CSE 546 - Cloud Computing

Project Report 3

Student Names

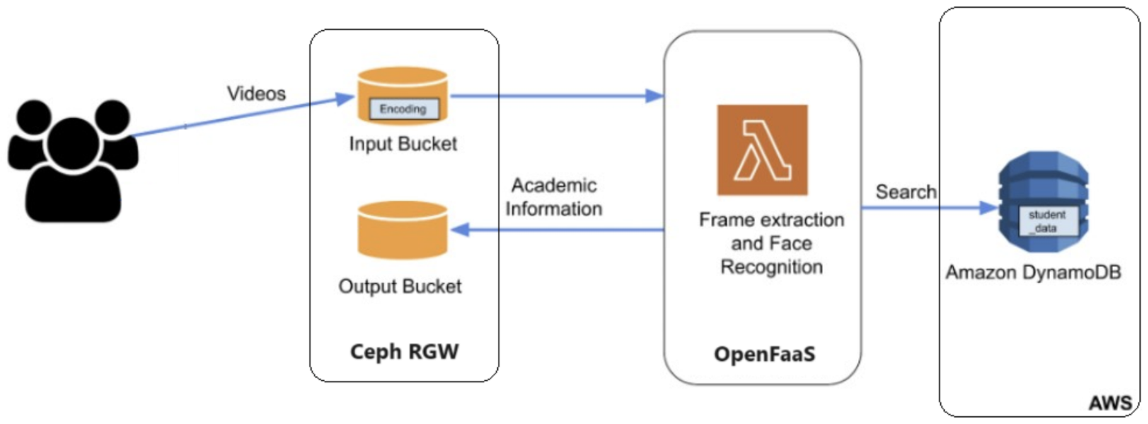
- Raviteja Reddy Guntaka
- Sai Manoja Gadde
- Karthik Chadalavada

1. Problem Statement :

This project focuses on the migration of an elastic application to a hybrid cloud environment using Amazon Web Services (AWS), OpenFaaS, and Ceph. The application, designed as a smart classroom assistant, employs face recognition on user-uploaded videos to extract academic information from AWS DynamoDB. The hybrid setup involves deploying OpenFaaS on Minikube for serverless functions and configuring an all-in-one Ceph storage cluster for object storage. Key tasks include establishing seamless communication between AWS, OpenFaaS, and Ceph components, deploying and executing OpenFaaS functions for video processing, integrating face recognition using the face recognition library, and querying AWS DynamoDB for academic details. Successful implementation will yield an efficient and scalable hybrid cloud environment, providing valuable insights into the seamless integration of serverless functions and object storage for real-world applications.

2. Design and Implementation :

2.1 Architecture



2.1.1 Cloud and Serverless Functions (OpenFaaS on Kubernetes)

OpenFaaS (Functions as a Service) is deployed on a Kubernetes cluster, creating a scalable platform for running serverless functions. These functions, packaged in Docker containers, offer seamless deployment and management. Events, such as new video uploads, trigger OpenFaaS functions.

2.1.2 Object Storage (Ceph S3)

Ceph storage buckets serve as repositories for input and output data. The input bucket stores classroom videos, while the output bucket holds processed data, such as results from face recognition.

2.1.3 Database (AWS DynamoDB)

DynamoDB houses students' academic information, queried by OpenFaaS functions to retrieve relevant data based on face recognition results.

2.1.4 Video Processing and Face Recognition

Videos in the S3 input bucket trigger OpenFaaS functions. This function, possibly using ffmpeg for frame extraction and the Python face recognition library for student identification, processes the video. Only the first recognized face in each video is used for subsequent processing.

2.1.5 Data Flow and Processing

New video uploads to the Ceph input bucket trigger an event, invoking an OpenFaaS function for video processing. This function extracts frames, performs face recognition, and queries DynamoDB. Processed data, like academic information, is stored in the Ceph output bucket.

2.1.6 Local Machine or VM (Private Cloud)

The local machine or VM hosts application code, monitoring Ceph buckets. It triggers the OpenFaaS function on new video detection and retrieves processed data from the output bucket, presenting it via CLI or a file.

2.1.7 Additional Components

- **Docker:** Used for containerizing OpenFaaS functions, ensuring consistency and ease of deployment.
- **Minikube (for Local Kubernetes Development)**: Simplifies running a local Kubernetes cluster for development and testing.
- **Networking and Security:** Implements proper IAM roles and policies in AWS for secure S3 and DynamoDB access. Secure handling of credentials and sensitive data is maintained, potentially through environment variables or cloud secrets management services.
- **Monitoring and Logging:** Utilizes tools for monitoring OpenFaaS functions and the Kubernetes cluster, along with logging mechanisms to track function executions, errors, and system messages.

2.2 Autoscaling :

OpenFaas is engineered to handle automatic scaling without the need for explicit auto-scaling algorithms or groups. When a request is made to a Ceph cluster, the cluster monitors logs details. If multiple requests come in, Openfaas automatically scales out by creating concurrent resources to manage these requests. Once the request activity subsides, Openfaas terminates these resources to scale back in.

2.3 Member Tasks

Raviteja Reddy Guntaka:

- Integrate the components developed by other team members.

Ensure seamless interaction between Ceph RGW, OpenFaaS functions, and DynamoDB.

- Developed comprehensive test plans and conducted testing (unit, integration, and system).
- Validated the end-to-end workflow, from video upload to data processing and storage.
- Assisted in deploying the application, including OpenFaaS functions and Ceph configurations.
- Set up monitoring tools to track system performance and health.
- Design and create the DynamoDB table for storing academic information.
- Configured IAM roles and policies for secure access to DynamoDB.

Sai Manoja Gadde:

- Developed the OpenFaaS function for video processing, including frame extraction and face recognition. Wrote the logic to interact with Ceph RGW (for video input/output) and DynamoDB (for querying academic information).
- Created Dockerfiles for the OpenFaaS functions, ensuring all dependencies were included.
- Managed container builds, ensuring they are optimized for performance.
- Developed scripts or mechanisms to monitor Ceph RGW buckets and trigger OpenFaaS functions.
- Handled data retrieval and processing from the output bucket.

Karthik Chadalavada:

- Installed and configured the Ceph storage cluster.
- Created and managed Ceph storage pools, ensuring they are optimized for the project's needs.
- Set up Ceph Object Gateway (RGW) to provide S3-compatible storage.
- Created buckets for input (video uploads) and output (processed data).
- Implemented bucket notifications or other mechanisms to trigger processing when new videos are uploaded.

3. Testing and evaluation

The project's testing strategy was meticulously designed to ensure the robust functionality and reliability of the application, particularly focusing on the integration of OpenFaaS functions with Ceph storage and DynamoDB. This involved a comprehensive, iterative testing process, emphasizing both individual component functionality and overall system integration.

- *Component-Level Testing:*
    - OpenFaaS Function Testing: Each OpenFaaS function, particularly handler.py, was tested for correct execution, including its ability to process inputs and interact with Ceph and DynamoDB.

- *Ceph Storage Interaction:*
    - Tested the connection to both input and output buckets in Ceph, ensuring proper configuration, access control, and file handling.

- *DynamoDB Connectivity:* Verified the function's ability to successfully query and update data in DynamoDB.

- *Integration Testing:*
    - Function and Storage Integration: Tested the integration between the OpenFaaS function and Ceph storage, ensuring that file uploads to Ceph trigger the function and processed results are correctly written back.

- *Function and Database Integration:* Ensured seamless interaction between the OpenFaaS function and DynamoDB for data retrieval and storage.

- *End-to-End System Testing:* Conducted comprehensive testing of the entire workflow, from uploading a video to the Ceph input bucket, processing it via the OpenFaaS function, querying DynamoDB, and storing results in the output bucket. Utilized a workload generator to simulate the upload of video files and trigger the processing pipeline. Verified the correctness and format of the output in the Ceph output bucket.

- *Logging and Monitoring:* Implemented extensive logging within the OpenFaaS function for real-time monitoring and troubleshooting. Analyzed logs post-execution to identify and address any operational issues.

4. Code

Folder Structure

Directory Structure and each file details:
- Readme.md: lists group member and there tasks, the AWS credentials, S3 bucket names, DynamoDB table name, Lambda Function detail, Elastic Container Registry Detail.
- Handler.py: Event processor, It handle the event generated when a file is uploaded to Input Ceph bucket. It is responsible for downloading video, parsing it into frames, classifying the frame and uploading the result csv to output Ceph bucket.
- .aws: Folder containing credential file for AWS
    - credentials -> AWS credential

- load_data_to_dynamoDB.py: Script to parse the student_data.json and upload items to DynamoDB.
- requirement.txt: list of python libraries used.
- Dockerfile: Configuration to build Docker Container image.
- mapping: As provided in the project github repo
- encoding: As provided in the project github repo
- entry.sh: As provided in the project github repo
- project3.yml: YAML configuration files for defining Ceph, S3 and DynamoD to the VM.
- Openfaas_invocation.py: Monitors any file upload to ceph inputbucket to invoke function for face recognition task
- Start.sh: Useful for initial docker build, deploy and run
- Test.py: You can save the file from outputbucket to view

Running the Application