CSE 546 — Project 2 Report

Karthik Aravapalli (1225611998)
Deepak Reddy Nayani (1225418259)
Nikhil Chandra Nirukonda (1223333995)

1. Problem statement

The aim of the project is to leverage PaaS services provided by *Amazon Web Services* i.e AWS Lambda. Lambda is a function-based server based computing service that is being used to build an elastic application that scales based on demand. Said application uses Python recognition modules to classify images on a pre-trained image recognition model, the application stores the output of the model and provides the relevant information about the subject. This problem is a valid subset of real world scenarios that involve creating scalable applications using face recognition APIs and cloud resources

2. Design and Implementation

2.1 Architecture

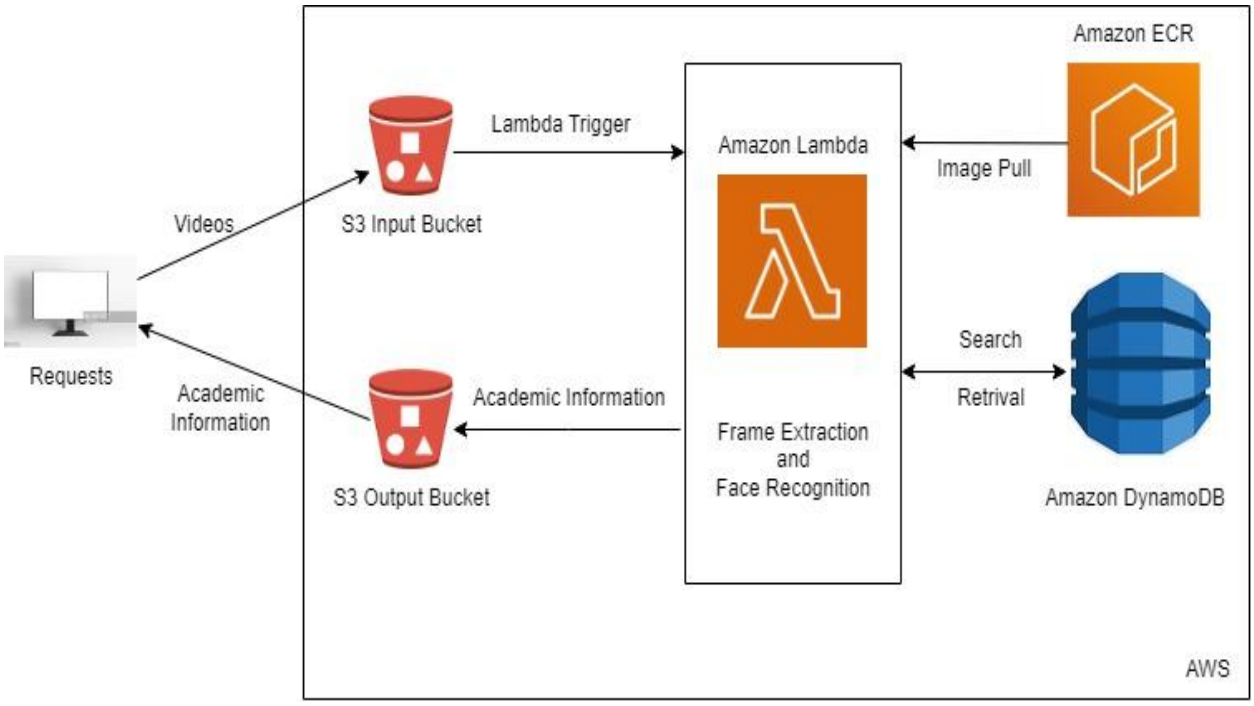

Fig. 1.1 Architecture Diagram

2.1.1 Architecture Description

From a cloud computing perspective, we are following a loosely-coupled architecture involving the use of various AWS Microservices such as Storage(S3), Computation(EC2), Deployment(Lambda) etc.

Referring to Fig 1.1, the workflow of the architecture is as follows, firstly, the video clips containing the subject are uploaded to the S3 Input Bucket. Upon uploading, an S3 event is triggered by AWS Lambda which is responsible for processing the videos in the input buckets and querying the output with DynamoDB and writing the output back into an S3 Output Bucket.

The lambda function is a critical part of the architecture. It contains the docker image stored into an AWS ECR repository, which copies all the dependencies and packages to run the face recognition. The lambda pipeline begins by downloading the input videos from S3 into Lambda and processing the input videos. The processing of the video involves creating the frame/thumbnail for each video, encoding the frame and using said encodings to run face recognition, finally query the output against a prefilled table on DynamoDB and write the output back into the S3 bucket as a .csv file. The lambda service is allocated 5GBs of storage memory, 3GBs of RAM to compute the above specified process, with a lambda timeout threshold of 15 minutes incase of a failure.

2.1.2 AWS Components

Simple Storage Service (S3) - Amazon S3 is a scalable object storage service provided by AWS, used to store the input videos.

Lambda - Lambda is a function that offers a service for another function. It is a computing service that facilitates the creation of functions that apply computation to cloud-based data. It is used to categorize photos and retrieve student data from DynamoDB based on classification.

DynamoDB - DynamoDB is a database management system that supports key-value storage. It can be utilized to store records of the students so that any student's academic information can be obtained using the student's name as the key.

2.2 Autoscaling

For this project, there isn't a user specified auto scaling policy. The scaling is done by the AWS Lambda function. In case of concurrent requests, Lambda provides a separate execution instance, and as the number of parallel requests increases , it increases the number of execution instances till the account threshold is reached (which is usually a 1000). In some cases, we can set the concurrency limit to a specified cap, so that one particular function does not use up all the computing resources. But, in this case, we create and use only one function, so the lambda scaling automatically adjusts itself with respect to the number of requests coming in and outputs the files efficiently.

2.3 Member Tasks

Deepak Reddy Nayani - I was responsible for writing the logic for handler.py and processing of the videos to ensure that the images formed where appropriate, with the correct name and stored in the correct folder. Further, I wrote the code that leverages the python face_recognition module to output the final result and uses boto3 to write the output as a .csv into the output bucket.

Karthik Aravapalli - I was responsible for writing the DynamoDB part to insert the given data to db and lambda function to get the results from db to fetch the academic results based on the face recognition results and return back to the output bucket and I also involved to writing some of the logic for handler.py and lambda function auto trigger and docker file deployment and image creation.

Nikhil Chandra Nirukonda - I was responsible for documentation and helping on the face recognition code and the lambda function trigger creation and validated the results which are generated by the lambda using the face recognition python libraries. Also, tested all the test cases which are given by the project instructor to complete all the tasks within 7 minutes.

3. Testing and evaluation

For testing the code, we have separated the entire process into modular tests to make sure every function is working as intended. The testing pipeline is as follows:

- ➔ Tested the workload generator to make sure that all the items in the test_cases folder were being uploaded as per and also making sure that the input and output buckets are emptied before uploading any files to either of them.
- ➔ Locally tested the face_recogition module with sample images to make sure the classification is accurate and the results produced are as per mapping.
- ➔ Checked the entries in dynamoDB manually after uploading the student_data.json object to verify all the data has been pushed to the database.
- ➔ Tested the lambda functions with various IAM roles to make sure that the reading and writing of files from and to the bucket and dynamoDB are seamless.
- ➔ Testing if the event produced by the S3 bucket is being parsed correctly. An example of the event is :

```
{
   "Records":[
      {
         "eventVersion":"2.1",
         "eventSource":"aws:s3",
         "awsRegion":"us-east-1",
         "eventTime":"2023-03-24T22:07:57.158Z",
         "eventName":"ObjectCreated:Put",
         "userIdentity":{
            "principalId":"AWS:AIDAWVOECYJ5QGQ5K7HK5"
         },
         "requestParameters":{
            "sourceIPAddress":"174.73.252.71"
         },
         "responseElements":{
            "x-amz-request-id":"1S02ZS27DMWD3MWM",
            "x-amz-id-2":"QZvsnM4TaV2dY1/0yLWStfFCHLvpd4yBtCQb4KLuS2/MdGrHVQe8Tr7L+zvDfExuNoIZAqjyyw+EK3kD5b+WQEYwTdVA1lUU"
         },
         "s3":{
            "s3SchemaVersion":"1.0",
            "configurationId":"b4d7981f-c4d7-4e31-8adc-e5a80a6a87fb",
            "bucket":{
               "name":"ndk-proj2",
               "ownerIdentity":{
                  "principalId":"A2MCSMR51B730P"
               },
               "arn":"arn:aws:s3:::ndk-proj2"
            },
            "object":{
               "key":"test_13.mp4",
               "size":1719624,
               "eTag":"e4af7894fcd0a4ad49c88fb8e7413f79",
               "sequencer":"00641E1F3C26C46447"
            }
         }
      }
   ]
}
```


- ➔ Testing that all the directories are empty after each iteration, to make sure that pre-saved results from a previous iteration are not outputted again and that the entire image recognition process happens for all the relevant files only.
- ➔ Tested the lambda function with various parameters (timeout, memory, ephemeral storage). Chose the upper limit of the free-tier option (3GB memory and 8GB ephemeral storage for optimal performance).

Finally, for the evaluation, since the output is an array of student information, such as graduation year, major, name. We cross checked the outputs of the result with the *mapping* provided by the TA to ensure correctness of the ML models output.

4. Code

Dockerfile - The dockerfile provided, contains code to install the OS, dependencies and packages (from requirements.txt) required to carry out the required operations. It sets the permissions of the system, creates the working directories and finally runs the lambda handler function upon triggering of the lambda function.

workload.py - The workload generator is responsible for clearing both the input and output S3 buckets and to upload the video files from the "test_cases" folder using python boto3 client. It is to be run locally.

uploadDB.py - This file is also run on the local machine and is required to be run only once. It is responsible for connecting to dynamoDB using boto3 resource, connecting to the table and uploading the data provided in *student_data.json* into the table using the get_item() method, to be queried later in the process.

handler.py - handler.py contains the majority of the code and the logic. The tasks performed by the code are as follows:

- Upon running, it first clears all the stored files for every iteration. i.e clears all the folders and files in "tmp".
- Parses the S3 event upon lambda invocation and stores the names of the input bucket and the filename upon uploading to S3.
- Downloads the videos from S3 into Lambda "/tmp" folder using the filenames that are parsed, as specified above through the *getVideosfromS3() function*.
- Creates a directory for storing the frames/thumbnail ("/tmp/test_frames)") of each video. Processes each video in the /tmp folder that is downloaded from S3 to create respective images (.jpeg) file using python-ffmpeg package and stores the images into said test_frames folder.
- Iterates through each image in the test_frames folder and feeds it to the python face_recognition API.
- The face recognition is performed in the *recognize_face()* function, which uses the encodings provided by the TA to classify the names for the unknown test images present in the test_frames folder.
- The output of the face_recognition is then parsed and used to query against dynamoDB using boto3 resource and create a .csv file.
- Finally, the created .csv file (which has the same file name as the corresponding .mp4 video file name) is written into the output S3 bucket through the *writeOutputToS3()* function.

Requirements.txt - Contains all the packages required to run the code.

The instructions on how to install and run the application are already specified in the readMe file.

It is as follows:

- ★ Download and install AWS CLI

  https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

- ★ Go to terminal, and run "aws configure", then type in the AWS_ACCESS_KEY, AWS_SECRET_KEY & REGION (not present in the README.md file for security purposes)

- ★ AWS_REGION = "us-east-1"

  aws_access_key_id = "AKIAWVOECYJ5VNG4XKTN"

  aws_secret_access_key = "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"

- ★ Git Clone the repository and cd into it.

  https://github.com/deepaknayani22/CSE546_Project2.git

  After cloning the repo, make sure to change the access keys to the relevant values.

- ★ Make the following changes (if using the TAs code)
  - In Dockerfile, add a line "COPY encoding /home/app/"
  - In requirements.txt, change ffmpeg to python-ffmpeg
- ★ Download and install Docker and Docker Desktop for deploying the image.
  - Run the following commands:
    - aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 458362110587.dkr.ecr.us-east-1.amazonaws.com
    - docker build -t proj2-docker-lambda .
    - docker tag proj2-docker-lambda:latest 458362110587.dkr.ecr.us-east-1.amazonaws.com/proj2-docker-lambda:latest
    - docker push 458362110587.dkr.ecr.us-east-1.amazonaws.com/proj2-docker-lambda:latest
- ★ Run python3 uploadDB.py (should not be required as the table is already populated)
- ★ Run python3 workload.py to upload the videos in the test folder into the S3 bucket.

Individual Contribution

Karthik Aravapalli (ASU ID: 1225611998)

Design:

In the design process all my team members collaborated with each other to decide the final design to complete all the tasks within 7 minutes so, the below architecture of our design is the AWS platform as a service using the amazon simple storage buckets, Lambda function, DynamoDB and ECR.

Implementation:

- S3 bucket set up:
  - ➢ We need two S3 buckets to finish our project so, I created two s3 buckets one is for input which is used to store the videos to process the face recognition to determine the name of the appeared person using the lambda function to automate the process and the second one is used to store the results which are the academic details for the each video's first frame recognized image. This bucket contains all the results of the input videos in csv format which is required for our project.
- Lambda function set up:
  - ➢ The Lambda function is used to process the input videos to face recognition python libraries to recognise the person name. First, we created a trigger for this lambda function to trigger the function once the input bucket is loaded with new input videos which are loaded using the workload generator. Also, the lambda function is set up to pull the ECR image which contains the docker file and all the python face recognition libraries which are face recognition. Once the videos are processed to find the first frame person name then, these names are searched in dynamoDB to find academic details to pull back to send output bucket to as in csv.
- DynamoDB set up:
  - ➢ I created the DynamoDB table 'ndk-proj2-table' in the Amazon Web Services database. I set the partition key of the table as 'name', this will make it simpler to look for students' information using their names as search criteria. I copied the given academic details to this table using the given 'student_data.json' file. So, this table will allow the lambda function to search the details to get the academic details for each person who is identified by python face recognition libraries.
- ECR set up:
  - ➢ A ECR repository is created to have the Docker file which contains the image which is used by Lambda function to pull the image to execute the handler and face recognition python codes and the input videos are divided into frames to take the first frame to determine the name.

Testing:

In the testing phase, To ensure that each function is operating as planned, the entire process has been divided into modular tests and verified the results. Tested the face recognition code initially in the local machine, then tested in the cloud using the lambda function and the dynamo db is tested whether it gives all the academic details for each name. Also, the S3 buckets are tested to ensure that all the videos are uploaded correctly by the workload generator and the output S3 should collect all the results.

Individual Contribution

Nikhil Chandra Nirukonda (ASU ID: 1223333995)

I participated in numerous phases of the project's development. I was eager to comprehend the project requirements and then formulate a plan for managing the task successfully. The "lambda" function was one of the primary components I designed and configured. The lambda function is essential to the project's functionality since it was used to retrieve images from the S3 bucket. It also contained the image evaluation code logic. It also includes the logic to get records from DynamoDB that contain information about the human subjects on which the facial recognition model was pretrained. Name, graduation year, key, and major are the various properties of the DynamoDB table's elements.

To accomplish this, I broke the assignment into sections and worked on them separately. Referring to the resources supplied by the lecturer and TAs, as well as the extensive documentation, was beneficial when confronting any problems. Docker must be installed on the local computer. In the case of Windows, we download "Docker Desktop" software, which enables us to create and run docker images. In installing docker, I setup the local wsl settings and proxies to ensure that we had access to the produced images. Docker may generate containers from predetermined image files. A background thread is used to invoke a lambda function in this project. Depending on the underlying platform, Lambda functions can take various forms, including Python, Node.js, Docker containers, etc. I developed two distinct lambdas, one for evaluating facial recognition models and the other for retrieving entries from DynamoDB based on the evaluation outcome.

To reduce the communication time between the two lambda functions, it was subsequently decided to use a single lambda. As the lambda function is provisioned to be able to run a docker container with complex calculation that would normally not be executed within a single script, such computation is now computable. A docker image containing the pre-trained model and evaluation script is built. The image is posted to the ECR repository, which is subsequently associated with lambda. Every lambda instance that is produced based on a docker image is accompanied by a lambda container. The docker image lambda handler is tasked with retrieving image and bucket information from the event and downloading the image from the S3 bucket. The facial recognition model examines the downloaded image in order to categorize the human subject into the predefined labels. The result is stored as a key in the DynamoDB alongside other information about the individual. These details are retrieved from DynamoDB and returned as the response to the REST API request that invoked the lambda function.

The Docker image had comprehensive setup instructions that could be used to construct containers with Dockerfile. The docker can be executed in either a single-stage or multi-stage configuration. After consulting with my team, we decided to implement a multi-step arrangement in our situation. With a multistage system, artifacts from a single image might be used selectively in multiple stages, saving a substantial amount of time. This allowed for the reuse of photos in several steps. The primary benefit of this was the reduction in time and complexity of the work. When a docker container is built, the entry points are often the execution of a file or script containing all the instructions to be executed. The Lambda function was integrated with Docker such that the image created based on the Docker file instructions was pushed to the ECR. Docker is a very practical technique to use a platform or machine that has its prerequisites met and can be executed practically immediately.

This provided me with an excellent opportunity to review the fundamentals of the Amazon lambda function and simultaneously apply those abilities to the development of a project. My future objectives are aligned with this domain, and this project has become increasingly significant to me. Not only did it expand my technical knowledge, but the teamwork and experience laid the framework for future prospects in the tech industry. Experience with a variety of cloud resources, such as Lambda functions, DynamoDB, S3, and other resources, will improve my performance as a developer when I am responsible for executing any activities as part of DevOps.