CSE 546 - Project 3: Hybrid Cloud - Project Report

Team: Quiz on Friday

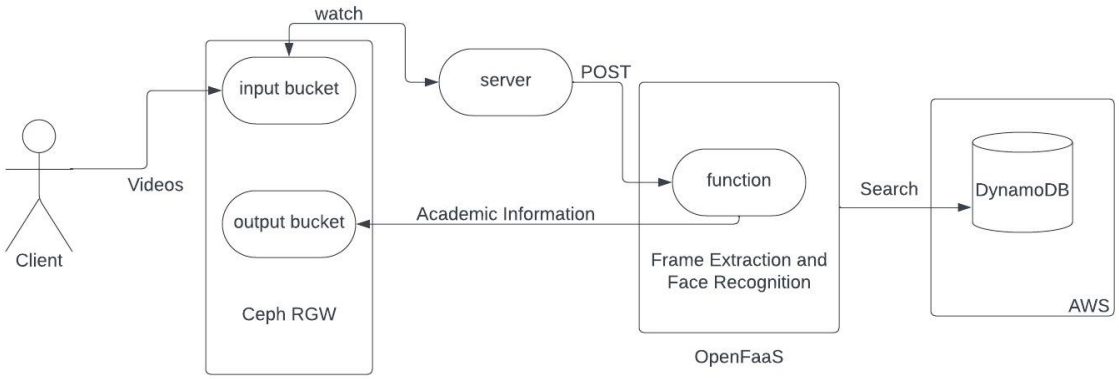
Kaustubh Manoj Harapanahalli, Shravya Suresh, Sreekar Gadasu

1. Problem statement

The aim of this project is to build an application for image recognition leveraging tools such as OpenFaas, Ceph, and AWS Resources. The classroom videos are stored in Ceph. OpenFaas is used to develop a function that is invoked using an API call made by the server whenever a new video is uploaded to the Ceph bucket. Once the OpenFaas function is invoked, it performs facial recognition on the video to get the student's name. This is then matched against the data present in Amazon DynamoDB and the output is stored in the form of CSV files in a Ceph bucket. The core motive of this project is to enable a hybrid cloud using OpenFaas, Ceph, and AWS, which seamlessly integrate and communicate with each other, thereby offering speed, security, and scalability.

2. Design and implementation

2.1 Architecture



Custom Cloud
Architechture

Bucket Storage:

- **Input Bucket:** This is a ceph bucket that holds all the input videos of the classroom uploaded by the users. When a video is uploaded, the server will capture the name of the file and invoke a REST API call to the OpenFaas function.

- **Output Bucket:** This is a ceph bucket created to hold the results. Once the OpenFaas function is executed, it stores the output in this bucket as a CSV file.

Server: This is a python script. This component is responsible for the function invocation. The server starts once the videos start uploading to the input bucket by the workload generator file. As soon as the videos start uploading in the input bucket, the server retrieves a list of files present in the bucket. It then compares it against the list of files for which the function has been already invoked. Furthermore, for each of the remaining files, it makes a REST API call to an OpenFaas function with the filename as payload.

Functions: We run a Face Extraction and Face Recognition function hosted on a Ubuntu-based server using OpenFaaS as the base. Here the function is deployed on a docker cluster orchestrated by Kubernetes and OpenFaaS. OpenFaaS provides a REST endpoint to which the user/application can make POST requests to get the results. Based on the traffic the OpenFaaS gets, OpenFaaS automatically increases the computational resources and produces results faster which is the core functionality of Lambda Functions.

DynamoDB: Once the face recognition code runs on the video, we get the person's name from a predefined list. Based on the name, the lambda function then retrieves students' information such as the student's name, major, and year from the dynamoDB tables. This information was already fed into the noSQL tables based on a JSON file provided as a part of the project.

2.2 Autoscaling

OpenFaas handles autoscaling using the HPA (Horizontal Pod Autoscaler) provided by Kubernetes'. This monitors all the metrics and based on the metrics such as CPU usage, it creates or deletes the number of function instances. OpenFaas has a controller which manages the deployment of a function, and the scaling configuration decides the number of replicas present. Hence, my doing this OpenFaas optimizes the performance of the project.

2.3 Member Tasks

1. **Kaustubh Manoj Harapanahalli:** I implemented an OpenFaaS function using its templates by first setting up OpenFaaS and its command-line interface (CLI). I created a new function using the CLI command faas-cli new --lang <language> <function-name>, selecting a template appropriate for my chosen programming language. After creating the function, I wrote the necessary code within the provided template. I then built the function using the command faas-cli build -f <function-name>.yml and deployed it onto my OpenFaaS cluster with faas-cli up -f <function-name>.yml. This process enabled me to efficiently deploy and manage the function. The detailed information related to the implementation is mentioned in the implementation section.

2. **Shravya Suresh:** My primary contribution to the project was setting up the Ceph environment This was done using the Microceph version on a linux machine(Ubuntu 22.04) using Virtual Box software. Configured Ceph cluster, dashboard , and OSDs

using microceph. Furthermore, set up the rgw using radosgw-admin. Once the set up was completed, testing the set up thoroughly and creating a python script to create two buckets and add the videos to the bucket. Also contributed to the server script which is responsible for the function invocation

3. **Sreekar Gadasu:** OpenFaaS relies on Kubernetes and Docker to enable autoscaling and rapid deployment of functions. For this project, I deployed a face extraction and recognition pipeline using OpenFaaS on an Ubuntu-based server cluster. The pipeline exposes a REST API allowing applications to submit images and receive back facial encodings or identification results. I leveraged OpenFaaS to provide serverless scaling - as traffic increases, OpenFaaS automatically spins up additional function containers. This eliminates capacity planning and ensures the face pipeline can handle spikes in usage. The containers run on Kubernetes for simplified management.

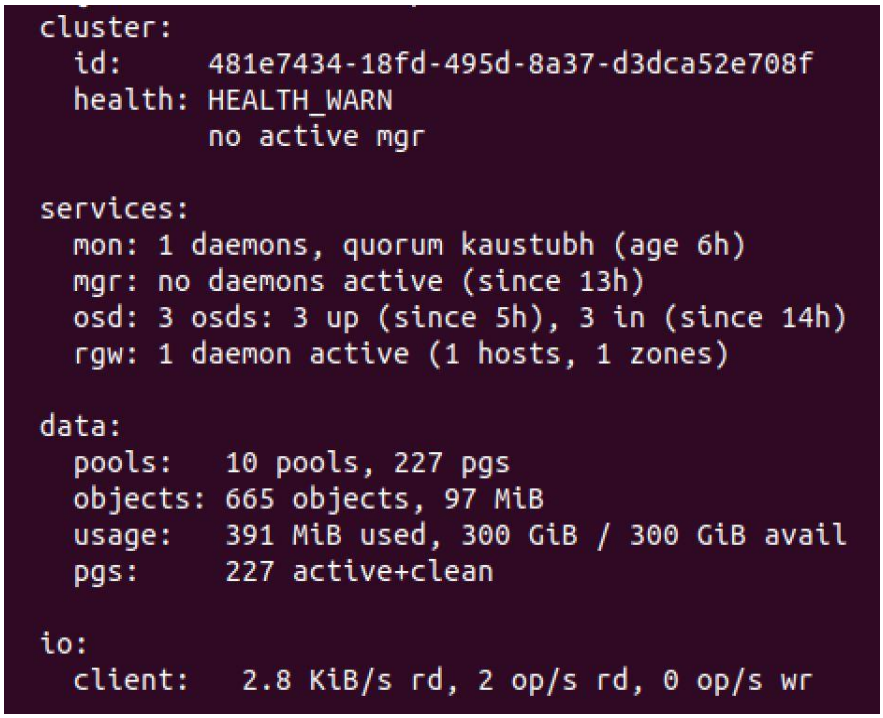
3. Testing and evaluation

Elaborate Testing was performed throughout the project.

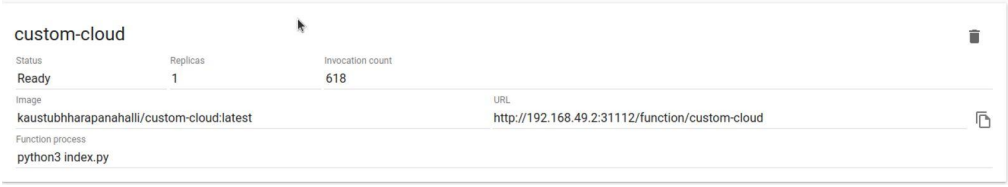
Unit Testing:

Unit Test Cases Performed:

- Ceph cluster is up and running [PASS]



- 100 invocations are made to the openFaas functions [PASS]



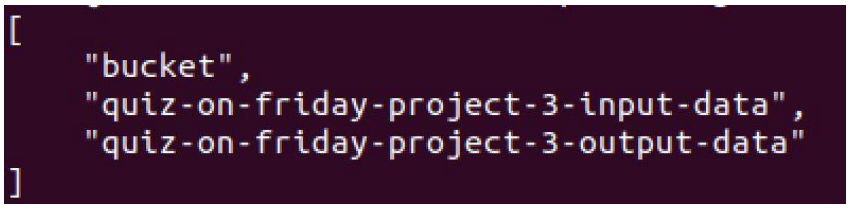
- Uploaded 100 videos to the input bucket [PASS]


```
{
    "bucket": "quiz-on-friday-project-3-input-data",
    "num_shards": 11,
    "tenant": "",
    "zonegroup": "5e1de2bc-d3ba-4019-ae05-b6275221e93c",
    "placement_rule": "default-placement",
    "explicit_placement": {
        "data_pool": "",
        "data_extra_pool": "",
        "index_pool": ""
    },
    "id": "e2a0f597-81ca-4680-8fa5-ab09dbeb60d4.24147.2",
    "marker": "e2a0f597-81ca-4680-8fa5-ab09dbeb60d4.24147.2",
    "index_type": "Normal",
    "owner": "user",
    "ver": "0#60,1#164,2#269,3#290,4#190,5#164,6#303,7#219,8#255,9#287,10#158",
    "master_ver": "0#0,1#0,2#0,3#0,4#0,5#0,6#0,7#0,8#0,9#0,10#0",
    "mtime": "0.000000",
    "creation_time": "2023-12-01T08:21:19.228266Z",
    "max_marker": "0#,1#,2#,3#,4#,5#,6#,7#,8#,9#,10#",
    "usage": {
        "rgw.main": {
            "size": 101207137,
            "size_actual": 101412864,
            "size_utilized": 101207137,
            "size_kb": 98836,
            "size_kb_actual": 99036,
            "size_kb_utilized": 98836,
            "num_objects": 100
        }
    },
    "bucket_quota": {
        "enabled": false,
        "check_on_raw": false,
        "max_size": -1,
        "max_size_kb": 0,
        "max_objects": -1
    }
}
```


- 100 csv files added to the output bucket [PASS]


```
{
    "bucket": "quiz-on-friday-project-3-output-data",
    "num_shards": 11,
    "tenant": "",
    "zonegroup": "5e1de2bc-d3ba-4019-ae05-b6275221e93c",
    "placement_rule": "default-placement",
    "explicit_placement": {
        "data_pool": "",
        "data_extra_pool": "",
        "index_pool": ""
    },
    "id": "e2a0f597-81ca-4680-8fa5-ab09dbeb60d4.24147.3",
    "marker": "e2a0f597-81ca-4680-8fa5-ab09dbeb60d4.24147.3",
    "index_type": "Normal",
    "owner": "user",
    "ver": "0#14,1#53,2#63,3#125,4#61,5#101,6#64,7#77,8#66,9#75,10#8",
    "master_ver": "0#0,1#0,2#0,3#0,4#0,5#0,6#0,7#0,8#0,9#0,10#0",
    "mtime": "0.000000",
    "creation_time": "2023-12-01T08:21:27.593506Z",
    "max_marker": "0#,1#,2#,3#,4#,5#,6#,7#,8#,9#,10#",
    "usage": {
        "rgw.main": {
            "size": 3285,
            "size_actual": 409600,
            "size_utilized": 3285,
            "size_kb": 4,
            "size_kb_actual": 400,
            "size_kb_utilized": 4,
            "num_objects": 100
        }
    },
    "bucket_quota": {
        "enabled": false,
        "check_on_raw": false,
        "max_size": -1,
        "max_size_kb": 0,
        "max_objects": -1
    }
}
```

- Input and Output buckets created [PASS]



Integration Testing Performed:

The primary objective of integration testing in this project is to confirm that the OpenFaas functions communicate and interact with external services like Ceph and DynamoDB as intended. Through an analysis of data flow, communication protocols, and data consistency, it makes sure that components work well together. Integration tests evaluate how well the OpenFaas functions handle various scenarios when communicating with these services. They include real-world interactions with external dependencies. This testing phase ensures that the system functions as intended in real-world scenarios and is crucial for spotting possible issues that could arise when different components interact.

4. Code

To setup ceph, we started with creating a VM in Virtualbox using Ubuntu 22.04 image. We created 3 additional virtual disks to assist in creation of OSDs. The following commands were implemented in the ubuntu terminal:

```
# Microceph setup
sudo su -
sudo snap install microceph --channel=quincy/stable && snap refresh --hold
microceph
sudo snap connect microceph:hardware-observe
sudo snap connect microceph:block-devices
sudo microceph cluster bootstrap
sudo microceph status
lsblk | grep -v loop
sudo microceph disk add /dev/disk1 --wipe
sudo microceph disk add /dev/disk2 --wipe
sudo microceph disk add /dev/disk3 --wipe
sudo microceph status
sudo ceph status
```

Once the setup is complete, we enable the rados admin to enable file storage.

```
# radosgw-admin setup
sudo microceph enable rgw --port 8080
```

```
export hostname=$(hostname -I)
sudo microceph.radosgw-admin realm create --rgw-realm=cephzonerealm
--default
sudo microceph.radosgw-admin zonegroup create --rgw-zonegroup=us
--endpoints=http://$hostname:8080 --rgw-realm=cephrealm --master --default
sudo microceph.radosgw-admin zone create --rgw-zonegroup=us
--rgw-zone=us-east --master --endpoints=http://$hostname:8080
sudo microceph.radosgw-admin zonegroup delete --rgw-zonegroup=default
--rgw-zone=default
sudo microceph.radosgw-admin period update --commit
sudo microceph.radosgw-admin zone delete --rgw-zone=default
sudo microceph.radosgw-admin period update --commit
sudo microceph.radosgw-admin zonegroup delete --rgw-zonegroup=default
sudo microceph.radosgw-admin period update --commit
sudo ceph osd pool rm default.rgw.control default.rgw.control
--yes-i-really-really-mean-it
sudo ceph osd pool rm default.rgw.data.root default.rgw.data.root
--yes-i-really-really-mean-it
sudo ceph osd pool rm default.rgw.gc default.rgw.gc
--yes-i-really-really-mean-it
sudo ceph osd pool rm default.rgw.log default.rgw.log
--yes-i-really-really-mean-it
sudo ceph osd pool rm default.rgw.users.uid default.rgw.users.uid
--yes-i-really-really-mean-it
sudo microceph.radosgw-admin user create --uid="user" --display-name="user"
--access-key=useraccesskey --secret-key=usersecretkey --system
sudo microceph.ceph dashboard set-rgw-credentials
sudo microceph.radosgw-admin period update --commit
sudo apt install -y awscli
sudo microceph.radosgw-admin user list
sudo microceph.radosgw-admin zone list
aws configure --profile=ceph-user
# Provide access-key = useraccesskey, secret-key = usersecretkey,
region=default, and the last value as json
```

Restart machine after all the above setup statements before running the below command to create the buckets

```
aws --profile ceph-user --endpoint-url http://192.168.64.6:8080 s3api
create-bucket --bucket input_bucket
aws --profile ceph-user --endpoint-url http://192.168.64.6:8080 s3api
create-bucket --bucket output_bucket
```

After the setup of ceph is completed, we setup openfaas using the following commands:

```
## OpenFaaS - Run these setup as user not as root
# Docker
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg
--dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg
echo \
  "deb [arch=$(dpkg --print-architecture)
signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
docker-buildx-plugin docker-compose-plugin -y
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker

# Minikube
curl -LO
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-arm6
4
sudo install minikube-linux-amd64 /usr/local/bin/minikube

# openfaas cli
curl -sSL https://cli.openfaas.com | sudo -E sh

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

sudo vi ~/.bashrc
## Add the following command to the last line of bashrc file and save it
using :wq
alias kubectl="minikube kubectl --"
source ~/.bashrc
```

```
# Setup openfaas on ubuntu
minikube start --driver=docker
kubectl apply -f
https://raw.githubusercontent.com/openfaas/faas-netes/master/namespaces.yml
helm repo add openfaas https://openfaas.github.io/faas-netes/
helm repo update
export PASSWORD=$(head -c 12 /dev/urandom | shasum| cut -d' ' -f1)
echo $PASSWORD
kubectl -n openfaas create secret generic basic-auth
--from-literal=basic-auth-user=admin
--from-literal=basic-auth-password="$PASSWORD"
helm upgrade openfaas --install openfaas/openfaas --namespace openfaas
--set basic_auth=true --set functionNamespace=openfaas-fn --set
generateBasicAuth=true
export Password=$(kubectl -n openfaas get secret basic-auth -o
jsonpath="{.data.basic-auth-password}" | base64 --decode)
echo $Password>dashboard_password.txt
echo $Password
# if you want to port forward gateway instead of using gateway external,
use the command below:
kubectl port-forward -n openfaas svc/gateway 8080:8085 &

# if you don't want to port forward, run the following commands
export OPENFAAS_URL=$(minikube ip):31112
echo -n $PASSWORD | faas-cli login -g http://$OPENFAAS_URL -u admin
--password=$Password
kubectl -n openfaas get deployments -l "release=openfaas, app=openfaas"
```

After OpenFaaS is installed, the OpenFaaS function is created and deployed to openfaas registry.

```
# Function definition
kubectl get pods -n openfaas
faas-cli template store pull python3-debian
faas-cli new custom-cloud --lang python3-debian

# create two files, access-key.txt, secret-key.txt and add the
corresponding values to these text files and run the following commands:

# NOTE: the $gw is going to be the URL of the openfaas dashboard with the
port
export gw="http://192.168.49.2:31112"
```

```
faas-cli secret create openfaas-aws-access-key --from-file=access-key.txt
-g $gw
faas-cli secret create openfaas-aws-secret-key --from-file=secret-key.txt
-g $gw

# navigate to custom-cloud folder, update requirements with
opencv-python-headless, face-recognition and boto3.
# open handler.py in the custom-cloud folder and update the code for
running the recognition there.
# go to template folder -> python3-debian folder -> Dockerfile. In the
dockerfile add ffmpeg in line 14, where there is apt-get install command.
# open custom-cloud.yml file, and add the following settings in indented
inside the function key:
```
  functions:
    custom-cloud:
      ...
      secrets:
        - openfaas-aws-access-key
        - openfaas-aws-secret-key
      environment:
        dynamodb_region: us-east-1
        dynamodb_table: CSE546-student-database
```
# Once all the above changes are made, lets deploy the function

# Update the image key in the custom-cloud.yml with your docker hub
username. If you don't have a docker hub account, create an account by
visiting hub.docker.com. login to docker hub on your machine using the
following command.

docker login

# Copy the encoding file to the custom-cloud folder.

faas-cli up -f custom-cloud.yml -g $gw

# You can check the status of the container pod using the following
command:

kubectl get pods -n openfaas-fn

#gw is same as above
```

Once the setup is complete, implement the following functions:

Code Snippets from the code:

- **Face Recognition:**

```
image_path = f"/tmp/{test_image_name}-{i:03d}.jpeg"
if os.path.isfile(image_path):
    # Load the image
    unknown_image = face_recognition.load_image_file(image_path)
    face_encodings = face_recognition.face_encodings(unknown_image)[0]
    matches = face_recognition.compare_faces(
        known_faces, face_encodings
    )
    name = "Unknown"

    # If there is a match, use the known face's name
    if True in matches:
        first_match_index = matches.index(True)
        name = known_names[first_match_index]
        result = search_database_table(name)
        output_file_name = req.split(".")[0]
        output = [
            result["name"],
            result["major"],
            result["year"],
        ]
```

- **Saving the Output:**

```
# If there is a match, use the known face's name
if True in matches:
    first_match_index = matches.index(True)
    name = known_names[first_match_index]
    result = search_database_table(name)
    output_file_name = req.split(".")[0]
    output = [
        result["name"],
        result["major"],
        result["year"],
    ]

    output_path = output_file_name + ".csv"
    with open("/tmp/" + output_path, mode="w") as file:
        writer = csv.writer(file)
        writer.writerow(output)
    # Upload CSV file to S3 bucket
    s3.upload_file(
        "/tmp/" + output_path, output_bucket, output_path
    )
```

The handle.py in the custom-cloud folder handles the receiving of the request, extracting the video from the Ceph bucket using the filename that is in the request, extracting faces, recognizing the faces using the encoding provided, and finally storing the output in a CSV file which is then stored in the output bucket.

- **custom-cloud.yml:**

```yaml
version: 1.0
provider:
  name: openfaas
  gateway: http://192.168.49.2:31112
functions:
  custom-cloud:
    lang: python3-debian
    handler: ./custom-cloud
    image: kaustubhharapanahalli/custom-cloud:latest
    secrets:
        - openfaas-aws-access-key
        - openfaas-aws-secret-key
    environment:
      dynamodb_region: us-east-1
      dynamodb_table: CSE546-student-database
```

The custom-cloud.yml is used to build the open-faas function. It contains the lang, handler name, AWS access keys to access dynamoDB and the gateway at which the function shall be deployed. This is used to deploy the open-faas function using the command:

```
faas-cli up -f custom-cloud.yml -g $gw
```

Portfolio Reports

Individual Contributions:

Shravya Suresh (1225488290)

Individual Role

The following were my individual contributions to the project.

- Ceph Set Up: My primary task was to set up Ceph on an Ubuntu 22.04 version on Virtual box using Microceph. Once the installation was complete, I set up the Ceph cluster and made sure all the warnings were fixed and HEALTH status was HEALTH_OK. I then attached the OSDs and set up ceph mgr and Ceph rgw. Finally set up users, zones, and zonegroup and realm using radosgw-admin.
- Bucket Creation : Created two buckets. Input Bucket that holds all the input videos of the classroom uploaded by the users. When a video is uploaded, the server will capture the name of the file and invoke a REST API call to the OpenFaas function. Output Bucket to hold the results. Once the OpenFaas function is executed, it stores the output in this bucket as a CSV file.
- AWS DynamoDB setup: Once the face recognition code runs on the video, we get the person's name from a predefined list. Based on the name, the lambda function then retrieves students' information such as the student's name, major, and year from the dynamoDB tables. This information was already fed into the noSQL tables based on a JSON file provided as a part of the project.
- Scripts: Created a Python script to create buckets in the Ceph cluster and load data into the bucket. Contributed to the server script. This script is responsible for the function invocation. As soon as the videos start uploading in the input bucket, the server retrieves a list of files present in the bucket. It then compares it against the list of files for which the function has been already invoked. Furthermore, for each of the remaining files, it makes a REST API call to an OpenFaas function with the filename as payload.

Skills/Knowledge Acquired through the project.

- AWS Cloud Services: I developed a thorough understanding of AWS services like DynamoDB as well as how to configure and integrate them for the creation of serverless applications.
- Containerization with Docker: I gained understanding on how to package programs and dependencies for quicker deployment by building and managing Docker containers.
- Ceph and OpenFaas: Understood set up and working with Ceph as well as creating functions and auto scaling invocations on OpenFaas.
- Automation and Troubleshooting: I became more adept at automating processes to ensure that activities are completed on time. I also developed my problem-solving skills, particularly in the area of debugging and resolving issues with cloud-based applications.
- Hybrid Cloud Integration: Gained knowledge on how to enable a hybrid cloud using OpenFaas, Ceph, and AWS, which seamlessly integrate and communicate with each other, thereby offering speed, security, and scalability.

Sreekar Gadasu | 1227891563 | gsreekar@asu.edu

Role:

● I was responsible for the setup of OpenFaaS on the Ubuntu server, this included the installation of OpenFaaS and the libraries that are required by the OpenFaaS.
● I was also responsible for the dynamoDB Implementation and integration.

Implementation:

The following tasks are involved:

OpenFaaS Setup:

● Installed minikube to provide a local single-node Kubernetes cluster for development
● Used a curl command to install the faas-cli tool for scaffolding and managing OpenFaaS functions
● Installed helm, the Kubernetes package manager, which is used to install OpenFaaS
● Started up the minikube cluster so there is a local Kubernetes master and worker to deploy pods.

Deploy OpenFaaS to minikube:

● Created the required Kubernetes namespaces for the OpenFaaS components and functions using the provided YAML file.
● Added the official Helm repo for OpenFaaS charts to get the needed Kubernetes package definitions.
● Ran Helm repo update so the local Helm client has the latest charts compatible with the installed OpenFaaS version.
● Generated a random password for the OpenFaaS admin account and stored it in an environment variable.
● Created a Kubernetes secret object containing the admin credentials so functions can authenticate.
● Used Helm to install OpenFaaS from the chart repo onto the cluster, configuring namespaces, basic auth, etc.

DynamoDB setup:

● Careful design of DynamoDB tables tailored for video metadata and processing results
● Optimization efforts focused on storage setup for efficiency
● Development of a specialized Lambda function to fetch data from DynamoDB using video names as references
● Streamlined approach for efficient data management during uploads
● Seamless integration between the Lambda function and DynamoDB tables for smooth data storage and retrieval.

Skills/Knowledge Acquired through the project:

Implementing an end-to-end video processing pipeline with OpenFaaS and DynamoDB fortified my modern cloud computing skills. This involved creating a scalable microservices architecture using containers and Kubernetes, crafting event-driven serverless functions via faas-cli, optimizing data persistence, and ensuring seamless integration. This hands-on project amplified my expertise in container orchestration, serverless architecture, efficient database management, and distributed system monitoring. Acquiring skills in API gateway management, infrastructure provisioning, and stream processor coordination laid a robust groundwork for cloud-native application design. Armed with deeper systems knowledge and broader full-stack comprehension, I'm equipped to develop robust, production-grade process automation solutions.