CSE 546 - Project 2 Report

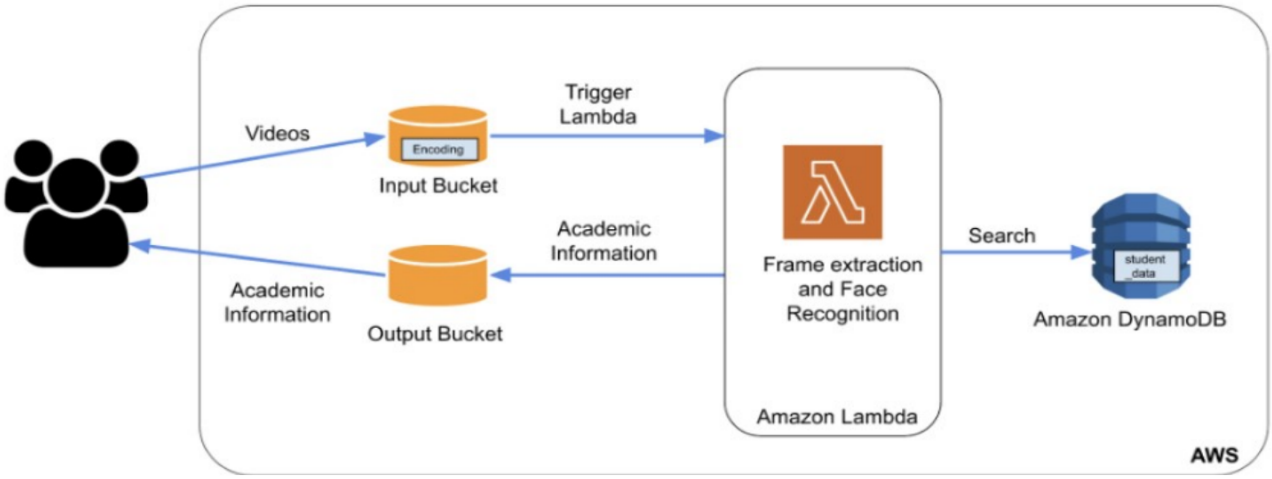
Adwait Patil
Dhaval Shah
Akul Rishi

Problem Statement

The goal of this project is to build an elastic web application that uses a PaaS cloud service to automatically scale out and scale in on demand and cheaply. The program receives videos, uses face recognition to categorize them, and then outputs pertinent data regarding the Amazon DynamoDB employee. This was built up using the AWS cloud. Utilizing services like Simple Storage Service (S3), AWS Lambda, and Amazon DynamoDB give a reliable mechanism to aid in running the program. Because we are developing an application that serves a big number of users, this issue is crucial for concurrent user requests very effectively, without relying on a single server. It instead utilizes the AWS cloud resources to provide an image recognition solution to the uploaded videos and returns relevant information to the user.

Design and Implementation

Architecture



AWS Services used in the project

1. **AWS Lambda:**
   You may run code for practically any kind of application or backend service with AWS Lambda, a serverless, event-driven computing service, without creating or managing servers. Our complete code is performed using AWS Lambda as a PaaS, and it then returns the appropriate academic data to the output bucket. The

creation of the Input Bucket object is the sole event that sets off the Lambda function.

2. **AWS S3:**
   We are using AWS S3 to store input videos (.mp4) that were uploaded to the input bucket by the task generator. After the picture recognition is finished, the Lambda function stores the student's academic data (.csv) in the output bucket so that the user can examine it. For persistence, we use the AWS S3 service.
3. **AWS ECR:**
   You can deploy, maintain, and grow containerized apps with ease using Amazon ECR, a fully managed container orchestration service. We are utilizing this service to store the Docker image that will be used to launch the AWS Lambda service.
4. **Amazon DynamoDB:**
   Amazon DynamoDB is a fully managed, serverless, key-value NoSQL database designed to run high-performance applications at any scale. We are utilizing Amazon DynamoDB to store the academic information of the person in a key-value format.
5. **Amazon Cloudwatch:**
   This service that is provided by AWS helps keep an eye on the logs generated by different AWS services. We used this service while developing the lambda function to understand the different intermediary results and their types.

Architecture Explanation

In our architecture, we have two AWS S3 buckets for storing input and output data and a single AWS Lambda function to perform image recognition on the frames of movies delivered to the system and return related data saved in Amazon DynamoDB. Videos make up the input data, while academic knowledge makes up the output data. A Docker image containing the software and packages needed to run the AWS Lambda function serves as its foundation. Locally created and kept in an ECR repository, this image. This lambda function is activated with information pertaining to the file that was dropped once a video is dropped into the input S3 bucket. The function uses this data to download the movie to the lambda environment, extract image frames from it, classify the image using the face_recognition python package, and search for the resulting name in our DynamoDB. The academic details of the person identified in the video were taken from the database. This is transferred to the output S3 bucket in.csv format for storage. Finally, before exiting, the intermediary files created by the function are deleted since their persistence is not necessary.

Autoscaling

Each Lambda function is executed by AWS Lambda as a separate process in its own environment. A default cap of 1000 concurrent Lambda functions applies. The environment is built, the function is executed, and the environment is destroyed each time an event source executes a Lambda function. 1000 Lambda functions are launched when there are 1000 invocations. It automatically "scales" by operating concurrently on the AWS infrastructure. AWS's responsibility is to make sure that their back-end infrastructure can scale to accommodate the total amount of Lambda functions being executed by all of their clients.

Member tasks

Adwait Patil

Design: I was in charge of writing the handler.py script for this project. I spent time storing the most recent video file that was uploaded to the input S3 bucket in the lambda temporary folder and extracting the film's frames for facial recognition. The input and output S3 buckets were also set up by me.

Implementation: My installation process started with setting up the AWS S3 buckets. The workload generator's videos were uploaded using the input bucket. On the input bucket, I configured an Object Creation event that would launch the AWS lambda function. As a result, the lambda function will be called each time a file with the prefix "test_" and suffix ".mp4" is added to the input bucket. I looked at the event log information before storing the video from the S3 bucket to the lambda environment. The video file for that event's filename was then extracted, and it was downloaded to the lambda function's '/tmp' directory. The name of the identified person was obtained from a one-hot encoded array that constituted the model's output. Later, this was utilized to query DynamoDB to retrieve their academic data. A ".csv" file with the DynamoDB result was created and sent to the output bucket.

Dhaval Shah

Design: I was given the task of putting up the AWS Lambda environment for this project, which contains the code in charge of doing the categorization and orchestrating the outcomes. This Docker container image served as the foundation for this AWS Lambda function.

Implementation: I combined the professor's provided docker file with my team's "handler.py" script, the deep learning model's "encoding" file, and the "entry.sh" shell script in order to construct the bespoke docker image. Then, I turned this bundle into a docker image and submitted it to Amazon Elastic Container Registry, often known as ECR, a place where Docker images are stored. I referred to the image in the ECR repository when configuring the Lambda function. I built a new image, uploaded it to the ECR

repository, and changed the AWS lambda function to point to the most recent image each time we made changes to the project's code base.

Akul Rishi

Design: Setting up the DynamoDB and creating a script to retrieve the academic data from the DynamoDB were my responsibilities for this project.
Implementation: I created a DynamoDB called "new students" and preloaded it with the academic data that the professor had provided to the student in the "student data.json" file. The primary key for this database was initially kept as the table's "ID" column. However, during testing, we discovered that this approach was somewhat ineffective for looking up academic data in the table. We ultimately came to the decision that since each name is distinct, the name of the person should serve as the primary key. We decreased the DynamoDB search time by implementing the table in this manner. After that, I created a script to get the categorized person's academic background.

Testing and Evaluation

We tested our system and acquired a ton of information about many elements. The majority of input videos for uploads are quickly uploaded to the S3 bucket. Our tests revealed that a video may be uploaded to the input bucket in as little as 1.106768847 seconds and as long as 9.617904186 seconds.

Code

1. handler.py:

This python file contains the code for the AWS Lambda function whose functionality is to download the input video to the ~/tmp folder, extract a frame from the video, perform facial recognition using the provided model, request the academic information from DynamoDB, upload the information to the output bucket and finally delete all the files stored in ~/tmp folder.

2. Dockerfile:

The Docker file contains the script to install the packages required for the code to run The image built on this file is used by the AWS Lambda function to perform the functionalities.

3. student_data.json:

This file contains the academic records data of different individuals. This is used to prepopulate the database that is then later queried.

4. encoding:
This file contains the trained deep learning model which the handler.py imports to classify the images extracted from the input video.

5. entry.sh:
This shell script is used to invoke the “handler.face_recognition_handler” function in the lambda environment when it is triggered.

How to run:

1. Install Python.
2. Install boto3 and awscli packages and configure the AWS environment.
3. Open the workload.py file and change the input and output bucket variables to point to the correct values as mentioned above.
4. Execute the workload file to upload videos to the input bucket. This triggers the lambda function to execute.
5. Open the output bucket on the browser to find the respective academic records being generated.

References

[1] https://docs.aws.amazon.com/lambda/latest/dg/with-s3-example.html
[2]https://stackoverflow.com/questions/36144757/aws-cli-s3-a-client-error-403-occurred-when-calling-the-he
adobject-operation
[3]https://docs.aws.amazon.com/AmazonECR/latest/userguide/docker-push-ecr-image.html
[4] https://docs.aws.amazon.com/cli/latest/reference/dynamodb/index.html