CSE 546 — Project Report

Janam Devang Vaidya (1222377325)

Rhythm Rakeshkumar Patel (1225583281)

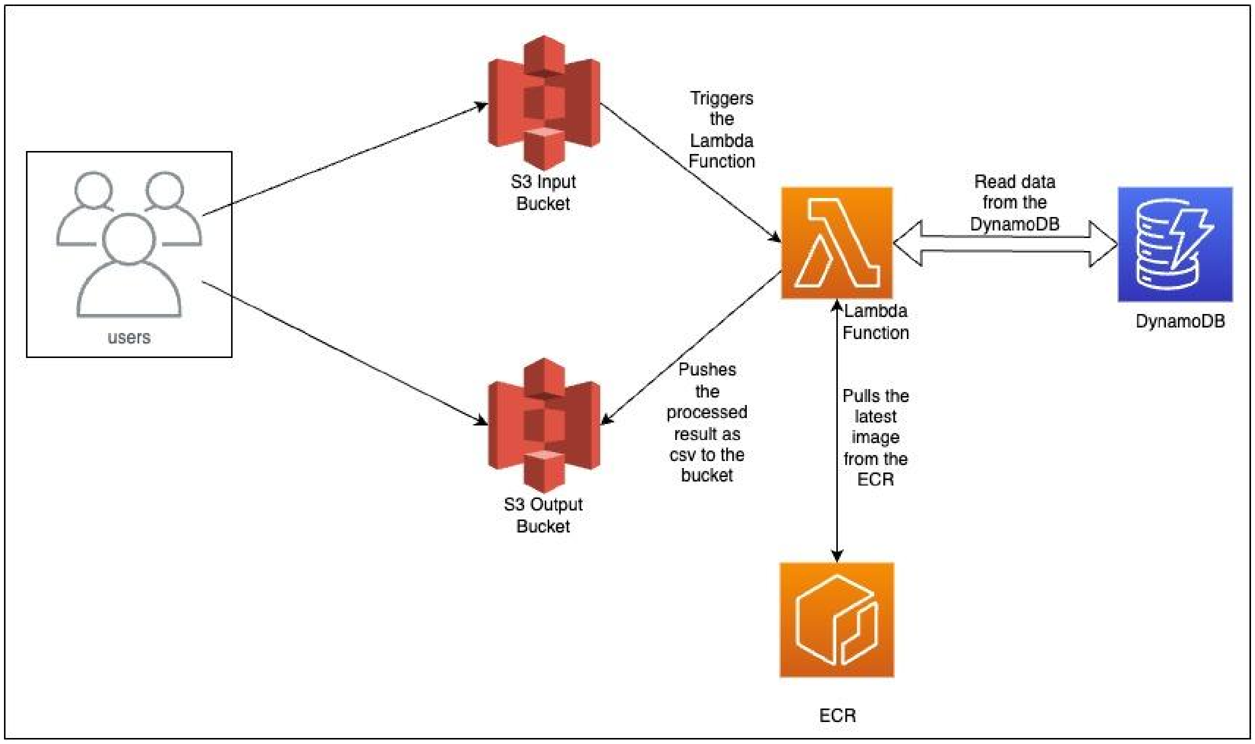
Rishabh Krishnakant Pandat (1222158509)

1. Problem statement

In this project, our aim is to build an elastic web application that automatically scales in and scales out on-demand and cost-effectively using PaaS cloud services. The PaaS resources, AWS Lambda and other resources used in building this application are from Amazon Web Services (AWS). AWS Lambda is the first and currently the most widely used function-based serverless computing service. Our cloud app will implement a smart classroom assistant for educators. This assistant takes videos from the user's classroom, performs face recognition on the collected videos, looks up the recognized students in the database, and returns the relevant academic information of each student back to the user.

2. Design and implementation

2.1 Architecture


users
S3 Input Bucket
Triggers the Lambda Function
Lambda Function
Read data from the DynamoDB
DynamoDB
Pushes the processed result as csv to the bucket
S3 Output Bucket
Pulls the latest image from the ECR
ECR

Components:

1. S3: There are two S3 buckets, one for input and another for output. The input bucket has an event notification set to trigger the lambda function whenever a new object is pushed to it.
2. ECR: We have deployed the lambda function using ECR. Hence whenever we want to update the code for lambda function we upload the build docker image to our Elastic Container Registry and Lambda function pulls the latest image.
3. Lambda Function: The lambda function takes the event object for s3 notification and reads the video file splitting it into frames and finding faces from it. It also connects to our dynamoDB which stores the additional information and process the output into a csv file to upload it to the S3 output bucket
4. DynamoDB: It stores the additional information which we query and search through our lambda function.

2.2 Autoscaling

An instance of the function is created by AWS Lambda and its handler method is run by it for processing the event when we invoke our function for the first time. When we receive some response from the function, it stays active and waits to process additional events and on invoking the function again while the processing of the first event is still going on, another instance is created by lambda, and two events are concurrently processed by the function. New instances are created accordingly as Lambda routes them to available instances upon arrival of more events. Unused instances are stopped by Lambda when there is decrease in the number of requests which frees up scaling capacity for other functions.

2.3 Member Tasks

Janam:

1. Created read and upload flows to and from S3 buckets.
2. Created the flow to query dynamoDB and generate csv for upload.
3. Worked on report and documentation.

Rhythm:

1. Wrote a function to read faces from video and find whose image it is.
2. Created the flow for docker image to push to ECR and pull directly to lambda function.
3. Worked on report and documentation.

Rishabh:

1. Responsible for setting up the necessary AWS infrastructure like S3, ECR, Lambda, DynamoDB instances.
2. Populated the dynamoDB.
3. Worked on report and documentation.

3. Testing and evaluation

Testing was done mainly in two ways:

1. Integration test: We tested end to end all the services are connected and working as expected by inserting some dummy and known data. After this was done we tested with actual test cases given by the TA
2. Final Test using Workload generator:We used the 100 videos given by the TA. During testing, we also checked whether the application can take more than one video at a given time as an input. Looking at the number of csv stored in the S3 output bucket is the same as the number of videos the user had sent.

4. Code

Dockerfile: This helps in building the container image with correct volumes and needed files. It is a very critical code as this ensures image creation for upload to ECR.

Handler.py: The functions inside this file work progressively in the following manner.

1. First we read the event that triggered the function.
2. We download the video file from the S3 bucket that triggered the function(input bucket here) and save in the tmp folder provided by lambda.
3. We read the video and split it into frames and check whether the frame contains any face or not using the face_recognition library.
4. After this we compare the faces found in a video to the known faces as provided in the encoding file which is read by the open_encoding() function.
5. We then use the name of the matched face to search for more information in DynamoDB and generate a csv to push to the S3 output bucket.

Workload.py: This tests our code in the lambda function by brute forcing test cases onto it.

To install and run our program:

Since it can vary from machine to machine on how to run we suggest using a dockerised environment to run the program.

1. Install Docker
2. Build the docker image from the given docker file
3. Run the built docker image
4. You can invoke the handler.py handler function by executing into the container.

To run it completely using the lambda function.

1. Build the docker image
2. Push it to the ECR
3. Create the lambda function using the ECR image URI.
4. Set the input S3 bucket as the trigger for this lambda function.
5. Push any video files to the input bucket and you will be able to see the processed results in csv format in the output bucket.