CSE 546 — Project 3 Report

Muskan Mehta (1225444701)

Sannidhya Pathania (1225329976)

Vaishnavi Amirapu (1225461211)

Table of contents

1. Problem Statement

In the third project, we are extending the elastic application we developed on AWS PaaS cloud to OpenStack. The application should automatically scale in and scale out on demand and cost-effectively in the Hybrid cloud environment. The application should be built using both resources from Amazon Web Services (AWS) and OpenStack (as a private cloud). OpenStack is a free, open standard cloud computing platform and is mostly deployed as IaaS in both public and private clouds where virtual servers and other resources are made available to users. AWS Lambda, which is a function-based serverless computing service should be used along with other supporting services from AWS and OpenStack. The application should implement a smart classroom assistant for educators. The application should take videos from the user's classroom and perform face recognition on the collected videos. The AWS Lambda function will be triggered by the application running on a VM in OpenStack with each video uploaded on AWS S3. Then it should look up the recognized students in the database. In the end, it should return the relevant academic information of each student to the user.

2. Design and implementation

2.1 Architecture

Amazon Web Services:

- AWS Lambda:

  AWS Lambda is a serverless, *event-driven* computing service. We are using AWS Lambda to run our application (backend service) virtually without provisioning or managing servers. The file will be downloaded from the Input S3 bucket by our Lambda function, which will then process it to recognize the face. In order to obtain more details about the student, it will next interact with DynamoDB (our database), build a CSV file, and upload it to the Output S3 bucket.

- Amazon DynamoDB:

  Amazon DynamoDB is a fully managed, serverless, key-value NoSQL database. It is designed to run high-performance applications at any scale. We are using it to store academic information about students. The schema includes 4 columns, id (primary key), name (Candidate key/global secondary index), major, and year.

- AWS S3:

  S3 is a storage service provided by AWS and is used to store videos that were provided by the workload generator. It also stores the resulting CSV files from DynamoDB after facial recognition. S3 is providing persistence for our videos and results. We are using 2 buckets, one for input .mp4 videos and one for storing the output CSV files as results from facial recognition.

- AWS SQS:

  SQS is a messaging service based on a queue data structure provided by AWS for decoupling applications. SQS is decoupling the AWS S3 from the OpenStack cloud. It will be queuing all the requests made by the workload generator to AWS S3. The queue will have *event* information related to the uploaded video which is being passed to OpenStack Nova VM.

- Amazon ECR:

  Amazon Elastic Container Registry is an AWS-managed container image registry service. It is highly secure, scalable, and reliable. ECR supports private repositories and permissions can be assigned based on AWS IAM. We chose ECR to upload our docker image rather than DockerHub (another available option) due to the following reasons:

  - ➢ better access control based on IAM user,
  - ➢ provides in-depth registry usage and,
  - ➢ having tight integration with Docker CLI.

- AWS EC2:

  EC2 is a computing service provided by AWS to create VMs. We are using an EC2 instance with Linux as an OS to run our OpenStack.

OpenStack:

- OpenStack Nova:

  OpenStack Nova is a component of the OpenStack cloud computing platform that provides compute services. It is responsible for managing and scheduling the VMs and associated resources, such as CPU, memory, and disk, in a cloud environment. We are using Nova to deploy a VM on which we have our application running. This application will trigger AWS Lambda with each video that will be uploaded on AWS S3. We are creating the Nova VM inside the default private network. Components of Nova used in our project are:

  - ➢ Nova image: A pre-configured disk image that contains a ready-to-run OS and any necessary software and applications. It is used to launch a new VM instance with the desired configuration.

- ➢ Nova instance: A VM that is launched from a Nova image. It is assigned its own set of resources, such as CPU, memory, and disk, which can be scaled up or down as needed.
- ➢ Keypair: A security credential used to access and manage a Nova instance. It consists of a public key stored on the instance and a private key kept by the user. Keypairs are used to securely authenticate and connect to the instance through SSH or other remote access protocols.

- OpenStack Neutron:

  OpenStack Neutron is a component of the OpenStack cloud computing platform that provides networking services. It is responsible for managing and provisioning network resources, such as virtual networks, subnets, routers, load balancers, and firewalls, in a cloud environment. We are using Neutron to access OpenStack Nova VM using Floating IPs.

  - ➢ Network: A logical entity that represents a group of interconnected resources, VM and routers, communicating with each other using a set of networking protocols and policies.
  - ➢ Floating IP: It is a type of IP address that is associated with a VM instance. It provides a way to assign a public IP address to an instance, used for external communication.

Architecture Design:

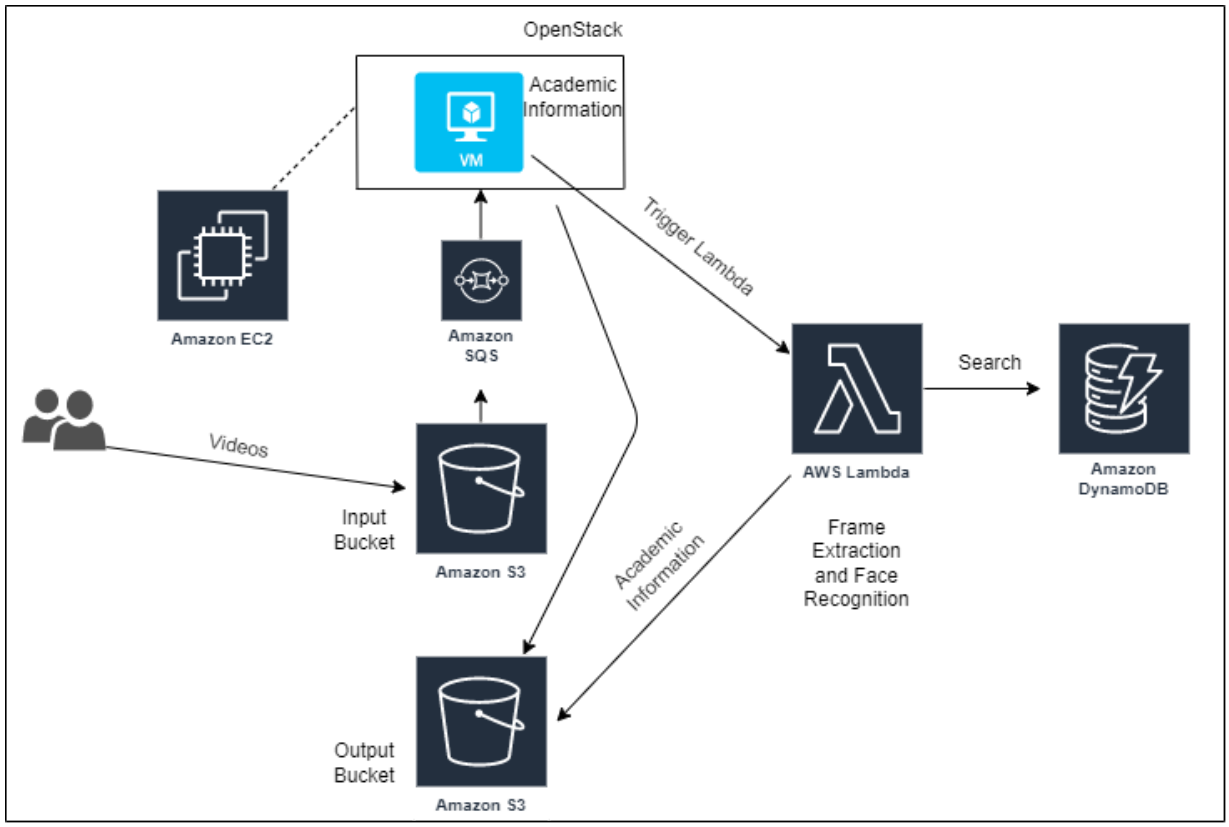

Fig.1 Architecture diagram

The Architecture diagram as seen above consists of OpenStack Nova VM, AWS EC2, AWS S3 buckets, AWS Lambda, and Amazon DynamoDB. There are 2 applications one is deployed on OpenStack Nova VM (“monitor” application) and the other on AWS Lambda (“handler” application). For accessing and running OpenStack, we have used an AWS EC2 instance with Linux OS. Then a Nova VM instance (“OpenStack-hybrid”) is created on the OpenStack cloud.

The workload generator uploads the video files to the input S3 bucket. With each video file uploaded, OpenStack Nova VM “OpenStack-hybrid” will trigger the AWS Lambda. The “monitor” application on “OpenStack-hybrid” (OpenStack Nova VM) will take the *event* information which is queued on AWS SQS after being notified by input S3 bucket for each video file uploaded. Then the "monitor" application OpenStack will pass the *event* information to AWS lambda while triggering. For instance, when a video is uploaded to the input S3 bucket, the Lambda function is invoked by the application running on OpenStack VM for that input video. At the same time, the Lambda function is handling the autoscaling based on a number of invocations whenever it is triggered by OpenStack VM’s application. The workload generator will make concurrent requests to our application. The Lambda function will automatically scale up the instances required to process the requests.

For a trigger event, AWS S3 notifies the *event* object to the AWS SQS, which is being long-polled by the "monitor" application, then these details are passed on to the handler function in Lambda. These objects contain information about the triggering event and function invocation, function configuration, etc. respectively. Our handler script uses the *event* object to get information about the input video to download from the input S3 bucket.

After the Lambda function is triggered and the input video is downloaded from the S3 bucket using the *event* information, the multimedia framework, *ffmeg* is used to extract the frames from the video. Then the script loops through the extracted frames, and upon detecting a face with known encoding, the script exits. The facial recognition result is used to fetch further details from Amazon DynamoDB. And these results are then written to a CSV file which is uploaded to the output S3 bucket.

2.2 Autoscaling

Concurrency is the number of in-flight requests our AWS Lambda function handles at the same time. For each concurrent request, Lambda will provide a distinct instance (container) of our docker image (execution environment) that was uploaded to the ECR. Lambda will automatically scale the number of execution environments as our function receives more requests up until the account's concurrency limit is reached.

The default concurrency limit for all functions in a region set by Lambda is 1,000. As our maximum limit is 100 requests at a time, we are using the in-built scaling provided by AWS Lambda.

2.3 Member Tasks

We have added the Individual Project contributions under section 5.

3. Testing and evaluation

- To test whether the OpenStack cloud was working correctly, we accessed the Horizon Dashboard using the public IP address of the EC2 VM.
- The monitor application was running continuously on our OpenStack VM to long poll the AWS SQS queue.
- The next steps mentioned were followed in order to test our application:
    - The application was tested with the 100 videos that were provided as a part of the assignment.
    - We then verified if the S3 input bucket is receiving the videos in the desired format.
    - We also monitored the Cloud Watch logs for the AWS Lambda function.
    - And verified the CSV files in the output bucket.
- The following are the results of testing and evaluation:
    - The results generated by the classification are stored in a .csv file in the S3 for persistent storage.
- We also used a script "verifier.py" which was used to list all the objects in the output S3 bucket and print their content. This script is executed in the OpenStack VM.

4. Code

4.1 Files

- Dockerfile
    - To build the docker image to encompass all the required source code files such as handler.py, encoding, and package dependency files such as requirements.txt for the application.
- encoding
    - This file contains the encoding for the known faces.

- entry.sh
    - This shell script contains the commands for the *ENTRYPOINT* of the dockerized application. This script runs every time a container of the application is created.

- handler.py
    - This Python script is the main function of the AWS Lambda service. And this is called when the Lambda function is triggered.
    - This file also contains the face recognition application logic.
- mapping
    - This file contains the input video name and the corresponding output.
- requirements.txt
    - The dependency file contains the name of all the required packages for the application to run.

- student_data.json
    - This file contains the list of all JSON objects, where each object corresponds to a single record in the database.

- monitor.py
    - This script runs inside the OpenStack VM which will be monitoring the AWS SQS queue. This queue receives the notification for each object from the input S3 bucket.

- resource.py
    - This file functions to configure the input S3 bucket and create a new AWS SQS queue to hold and buffer the messages. The monitor.py function then long polls the created SQS queue.

- verifier.py
    - This file functions and lists all the objects from the Output S3 bucket and the content of each object is displayed on the terminal of the environment. This script is executed in the OpenStack VM after all images are processed.

4.2 Installation and Running

For this project, we have set up two cloud setups - OpenStack, the private cloud, and AWS, the public cloud. The private cloud is set up in an EC2 VM in AWS. In order to set up the OpenStack project the following commands are executed:

```
>> git clone https://opendev.org/OpenStack/devstack
```

```
>> cd devstack
>> vim local.conf
>> ./stack.sh
```

The following commands are executed to create the VM instance, Security Group, a Keypair to access the VM and a Floating IP in the OpenStack project:

```
>> cd devstack
>> source openrc admin
>>  wget https://cloud-images.ubuntu.com/focal/current/focal-server-cloudimg-amd64.img
>> openstack image create --disk-format qcow2 --container-format bare --public --file
./focal-server-cloudimg-amd64.img hybrid
>> openstack security group create default2
>> openstack security group rule create --proto tcp --dst-port 22 default2
>> openstack security group rule create --proto icmp default2
>> openstack security group  list
>> openstack security group rule list default2
>> openstack network list
>> openstack keypair create unlock>unlock
>> openstack server create --flavor ds1G --image $(openstack image list | grep hybrid| cut -f3 -d '|')
--nic net-id=$(openstack network list | grep private | cut -f2 -d '|' | tr -d ' ') --key-name unlock
--security-group default2 openstack
>> openstack floating ip list
>> openstack floating ip create public
>> openstack server list
>> openstack server add floating ip openstack <floating_ip>
```

The following command will SSH into the VM created from above steps:

```
>> ssh -i unlock -v ubuntu@<ip>
```

The following commands are used to perform the DNS set up on the VM to install the required packages for running the “monitor” application.

```
>> sudo nano /etc/netplan/50-cloud-init.yaml
>> sudo netplan apply
```

5. Individual contributions

5.1 Muskan Mehta (1225444701)

DESIGN:

I was responsible for designing the Private Cloud setup using the OpenStack tool.

- The main objective of the project was to build Project 2 (PaaS implementation) in a Hybrid Cloud environment. As the name suggests, the environment involves two types of cloud - Private Cloud is set up in a private network and is not accessible to the outer world, securing the data.
- And the other is a Public Cloud using a known public Cloud provider like AWS in this project. The project also required us to set up applications respectively on each cloud and allow them to communicate with one another.

IMPLEMENTATION:

Phase 1:

- Once the design was complete, I implemented the Private Cloud setup on an EC2 VM instance in AWS using the *Devstack* tool.
- I created an EC2 VM instance in AWS to host the private cloud, *Devstack* for the project. This task required me to create a *t2.large* VM with around 40 GB of root volume to hold and provide enough capacity to provision resources on the OpenStack cloud.
- After creating the VM, the *Devstack* tool was downloaded and installed onto the VM. The complete deployment process of the tool took around 15 mins as it required setting up all the virtualization modules and services of the tools such as Nova, Neutron, Keystone, and so on.
- Once the *Devstack* tool was deployed, the Horizon dashboard of the service was up and ready for users to login and use the services. I then accessed the Horizon dashboard of the OpenStack cloud using EC2 public IP address.

Phase 2:

- I created another script (verifier.py) to verify the correctness of our implementations and print out all the results CSV file's names and contents on the terminal in plain text.

RESULTS:

- I learned about private cloud implementation and OpenStack (*Devstack*) and its components.
- I also learned about how the private cloud is deployed and how secure communication can be made between private and public cloud in this project.
- As the Horizon dashboard was up and running, I was able to verify that the OpenStack cloud was set up correctly.
- I also learned about various Linux commands.

5.2 Sannidhya Pathania (1225329976)

DESIGN:

- This is a hybrid cloud implementation project which has two clouds, OpenStack (Private cloud) and AWS (Public cloud).
- I was responsible for implementing the Private cloud of the project by creating the Nova and Neutron service components. The resources built in the OpenStack cloud used the resources of the underlying EC2 VM instance.
- I was responsible for understanding the requirements of the project and modifying the PaaS implementation to meet those requirements.
- I designed the modifications to the PaaS implementation of the application to enable communication with the Private Cloud components.

IMPLEMENTATION:

Phase 1:

- After the private cloud was set up, I created the Network (Neutron) components such as Floating IP and Security Groups for access to the VM.
- Compute (Nova) components such as Image, Instance and Key pair were created to deploy the application.
- A simple Ubuntu-flavored image was used to build the VM.

Phase 2:

- As a part of this phase, I had to create and deploy services to support our application that would trigger the AWS Lambda function of our PaaS application. I implemented the modifications for the PaaS implementation of the application to enable communication with the Private Cloud components. I made sure that the “monitor” application is able to communicate with the Private Cloud components and trigger the AWS Lambda function.
- In this phase, we deployed the PaaS application with the exception of omitting the trigger configuration for this service. The function is triggered from the application in the private cloud (*Devstack*).

RESULTS:

- I gained knowledge about private cloud implementation and OpenStack (*Devstack*) and its components.
- I tested and verified the correctness of the modified PaaS implementation, including the communication with the Private Cloud components and the triggering of the AWS Lambda function.

5.3 Vaishnavi Amirapu (*1225461211*)

DESIGN:

- The project is a Hybrid cloud implementation using OpenStack project (private cloud) and AWS (private cloud). This project is implemented in 2 phases.
- As a part of replacing the default trigger event of the AWS Lambda function, I worked on designing the AWS SQS queue and its configuration to monitor the input S3 bucket and trigger the AWS Lambda function in the Public Cloud. The input S3 bucket sends a notification to AWS SQS whenever an object is uploaded to it.
- And the "monitor" application in the private cloud will long-poll the SQS queue in AWS and trigger the Lambda function if it receives a message regarding the object created in S3.

IMPLEMENTATION:

Phase 1:

- Once the VM was created in the OpenStack cloud, a Floating IP was attached to the VM in order to be able to ping and SSH into the VM from the AWS EC2 VM.
- Upon remotely accessing the VM, the Python environment was set up using the *conda* tool and the required packages and scripts were downloaded and executed on this VM.
- While trying to set up the packages in the OpenStack Nova VM, we faced an issue with the DNS nameservers for Google (8.8.8.8 and 8.8.4.4) which were later added allowing the VM to download the required packages.
- Upon updating the network interface, we were able to install and set up the environment of the private cloud to run the "monitor" application.

Phase 2:

- An AWS SQS queue was created to monitor the input S3 bucket and the input S3 bucket was modified to send push notifications for object creation events to this created SQS queue. Every time an object is pushed, S3 will send a message to the SQS queue.
- This queue is long-polled by the "monitor" application in the private cloud and it will trigger the AWS Lambda function in the public cloud.

RESULTS:

- The private cloud implementation was a novice concept for me and I got the opportunity to learn about OpenStack (Devstack) and its components. And how the components communicate with each other.
- Also learned a new design implementation on how the AWS Lambda service can be triggered (using SQS queue) without using the default trigger configuration.