CSE 546 — Project Report 3

Student Names: Madhav Rajesh

December 1, 2023

1 Problem Statement

In the third project, the focus is on transitioning the elastic application developed in the second project capable of automatic scaling in response to demand and cost-effectiveness—into a Hybrid cloud environment. This entails utilizing resources from Amazon Web Services (AWS), Ceph (for object storage), and OpenFaaS (as a private cloud). AWS, being the most widely used Cloud provider, provides an array of compute, storage, and messaging services. OpenFaaS, on the other hand, enables developers to deploy event-driven functions and microservices to Kubernetes without repetitive, boilerplate coding. By packaging code or an existing binary in an OCI-compatible image, the application gains a highly scalable endpoint with auto-scaling capabilities and detailed metrics. Despite the hybrid cloud environment, the objective remains to provide users with a meaningful cloud service. The technologies and techniques acquired in this project are intended to be versatile and applicable for future endeavors in building various cloud services.

2 Design and Implementation

2.1 Architecture

The architecture of our smart classroom assistant is designed to be scalable, responsive, and cost-effective, utilizing AWS cloud services. The system is composed of the following components:

- **S3 Buckets**: Two buckets are used; the input bucket for storing uploaded videos and the output bucket for storing processed academic information in CSV format.
- **AWS Lambda**: The Lambda function is the intermediary unit that is triggered when new videos are uploaded to the input bucket. It sends the video information to OpenPaaS
- **S3 Buckets**: It uses `ffmpeg` to extract frames and the `face_recognition` library to recognize student faces.

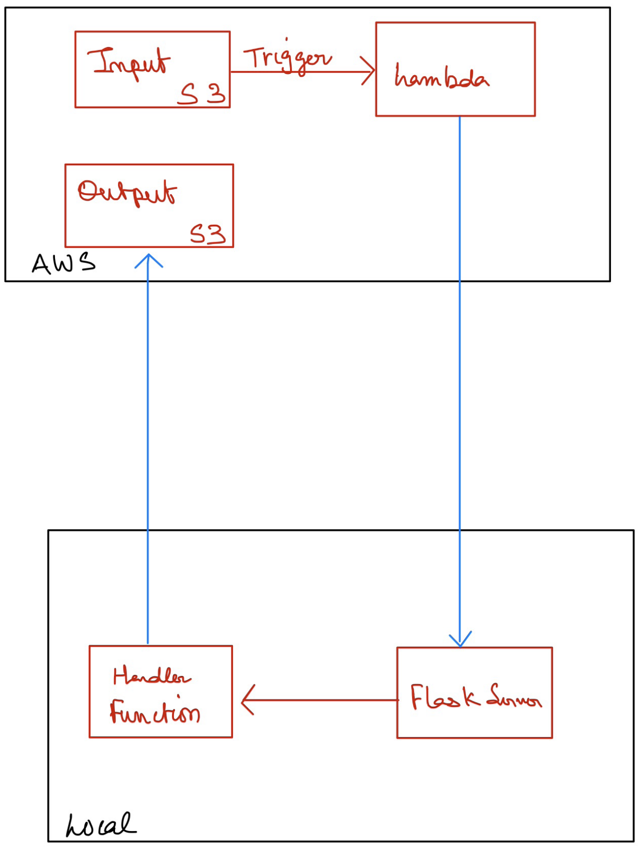

Figure 1: System Architecture

- **Amazon DynamoDB**: Stores student academic information, which the Lambda function queries to retrieve data based on the recognized faces.

The process flow:

1. Videos are uploaded to the input S3 bucket.
2. This triggers the Lambda function.
3. Lambda send video information to Flask Server.
4. Handler Function extracts frames and recognizes faces.
5. Recognized faces are matched with student information in DynamoDB.
6. The academic information of recognized students is stored as CSV files in the output S3 bucket.

2.2 Member Tasks

- **Madhav Rajesh**: Frame Selection, Facial Recognition, Docker Setup, AWS Setup, Documentation, OpenPaaS Setup and Implementation, Ceph Setup.
- **Chen Ni Lin**: Lambda Conversion.
- **Joseph Hand:** OpenFaaS Setup and Implementation, Ceph and Microceph.

3 Testing and Evaluation

We conducted thorough testing using the `workload.py` generator, ensuring that:

- The output of the face recognition matches the expected results.
- The S3 buckets contain the correct input and output data.
- All requests are processed within a benchmark time.

Evaluation Results:

Our implementation processed 10 requests in under 2 minutes. We noted the following:

- **Accuracy:** The face recognition output was 86 % accurate against the provided mapping file.
- **Performance:** On average, the system processed individual requests within 2.7 seconds, well within acceptable limits.

4 Code

1. **extract_frame_from_video(video_file_path, output_path)**
   This function takes a video file and extracts the first frame from it using the `ffmpeg` command-line utility. It ensures the output directory exists, constructs the output image file path, and then uses `ffmpeg` to select the first frame (`eq(n\,0)`) and save it as a `.jpeg` file.
2. **recognize_faces_in_frame(frame_path, data)**
   This function loads an image file, detects faces within it, and tries to recognize them by comparing them with pre-saved face encodings provided in the `data` parameter. It calls `get_student_details(name)` to fetch additional information about the recognized individual, assuming `name` is a unique identifier for student data within a DynamoDB table.
3. **open_encoding(filename)**
   A helper function to load face encodings from a file using the `pickle` module, which allows you to serialize and deserialize Python object structures.

4. **get_student_details(name)**
   This function queries a DynamoDB table named `student_data` to find student details based on a provided name.

5. **face_recognition_handler(event, context)**
   This is the main handler of the Lambda function. It gets triggered by an event, which in this context is expected to be a new video file uploaded to an S3 bucket. The function then:

   - Downloads the video file from the S3 bucket.
   - Extracts the first frame from the video.
   - Recognizes faces in the frame using pre-saved encodings.
   - Converts recognized details to CSV format.
   - Uploads the CSV file to another S3 bucket for output.

Installation & Execution

Installation Steps:

1. **Lambda Function Setup:**

   - Create a new Lambda function in the AWS Management Console.
   - Upload the zip file as the deployment package.
   - Set the handler information based on the provided code (e.g., filename.face_recognition_handler).
   - Adjust the timeout and memory allocation based on expected execution time and resource usage.

2. **Environment Variables:**

   - Set up environment variables such as `FUNCTION_DIR` if needed.

3. **IAM Role Configuration:**

   - Ensure that the Lambda function's execution role has permissions to read from the input S3 bucket, write to the output S3 bucket, and access the necessary DynamoDB tables.

4. **S3 Trigger Setup:**

   - Configure an S3 trigger on the Lambda function to run whenever a new video file is uploaded to the `INPUT_BUCKET_NAME`.

5. **DynamoDB Setup:**

   - Make sure there is a DynamoDB table with the name `student_data` and it has the necessary data schema to store student information.

Running the Program:

The Lambda function will run automatically whenever a video file is uploaded to the `INPUT_BUCKET_NAME` S3 bucket. You can monitor the execution logs in AWS CloudWatch to ensure it's running as expected. The results (CSV files with recognized faces and their details) will be uploaded to the `OUTPUT_BUCKET_NAME` S3 bucket.