CSE 546 — Project Report

Smart Classroom Assistant for Educators (PaaS)

Sai Vikhyath Kudhroli - 1225432689

Gautham Maraswami - 1225222063

Abhilash Subhash Sanap - 1225222362

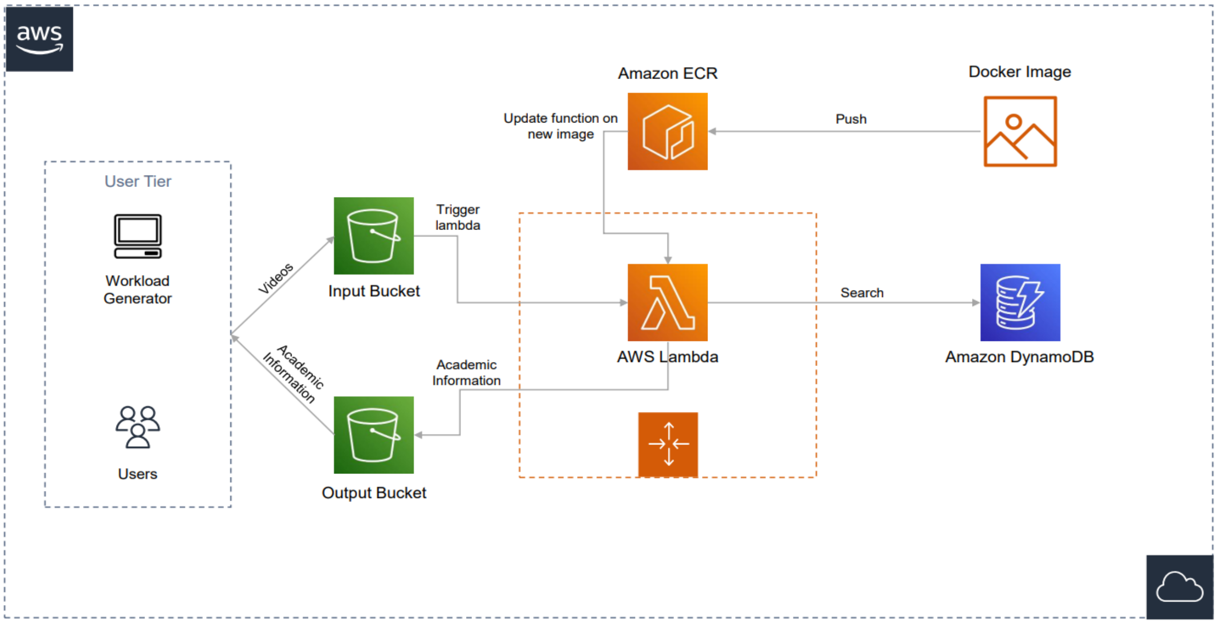
1. Problem Statement

The project aimed to build a scalable application that allows educators to look up information about any student in a smart classroom. The application is supposed to recognize the face of the student in the video and fetch the academic details. To build this application, it was expected to leverage the Platform as a Service (PaaS) functionality offered by AWS - serverless computing engine AWS Lambda, a simple storage S3 and a database service DynamoDB. To extract the frames from the video and to perform face recognition, Python libraries were to be used. All in all, an elastic application that automatically scales out and in on-demand and does so cost-effectively is to be built using the PaaS cloud.

2. Design and implementation

2.1 Architecture

The system consists of four major components: S3 buckets, Lambda function, Amazon Elastic Container Registry, and Amazon Dynamo Database.



All these components along with the docker image are collectively responsible for an elastic Platform as a Service application that provides the academic information of an individual.

2.1.1 S3 Buckets

There are two S3 buckets that are utilized: one input bucket and one output bucket. The input S3 bucket stores the input video files, and the MP4 videos that are uploaded by the user or workload generator are pushed into the input S3 bucket. There is also a trigger defined on the input S3 bucket, whenever a video is uploaded it triggers the lambda function. The output bucket is used to store the CSV files corresponding to the video uploaded which consists of academic information about the first person in the image. The output bucket receives the CSV files from the lambda function. We have used himaliaInputBucket as the input bucket and himaliaOutputBucket as the output bucket name

2.1.2 Lambda function

The lambda function is the core of the system. This lambda function is triggered when a video is uploaded into the input S3 bucket. A docker file is used along with the lambda function to support the following functionalities:

- Lambda function handler reads the encoding file which consists of the encodings of all the individuals in the class.
- Downloads the video from the input S3 bucket.
- Extracts all the frames in the video, then selects the first frame which has a person.
- Extracts the name of the person in the video by comparing the encoding of the first frame and all the encodings.
- Using the name, fetch the details of the person stored in the dynamo database.
- Creates a CSV file with all the academic information and pushes it to output S3 bucket.

2.1.3 Elastic Container Registry

Amazon ECR is a fully managed container registry that can be used to deploy application images. The docker image is deployed on ECR which enables high availability and scalability. This eliminates the need of defining a scale-in and scale-out policy based on demand.

2.1.4 Dynamodb

Amazon DynamoDB is a fast, fully managed, and serverless key-value NoSQL database that is used to store the academic information of all individuals. All the academic information pertaining to all the individuals is pushed into the database. Then after the name of the person is extracted, it is used as the key to extracting all the academic information pertaining to that individual. We have used student_data as the table name.

2.2 Autoscaling

The services used such as AWS lambda and AWS DynamoDB are capable of handling varying demands and provide auto-scaling without the need to explicitly define auto-scaling policies. As the lambda function receives more requests, it automatically handles scaling the number of execution environments until the account’s concurrency limit is exhausted.

2.3 Member Tasks

2.3.1 Sai Vikhyath Kudhroli (1225432689)

1. Created the face recognition handler that is executed when the lambda is triggered.
2. Loaded the encoding data which is used to recognize the faces in the video.
3. Extract the key of the video file which is uploaded to S3 and download the video from the input bucket.
4. Use the downloaded video to extract all the frames in the video.
5. Extract the first frame that contains a person and generate encoding for it.
6. Developed the face recognition logic to extract the name of the person in the video.
7. Use this name to extract academic information about the person from DynamoDB.
8. Create a CSV file with all the academic information extracted from DynamoDB.
9. Push this CSV file into the output S3 bucket.

2.3.2 Gautham Maraswami (1225222063)

1. Set up S3 Buckets programmatically using boto3 libraries.
2. Configured buckets, so that the workload generator and lambda functions are able to upload and download files to the bucket seamlessly.
3. Set up DynamoDb using Boto 3 libraries.
4. Defined Schema, keys, attribute definitions, and Throughput so as enable correct configurations of DynamoDb.
5. Set up user roles for lambda functions for correct access to Log files, S3 Buckets, and DynamoDb.
6. Developed a testing mechanism to check if the generated outputs are correct.
7. Manually and programmatically compared the generated outputs in the S3 Bucket and the mapping files provided in the bootstrap file.
8. Loaded the data provided in the JSON file into DynamoDB.

2.3.3 Abhilash Subhash Sanap (1225222362)

1. Read and understood the DockerFile and entry.sh file.
2. Created an AWS Elastic Container Registry (ECR) repository.
3. Built the docker image from the given Docker File.
4. Configured the programmatic access to the ECR and pushed the image to it.
5. Configured an AWS Lambda function through the console using the “latest” image.
6. When the lambda function failed to run, I debugged the issue and set the memory requirement and timeout to an appropriate value.
7. Understood the triggers Defined a trigger on the lambda function such that it is invocated every time an object is added to the input bucket.

3. Testing and evaluation

3.1 Reliability Test

The application was tested to make sure the expected reliability and accuracy are achieved. It was tested and confirmed that as many lambda functions are invocated as there are videos in the input bucket. The application was tested to check if it can handle up to one hundred requests concurrently and it successfully does.

3.2 Flow of Operations Test

The expected flow for the application is – the user uploads the video to the input bucket, which triggers an event that launches a lambda function, which extracts the image, performs face recognition, queries the DynamoDB to get academic information, and pushes it to the output bucket in a CSV format. It was ensured that this happens in the expected order.

3.3 Output validity test

Response at the workload generator is checked against the source of truth document provided to make sure all the details fetched from the DynamoDB are correct. The time taken for the execution of one hundred images is noted to be around three and a half minutes.

4. Code

Technologies used: Python, Boto3, AWS S3, AWS DynamoDB, AWS Lambda, AWS Cloud watch (For Monitoring purposes)

The file Handler.py has all the logic required for downloading the file from S3, Image Recognition, getting additional information from DynamoDb, writing the output to S3. Even though DockerFile and entry.sh were already provided in the bootstrap code but required a more profound understanding of what was happening inside the code as running the code had multiple failures and needed debugging on the same.

Files:

1. Handler.py: This file contains the core logic of the project containing, Image recognition, S3 Bucket access, and DynamoDB Access.
2. DockerFile: This file contains the dependencies installation and triggers to be created on lambda function creation.
3. Entry.sh contains the command to be executed on lambda function execution

Handler.py

1. This file is responsible for the core operations of the project.

2. This file is triggered when any object is put in the S3 Bucket. Face_recognition_handler is invoked.
3. Method download_from_s3 is to download all the videos from S3 Bucket.
4. The downloaded file is stored in /tmp directory
5. Method generate_encoding extracts the first frame from the video and generates the encoding of the image (Frame).
6. Recognise_face method compares the encoding from the image and the encoding input provided and returns the corresponding name of the identified person.
7. fetch_data_from_dynamoDB: gets the year and major information from the dynamo database corresponding to the name.
8. Write_to_csv is created as a CSV file containing the name, major, and year corresponding to every video uploaded. The CSV files will be available in /tmp
9. The calling method then uploads the files to the S3 Bucket.

Docker File

1. The docker file is provided in the bootstrap code.
2. Function directory is specified in the docker file where image/video/csv files will be stored.
3. This file installs all the dependencies in the container required for the execution of the function.
4. The dependencies include, GCC, python, python AWS Lambda runtime interface client, AWS Lambda runtime interface emulator, FFmpeg library for image recognition, OpenCV, boto3, and NumPy.
5. Necessary permissions for the executable files are also provided using this file.

4.1 Project setup and execution

1. Create S3 Bucket for input and output using AWS Console
2. Set Bucket permissions to public
3. Use the handler file provide and create a docker image using the file.
4. Create a Docker image and push it to AWS ECR
    a. Use the following commands:
    b. Retrieve an authentication token and authenticate your Docker client to your registry. Use the AWS CLI:
        i. aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 630075306220.dkr.ecr.us-east-1.amazonaws.com
    c. Build your Docker image using the following command:
        i. docker build -t face_recoginition_sample .
    d. After the build completes, tag your image so you can push the image to this repository:

        i. docker tag face_recoginition_sample:latest 630075306220.dkr.ecr.us-east-1.amazonaws.com/face_recoginition_sample:latest

    e. Run the following command to push this image to your newly created AWS repository:

        i. docker push 630075306220.dkr.ecr.us-east-1.amazonaws.com/face_recoginition_sample:latest

5. Create a Lambda function from the AWS Console and define a trigger so that it is invocated when a new object is added to S3.
6. Create role for which has the following permissions
    a. AmazonS3FullAccess
    b. AmazonDynamoDBFullAccess
    c. AWSXRayDaemonWriteAccess
    d. AWSLambdaBasicExecutionRole

7. Assign the role to the AWS Lambda function
8. Update input bucket name in Workload generator and run the to input bucket.

5. Individual Contributions – Portfolio

5.1 Sai Vikhyath Kudhroli (1225432689)

Introduction

The aim of the project is to develop an elastic application that can scale in and scale out on demand that extracts academic information about the student based on the video that comprises that student using AWS PaaS.

Solution

The application consists of two S3 buckets, Lambda function, Amazon ECR, Docker file, and Amazon DynamoDB along with a face recognition module that provides academic information based on the video.

Design Contributions

- My significant contribution to this project is the design of the lambda function and its integration with S3 buckets and DynamoDB.
- I designed the face recognition module and its integration with the lambda function. I designed the approach to utilize the encodings and recognize the face in the video using the face recognition module.

Implementation Contributions

- Implemented the face recognition handler (handler.py) which is triggered when the video is pushed to the S3 bucket.
- Implemented the method to read the encodings and utilize them later for face recognition.
- Extracted the key of the video pushed to input S3 bucket, using the key downloaded the video from input S3 bucket.
- Using the downloaded video, extracted all the frames in the video using FFmpeg.
- From all the frames extracted, identified the first frame that consists of a person and generated encoding for this frame.
- Implemented the face recognition logic that returns the name of the person in the video by comparing all the encodings and the encoding extracted from the image.
- Use the name extracted to query Amazon DynamoDB to retrieve all the academic information for that individual.
- Generate a CSV file using the academic information obtained from DynamoDB
- Push the CSV file generated to the output S3 bucket.

New Skills and Technologies Acquired

- Using AWS S3, AWS Lambda, AWS ECR, AWS DynamoDB, Python's Face Recognition module, and Boto3 AWS SDK to develop an elastic, on-demand autoscaling application to extract information about an individual from a video of the person and deploy on cloud PaaS resources.
- Utilize the face-recognition module to implement the face recognition functionality.
- Utilizing AWS Lambda to create serverless compute services that can scale up and scale down automatically on their own.
- Utilize DynamoDB to achieve high availability and scalability.

5.2 Gautham Maraswami

Task: My responsibilities in this project is mainly to set up S3, Buckets, and DynamoDB. I also individually spend sufficient time understanding various features of DynamoDb and AWS Lambda.

S3: Using the skills learnt from the previous project I was able to use boto3 to set up S3 buckets. Individually experimented with the boto3 library and after analyzing the documentation. It was important to set up bucket policies so we could upload files into S3 from the workload generator. Also, Lambda functions must be able to download video files from the input bucket and upload CSV files to the output bucket.

DynamoDb: After analyzing the documentation setup DynamoDb using AWS Boto3. Had to understand the significance of parameters like Keyschema and the difference between KeyType Hash VS KeyType Range.
Learnings

1. ProvisionedThroughput: used to set up configurations like the maximum number of strongly consistent reads consumed by the database.
2. Queries to create, read, update, and delete elements from the DynamoDb.
3. Scan Commands which contain expressions and start keys and how to use them.
4. Using batch updates to update multiple entries into a table vs adding individual columns into the table.

AWS Lambda: Even though the actual contributions to this project were majorly taken up by other team members. Significant effort was put into understanding the various configurations of lambda functions.

1. Understanding the creation of lambda function from Docker image from the container.
2. Types of triggers available to the lambda function, including S3, DynamoDB, and SQS.
3. Creation of roles for resource access by the Lambda function.
4. Create functions from Aws console, blueprint, etc.
5. Understood the time and memory restrictions of AWS Lambda functions.

Testing the generated output: Compared the output generated by the lambda function available in the S3 Bucket with the given mapping file. As the number of files generated was large. We used Aws CLI to download the files from the S3 bucket. Used Pandas libraries to merge the outputs, into a data frame, created join with the given mapping file to compare the generated output with the available outputs.

Learning from the project
Ability to Setup Docker create images and upload the image to the repository. Set up lambda functions using various approaches including using container images. Setup and usage of S3 Buckets. Setup and perform create, read, update, and delete operations on dynamo DB. Setting up test frameworks to validate the output using pandas libraries.

5.3 Abhilash Subhash Sanap
ASU ID - 1225222362

Responsibilities

- Conceptualise the overall design of the application in a collaborative way with other teammates
- Build a Docker image and push it to AWS Elastic Container Registry (ECR).
- Set up the Lambda function using a Docker image.
- Define a trigger on the lambda to handle a new object being added to S3.

Contributions

Architecture

- While other team members implemented the face recognition code and set up the S3 buckets and dynamoDB, I actively contributed to the group discussions to decide the configurations required for the project.

AWS Elastic Container Registry

- I read and understood the functionalities offered by AWS ECR.
- I set up an ECR repository to which we pushed the Docker images during development and finally used the “latest” image in the Lambda function.

AWS Lambda

- I read and understood AWS Serverless Computing and specifically the inner workings of Lambda.
- The face recognition code took more than 3 seconds to run locally which led to a timeout when it was deployed on Lambda. I realized that the default timeout and memory limit settings of the Lambda function were inadequate for our use case.
- In order to solve the problem, I increased the time limit of the function to 5 minutes and allocated a bigger chunk of memory.
- After the Lambda function ran successfully, I packed the function into a docker image and tested the entire flow end to end with the help of my teammates.
- I monitored the CloudWatch Metrics like Invocations, Error rates, and Logs to ensure that the lambda function is working perfectly.

Learning Outcomes

- Using Platform as a Service modules like AWS ECR, and AWS Lambda, including various ways to define and invoke lambda functions, AWS Cloud Watch, and AWS S3.
- Learning AWS CLI and boto3 AWS SDK to develop applications based on AWS
- Understanding of using a Docker image to create a function.
- In-depth understanding of Triggers on Lambda functions, how they can be set on S3, DynamoDB, and SQS events
- Understanding various configuration settings in Lambda like Timeout, Memory allocation, and ephemeral Storage.