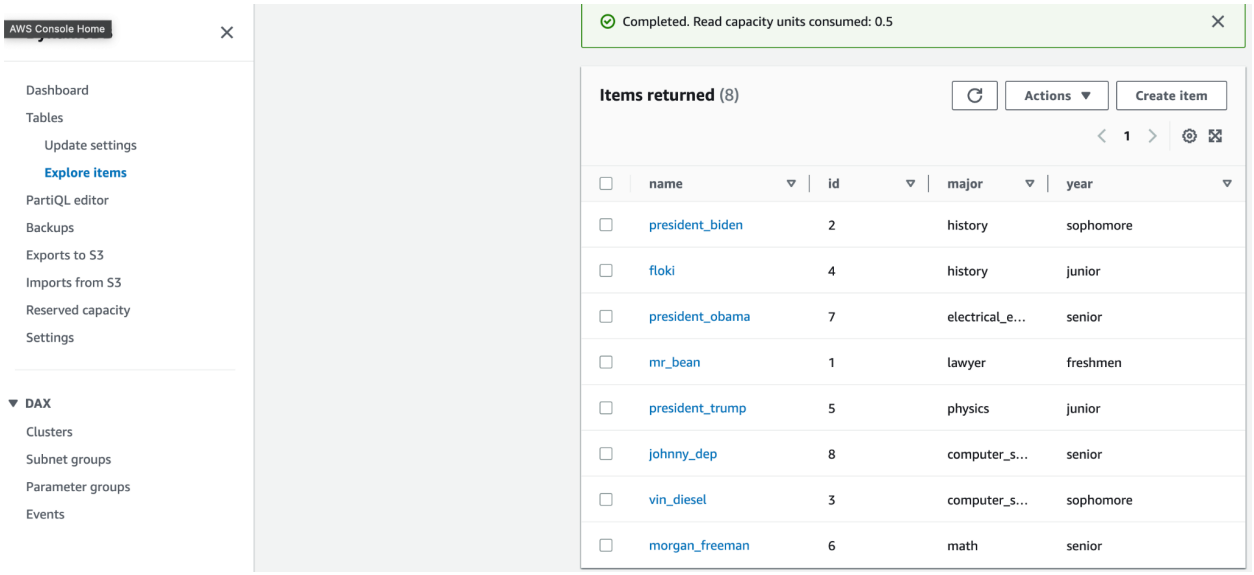

Figure 5. Returned results

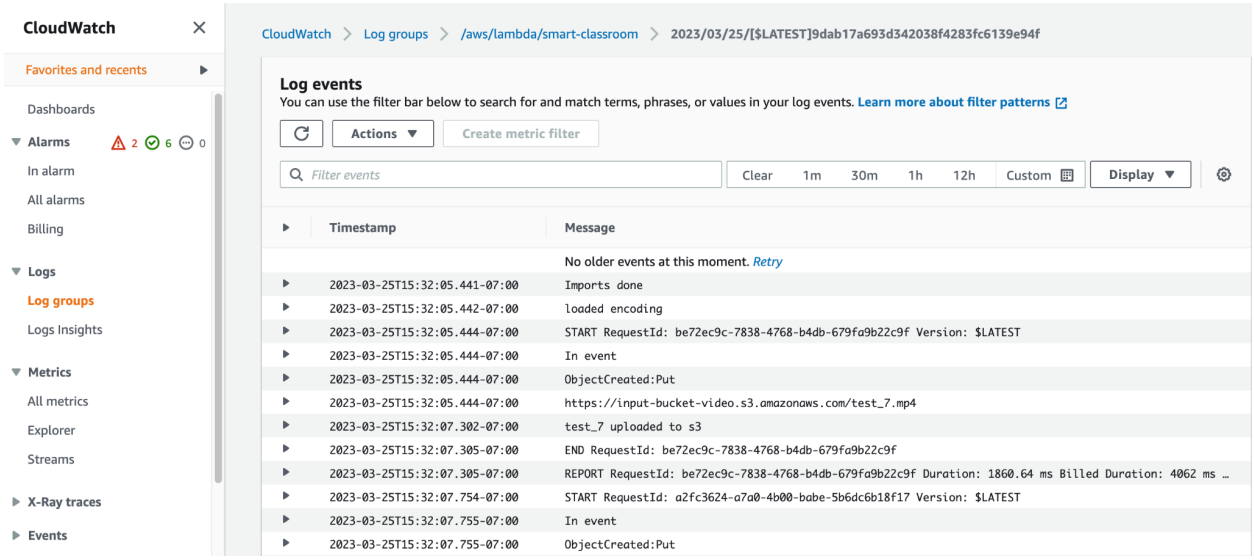

Figure 6. Logs returned by the Lambda function

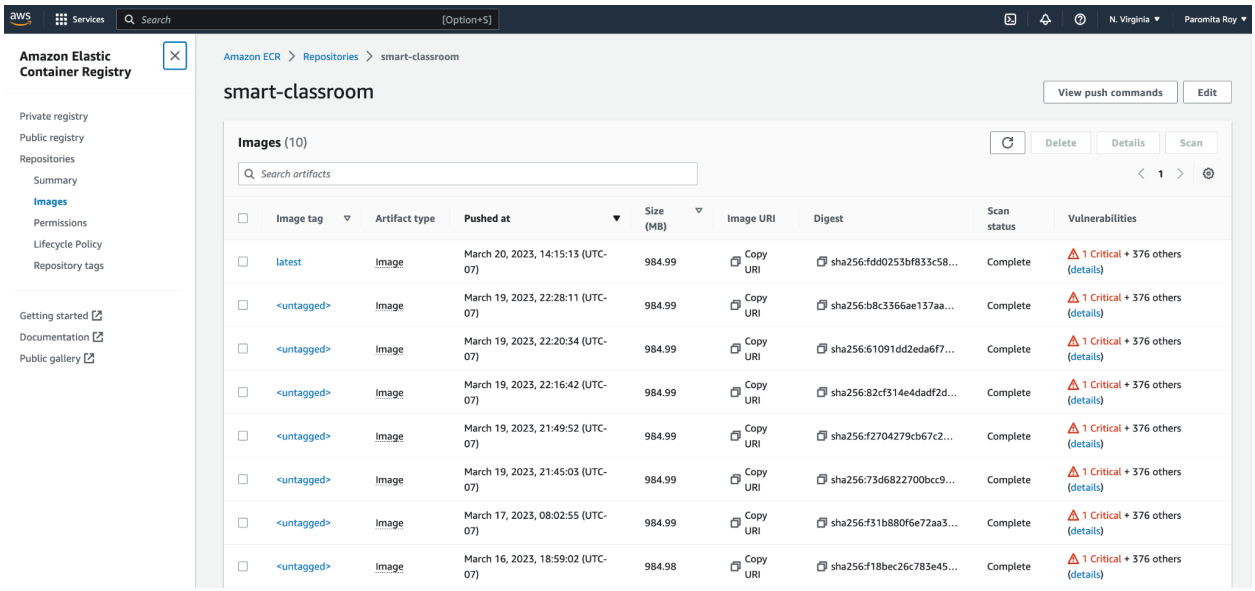

Figure 7. ECR Images

6. References:

1. https://docs.aws.amazon.com/AmazonECR/latest/userguide/getting-started-cli.html
2. https://docs.aws.amazon.com/lambda/latest/dg/images-create.html