CSE 546 — Project Report

Atul Prakash (1225542214)

Abhi Teja Veresi (1225506321)

Kedar Sai Nadh Reddy Kanchi (122527164)

1. Problem statement

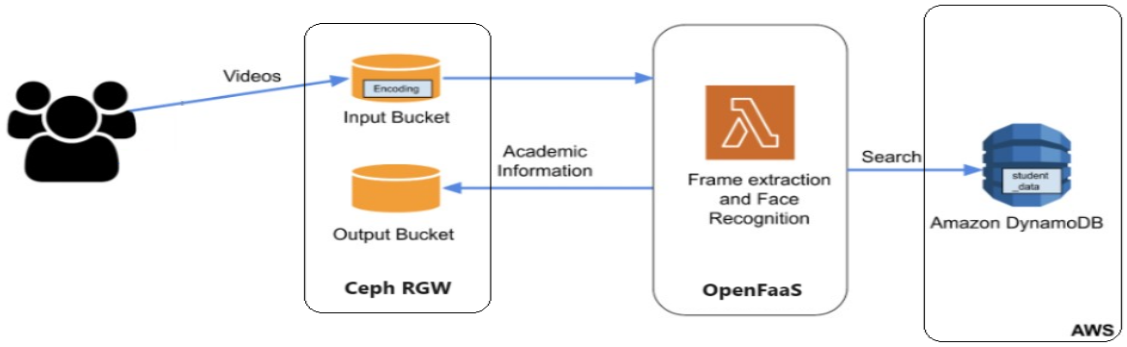
This project's objective is to create an adaptable cloud application within a hybrid cloud setting, leveraging OpenFaaS for event-driven trigger functions that autonomously scale according to demand. Additionally, AWS services will be employed to function as a smart classroom assistant for educators, akin to Project 2. The application's functionality will allow users to upload classroom videos to an S3 input bucket. Following the video upload, the system will automatically initiate an OpenFaaS event-driven function to process the video.

- Extract frames from the video using the multimedia framework, ffmpeg.
- Utilize the Python face_recognition library to identify and classify the first detected face in the video.
- Search for the recognized face's name in DynamoDB to retrieve the associated academic information.
- Store the student's academic information as a CSV file in the S3 output bucket, with the
- file name corresponding to the video name.

The system is required to prepopulate student data into DynamoDB and establish an event-driven function within OpenFaaS, utilizing a customized container image that incorporates ffmpeg and the face_recognition library. To ensure precise and efficient processing, comprehensive testing using provided sample videos and a workload generator is essential. Evaluation criteria encompass the accuracy of face recognition, precision of S3 input and output content, and the timely execution of requests. The benchmark for assessment involves accomplishing 100 requests. The successful implementation of this cloud-based smart classroom assistant is anticipated to provide a valuable service to educators, showcasing proficiency in cloud programming and hybrid cloud technologies such as OpenFaaS, AWS, and minikube.

2. Design and implementation

2.1 Architecture



1. User Interaction:
    a. Users upload classroom videos to an S3 input bucket, which serves as the entry point for video processing.
2. AWS S3 (Input Bucket):
    a. The S3 input bucket stores the uploaded classroom videos.
3. OpenFass event driven Function:
    a. Upon the availability of a new video in the input bucket, an event driven function is triggered in OpenFass. The function is the core component responsible for video processing.
4. Video Processing by OpenFass:
    The OpenFass function employs a series of steps to process the video, which include:
        a. Using the multimedia framework, ffmpeg, to extract frames from the video.
        b. Utilizing the Python face_recognition library to perform face recognition on the extracted frames. Only the first detected face is classified, and others are ignored.
        c. Retrieving the name of the recognized face and searching DynamoDB for the corresponding academic information.
5. DynamoDB:
    a. DynamoDB is used as the database to store and retrieve academic information related to recognized faces. Student data is preloaded into DynamoDB.
6. Academic Information Retrieval:
    a. The OpenFass event driven function uses the name of the first recognized face to query DynamoDB and retrieve the academic information associated with that student.
7. AWS S3 (Output Bucket):
    a. The OpenFass function stores the student's academic information as a file in the S3 output bucket. The name of the file is derived from the video name, and the content is a CSV file containing three fields: name, major, and year. This output is used to provide the relevant academic information back to the user.
8. User Output:

a. Users can access the academic information of students from the S3 output bucket, where the processed data is stored.

9. Testing and Workload Generator:
    a. To ensure the accuracy and efficiency of the application, a workload generator is provided for testing. This generator uses a mapping file to verify the correctness of the application's output against expected results.
10. Monitoring and Scalability:
    a. The application leverages the elasticity of OpenFass function to automatically scale out and in based on demand. As more video processing requests are made,event driven functions can scale to handle the workload efficiently. This ensures that the system can accommodate varying levels of usage and operate cost-effectively.
11. Development Container Image:
    a. The OpenFass function is created using a customized container image that is preinstalled with ffmpeg and the face_recognition library. This container image is built using a Dockerfile.

2.2 Autoscaling

Has not been implemented for this project due to setup constraints.

2.3 Member Tasks

Atul Prakash (1225542214):

- **Deploy OpenFaas to Minikube:** I have worked on the setup part with respect to deploying OpenFaas to minikube. I have followed each of the 10 steps in order to achieve the desired result. There were a few errors that came up which I have resolved successfully by using some of the good solutions available online.
- **Trigger to the OpenFaas function**: I have worked on the trigger file which is used to trigger the OpenFaas function everytime a new file is uploaded into the S3 input bucket.
- **Report and Readme creation:** Helped in writing the report using the provided template and also created the README file which contains the details of the installation requirements and the steps to run the application.

Abhi Teja Veresi (1225506321)

- **AWS EC2 Virtual Machine Setup:** In order for the entire team to work on one system, we have decided to create an AWS EC2 Instance and use that as our common virtual machine. So, I have created the instance using the Ubuntu Platform. I have created a new IAM role with the permission policies for this project.
- **Minikube start**: I have worked on setting up the minikube required for the OpenFaas. I have followed each and every step as mentioned in the canvas description to complete the setup. There were a few errors which have come up with respect to the docker when i tried to execute the command minikube start. But following a good stack overflow article I was able to resolve this issue and successfully execute the minikube start command.

- **Report and Readme creation:** Helped in writing the report using the provided template and also created the README file which contains the details of the installation requirements and the steps to run the application.

Kedar Sai Nadh Reddy Kanchi (1225297164)

- **Creating a new OpenFaas Function**: Firstly, I have worked on creating a new OpenFaas function following the articles given to us in the canvas description. As mentioned in the article, the OpenFaas CLI template engine built-in can create new functions in a given programming language. Therefore, before creating a new function I pulled in the official OpenFaaS language templates from GitHub via the templates repository. Then, finally I used the new command to create a new open faas function.
- **Setting the code for the OpenFaas function**: I have used the python template for creating the OpenFaas function. Post the creation of the new OpenFaas function, I worked on making the docker build required for this project.
- **Handler.py** Code: I worked on the handler.py file which by default gets built with the handle function. Firstly, I worked on the handle function, which would first take in the payload json that is sent to the handler.py file from the trigger file when the OpenFaas function is invoked. Then I proceeded to firstly download the video in my tmp directory that I have created in my working system. Then, I proceeded to delete the video file from the input bucket. Then, I accessed the video saved in the tmp directory and then extracted the images from the video using the ffmpeg library. Then using the face_recognition library I worked on comparing the encodings from the extracted image and the encodings given to us in the encodings file. Once a definitive result is returned by the face_recognition library, I proceed to retrieve the data corresponding to that name from the dynamoDB. Finally, I created a csv file and pushed it into the output bucket.

3. Testing and evaluation

We conducted a comprehensive evaluation of the application by submitting videos and validating the accuracy of the application's output against provided results. To initiate the testing process, we set up the AWS infrastructure, configuring S3 buckets, DynamoDB, Minikube, and OpenFaaS in a VM, ensuring all components met the specified requirements.

After successfully establishing the infrastructure, we systematically tested each phase of the application. This involved confirming the successful upload of videos to the S3 buckets and verifying the triggering of the OpenFaaS function from the input bucket, where videos underwent processing for face recognition. Initially, we conducted tests with a single video, ensuring the desired image recognition results were achieved. Subsequently, we scaled up the testing to ten videos, assessing the outcomes. Finally, we processed all videos and verified the results.

Throughout the testing process, we meticulously recorded the time taken at each step, including the time for video uploads into the input bucket, the processing time for videos, and the time taken to store results in the output bucket. These timings were then documented in a CSV file for further analysis.

4. Code

The code for the application contains the following files and its function.

handler.py:

The Python code is meant to be run as an OpenFass event driven function. Face_recognition_handler function is the entry point for the OpenFass function. It takes an event and context as input. The event is triggered when a new video file is added to an S3 bucket. It extracts the S3 bucket and object key from the event. It then downloads the video from S3, extracts images from the video, and processes those images using the face_recognition library to identify faces. It looks up additional information about the identified faces in a DynamoDB table named 'CSE546Project2StudentData'. It creates a CSV file containing the information and uploads it to another S3 bucket named 'cc-output-546'. If an error occurs at any step, it prints an error Message.

loadStudentData.py:

This file is to upload data from a JSON file to an Amazon DynamoDB table. It reads data from a JSON file and inserts each item from the JSON file into a DynamoDB table specified by the table 'CSE546Project2StudentData'.

4.1 Installation and running requirements of the application:

Access Key: `AKIAQOZDHDE5H5YOCBIJ `

Secret Key: `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`

S3 bucket names: cc-input-546 and cc-output-546

IAM role: CSE546PROJECT2ROLE

arn:aws:iam::031749249338:role/CSE546PROJECT2ROLE

AWS ID: kkanchi@asu.edu

AWS Password:

CSE535CloudComputingAWSProjectAccountPasswordForKedarSaiNadhReddyKanchi@Fall20

23

Requirements.

a. Python 3
b. pip3 install boto3
c. Install Docker Desktop
d. Create an Input S3 bucket to upload videos
e. Create the Output S3 bucket to store the output file.
f. Set Up the AWS DynamoDB and load the provided student_data.json file into it.

4.2 Installation:

1. Clone the repo - https://github.com/nehavadnere/cse546-project-lambda
2. Set the AWS Account Access Key and the Secret Access Key to the directory in which the code is cloned.
3. Configure your S3 bucket names and the DyanamoDB table details in the handler.py file
4. OpenFass setup:

   Install Minikube

   a. curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
   b. sudo install minikube-linux-amd64 /usr/local/bin/miniku

   Install faas-cli

   c. curl -sL cli.openfaas.com | sudo sh

   Install Helm

   d. $ curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
   e. $ chmod 700 get_helm.sh
   f. $ ./get_helm.sh

   Start cluster

   g. minikube start

   Deploy OpenFaaS to minikube

   Create namespaces for OpenFaaS core components and OpenFaaS Functions

   h. kubectl apply -f https://raw.githubusercontent.com/openfaas/faas-netes/master/namespaces.yml

   Add the OpenFaaS helm repository

   i. helm repo add openfaas https://openfaas.github.io/faas-netes/

   Update all the charts for helm

   j. helm repo update

   Generate a random password

   k. export PASSWORD=$(head -c 12 /dev/urandom | shasum| cut -d' ' -f1)

   Get Password

   l. echo $PASSWORD

   Create a secret for the password

   m. kubectl -n openfaas create secret generic basic-auth --from-literal=basic-auth-user=admin --from-literal=basic-auth-password="$PASSWORD"

Install OpenFaaS using the chart

n. elm upgrade openfaas --install openfaas/openfaas --namespace openfaas --set basic_auth=true --set functionNamespace=openfaas-fn --set generateBasicAuth=true

Set the OPENFAAS_URL env-var

o. export OPENFAAS_URLLinks to an external site.=$(minikube ip):31112

login using the CLI

p. echo -n $PASSWORD | faas-cli login -g http://$OPENFAAS_URLLinks to an external site. -u admin -- password-stdin

Check OpenFaaS pods installation status

q. kubectl get pods -n openfaas

Build, push and deploy function in OpenFass

r. faas-cli new --lang python3 app1
s. faas-cli build -f app1.yml
t. faas-cli push -f app1.yml
u. export gw=http://$(minikube ip):31112
v. faas-cli deploy -f app1.yml --gateway $gw
w. curl -v http://192.168.49.2:31112/function/app1
x. echo test | faas-cli invoke app1 --gateway $gw