CSE 546 — Project Report

Suhas Suresh *Shambhulinganagouda Patil* *Charan Raju*

1. Problem statement

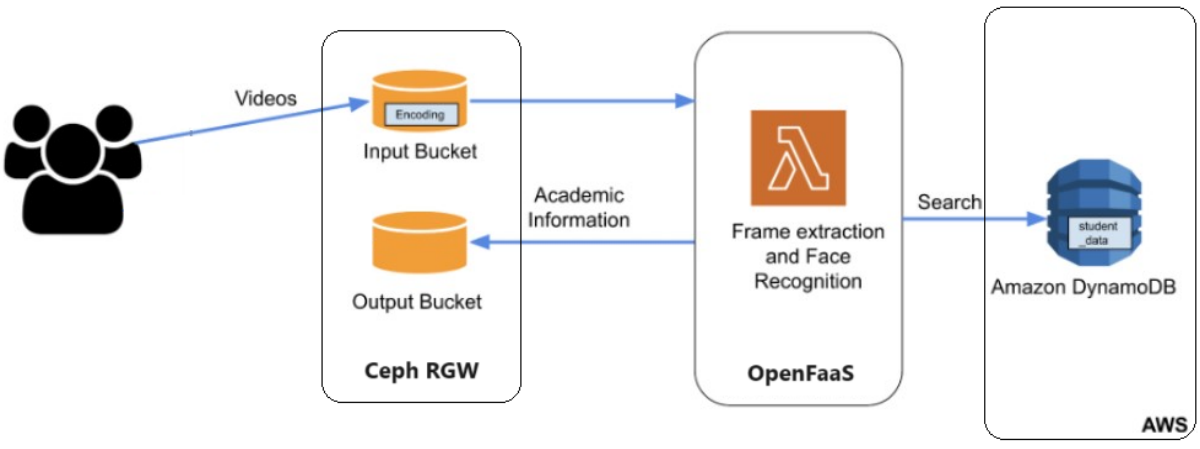
In the rapidly advancing landscape of educational technology, there exists a critical need for tools that can streamline the processing and analysis of classroom data, empowering educators with efficient means of gaining insights into student performance. Traditional manual methods are proving insufficient, prompting the exploration of innovative solutions to enhance educational efficiency.

To meet this challenge, our project focuses on the creation of a Smart Classroom Assistant, a cloud-based solution that harnesses the power of hybrid infrastructure. Incorporating leading-edge technologies such as OpenFaaS for serverless computing, Ceph RGW for scalable object storage, and AWS services for additional cloud resources, the goal is to provide educators with a transformative tool for data-driven decision-making.

In the current educational landscape, where the demands on educators continue to grow, the need for streamlined and automated processes has never been more critical. The Smart Classroom Assistant serves as a catalyst for enhancing educational efficiency by automating the extraction and analysis of academic information from classroom videos. This shift towards automation not only reduces the burden on educators but also promises a more streamlined and personalized approach to student engagement.

2. Design and implementation

2.1 Architecture



This hybrid cloud application is used as a smart classroom assistant. The application uses OpenFaaS, a serverless platform for deploying microservices, and Ceph, a distributed storage platform. The architecture of the application is described below:

1. **Ceph RGW**: A distributed storage platform that provides a RESTful interface for storing and retrieving data. In this architecture, Ceph RGW is used to store the input and output videos for the OpenFaaS function.

2. **Local application**: Monitors the input bucket in Ceph RGW and triggers the OpenFaaS function when a new video is uploaded. Monitors the output bucket in Ceph RGW and gathers the Academic information generated by the OpenFaaS function. Prints the academic information on the VM's CLI or saves it in a file.

3. **OpenFaaS function**: A serverless platform for deploying microservices. In this architecture, the OpenFaaS function is used to process the videos, recognize faces, and retrieve academic information.

4. **Ffmpeg**: A multimedia framework that can be used to extract frames from videos.The OpenFaaS function uses ffmpeg to extract frames from the videos that are uploaded to the input bucket.

5. **face_recognition library**: A Python library that can be used to recognize faces from images.The OpenFaaS function uses the face_recognition library to recognize faces from the frames that are extracted from the videos.

6. **AWS DynamoDB**: A NoSQL database service that is offered by Amazon Web Services.The OpenFaaS function queries AWS DynamoDB to retrieve the academic information of the recognized students.

7. **Docker**: A platform for building, running, and managing containerized applications. The OpenFaaS function is packaged as a Docker image, which makes it easy to deploy and manage.

8. **Kubernetes**: A container orchestration platform that is used to manage the OpenFaaS function.

9. **Helm**: A package manager for Kubernetes that is used to install OpenFaaS.

2.2 Member Tasks

Suhas Suresh: Suhas implemented facial recognition using ffmpeg and face-recognition libraries.. He configured the Ceph input and output buckets to serve as the storage components for the video files and processed academic information, respectively. Suhas took charge of configuring and setting up DynamoDB, a NoSQL database service provided by AWS. DynamoDB served as the backend database to store and retrieve academic information related to recognized faces in the videos.

Shambhulinganagouda Patil: Shambhu was responsible for setting up the OpenFaaS function for video processing. To achieve this, he utilized a Dockerfile that he built. A Dockerfile is a script containing instructions to assemble a Docker image. This image encapsulates the OpenFaaS function along with its

dependencies, ensuring consistency in deployment across different environments. After constructing the Dockerfile, Shambhu pushed the resulting image to the Docker Hub, making it accessible for deployment. He set up the Ceph OSDs and the RADOS daemon. He also updated the workload.py with the right Ceph credentials.

Charan Raju: Charan primarily focused on the code to retrieve input videos and store them in the OpenFaas function, upload the final results into output Ceph bucket and finally delete the local copy of the input video. Charan took on the responsibility of testing the OpenFaaS function and the Ceph storage system using a workload generator. He also worked on completing the project report that involved documenting the design, implementation, testing procedures, and outcomes of the project

3. Testing and evaluation

The application was thoroughly tested using a workload generator that encompasses two test cases. Test case 1 involves the upload of 8 videos to the Ceph bucket, while test case 2 involves the upload of 100 videos. The code can handle the automatic deletion of old files in the Ceph buckets. Upon uploading, the event triggers the monitor.py function, which executes the frame extraction, facial recognition, and data extraction code. The resultant output is saved in a CSV file and uploaded to the output Ceph bucket. The code for the workload generator can be found in workload.py. When the OpenFaaS function processed a video file, the resulting academic information was stored in a CSV file. Subsequently, this CSV file was uploaded to the output Ceph RGW bucket. Therefore, the count of invocations directly correlates with the number of video files processed and the subsequent creation of corresponding output files in the output bucket. This correlation ensures a one-to-one relationship between the processing of input videos and the generation of academic information stored in CSV files in the output bucket.

4. Code

Files:

1. **Dockerfile** -This Dockerfile is used to build a container image for the OpenFaaS Python function. We have modified the base python3-http dockerfile template to install ffmpeg, g++ and cmake for our application. The dockerfile also copies the encoding file into the handler.py's location.
2. **Encoding** - In the face recognition library, the encoding file refers to a file that contains the facial embeddings of a person's face.
3. **Handler.py** - The handler.py file in the context of the project contains the main entry point function that is executed when the Openfaas function is invoked. This file contains the implementation of the handle function.
4. **Workload.py** - Contains the code for testing the project on 2 given test cases
5. **Monitor.py** - It monitors the Ceph input bucket and checks if there is a new video file that has been uploaded and this inturn invokes the Openfaas function to download the video file from the input bucket, perform face recognition and upload the results to the output bucket.
6. **Requirements.txt** - This is a text file that includes the python packages that are required for the project.

7. **ccproject.yml -** This YAML configuration file is for defining the OpenFaaS function.
8. **Mapping** - File that lists the expected result of the workload generators' requests.
9. **Student_data.json** - It is a JSON file that is used to populate the DynamoDB table.

Setup and Execution:

1. Install Minikube and Helm.
2. Start a local cluster and deploy OpenFaas to minikube.
    a. Create namespace for OpenFaas core components and functions.
    b. Add Openfaas Helm repository
    c. Create a Kubernetes Secret for basic authentication.
    d. Install OpenFaas using Helm chart.
    e. Login to OpenFaas CLI
3. To install Ceph using microceph:
    a. Install microceph.
    b. Initialize the cluster.
    c. Add storage (We added 3 disks of 12GB each).
4. Setup RGW service and create realm, zonegroup, zone and user.
5. Using the data in student_data.json file populate the DynamoDB table.
6. Using handler.py and Dockerfile, create OpenFaas function based on python3 - http template.
7. Push the function into Docker Hub.
8. Run the monitor.py file in the background.
9. Execute the workload.py to upload the files into the Ceph bucket.