CSE 546 — Project 2 Report- Group Skyline Surfers

Atul Prakash - 1225542214

Abhi Teja Veresi- 1225506321

Kedar Sai Nadh Reddy Kanchi - 1225297164

1. Problem Statement

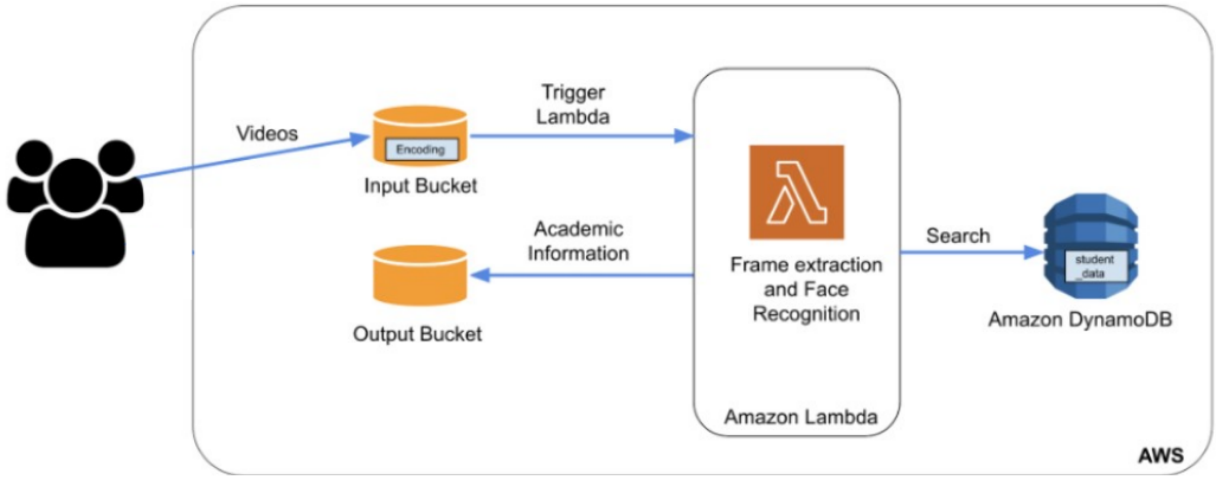
This project aims to develop an elastic cloud application utilizing AWS Lambda and other AWS services that function as a smart classroom assistant for educators. The application should enable users to upload classroom videos to an S3 input bucket. Upon video upload, the system should automatically trigger a Lambda function to process the video. The Lambda function must perform the following tasks:

- Extract frames from the video using the multimedia framework, ffmpeg.
- Utilize the Python face_recognition library to identify and classify the first detected face in the video.
- Search for the recognized face's name in DynamoDB to retrieve the associated academic information.
- Store the student's academic information as a CSV file in the S3 output bucket, with the file name corresponding to the video name.

The system should preload student data into DynamoDB and create the Lambda function using a customized container image containing ffmpeg and the face_recognition library. To ensure accurate and efficient processing, the application should be thoroughly tested using the provided sample videos and a workload generator. The evaluation criteria include the correctness of face recognition, the accuracy of S3 input and output content, and the timely processing of requests, with a reference workload of 100 requests completed within seven minutes. The successful implementation of this cloud-based smart classroom assistant is expected to offer a valuable service to educators and demonstrate proficiency in cloud programming and PaaS cloud technologies.

2. Design and implementation

2.1 Architecture



1. User Interaction:
    a. Users upload classroom videos to an S3 input bucket, which serves as the entry point for video processing.
2. AWS S3 (Input Bucket):
    a. The S3 input bucket stores the uploaded classroom videos.
3. AWS Lambda Function:
    a. Upon the availability of a new video in the input bucket, a Lambda function is triggered. The Lambda function is the core component responsible for video processing.
4. Video Processing by Lambda:
    a. The Lambda function employs a series of steps to process the video, which include:
        i. Using the multimedia framework, ffmpeg, to extract frames from the video.
        ii. Utilizing the Python face_recognition library to perform face recognition on the extracted frames. Only the first detected face is classified, and others are ignored.
        iii. Retrieving the name of the recognized face and searching DynamoDB for the corresponding academic information.
5. DynamoDB:
    a. DynamoDB is used as the database to store and retrieve academic information related to recognized faces. Student data is preloaded into DynamoDB.

6. Academic Information Retrieval:
    a. The Lambda function uses the name of the first recognized face to query DynamoDB and retrieve the academic information associated with that student.
7. AWS S3 (Output Bucket):
    a. The Lambda function stores the student's academic information as a file in the S3 output bucket. The name of the file is derived from the video name, and the content is a CSV file containing three fields: name, major, and year. This output is used to provide the relevant academic information back to the user.
8. User Output:
    a. Users can access the academic information of students from the S3 output bucket, where the processed data is stored.
9. Testing and Workload Generator:
    a. To ensure the accuracy and efficiency of the application, a workload generator is provided for testing. This generator uses a mapping file to verify the correctness of the application's output against expected results.
10. Monitoring and Scalability:
    a. The application leverages the elasticity of AWS Lambda to automatically scale out and in based on demand. As more video processing requests are made, Lambda functions can scale to handle the workload efficiently. This ensures that the system can accommodate varying levels of usage and operate cost-effectively.
11. Development Container Image:
    a. The Lambda function is created using a customized container image that is preinstalled with ffmpeg and the face_recognition library. This container image is built using a Dockerfile.

2.2 Functionality and Infrastructure(Autoscaling)

The AWS lambda function has the ability to automatically increase and decrease according to the inputs to the system. By default, AWS sets a limit of 1000 concurrent lambda functions for the number of AWS lambda functions running at the same time. Each time a source instance executes a lambda function, an environment is initially configured, the lambda function is implemented and executed after that, and the environment is then torn down. If the maximum limit of lambda function invocations is reached, all lambda functions will be executed. AWS Lambda scales the backend infrastructure to handle the number of lambda functions running by all customers at the same time. In our application, AWS Lambda scales up and down based on the number of inputs delivered to the system. We found that this architecture provides an overall optimal efficiency for our project. The services used during the development of our application are:

1. AWS Lambda - In our application, we utilize AWS Lambda services as a PAAS to run our code on it and return the academic information stored on the

AWS S3 output bucket. The S3 input bucket object creation triggers the Lambda function.

2. AWS ECR - Application deployment and management are made possible by Amazon ECR, is a fully managed docker service provided by AWS. The AWS lambda service that we utilize is created by using the docker images stored in Amazon ECR.

3. Amazon S3 – The AWS S3 buckets are used for storing provided input videos uploaded by the workload generator in the S3 input bucket. The The lambda function stores the output file or student's academic information in the output bucket to be viewed by the user after the image recognition is Complete.

4. Amazon DynamoDB - Amazon DynamoDB is a fully managed, serverless, key-value NoSQL database designed to run high-performance applications at any scale. The application uses Amazon DynamoDB to store the personal student information of the student provided in a key-value format.

2.3 Member Tasks

Atul Prakash (1225542214):

Setting up Infrastructure: I have created the infrastructure requirements for the project and some testing of the project, which includes setting up all the buckets (input bucket and the output bucket), DynamoDB, and AWS Lambda along with Kedar. The workload generator was used to upload the input video to the input bucket and use it as a target. As the video is uploaded to the bucket the Lambda function gets triggered as an object creation event occurs in the input bucket. After uploading the file to the AWS S3 input bucket, it will automatically trigger the lambda function. The video file was then downloaded to the Lambda function's directory before extracting the name of the video for the event. For DynamoDB, the student's academic information provided (students_data.json) was uploaded to the database, and the unique ID given was used as the primary key for the database. I also helped in testing the code and checking whether the data was being processed correctly or not and whether we were getting the desired results or not and fixing the problem if any issue occurred.

Code: Generated code in the handler.py file to create the CSV file from the given data and uploaded the file along with the results to the output S3 bucket. I have also implemented the dynamodb_result function to get the data from DynamoDB that matches the results from face recognition. And initially, upfront I have written the code file "loadStudentData.py" to upload data from the provided student_data JSON file to an Amazon DynamoDB table.

Report and Readme creation: Helped in writing the report using the provided template and also created the README file which contains the details of the installation requirements and the steps to run the application.

Abhi Teja Veresi (1225506321):

Code Generation: Generated code in the handler.py file for the image processing function. In this function, the first thing I did was load one of the image files that were extracted from the video. In this project, I have chosen to go with the 2nd extracted image “image-002.jpeg”. The code command to load the image file was present in the face_recognition Python documentation provided by the professor in the project description on Canvas. After loading the image, I then moved to extract the face encodings for that image using the code command present in the documentation. The extracted encoding is the unknown encoding. Now, the main functionality of this function was to compare this unknown encoding with all the known encodings present in the encoding file. So, for this, I now moved to load the encoding file from the working directory using the pickle library. After loading the encoding file, I extracted the names into one list and the corresponding face encodings into one list. Then, we used the comapre_faces function which is provided in the documentation to compare the unknown encoding which we extracted with each of the provided known encodings. Finally, I find the index at which the unknown encoding matches with some known encoding and return at that name at that index.

Testing: Tested the results after completion of the application by sending videos by increasing the input limit and verifying the output with the provided mapping file.

Report creation: Helped in writing the report for testing and code sections in it using the provided template.

Kedar Sai Nadh Reddy Kanchi (1225297164):

Setting up Infrastructure: To set up the Lambda function, a lot of prerequisite things had to be set up. For this, the first thing me and Atul did was to create a new IAM Policy “S3-DYNAMODB-CLOUDWATCHLOGS-ACCESS-FOR-LAMDA-FUNCTION-PROJECT2” within which I have defined the permissions for the CloudWatch Logs, DynamoDB, and S3 buckets. Then I and Atul created a new IAM Role “CSE546PROJECT2ROLE” to which the above-mentioned IAM Policy was attached. Now, with the access set, the next step was to build the Elastic Container Registry Image. So, for this, me and Atul installed the Docker Desktop for Windows. Then we opened the Docker Desktop and started a new session to help us with the next step. Now, we opened the command prompt and created the Elastic Container Registry Image. Finally, using this Elastic Container Registry Image we have created the Lambda Function “AWSLAMDAFUNCTIONPROJECT2CSE546” and while creating the Lambda Function we have attached the new IAM Role that we created for this project. To finish the infrastructure, we made small changes in the Lambda Function, such as increasing the memory size to 512 MB from the default set value of 128 MB and the timeout from 3 seconds to 5

minutes. Lastly, we have set the trigger for the Lambda function to be when there is any create event specified by “ALL CREATE EVENTS” in the input S3 bucket “cc-input-546”.

Code Generation: I specifically have worked on extracting the new event triggered when a new file is uploaded into the S3 bucket. This is basically to extract the newly uploaded video from the input S3 bucket and save it in a local directory “videos” and subsequently delete the event which basically deletes the newly uploaded video from the input S3 bucket. Post implementing this I then moved on to implement the function to extract the images from the currently extracted video. This function was simple to implement with the help of the professor’s sample command provided to us in the project description.

Report creation: Helped in writing the report for testing and code sections in it using the provided template.

3. Testing and evaluation:

We tested and evaluated the application by sending the videos and validating whether the output from the application is correctly matched or not with the provided results. First set up the AWS infrastructure like s3 buckets, AWS ECT, Dynamodb, etc, and verified the infrastructure and its specifications.

After setting up infrastructure we tested each step like whether the videos are uploaded in S3 buckets or not. Tested if input bucket is triggering lambda function where the videos are processed for face recognition. Initially, we tested a sample of one video, and whether the image recognition results were desired or not. Then we scaled the videos to ten and tested the results. Finally, we processed all the videos and verified the results.

We calculated the time taken at each step i.e., the time taken for the videos to get uploaded in the input bucket, the time taken for processing the videos, and results to store in the output bucket and store it in a CSV file.

4. Code:

The code for the application contains the following files and its function.

handler.py:
The Python code is meant to be run as an AWS Lambda function. face_recognition_handler function is the entry point for the AWS Lambda function. It takes an event and context as input. The event is triggered when a new video file is added to an S3 bucket. It extracts the S3 bucket

and object key from the event. It then downloads the video from S3, extracts images from the video, and processes those images using the face_recognition library to identify faces. It looks up additional information about the identified faces in a DynamoDB table named 'CSE546Project2StudentData'. It creates a CSV file containing the information and uploads it to another S3 bucket named 'cc-output-546'. If an error occurs at any step, it prints an error message.

loadStudentData.py:
This file is to upload data from a JSON file to an Amazon DynamoDB table. It reads data from a JSON file and inserts each item from the JSON file into a DynamoDB table specified by the table 'CSE546Project2StudentData'.

4.1 Installation and running requirements of the application:

Access Key: `AKIAQOZDHDE5H5YOCBIJ `

Secret Key: `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`

S3 bucket names: cc-input-546 and cc-output-546

IAM role: CSE546PROJECT2ROLE - arn:aws:iam::031749249338:role/CSE546PROJECT2ROLE

AWS ID: kkanchi@asu.edu

AWS Password:
CSE535CloudComputingAWSProjectAccountPasswordForKedarSaiNadhReddyKanchi@Fall2023

Requirements:

1. Python 3
2. **pip3 install boto3**
3. Install Docker Desktop
4. Create an Input S3 bucket to upload videos
5. Create the Output S3 bucket to store the output file.
6. Configure AWS Elastic Container Registry which stores the Docker container images.
7. Set Up the AWS DynamoDB and load the provided student_data.json file into it.
8. Have the handler.py file ready with the code that you want to deploy.
9. Keep all the required files: handler.py, entry.sh, Docker, requirements.txt, encoding, student_data.json, workload.py, mapping, and the loadStudentData.py in file directory.
10. Open the Entry.sh file in Notepad++

a. Then click on edit tab
b. Then click on the end of file section
c. And select Linux.

4.2 <u>Installation:</u>

1. Clone the repo - https://github.com/nehavadnere/cse546-project-lambda
2. Set the AWS Account Access Key and the Secret Access Key to the directory in which the code is cloned.
3. Configure your S3 bucket names and the DyanamoDB table details in the handler.py file.
4. Open the AWS account and do the following steps:
    a. Create a new IAM policy that gives the Lambda function permissions to access the S3 buckets, DynamoDB, and the CloudWatch Logs.
    b. Create a new IAM Role and attach the above policy to it.
    c. Now open Elastic Container Registry and create a new repository.
5. Now open the newly created Elastic Container Registry repository in 4.c
    a. Then click on the view push commands button.
    b. Open the macOS / Linux tab and execute the 4 commands specified in that one by one as specified in the following points from (6 - 9).
6. Retrieve an authentication token and authenticate your Docker client to your registry. (Use the AWS CLi:)
    a. aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 031749249338.dkr.ecr.us-east-1.amazonaws.com
7. Build your Docker image using the following command.
    a. docker build -t cse546project2lamdarepository .
8. After the build completes, tag your image so you can push the image to this repository:
    a. docker tag cse546project2lamdarepository:latest 031749249338.dkr.ecr.us-east-1.amazonaws.com/cse546project2lamdarepository:latest
9. Run the following command to push this image to your newly created AWS repository:
    a. docker push 031749249338.dkr.ecr.us-east-1.amazonaws.com/cse546project2lamdarepository:latest
10. Finally, using this Elastic Container Registry Image create the Lambda Function
    a. While creating the Lambda Function we have attached the new IAM Role that we created for this project.
    b. To finish the infrastructure, we made small changes in the Lambda Function, such as increasing the memory size to 512 MB from the default set value of 128 MB and the timeout from 3 seconds to 5 minutes.
    c. Lastly, set the trigger for the Lambda function.

References:

1. Group Project 2: PaaS - Canvas Project Description

2. face-recognition 1.3.0 - https://pypi.org/project/face-recognition/
3. Cse546-project-lambda - https://github.com/nehavadnere/cse546-project-lambda