CSE 546 — Project 2 Report

Group 11 Team Members:
Akash Kant (akant1)
Ayush Kalani (akalani2)
Nakul Vaidya (nvaidya7)

1. Problem statement

In the second project, we are leveraging our learning and understanding of Project 1 to Identify and solve a real-world problem. We will build a distributed application that utilizes PaaS services and IoT devices to perform real-time face recognition on real videos recorded by the devices. Specifically, we will develop this application using Amazon Lambda-based PaaS and Raspberry Pi-based IoT. Lambdas are the first and the most widely used Function as a Service PaaS. Raspberry Pi is the most widely used IoT development platform. This project will provide you with experience in developing real-world, IoT data-driven cloud applications. The second major objective of our project is to familiarize us with the functionality of cloud-based services (AWS in this case). As promised this project did help provide us with the experience in developing real-world, IoT data-driven cloud applications.

Familiarity with FaaS and other services like DynamoDB, Docker, ECR, and S3 can be used by us in varied projects and personal/professional work. We feel having done this project makes us capable to challenge our project head-on and solve and put forth a quality solution. Learning about IAM and giving access to users and defining the roles attached to each user, the access control list attached to each functionality and control that is provided by a cloud service like AWS, GCP, or Azure is nothing short of an admirable and self-sufficient solution. Moving on to the high-level problem statement as stated in the Project Document.

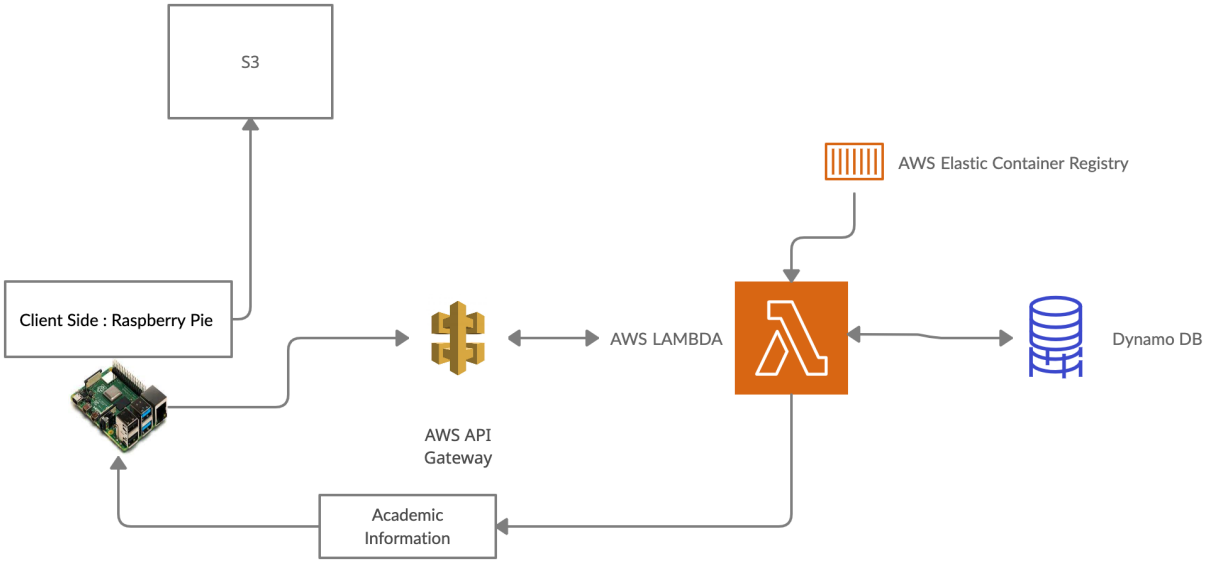
“The Raspberry Pi records the videos using its attached camera. The cloud performs face recognition on the collected videos, looks up the recognized

students in the database and returns the relevant academic information of each recognized student back to the user."

Our program on the PI should continuously record videos and send them to the cloud. The program should receive the output from the cloud for each recognized person in the videos and output the person's name and academic information in the following format: The #NUMBER person recognized: NAME, MAJOR, YEAR. There are additional performance specifications for this project regarding the accuracy and latency of our service.

2. Design and implementation

2.1 Architecture



2.2 Components

Client Side : Raspberry Pi

We used Raspberry Pi to learn programming skills, build hardware projects, do home automation, implement Kubernetes clusters and Edge computing, and even

use them in industrial applications. The Raspberry Pi is a very cheap computer that runs Linux, but it also provides a set of GPIO (general purpose input/output) pins, allowing you to control electronic components for physical computing and explore the Internet of Things (IoT).

Pi is recording videos indefinitely and uploading them to the cloud. For each recognized individual in the videos, our application is getting the cloud input and outputting the person's name and academic information in the following format -

{"name": "akash_kant", "major": "computer_science", "year": "graduate"}

Middleware : AWS API Gateway

The Amazon API Gateway service is a fully managed service that allows developers to easily construct, publish, maintain, monitor, and protect APIs at any size. APIs allow apps to access data, business logic, and functionality from your backend services through a "front door." RESTful APIs and WebSocket APIs may be created using API Gateway to allow real-time two-way communication applications. Web applications, as well as containerized and serverless workloads, are supported by API Gateway.

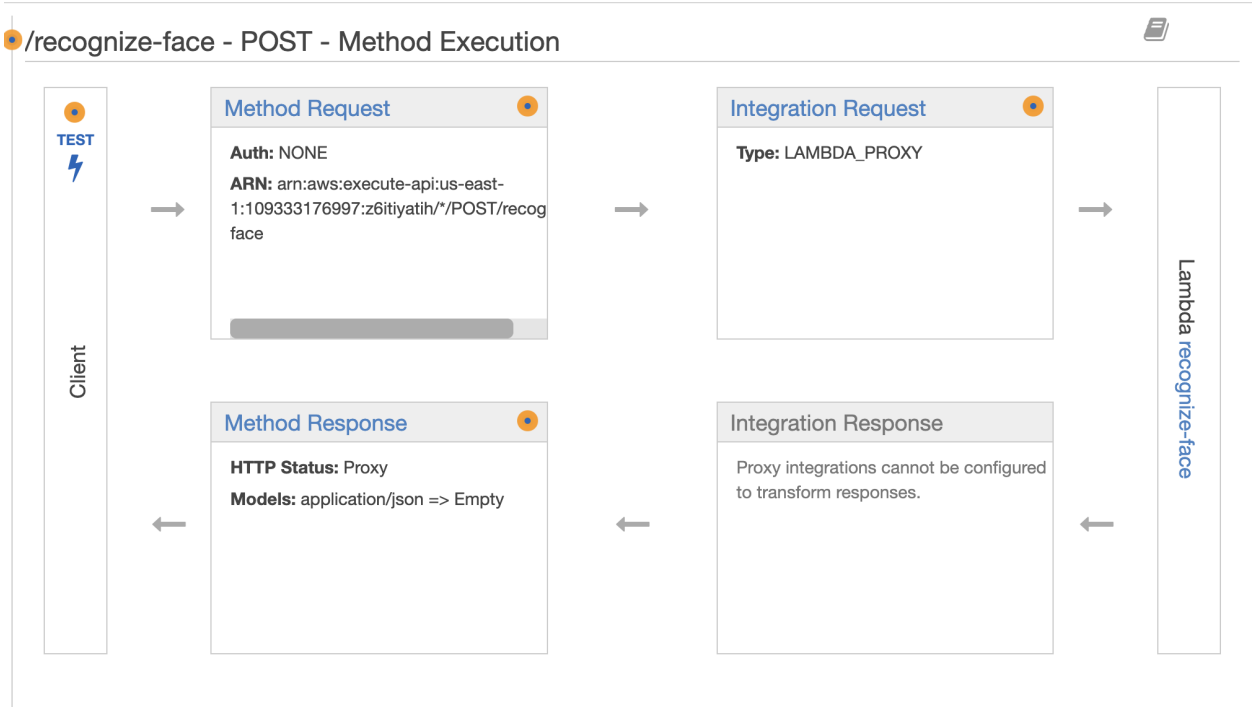
API Gateway takes care of everything from traffic management to CORS support to permission and access control to throttling, monitoring, and API version management for hundreds of thousands of concurrent API requests. There are no minimum fees or initial charges with API Gateway. You pay for the API calls you get and the data you send out, and you may save money by using the API Gateway tiered pricing model as your API usage grows.

Advantages of using api gateway -

1. Cost savings at scale
2. Efficient API development
3. Easy monitoring
4. Performance at any scale

We are encoding our image into base64 and sending it as a POST request on our REST API hosted at AWS API gateway. The api gateway triggers lambda and sends

this base64 encoded string to aws lambda. The flow of requests can be visualized in the diagram shown below.



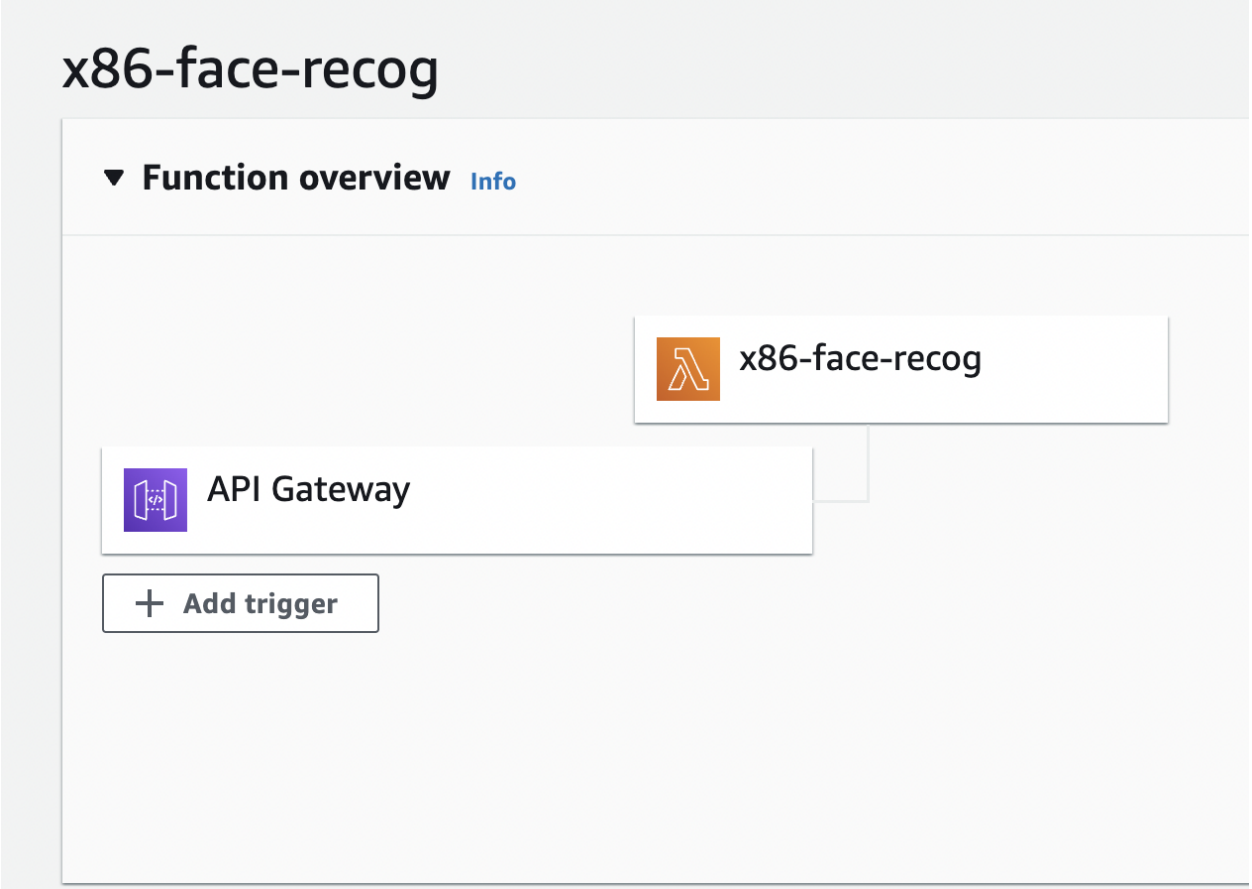
Serverless Computing : AWS Lambda

AWS Lambda is a serverless, event-driven computing solution that allows you to run code for almost any application or backend service without having to set up or manage servers. There are over 200 AWS services and SaaS apps to trigger Lambda, and we only pay for what you use.

Advantages of using AWS Lambda in our project -

1. Minimized Cost
2. No servers to manage, Less Operational Management
3. Continuous scaling
4. Millisecond metering
5. Loosely Coupled
6. Quicker Iterative Development

As shown in the diagram below we are triggering our lambda function via api gateway, and we are sending requests from raspberry pi to our API gateway using POST REST API and sending base64 encoded images there.



Lambda Creation:

Our lambda function is created from the docker container that we have created using the dockerfile. We created a docker image in our local and then pushed the same to Amazon Elastic Container Registry (Amazon ECR) using docker push commands.

Then we have created our lambda function to start from this docker image pushed to ECR.

Database : DynamoDB

Amazon DynamoDB is a serverless, fully managed key-value NoSQL database built to run high-performance applications at any size. Built-in security, continuous backups, automatic multi-Region replication, in-memory cache, and data export tools are all features of DynamoDB.

We query a dynamo db table by passing the name to the database and then the db returns the student data by filtering/querying the table.

Schema Definition -

Table Name - group11_students_table

Partition Key - name(string)

Columns - id, name, major, year

Object Storage : S3

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. S3 is designed for 99.999999999% (11 9's) of durability, and stores data for millions of customers all around the world. It is scalable and has virtually limitless storage. In this project we have made one bucket called as *face-recognition-videos*, this bucket is used to persist videos sent via pi.

2.3 Concurrency and Latency

To attain concurrency and minimize the latency we extracted the frames from the video recorded by the pi camera while video is being recorded and sent it to AWS lambda for classification. We have used Async await to attain the parallel processing for frames recorded every 0.5 second and once the complete video is being recorded we store it in S3 server.

After the function catches up to demand, requests fade away, and unneeded instances of the function are terminated after a period of inactivity. Unused

instances are frozen and do not incur any costs while they wait for requests. The time it takes to load and set up your code has an influence on the first request serviced by each instance as the function scales up. The impact on average and percentile latency might be severe if the startup code takes a lengthy period.

Latency is the delay between when a face is captured by the camera and when the student's academic information is returned to the Pi in our program. There are several steps to this. The picture is taken first, followed by a request to the API gateway. The request is passed from the gateway to the lambda function, and the picture is recognized the following time. Following picture recognition, the details are retrieved from DynamoDB and sent to lambda, then api, and lastly to the edge (raspberry pi). Latency is the amount of time it takes to perform all of these procedures. The lambda function must be invoked and made active, therefore the latency will be considerable for the first few queries.

3. Testing and evaluation

3.1 Precursors

As a precursor to the demo, we have taken the images for the raspberry pi to train and validate the model. We have deployed the model onto lambda for the end-to-end hit via API gateway added to the lambda hosting the image.

1) The videos are sent from the Pi to the cloud continuously

2) All the received videos in the cloud are properly saved in S3

3) All the faces in the sent videos are recognized (every 0.5 second)

4) The face recognition results are correct. More than 60% of the face recognition results are correct.

5) The end-to-end latency (the time between when a face is recorded by the camera and when the student’s academic information is returned to the Pi) should be reasonably short.

3.2 Evaluation

1. Verified correctness of the returned information.

2. The requests are received by the Lambda function after each frame from the videos is extracted at a 0.5 second interval.
3. The lambda function was invoked 600 times in total.
4. Checked concurrency to be at least 3.
5. Substantiated autoscaling of Lambda functions in the Lambda Web UI.
6. Cross checked received videos in S3 bucket face-recognition-videos. AWS S3 bucket was empty before starting the app, and after it was filled with videos.
7. Validated academic information stored in DynamoDB.

3.3 Results

We have fulfilled all the requirements of the project as mentioned in project evaluation rubric.

4. Code

4.1 Implementation

Dockerfile

Used the boilerplate file and added our own functionalities such as copying the dependencies such as torch, torchvision and tensorflow. We also copied the model, checkpoint and the eval scripts to our docker container so that we can run inference on lambda.

handler.py

eval_face_recognition.py - File that accepts the image path and runs the inference on our trained model and return the results in the form of string

train_face_recognition.py

Used to train the model by supplying images and labels.

models/

It has the necessary files such as inception_resnet_v1.py, mtcnn.py etc that are libraries used for running the model and getting the testing results for face recognition.

checkpoints

Has the required input configuration parameters that are used to train the model. The mode.pth file has the weights and parameters that the model is trained on. This folder has these two files which are absolutely required for running the inference-labels.json model_vggface2_best.pth.

camera_photo.py

Used for capturing videos and images and sending them to S3 and posting via rest api to aws api gateway. Used to cut frames from the video and send it to AWS API gateway.

requirements.txt

Used for installing all the python libraries from pypi.

4.2 Running the project

Client Side : Raspberry pie

python3 send_data_to_api_gateway.py

5. Individual Contributions

Akash Kant

IAM, Roles, Users, and Access

In this project, I did set up my AWS as a root user, wherein I AM I attached the new users as my two other teammates and defined their roles for access to the different services provided by AWS. I also set the access control list and the rootkey access for connecting the different services in ECR and others.

Setting up PI, Model training, and photos

I had some previous experience with a Raspberry Pi B+ and this one being the 3rd version was relatively easy to use and access. Raspberry PI is a small but powerful tool. Its utility is numerous and the ease of use is as easy as they come. I took the photos via a python script to 30-click images on the keyboard, interrupt for all our teammates, then connect to our EC2(spun out of the AMI provided) and then use these photos to **“scp”** the images to EC2. Here I trained the model and checked out the file on my laptop for creating the docker images. I also installed Raspbian on the pie.

Docker, ECR, Lambdas, DynamoDB

The model is downloaded with all the files and its movement in a docker container is set. I have also tested the via rest call to lambda via the code and test facility given in AWS. Then I tested sending it as a Base64 image and retrieving it in Lambdas as a log on CloudWatch and also as an image loaded in s3 and saved in lambdas "/tmp/" folder as it can be written there. With all the moving parts figured out, I created the DockerFile with all the details and the handler code and deployed it to ECR to create a lambda from container images. This took numerous tries to finally get it correctly running. Then I also called the eval_face_recognition.py as a subprocess call to finally give the output and also return as a JSON to the final rest callee. This was the major chunk of the setup and this got me to look for different solutions. I also tackled the latency issues, as the initial latency from the cold start took around 8-9 seconds. I employed multiple ways to get our latency to where it is. I also set up the API gateway and also connected with S3 and dynamoDB. I also edited the lambdas session timeout time for it to allow me to get the initial output

PI, Parallel sending of Request, Frames cutting

The PI would record the 5 min (300 sec) video which translates to 600 rest calls and outputs required. I was also skeptical about the multiple threads that a PI can spin out keeping the latency in mind and its physical limits. I also tried the asyncio process for each frame to get output from the lambda.

Ayush Kalani

AWS Infrastructure Setup

I set up the AWS infrastructure account under our team organization and created a root key to be able to access ec2 instances. Hence, I was able to get administrator access to the AWS services.

I created the infrastructure on AWS to develop such a reliable and cost-effective system.

System Pipeline - Docker, ECR, Lambda, S3

I was responsible for system design and architecture and proposing the lambda and api gateway. I developed a system that is fault-tolerant and designed to isolate points of access to remote systems, services, stop cascading failure and enable resilience in the distributed systems where failure is inevitable. I troubleshot and debugged the issues while testing and running the application on AWS, and fixed bugs and issues as I went ahead.

Created the docker image by running the docker build commands and using the aws ecr sdk commands. And added the files using the COPY directive in the docker and pushed the image to AWS ECR using the AWS ECR CLI.

After my teammate created the model, I created the database data to hold the student academic information. Then I debugged sending it as a Base64 image and obtaining it as a CloudWatch log in Lambdas. I built a DockerFile with all the information and the handler code and deployed it to ECR to construct a lambda from container images. I also created a bucket called face-recognition-videos on S3 that will store all the videos.

Testing

I helped in the AWS code and test feature to test the through rest call to lambda, and participated extensively in troubleshooting and debugging the project at various stages.

Miscellaneous

In addition, I worked with my other team members to properly integrate their various system components into an uniform pipeline on AWS. We worked together to install our application on AWS, verify the findings, show the TA the demo, and write the results.

Nakul Vaidya

Setting up Raspberry Pi

Plugging in all the components together and flashing with the provided sd card with Raspberry Pi OS. Wrote the camera code to capture 160×160 RGB images to train the provided model.

Pi camera client

wrote the code for capturing video and extracting RGB frames on the pie, used the pi camera module in python to complete the above tasks. Sent the extracted frames every 0.5 secs to lambda and delivered the final 5min video to Amazon S3. Used Async await to achieve concurrency in python. Handled the response provided by the AWS lambda and provided it to display the result along with the overall latency of the request.

Miscellaneous

collaborated with other teammates towards debugging the code and contributed to writing the final report majorly the part related to the pi camera client and achieving the Concurrency and Latency.