CSE 546 — Project Report

Smart Classroom Assistant for Educators (PaaS)

Sai Vikhyath Kudhroli - 1225432689

Gautham Maraswami - 1225222063

Abhilash Subhash Sanap - 1225222362

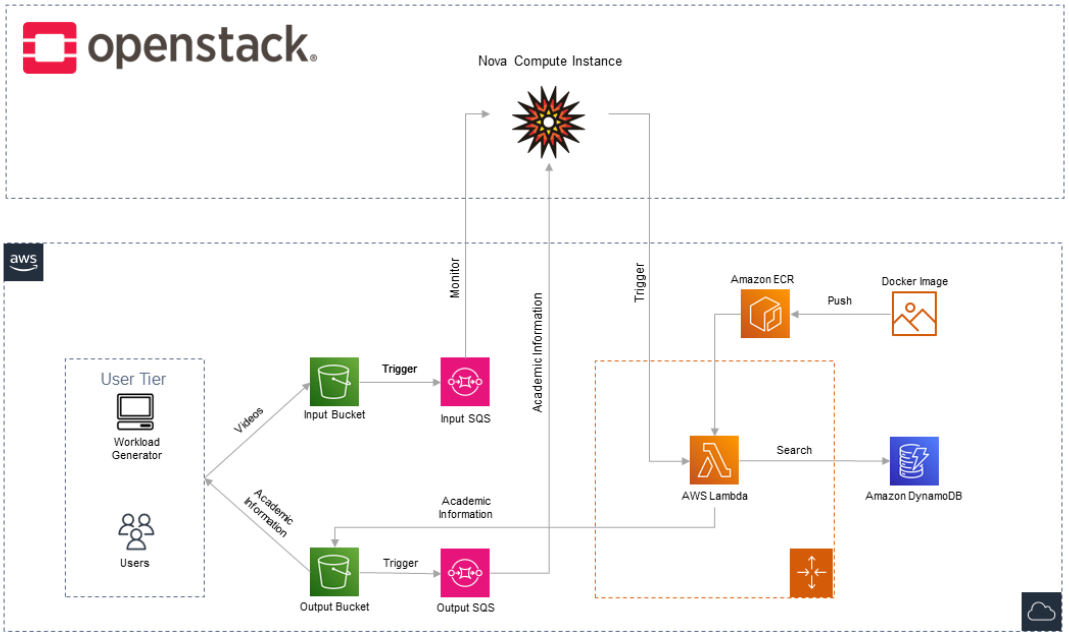
1. Problem Statement

The main objective of the project is to develop an elastic application that allows educators to look up information about any student in a smart classroom in a hybrid cloud environment. The project aims at implementing a hybrid cloud using AWS and OpenStack to recognize the face of the student in the video and fetch their academic details. Platform as a Service (PaaS) functionality offered by AWS is leveraged by using functionalities such as AWS Lambda, S3, ECR and DynamoDB. The private cloud is set-up using virtual compute, storage and networking resources offered by OpenStack. To extract the frames from the video and to perform face recognition, Python libraries were to be used. The hybrid cloud application will ultimately provide a solution to fetch a student's academic details by utilizing face recognition modules.

2. Design and implementation

2.1 Architecture

The system primarily consists of a public cloud service implemented using AWS services and a private cloud implemented using OpenStack.



AWS Lambda, AWS S3, AWS SQS, AWS ECR and AWS DynamoDB services from AWS and Nova Compute, Cinder storage, Neutron networking resources from OpenStack are used to build and support the application in a hybrid cloud set-up. The videos are uploaded to the AWS S3 input bucket either by a user or through a workload generator. As soon as the video gets uploaded into the input bucket, a trigger is created to push the URL of the video to an input SQS queue. This input SQS queue is monitored by the instance deployed in OpenStack which triggers AWS Lambda to perform face recognition. Once face recognition is complete, the details of the student are retrieved from Amazon DynamoDB. These details are formatted as a CSV and are pushed to the output S3 bucket. The trigger created on the output S3 bucket pushes the URL of the CSV containing academic details to the output SQS queue. The data in this queue is read by the instance in OpenStack. This proves that the Lambda is indeed triggered by the OpenStack instance and the utility of a hybrid cloud is satisfied.

2.1.1 AWS Resources

2.1.1.1 AWS S3

There are two S3 buckets that are utilized: one input bucket and one output bucket. The input S3 bucket stores the input video files, and the MP4 videos that are uploaded by the user or workload generator are pushed into the input S3 bucket. There is also a trigger defined on the input S3 bucket, whenever a video is uploaded it pushes the URL of the video to an input SQS queue. The output bucket is used to store the CSV files corresponding to the video uploaded which consists of academic information about the first person in the image. The output bucket receives the CSV files from the lambda function. There is a trigger defined on the output S3 bucket as well which pushes the URL of the CSV file containing academic information of the student into an output SQS queue. We have used himaliaInputBucket as the input bucket and himaliaOutputBucket as the output bucket name

2.1.1.2 AWS SQS

There are two SQS queues that are used: one input queue and one output queue. URLs of the videos uploaded to input S3 bucket are pushed into the input queue. This input queue is monitored by the instance deployed in OpenStack, which triggers the Lambda as soon as some message appears in the input queue. The output SQS queue receives the URLs of the CSV files comprising the academic information of the students. This data is pushed when the Lambda pushes the files to the output S3 bucket, the trigger defined on the output S3 bucket pushes the URLs of the CSV files to the output SQS queue. We have used s3_input_notification as the input SQS queue and s3_output_notification as the output SQS queue.

2.1.1.3 Lambda function

The lambda function is the core of the system. This lambda function is triggered by the instance deployed in OpenStack when it detects a message in the input SQS queue. A docker file is used along with the lambda function to support the following functionalities:

- Lambda function handler reads the encoding file which consists of the encodings of all the individuals in the class.
- Downloads the video from the input S3 bucket.
- Extracts all the frames in the video, then selects the first frame which has a person.

- Extracts the name of the person in the video by comparing the encoding of the first frame and all the encodings.
- Using the name, fetch the details of the person stored in the dynamo database.
- Creates a CSV file with all the academic information and pushes it to the output S3 bucket.

2.1.1.3 Elastic Container Registry

Amazon ECR is a fully managed container registry that can be used to deploy application images. The docker image is deployed on ECR which enables high availability and scalability. This eliminates the need of defining a scale-in and scale-out policy based on demand.

2.1.1.4 DynamoDB

Amazon DynamoDB is a fast, fully managed, and serverless key-value NoSQL database that is used to store the academic information of all individuals. All the academic information pertaining to all the individuals is pushed into the database. Then after the name of the person is extracted, it is used as the key to extracting all the academic information pertaining to that individual. We have used student_data as the table name.

2.1.2 OpenStack Resources

2.1.2.1 Nova Compute

OpenStack's Nova compute is used to upload an image using and to create an image from the uploaded image. An Ubuntu cloud image of 18.04 (Bionic Beaver) version is uploaded using glance. Once this image is uploaded, it is utilized to launch an instance in OpenStack which acts as a controller by monitoring the input SQS queue to trigger AWS Lambda and fetch the results from the output SQS queue.

2.1.2.2 Cinder

OpenStack's Cinder is used to create a virtual storage which is then mounted on to the virtual instance that is created.

2.1.2.2 Neutron

OpenStack's neutron is used to create a virtual network to which the virtual instance is connected. A virtual router is created first which is then attached to the virtual network. Floating IP is also allocated and associated with the instance which is used to access the instance from the outside network.

2.2 Autoscaling

The services used such as AWS lambda and AWS DynamoDB are capable of handling varying demands and provide auto-scaling without the need to explicitly define auto-scaling policies. As the lambda function receives more requests, it automatically handles scaling the number of execution environments until the account's concurrency limit is exhausted.

2.3 Member Tasks

2.3.1 Sai Vikhyath Kudhroli (1225432689)

1. Launched an EC2 instance with the hibernation option used to install openstack.
2. Configured AWS CLI on the openstack instance and git in the EC2 instance.
3. Cloned the devstack git repository which is then used to install openstack.
4. Create the local.conf in the devstack directory which specifies the configurations for installing openstack and run stack.sh to install openstack.
5. Download a lightweight Ubuntu cloud image to launch an instance in openstack.
6. Upload the Ubuntu image to openstack using glance.
7. Create a security group to allow TCP traffic and ICMP traffic across the instance from all the devices.
8. Create a key pair for the instance which is used to SSH into the instance.
9. Setup a private network by creating a virtual router and virtual network to which the instance is attached.

2.3.2 Gautham Maraswami (1225222063)

1. Create a nova instance inside openstack using the image and flavor.
2. Set resource utilization ratio to allow higher resources for openstack from host os.
3. Setup volume for the created instance using the horizon dashboard.
4. Enable Icmp, ssh and tcp traffic for the created openstack instance.
5. Enable DNS resolution for the created instance by updating the ip address in /etc/resolve
6. Install all the necessary libraries including python3, pip, boto3, json etc.
7. Resolve version compatibility issues related to python/ boto3
8. Run the controller code on the created instance.

2.3.3 Abhilash Subhash Sanap (1225222362)

1. Built the docker image from the given Docker File.
2. Created an AWS Elastic Container Registry (ECR) repository, configured the programmatic access to the ECR and pushed the image to it.
3. Configured an AWS Lambda function through the console using the “latest” image.
4. Modified the AWS lambda function so it receives the key for downloading the image from S3 from the code running in the private cloud running on an Openstack.
5. Authored the controller.py file running on the private cloud.
6. Set up the queues s3_input_notification and s3_output_notification.
7. Defined a trigger from s3_input_bucket to s3_input_notification queue and s3_output_bucket to s3_output_notification queue.
8. Wrote the code to invoke lambda from the controller running on a private cloud.

3. Testing and evaluation

3.1 Reliability Test

The application was tested to make sure the expected reliability and accuracy are achieved. It was tested and confirmed that as many lambda functions are invoked as there are videos in the

input bucket. The application was tested to check if it can handle up to one hundred requests concurrently and it successfully does.

3.2 Flow of Operations Test

The expected flow for the application is – the user uploads the video to the input bucket, which triggers an event into a input_notification queue, which is picked up by the monitoring code that is running on the private network set up in Openstack, which in turn launches a lambda function, which extracts the image, performs face recognition, queries the DynamoDB to get academic information, and pushes it to the output bucket in a CSV format, which sends an event notification to the ouptut_notification queue. This queue is also being monitored by the monitoring code in the private network, which picks up the event and downloads the csv and prints it to the console. It was ensured that this happens in the expected order.

3.3 Output validity test

Response at the workload generator is checked against the source of the truth document provided to make sure all the details fetched from the DynamoDB are correct. The time taken for the execution of one hundred images is noted to be around three and a half minutes.

4. Code

Technologies used: Python, Boto3, AWS S3, AWS DynamoDB, AWS Lambda, AWS Cloud watch, AWS SQS, Openstack, Linux (For Monitoring purposes)

The file Handler.py has all the logic required for monitoring the notification queues, downloading the file from S3, Image Recognition, getting additional information from DynamoDb, writing the output to S3. Even though DockerFile and entry.sh were already provided in the bootstrap code but required a more profound understanding of what was happening inside the code as running the code had multiple failures and needed debugging on the same.

Files:

1. Handler.py: This file contains the core logic of the project containing Image recognition, S3 Bucket access, and DynamoDB Access.
2. DockerFile: This file contains the dependencies installation and triggers to be created on lambda function creation.
3. Entry.sh contains the command to be executed on lambda function execution
4. Controller.py: This file contains the logic for communication between the public and the private cloud and it runs on the private cloud.

Handler.py

1. This file is responsible for the core operations of the project.

2. This file is triggered when any object is put in the S3 Bucket. Face_recognition_handler is invoked.
3. Method download_from_s3 is to download all the videos from S3 Bucket.
4. The downloaded file is stored in /tmp directory
5. Method generate_encoding extracts the first frame from the video and generates the encoding of the image (Frame).
6. Recognise_face method compares the encoding from the image and the encoding input provided and returns the corresponding name of the identified person.
7. fetch_data_from_dynamoDB: gets the year and major information from the dynamo database corresponding to the name.
8. Write_to_csv is created as a CSV file containing the name, major, and year corresponding to every video uploaded. The CSV files will be available in /tmp
9. The calling method then uploads the files to the S3 Bucket.

Docker File

1. The docker file is provided in the bootstrap code.
2. Function directory is specified in the docker file where image/video/csv files will be stored.
3. This file installs all the dependencies in the container required for the execution of the function.
4. The dependencies include, GCC, python, python AWS Lambda runtime interface client, AWS Lambda runtime interface emulator, FFmpeg library for image recognition, OpenCV, boto3, and NumPy.
5. Necessary permissions for the executable files are also provided using this file.

Controller.py

1. This file runs in the private cloud and handles the communication between private cloud and the public cloud.
2. It keeps monitoring the s3_input_notification queue and when it gets a new message in the queue, it invokes the lambda function with that image.
3. It also monitors the s3_output_notification queue and when it receives the message, it prints the contents to the console.
4. receive_message_from_queue() function keeps polling the queue for new messages.
5. purge_message_from_queue() function deletes the message from the queue after it is processed.
6. invoke_lambda() function invokes the lambda function and passes the event to the lambda function.

4.1 Project setup and execution

1. Launch an EC2 Instance
    a. Create an instance with configurations at least 4 VCPU 8GB Ram 50 GB Disk space.
    b. Enable hibernation for the instance by allowing disk encryption and choosing the hibernation option.

2. Download devstack openstack from github
    a. Run command git clone https://git.openstack.org/openstack-dev/devstack
    b. Update the configuration file local.conf to accommodate minimal configuration.
3. Update Security config of the host VM to enable HTTP traffic both inbund and outbound.
4. Run the command ./stock.sh to start the installation.
5. The installation will take 20 mins to complete.
6. Now access the openstack-dashboard horizon using the web address https://<publicIP of the host OS>/dashboard.
7. Now to create an ubuntu instance download OS- image from the ubuntu webpage link https://cloud-images.ubuntu.com/bionic/current/
8. Create an image by uploading the image file to Horizon dashboard.
9. Using the os uploaded image launch an instance inside openstack from horizon.
    a. Select the ssh key to login into the instance.
    b. Select a 10 GB disk 2 GB Ram as the flavor for the instance.
    c. Choose a private network as the network for instance.
    d. For the security group allow TCP/ ICMP/SSH and HTTP Traffic both inbound and outbound.
10. Create a floating IP and assign it to the created instance
11. Setup DNS Server inside the cloud image to google server 8.8.8.8
12. Login into the created instance and install python dependencies.
13. Run the controller.py code inside the created instance.
14. Create S3 Bucket for input and output using AWS Console
15. Create input and output SQS to enable notification from the S3 Buckets
16. Add triggers in the bucket to push notifications to the queues
17. Set Bucket permissions to public
18. Use the handler file provide and create a docker image using the file.
19. Create a Docker image and push it to AWS ECR
    a. Use the following commands:
    b. Retrieve an authentication token and authenticate your Docker client to your registry. Use the AWS CLI:
        i. aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 630075306220.dkr.ecr.us-east-1.amazonaws.com
    c. Build your Docker image using the following command:
        i. docker build -t face_recoginition_sample .
    d. After the build completes, tag your image so you can push the image to this repository:
        i. docker tag face_recoginition_sample:latest 630075306220.dkr.ecr.us-east-1.amazonaws.com/face_recoginition_sample:latest
    e. Run the following command to push this image to your newly created AWS repository:
        i. docker push 630075306220.dkr.ecr.us-east-1.amazonaws.com/face_recoginition_sample:latest

20. Create a Lambda function from the AWS Console and define a trigger so that it is invoked when a new object is added to S3.
21. Create role for which has the following permissions
    a. AmazonS3FullAccess
    b. AmazonDynamoDBFullAccess
    c. AWSXRayDaemonWriteAccess
    d. AWSLambdaBasicExecutionRole

22. Assign the role to the AWS Lambda function
23. Update input bucket name in Workload generator and run the to input bucket.
24. The controller code in the openstack VM will be monitoring the SQS Queues, whenever a new message is received in the input queue, lambda function will be triggered from the openstack VM.
25. The console of the openstack VM will be printing the input file names.
26. Whenever the output is available in the output bucket, SQS Queues will be notified, The controller code in openstack VM will download the csv file from the Bucket and print its content in the console.

5. Individual Contributions – Portfolio

5.1 Sai Vikhyath Kudhroli (1225432689)

Introduction

The objective of the project is set-up a hybrid cloud environment using AWS and OpenStack to develop an elastic application that can scale in and scale out on demand that extracts academic information about the student based on the video that comprises that student.

Solution

The application is supported by the private cloud set-up using OpenStack and resources from AWS. It consists of a virtual network and virtual instance created using openstack which triggers AWS Lambda to perform face recognition and return the results. It also comprises two S3 buckets, two SQS queues, Amazon ECR, Docker file, Amazon DynamoDB and a face-recognition module using python.

Design Contributions

- My significant contribution to this project is the design of the openstack set-up in AWS EC2 and the design of the private cloud in openstack.
- I designed the virtual network that will form the private cloud using openstack. I prepared the list of all security rules and interfaces that apply to the instance in openstack.

Implementation Contributions

- Launch an AWS EC2 instance with 8GB Memory, 60GB storage, hibernation option and appropriate security groups to later install openstack.
- Configured git in EC2 instance and AWS CLI on openstack instance.
- Cloned the devstack git repository which is then used to install openstack.
- Analyze different Ubuntu images that can support an instance on openstack. Downloaded Ubuntu 18.04 (Bionic Beaver) version using curl.
- Upload the Ubuntu image downloaded to openstack using glance.
- Create a private network by creating a virtual router and attaching it as an interface in a virtual network. The instance is then attached to this private network.
- Create a key pair for the instance which is used to SSH into the instance.
- Create a security group to allow all TCP traffic and ICMP traffic across the instance from all the devices to enable SSH connections and other traffic.

Challenges Faced

One major challenge faced in the project was to set-up a working openstack. Our team had two approaches, one to install openstack in a Virtual Machine hosted using VMWare/Virtual Box. Another option was to install openstack on an EC2 instance. I tried installing openstack on a virtual machine with 8GB memory and 50GB storage. But the issue was SSH was not working when the set-up was installed using a virtual machine.

New Skills and Technologies Acquired

- Utilizing openstack and AWS to set-up a hybrid cloud environment.
- Downloading and installing openstack using devstack.
- Set-up an instance in openstack using a custom image and set-up a virtual network in openstack and create security groups to allow traffic to integrate the private cloud to a public cloud.

5.2 Gautham Maraswami

Task: My responsibilities in this project is mainly to set up an instance inside openstack and make sure that all the required interaction with S3, SQS etc is working smoothly.

Setup in virtual box: I have tried to set up the openstack in the local machine using virtual box. I created a new instance with 6 GB Ram and 50 GB Disk inside a virtual box instance. Initially i faced issues because of trying to use ubuntu 20.04. After using ubuntu 22.04, installation of openstack was successful. I also tried configuring the ip address in local.conf file with the ip address of the network. The major issue here is the network ip inside the virtual box keeps changing after the timeout period making the openstack consoles inaccessible. Also as the disk and memory is limited inside the virtual box, we failed to launch an ubuntu/ cento OS systems in the local setup.

Setup in EC2: using the methods provided by the other teammate, I was able to replicate and setup the openstack on an EC2 instance. Inside this instance, I have experimented with modifying the volume of the created instance, demo vs admin role and its permissions. Ingress and egress configurations of the network group. I also worked on learning about Private, public and shared networks. I experimented with the host vs openstack resource utilization ratio.

Setup Nova Instance: using the image created by the teammate, I launched an instance inside openstack. I tried creating an instance of various flavors which can be used for ubuntu 18.04 image, we found out minimum 1 VCPU 4 GB ram and 10 GB Disk is necessary for reasonable performance of the instance. I also tried creating the instance using openstack cli.

Setup floating IP: created a floating IP using openstack cli, assigned it to the created instance. We made sure the assigned network has ICMP, SSH, and HTTP ingress and egress permissions.

Setup DNS: Even though i was able to ssh into the created instance, I was able to ping ip addresses, but running sudo apt update was throwing address resolution error. We tried changing the ubuntu mirror but the error persisted, after doing some research, we decided the error was in DNS resolution server address, changing which the solution was successful.

Setup Environment: I installed python, pip and boto3 to run the code of controller, but due to default python version in ubuntu 18,we faced compatibility issues with boto3. Reinstalling required python resolved the issues.

Learning from the project: How to setup openstack from scratch inside a linux environment. Network configurations and ip addressing for instance. Using Nova, neutron features of openstack for creating topologies. Defining firewall restrictions for access inside openstack. Using openstack cli for monitoring, testing and configuration of resources inside openstack.

5.3 Abhilash Subhash Sanap

Responsibilities

- Conceptualize the overall design of the application in a collaborative way with other teammates.
- Build a Docker image, push it to AWS Elastic Container Registry (ECR) and set up the Lambda function using a Docker image.
- Weighing the options for establishing the communication between public and private cloud and implementing the most suitable alternative.

Contributions

Architecture

- I set up the notification queues and defined the publishing of S3 upload events to these queues.
- I set up the private cloud locally using VMware Workstation and an Ubuntu image. (We used the private cloud set up using AWS EC2 for the final demo)
- I actively contributed to the group discussions to decide the configurations required for the project.

AWS Lambda

- I set up the AWS Lambda function with the necessary configurations so that it can be triggered from the Private Cloud.
- I monitored the CloudWatch Metrics like Invocations, Error rates, and Logs to ensure that the lambda function is working perfectly.

Communication between Private and Public Cloud

- The prime objective of the project is to run the application on a Hybrid cloud and the most important aspect of that is to establish a communication between the Public cloud realized using AWS and private cloud realized using Openstack.
- I wrote the controller code that acts as the bridge between the Public and Private cloud.
- I wrote the queue monitoring code that reads from the queue and deletes the message once it is processed.
- I wrote the code that invokes the AWS Lambda function to be executed on the Public cloud from the controller.

Learning Outcomes

- Learning about Openstack and its components like Nova, Neutron and so on.
- Setting up Openstack in Linux with an Ubuntu image
- Learning about Hybrid clouds and the communication between the Public and Private clouds.
- Using Platform as a Service modules like AWS ECR, and AWS Lambda, including various ways to define and invoke lambda functions, AWS Cloud Watch, AWS SQS and AWS S3.