CSE 546 — Project2 Report

Hemendu Roy
Radhika Ganapathy
Yihui ZENG

1. Problem statement

In this project, we will be building a distributed application that utilizes PaaS services and IoT devices to perform real-time face recognition on real videos recorded by the device. Specifically, we will develop this application using Amazon's Lambda based PaaS and Raspberry Pi based IoT. Lambda is the first and the most widely used Function as a Service PaaS. Raspberry Pi is the most widely used IoT development platform. This project is crucial as it will provide us with experience in developing real-world, IoT data driven cloud applications like help finding missing people/criminals, improved medical treatment etc. to name a few.

1.1 Architecture

We have built a distributed application consisting of the Raspberry Pi and various cloud services provided by AWS. The Raspberry Pi records the videos using its attached camera. The cloud then performs face recognition on the collected videos and looks up the recognized students in the stored database. Finally, it returns the relevant academic information of each recognized student back to the user on the Raspberry Pi.

- Amazon Lambda – It is an event-driven, serverless computing platform provided by Amazon as a part of Amazon Web Services. It is a computing service that runs code in response to events and automatically manages the computing resources required by that code. In our application, the lambda function is triggered using an API request which then goes on to extracting the frame from the recorded video and triggers the face recognition model. This is followed by extraction of the recognized face's details from the database.

- DynamoDB - It is a fully managed proprietary NoSQL database service that supports key–value and document data structures. In our application we utilize the services of DynamoDB to store the details of the students in our data set. The details include the student's academic information like name, major and year. Once the face is detected, the results for that student are retrieved from this database.

- S3 - It is a service offered by Amazon Web Services that provides object storage through a web service interface. In our application, the storage services are used to store the videos recorded by the pi.

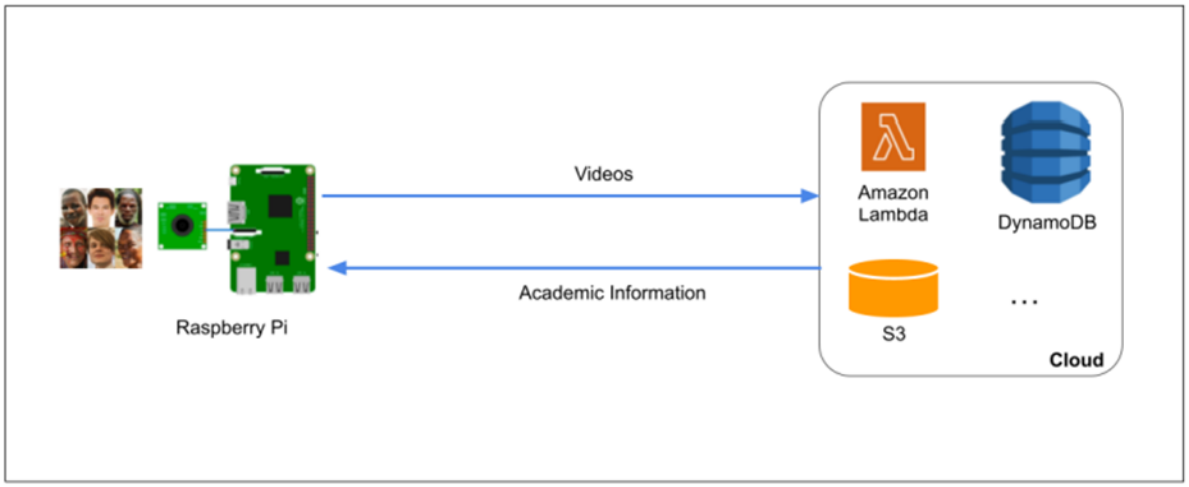


Fig. 1

1.2 Concurrency and Latency

The concurrency is achieved using worker threads. We have one thread assigned to record the video on the Pi and 20 additional worker threads that will grab tasks from the work queue and start processing the assigned task. Each worker thread will upload the video to S3 and then request the HTTP endpoint on lambda. Once it receives the result from lambda, it will return the result.

The latency calculation for our application starts when a worker thread is assigned with a task from the worker queue and ends with getting the result from Lambda.

2. Testing and evaluation

In this section, we discuss in detail the testing and evaluation results.

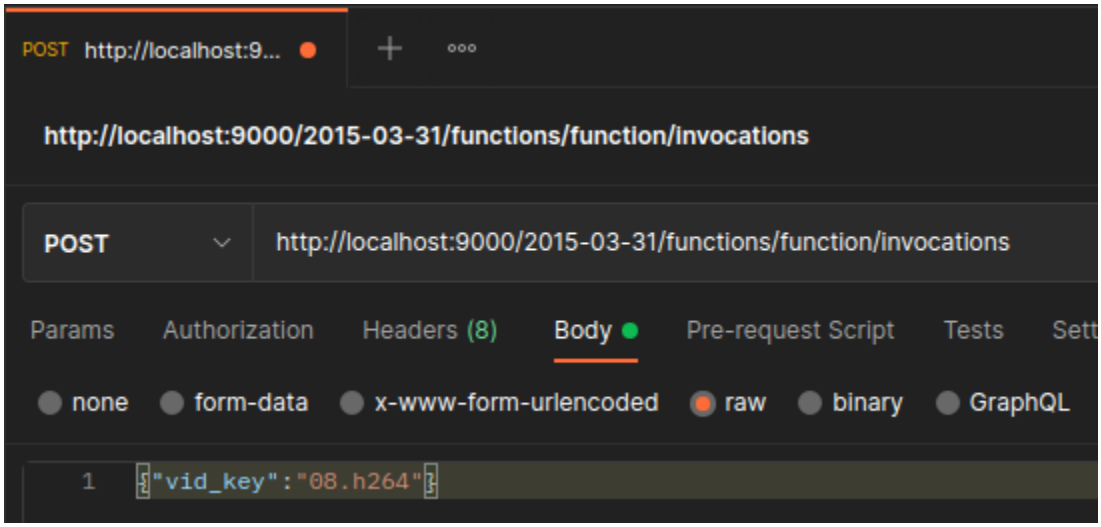
2.1 Creating the Docker Image

We wrote the *app.py* script with a handler function as the entry point to the Docker Image. This script would receive the name of the video stored in S3, download it, extract a frame, use the model to recognise a face, query DynamoDB using the recognition result and finally, return the correct data back to the requester.

Initially, the docker image was tested and run locally.

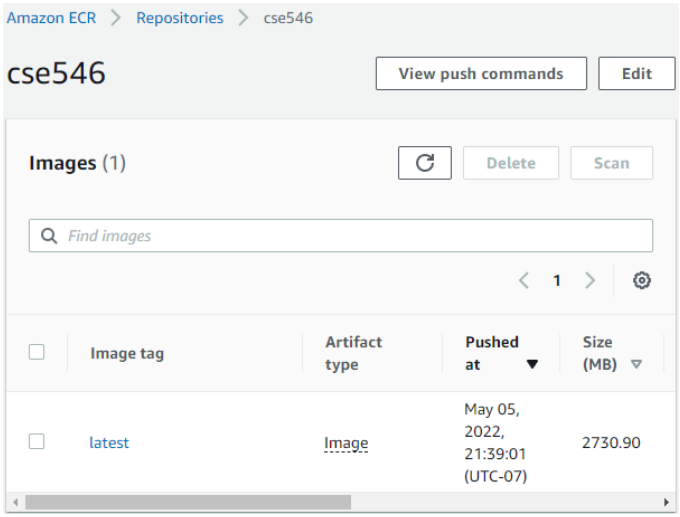
```
roy@ubuntu:~/CSE546-project2/lambda$ sudo docker run -p 9000:8080 cse
546
[sudo] password for roy:
06 May 2022 22:44:42,897 [INFO] (rapid) exec '/var/runtime/bootstrap'
```

We stored arbitrary 0.5 second long videos in S3 and sent their names to the docker image running on localhost using Postman.

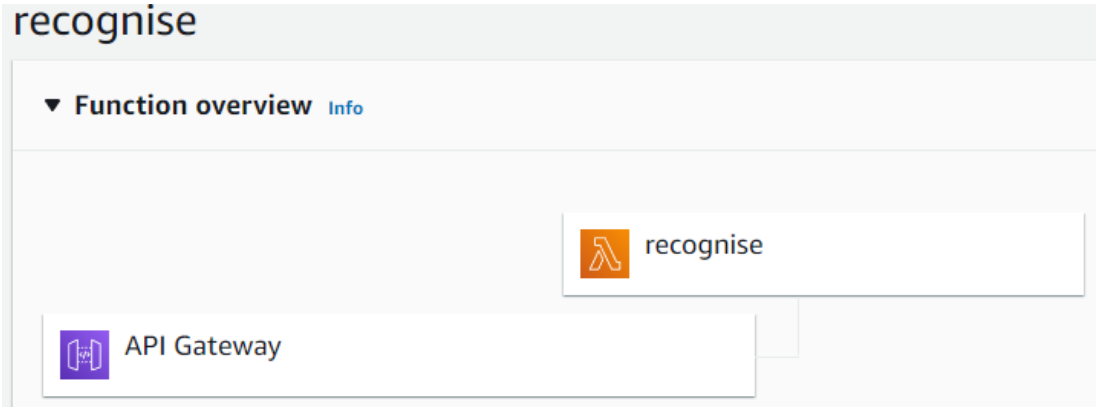


2.2 Creating the Lambda Function

Once we received an expected response, we pushed this image to the Amazon Elastic Container Registry (ECR).

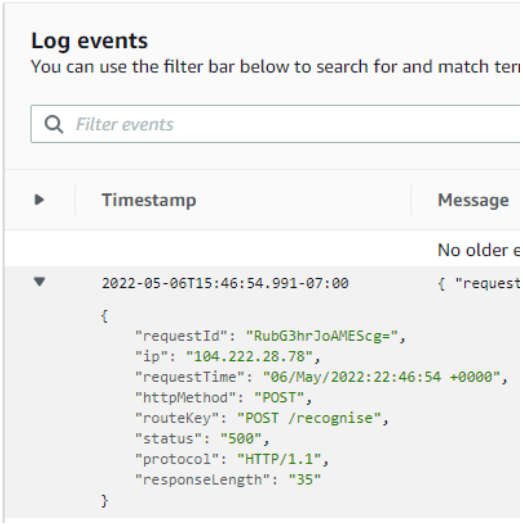


We then configured a Lambda function with an API Gateway trigger to use this image from ECR.



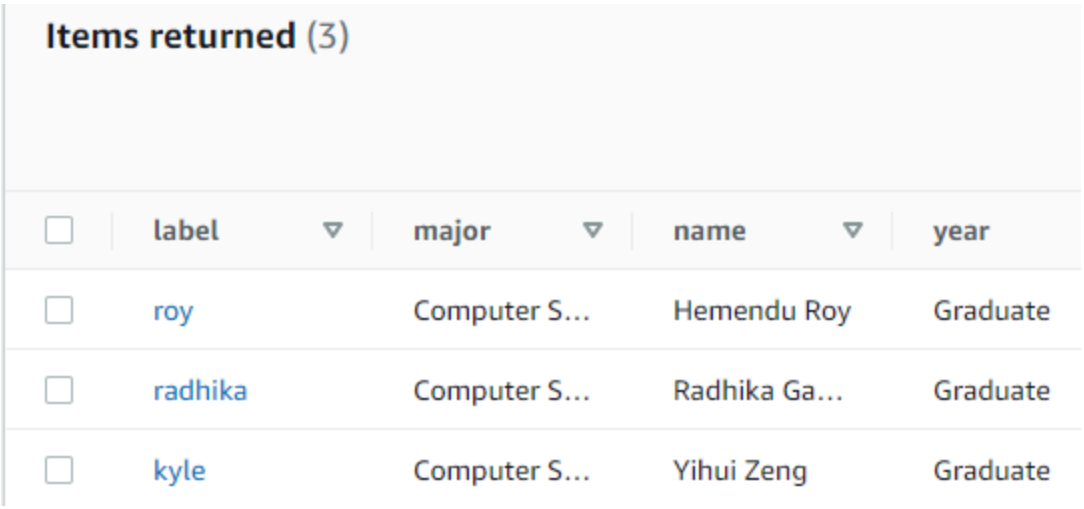
2.3 Configuring the API Gateway

Once again, to verify end-to-end functionality, we used Postman to test the API Gateway. To diagnose issues such as 500 Internal Server Errors, we enabled Cloudwatch logging and inspected the concerned log group.



2.4 Configuring DynamoDB

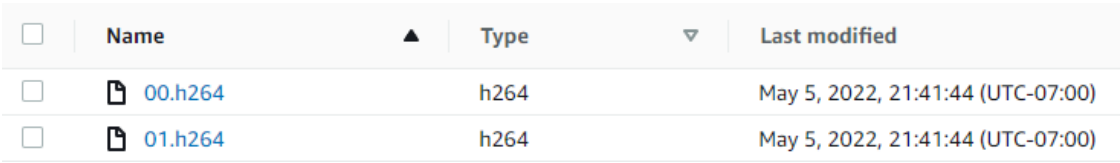
We stored our individual records in a table in DynamoDB as shown below.



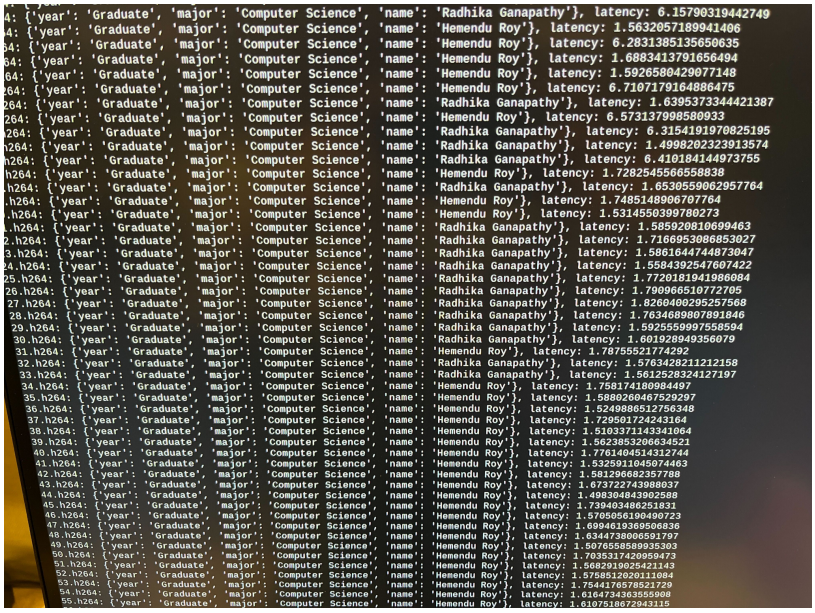
2.5 Setting up the Raspberry Pi

After verifying complete functionality with a stored video, we began work on the Raspberry Pi. We wrote a script, *record_videos.py* to record and store 0.5 second long videos in S3 in a multi-threaded fashion.

We then verified this by inspecting the contents of the S3 bucket using the AWS S3 console.



The script was finally made to send a POST request to the API Gateway endpoint of the Lambda function with the name of the video in S3. The Lambda function would then process it and return a JSON containing data of the recognized individual from DynamoDB. Finally, this data along with overall latency is displayed on the terminal.



We now had a functioning serverless facial recognition system using a Raspberry Pi camera.

3. Code

3.1 Functionality of Programs

- *config.py*

This file stores all the AWS credentials needed to run our application.

- *raspberry/record_videos.py*

This is the main script used to run the Raspberry Pi. The script runs the Pi to record a video continuously, and then assigns each 0.5 second video to a worker thread through a worker queue.

- *lambda/app.py*

This stores the code that runs inside the lambda function. The frame for face recognition is extracted using the video stored in S3 in this script. After extracting the frame, the image is run through the image recognition model and the result obtained is used to retrieve student information from the database.

- *lambda/eval_face_recogniton.py*

This is the provided script that invokes a pre-trained facial recognition model to recognise a frame supplied using the "--img_path" argument. In our implementation, we modified the module to accept an image object directly instead of using the command line.

3.2 Installation Instructions

To use this repository, clone it into the home directory of the raspberry Pi. We will use the *record_videos.py* script to interact with the Lambda function. The *config.py* file must be modified with AWS credentials, the API Gateway endpoint and the name of the DynamoDB table.

To set up the Lambda function, we must build a docker image using the provided *Dockerfile*, tag it, and push it to a repository in the Amazon Elastic Container Registry. The Lambda function can then be configured to use this image through the AWS console.

The API Gateway must then be added as a trigger, with default settings. A table must be created in DynamoDB and entries may be added manually using the web interface.

Now that the entire cloud infrastructure has been setup, we may simply run the script on the Raspberry Pi using the command *python record_videos.py* and point the camera towards a face to obtain results in a terminal window.

4. Individual contributions (optional)

Hemendu Roy

I wrote the app.py script to perform the following tasks,

1. Receive an HTTP POST request
2. Extract the name of a video stored in S3 from the JSON body
3. Extract a single frame from the video -> Done by Radhika
4. Use a pre-trained facial recognition model to recognise the face in the image
5. Use the name of the recognised individual to retrieve data from DynamoDB
6. Send this data back to the source of the HTTP request.

Additionally, I created a docker image and pushed it to AWS ECR and configured a lambda function to use it. I added an API Gateway trigger to the lambda function. I tested these components thoroughly using local docker containers, Cloudwatch and Postman. I also aided in training the facial recognition model with our datasets.

Radhika Ganapathy

I built a pipeline to handle the video recorded by the Raspberry Pi.

1. Once the 0.5 seconds videos are recorded, it is pushed to S3.
2. Trigger Lambda using HTTP requests.
3. Built and tested lambda function to extract frames from o.5 second videos.
4. Built an additional pipeline to extract frames in the pi for comparison.
5. Send the image to the face recognition model.

In addition to this I served as an aid in training the facial recognition model with facial datasets. I worked on writing and integrating the final report for Project 2.