# CSE 546 — Project Report - Hybrid Cloud

*Sai Krishna Chilvery*

*Akash Reddy Maligireddy*

*Sadaf Shaik*

## 1. Problem Statement

This project aims to develop a hybrid cloud-based application that integrates Amazon Web Services (AWS), Ceph (as object storage), and OpenFaaS (as a private cloud) to transition an elastic application into a hybrid cloud environment. The focus is to create a scalable, on-demand, and cost-effective cloud service that can efficiently manage automatic scaling.

**Core Requirements:**

1. **OpenFaaS:** Using Kubernetes through Minikube, OpenFaaS is set up to allow the deployment of serverless functions that can handle video processing tasks.

2. **Ceph**: Ceph offers object storage that is compatible with AWS S3. It's used here as a storage solution for the input and output buckets. The Ceph Rados Gateway (RGW) provides an S3-compatible interface for the application.

3. **Cloud Application**: The smart classroom assistant is the core application that integrates the OpenFaaS functions with the storage and database services. It automates the process of uploading videos, extracting frames, recognizing faces, and retrieving academic information from DynamoDB.

4. **System Evaluation:** Conduct tests using sample videos and a workload generator. Assess scalability, performance, and functionality in a simulated classroom setting. Evaluate face recognition accuracy, data correctness in buckets, and processing time.

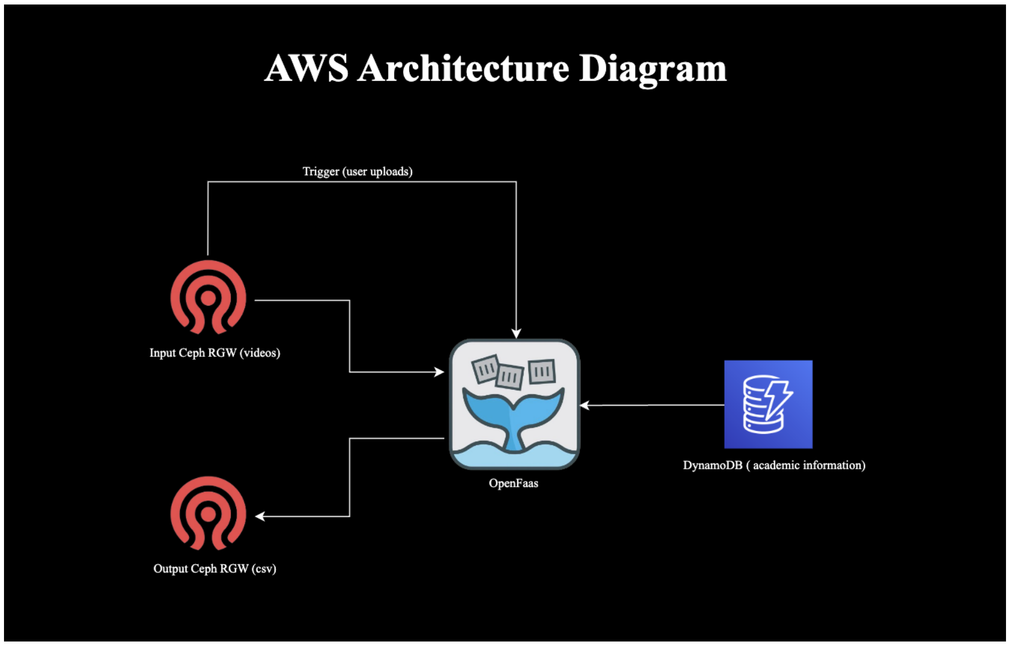
## 2. Design and Implementation

### 2.1 Architecture

1. **Hybrid Cloud Environment**
   - Integration of Amazon Web Services (AWS) and Ceph (as object storage), coupled with OpenFaaS deployed on a private cloud environment. This setup is

designed to leverage the strengths of both public and private cloud solutions, ensuring scalability, flexibility, and cost-effectiveness.

2. **Input Mechanism (Ceph RGW):**
    - An input bucket is set up to receive classroom video uploads. This bucket acts as the initial interface for users to interact with the system.
3. **OpenFaas Function:**
    - These functions are triggered upon video uploads. They are responsible for processing the videos, which includes tasks like face recognition using Python libraries and other processing needs.
4. **AWS DynamoDB:**
    - A connection to AWS DynamoDB is established for accessing preloaded academic data of recognized individuals. After processing the videos, OpenFaaS functions query this database to retrieve relevant academic information based on the faces recognized.
5. **Output Storage (Ceph RGW):**
    - The results from the OpenFaaS functions, including the academic information retrieved from DynamoDB, are compiled and stored in an output bucket. The stored data is organized to correspond with the processed video files, including details like student names, majors, and academic years.
6. **Monitoring and Orchestration (Kubernetes via Minikube):**
    - Kubernetes, running on Minikube, is used for orchestrating and managing the containerized OpenFaaS functions. This setup ensures efficient management of resources and scalability of the application, suitable for handling varying workloads in a classroom environment.



## 2.2 Autoscaling

1. **Event-Driven Architecture:**
   The system will utilize an event-driven architecture, where specific events, such as the uploading of a new classroom video, automatically initiate the processing sequence. This approach is inherently scalable, allowing each event to be handled independently and efficiently, ensuring that the system can adapt to varying workloads seamlessly.
2. **Stateless Components:**
   Key components of the system, particularly the OpenFaaS functions, are designed to be stateless. This means they do not depend on the state of other services or the context of previous interactions for their operations. Such a design ensures that each component can scale independently and responsively, enhancing the system's ability to handle fluctuating demands without bottleneck issues.
3. **Microservices Approach:**
   The application will be segmented into a series of microservices, with each one dedicated to a specific function, such as handling video uploads, processing the videos, and managing data storage. By employing a microservices architecture, the system enables individual scaling of each service. This modular approach allows for more precise and efficient scaling, where each service can independently adjust its resources based on specific demand, resulting in overall better system performance and resource utilization.

### 2.3 Member Tasks

In the described project structure, each member has been assigned specific responsibilities relating to different components of the AWS architecture.

1. **Sai Krishna Chilvery - Storage Management (Ceph RGW):** Sai Krishna Chilvery is responsible for the development of OpenFaaS functions for video processing and face recognition. This entails
   - Deploy CephFS and setup RGW in a Virtual Box
   - Facilitating the connection between the storage components and the OpenFaaS functions.
   - Implementing scripts for the efficient transfer of files between the system and the storage buckets.
   - Establishing triggers in Ceph RGW to initiate OpenFaaS functions when videos are uploaded.
   - Writing the necessary scripts in setup.py for setting up and initializing the storage buckets and their triggers.
2. **Akash Reddy Maligireddy - OpenFaas:** Akash is tasked with the development of OpenFaaS functions for video processing and face recognition. Key responsibilities include:
   - Deploy OpenFaas On a Minikube Kubernetes Cluster

    - Designing and implementing functions for video processing using tools like FFMPEG and integrating face recognition algorithms.
    - Ensuring smooth interaction between these functions and the video uploads from Ceph RGW.
    - Developing a deployment script within setup.py for the easy and efficient setup of OpenFaaS functions.
3. **Sadaf Shaik - AWS DynamoDB Integration and Management:** Sadaf is responsible for managing AWS DynamoDB, focusing on storing and retrieving academic data. Her role includes:
    - Coordinating with OpenFaaS functions for data retrieval after face recognition processes.
    - Creating a query_dynamo_table function for extracting academic information based on recognized student names.
    - Ensuring the database is preloaded with necessary data and integrating these processes into setup.py for streamlined deployment.
    - Collaborating with her team members, she played a crucial role in connecting the OpenFaaS function to AWS DynamoDB for efficient data management.

The team conducted collective reviews and discussions to ensure that each aspect of the hybrid cloud system, including OpenFaaS functions, storage solutions (Ceph RGW), and DynamoDB, operates in unison. This collaborative effort is essential to validate the integrated functionality and efficiency of all system components, ensuring a cohesive, effective, and responsive hybrid cloud solution.

## 3. Testing and evaluation

The application underwent comprehensive testing through a workload generator, encompassing two separate scenarios. In the first scenario, the task involved uploading eight videos to a Ceph storage bucket. The second scenario expanded this task to include the upload of 100 videos. A notable feature of the code is its ability to automatically delete older files in the Ceph buckets to manage storage space efficiently.

During each upload, the `monitor.py` function is activated. This function is responsible for coordinating various tasks such as extracting frames from the videos, identifying faces within these frames, and extracting relevant data. The collected data is then carefully compiled into a CSV file. This file, containing the processed information, is subsequently uploaded to a specified output bucket within the Ceph storage system. The entire process, from uploading videos to generating and storing data, is managed by the code in `workload.py`.

The OpenFaaS function plays a crucial role in this setup. As it processes each video file, it gathers academic-related information derived from the videos. This information is systematically organized into a CSV file. Once the data compilation is complete, the CSV file is uploaded to the output bucket in the Ceph RGW (Rados Gateway) storage system. This approach ensures a consistent and direct relationship between the number of video files processed and the corresponding output files generated. Each video file processed results in the creation of a unique CSV file containing academic information, which is then stored in the output bucket. This method guarantees a streamlined and efficient one-to-one correspondence between the processing of input videos and the generation of output data files.

## 4. Code

1. **Handler.py**: In the project structure, the handler.py file holds the primary entry point function executed upon invoking the OpenFaaS function. It encompasses the implementation of the handle function.

2.**Workload.py**: This particular file encapsulates the essential code for testing the project, focusing specifically on two provided test cases.

3. **Monitor.py**: Assigned with the responsibility of monitoring the Ceph input bucket, this file examines if a new video file has been uploaded. It subsequently triggers the OpenFaaS function to download the video from the input bucket, perform face recognition, and upload the results to the output bucket.

4. **handler.yml**: The YAML configuration file, handler..yml, plays a crucial role in defining the OpenFaaS function.

5. **Dockerfile**: Deployed for constructing a container image for the OpenFaaS Python function, this Dockerfile has been customized from the base python3-http template. It installs ffmpeg, g++, and cmake specific to the application. Additionally, the docker file copies the encoding file into the location of handler.py.

6. **Mapping**: Outlining the expected outcomes of requests generated by the workload generator, the mapping file serves a crucial purpose.

7. **Encoding**: In the context of the face recognition library, the encoding file refers to a document containing the facial embeddings of an individual's face.

8. **Setup_aws.py** : It sets up the dynamoDB and loads with given student data.

9. **Setup_Ceph.py**: Creates input and output buckets in the Ceph RGW

**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 (3 disks of 5 GB 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.

## Contributions - Sai Krishna Chilvery

### Contributions:

1. **CephFS Deployment**: Executed the deployment of CephFS within a Virtual Box environment, laying the groundwork for scalable storage management.
2. **RGW Setup**: Orchestrated the setup of RGW (Rados Gateway), ensuring robust and secure object storage capability for video processing tasks.
3. **OpenFaaS and Storage Integration**: Spearheaded the integration of OpenFaaS functions with storage components, enabling seamless video data handling.
4. **File Transfer Scripts**: Authored efficient scripts to facilitate the swift transfer of files between the system and Ceph storage buckets.
5. **Ceph RGW Triggers**: Innovated triggers within Ceph RGW to automatically launch OpenFaaS functions upon video upload, streamlining the processing workflow.\

### Skills Learned:

• **Ceph Storage Solutions**: Gained expertise in Ceph storage systems, focusing on the deployment and management of distributed storage solutions.

• **OpenFaaS Functions**: Enhanced proficiency in creating and deploying serverless functions tailored for video processing and facial recognition.

• **Python Scripting**: Advanced scripting abilities in Python, particularly for automating interactions between storage systems and function-as-a-service platforms.

• **Event-Driven Triggers**: Acquired knowledge in setting up event-driven mechanisms to activate cloud functions, reinforcing the responsiveness of the system.

• **Virtualization with Virtual Box**: Developed a solid understanding of virtualization tools, specifically using Virtual Box for creating isolated development environments.

• **System to Storage Communication**: Learned to establish and maintain secure and efficient communication channels between compute instances and storage services.

• **Kubernetes and Minikube**: Mastered the deployment and management of containerized applications using Minikube in a local Kubernetes environment.

## Contributions - Akash Reddy Maligireddy

### Contributions:

• **OpenFaaS Deployment**: Pioneered the deployment of OpenFaaS on a Minikube Kubernetes cluster, establishing a scalable serverless environment.

• **Video Processing Functions**: Designed and implemented video processing functions employing FFMPEG, coupled with face recognition algorithms for advanced data analysis.

• **Function and Storage Interactions**: Engineered the seamless interaction between video processing functions and video uploads from Ceph RGW, ensuring a fluid data pipeline.

• **Deployment Scripting**: Crafted a deployment script within setup.py, simplifying the setup and management of OpenFaaS functions for end-users.

**Skills Learned:**

• **Kubernetes and Minikube**: Mastered the deployment and management of containerized applications using Minikube in a local Kubernetes environment.

• **Serverless Function Development**: Gained significant experience in developing serverless functions with OpenFaaS, focusing on video processing and face recognition.

• **FFMPEG for Video Processing:** Acquired technical skills in using FFMPEG for video manipulation and processing tasks within cloud functions.

• **Scripting with setup.py:** Enhanced automation skills by creating scripts that streamline the deployment process of serverless applications.

• **Cloud Storage Integration:** Learned to integrate cloud-based storage solutions with serverless functions, facilitating efficient data handling and processing.

## Contributions - Sadaf Shaik

**Contributions:**

• **DynamoDB Management**: Managed AWS DynamoDB services, focusing on the storage and retrieval of academic data, crucial for the application's core functionality.

• **Data Retrieval Coordination**: Coordinated with OpenFaaS functions to retrieve data seamlessly post face recognition, linking identification processes with database queries.

• **Query Function Development**: Developed a query_dynamo_table function, enabling precise extraction of academic information by recognized student names.

• **Database Preloading**: Ensured the database was preloaded with essential data and integrated these processes into setup.py for efficient deployment and management.

• **Team Collaboration**: Worked collaboratively with team members to facilitate the connection between OpenFaaS functions and AWS DynamoDB, enhancing data management efficiency.

**Skills Learned:**

• **AWS DynamoDB**: Enhanced skills in managing non-relational databases, specifically AWS DynamoDB, for robust and scalable data storage solutions.

• **OpenFaaS and AWS Integration**: Gained experience in integrating serverless functions with cloud databases, ensuring efficient data flow.

• **Database Querying**: Acquired expertise in crafting database query functions, allowing for targeted data retrieval based on specific criteria.

• **Scripting for Deployment**: Improved scripting capabilities within setup.py for streamlined deployment and initialization of cloud services and databases.

• **Collaborative Development**: Strengthened collaborative skills in a cloud computing environment, contributing to a cohesive and integrated team effort.