CSE 546 — Project 3 Report

Prasidh Aggarwal, Revanth Suresha, Shriya Srinivasan

1. Problem statement

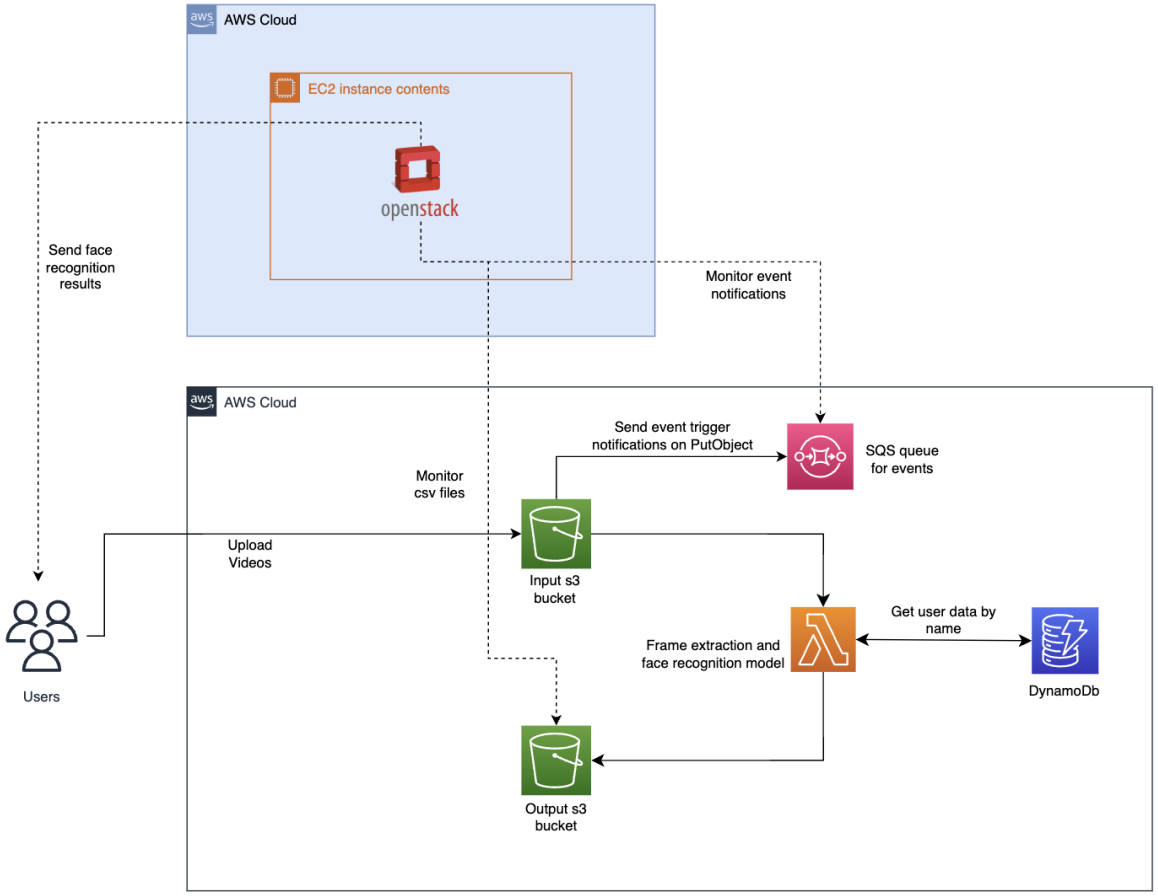
This project aims to deploy the elastic application developed in Project 2 into a hybrid cloud environment, leveraging both Amazon Web Services (AWS) and OpenStack. The project will build upon the previous implementation, which utilized AWS Lambda and other AWS services to create a scalable, cost-effective, and resilient application. The hybrid cloud environment will offer additional resources for the application to scale out automatically and on demand, ensuring optimal resource usage and cost-effectiveness.

The first part of the project will involve setting up OpenStack as a private cloud, either on a physical machine or a virtual machine in AWS EC2. DevStack will be used to deploy an OpenStack cloud from git source trees quickly. The minimum requirements for this step include a fresh Ubuntu 22.04 installation, 4GB RAM, 2 vCPUs, a hard disk capacity of 10GB, and an internet connection. Once DevStack is installed, users can access OpenStack Horizon and create a virtual machine to run their code, monitor S3 buckets, and trigger Lambda functions.

The second part of the project will focus on deploying the cloud app, which implements a smart classroom assistant for educators. The application will utilize AWS Lambda and other supporting services from AWS and OpenStack to provide users with a valuable and efficient cloud service. The app will perform face recognition on classroom videos, look up the recognized students in a database, and return relevant academic information to the user. The application will leverage the benefits of PaaS cloud technology to streamline development, deployment, and management. The project aims to enhance its scalability, resiliency, and cost-effectiveness by deploying the app in a hybrid cloud environment. The ultimate goal is to create a versatile and efficient cloud service that offers the benefits of both AWS and OpenStack, providing users with a valuable service while also serving as an opportunity to learn and hone skills in developing advanced cloud applications.

2. Design and implementation

2.1 Architecture



The major components of the architecture include:

1. Two S3 buckets, one each for video requests (uploaded by the user) and result responses.
2. A Private ECR to store docker images.
3. AWS lambda to create a lambda function triggered whenever the input bucket gets a new video uploaded.
4. AWS lambda trigger to connect the S3 put operation with the lambda function handler.
5. Handler of the lambda function that handles the bucket operations and the face recognition logic.
6. Dynamo DB to store student information.
7. OpenStack Nova, Neutron, and Glance.
8. An SQS queue to keep track of event notifications whenever something is pushed into the s3 bucket.

2.2 Autoscaling

AWS Lambda handles auto-scaling by automatically adjusting the number of concurrent function executions in response to incoming requests. This allows the service to maintain high performance and availability without manual intervention. When a Lambda function is triggered, AWS provisions one or more instances of the function to process the event. As the volume of requests increases, Lambda seamlessly scales the number of instances to accommodate the additional load. Behind the scenes, AWS Lambda manages capacity and resource allocation by utilizing "concurrency." Concurrency refers to the number of simultaneous executions of a Lambda function. When a function experiences a spike in requests, Lambda increases the concurrency level accordingly, creating new instances to handle the incoming events.

Lambda's auto-scaling capability ensures that the application remains responsive and performant even under high workloads without requiring developers to manage infrastructure scaling manually. This eliminates the need to pre-allocate resources or estimate peak demand, resulting in cost savings and optimized resource utilization. The automatic scaling of AWS Lambda ensures that the application benefits from the elasticity and efficiency of serverless computing, thereby enhancing the overall user experience.

2.3 Member Tasks

Prasidh Aggarwal (paggar10) – Efficiently executed the tasks of setting up the project infrastructure. Created an AWS account and defined IAM users with appropriate permissions, ensuring secure access to the required resources. Configured user roles for the Lambda function, establishing the necessary privileges for seamless operation. Launched an EC2 instance to serve as the infrastructure for our OpenStack. Created a user role that the s3 bucket can assume, to be able to push event notifications into the SQS queue. Created the required JSON policies for the s3 buckets and the SQS queue.

Shriya Srinivasan (ssrin103) – Wrote a python script that continuously polls the input SQS queue for incoming event notifications from the s3 bucket whenever a new video is added. Once the event is received, then processed it to retrieve the video information/name and then pass the required details to the lambda function and invoking it at the same time. Finally, added functionality to look for csv files corresponding to the uploaded videos in the output s3 bucket and print their respective outputs in the OpenStack console.

Revanth Suresha (rbangal5) – Handled the installation and setup of OpenStack with OVN using devstack. Configured the local.conf file with the right environment variables. Added a private subnet and the required routers for the correct network topology. Ran devstack to configure OpenStack. Installed the required libraries such as aws cli, boto3, python3. Troubleshooted any network errors/blockers during the setup of the OpenStack.

3. Testing and evaluation

After running Centos through OpenStack on an EC2 instance running Ubuntu, we tested the network, and compute configuration status using OpenStack Python client. Then we ran the Python code that read S3 push notifications from SQS and triggered the Lambda function with the message's contents. Monitored the Lambda function metrics and output S3 bucket to verify the end-to-end functionality. During the testing phase we encountered errors such as below:

- Not being able to ping or ssh into the cirros image (for testing).
- Not being able to ping or ssh into the ubuntu image due to credentials issues.
- Switched to the centOS image and faced issues with being able to SSH due to some private subnet networking issues.

After overcoming all the above flaws we were able to test the project end to end with our script to trigger the lambda function whenever an event notification is added to the SQS queue.

4. Code

4.1 *app.py*

This Python script is designed to work with Amazon Web Services (AWS) to process video files using face recognition technology. The script continuously listens to an AWS Simple Queue Service (SQS) input queue, waiting for incoming messages. Once a message is received, the script extracts the object key name from the message body and creates an event object containing the object key name as a parameter. The script then invokes an AWS Lambda function using the event object as the function payload. After the Lambda function is invoked, the script waits for a file with the same name as the input video file in an AWS Simple Storage Service (S3) output bucket. If the file is found, the script prints the contents of the file. If the file is not found, the script waits for some time before checking again. Once the file has been found, the script deletes the message from the SQS input queue. Overall, this script is designed to automate the process of processing video files using face recognition technology and provides an efficient way to handle large numbers of video files in an automated manner using AWS services.

5 OpenStack

5.1 Installation steps

The steps involved are:

1. Installing OpenStack with OVN using DevStack on an AWS EC2 instance.
2. Cloning the DevStack repository.
3. Setting environment variables in the local.conf file.
4. Running DevStack to install and configure OpenStack.

5. Once DevStack is running, the OpenStack command line tools can be used to interact with the APIs.
6. The remaining steps involve installing python3.6, aws-cli, and boto3 libraries on the OpenStack VM and running an app.py script to read messages from an SQS queue and trigger a Lambda function.