CSE 546 — Project Report

Hari Sai Charan Challa(1225461861)

Sathwik Katakam(1225445585)

Hari Murugan Ravindran(1222324688)

1. Problem statement

In this project, we are building an elastic application which is used as a smart classroom assistant for educators. Our application can Scale Out and In on-demand and cost-effectively in a Hybrid Cloud Environment. In this application, it takes the videos from the user's classroom as input then it performs face recognition on the frames extracted from these videos, identifies the student's name and retrieves the details of the identified student from the database and sends the academic records of all the students to the user. This application makes use of two resources namely Amazon Web Services for handling its public cloud infrastructure and OpenStack for handling its private cloud infrastructure.

2. Design and implementation

2.1 Architecture

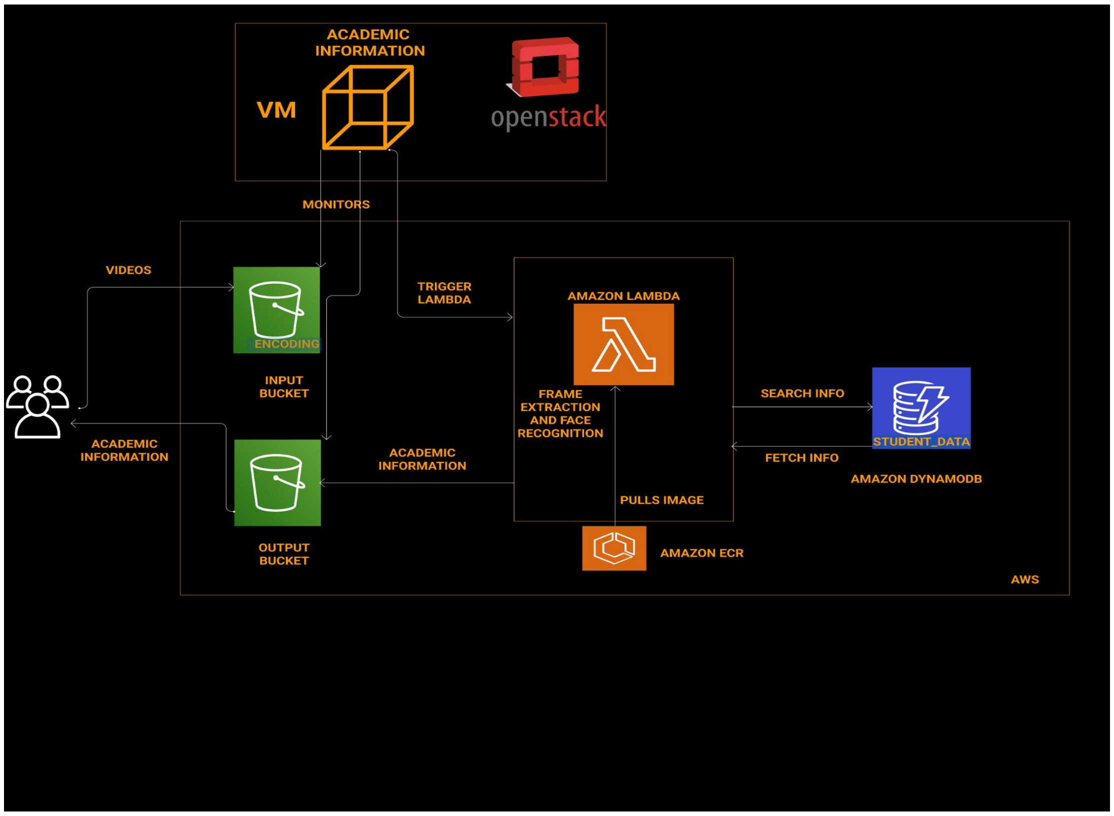

Figure 1

Description:

We begin this project by setting up two buckets in S3 which will act as our input bucket and output bucket. The Input bucket named "input-video-hsh" is used for receiving the input videos from the user and the Output bucket named "output-video-hsh" will store the output of the lambda function which gives the academic information of the students in the videos in the CSV format. In this Project, we are trying to build a hybrid cloud application. We use OpenStack for Private Cloud which will be hosted on our personal computer and AWS which will act as our Public Cloud. The Private cloud is hosted with the help of OpenStack which is connected to our two buckets in S3 with the help of two Queues. The input bucket is connected to the OpenStack through "event-sqs-queue" and the output bucket is connected to the OpenStack via "response_queue-hsh".The OpenStack is responsible for triggering the lambda to initiate the processing. All the Processing of the videos is carried out in our lambda function created with the help of the customized container image pre-installed with face_recognition and ffmpeg library. It is where the frame extraction and face recognition processing is done in our application. The Data in our application is handled at the rear end of the application by Amazon DynamoDB which is a NoSQL database service which contains the student's academic records. We set up the table named "student_table_two" for storing the academic records of the students. It is where the lambda function looks up the student's records who are identified from the videos and stores them in the output bucket. Then finally it is sent to the Command Line Interface of the VM on the OpenStack.

Our application execution can be divided into four major phases:

- Input Upload and Triggering Lambda Function.
- Input Frame Extraction and Face Recognition Processing.
- Querying Academic Records in DynamoDB.
- Storing the Results.

Input Upload and Triggering Lambda Function:

In this initial phase, the videos from the user are gets uploaded to the Input bucket "input-video-hsh".Here in this input bucket, we have created an event notification which is used to send a notification to the queue named "event-sqs-queue".The queue response is received by the OpenStack it gets the details of the bucket and it triggers the lambda function.

Input Frame Extraction and Face Recognition Processing:

In this phase, the trigger from the OpenStack invokes the lambda function named "image-function".This lambda function is helpful for us in utilizing the Function as a Service(FaaS) provided by AWS. We have created our lambda function by making use of the customized container image pre-installed with the face_recognition and the ffmpeg library. In this part of the application once the lambda is invoked by the OpenStack processing is started. Here the processing by the lambda is divided into two major steps. In the first step, the frame to be recognized is extracted and in the next step the extracted frame is recognized and finally, we get the name of the student in the video.

Querying Academic Records in DynamoDB:

The final part of the processing is carried out in this part it is where we get the academic record of the recognised student by using their name and querying the table which we have created on AWS DynamoDB.We will get the academic record from the DynamoDB and use them for generating our results.

Storing the Results:

Our application execution ends by pushing the generated result from the lambda into the Output Bucket named "output-video-hsh".Then finally the academic information is printed on the Command Line Interface of the VM on the OpenStack.

Usage of AWS Services:

- **AWS S3:**
  - We use two buckets in our Hybrid Cloud Application. The first one acts as the input bucket named "input-video-hsh" which is where the input from the user is uploaded. It also has an event notification feature assigned to notify the queue named "event-sqs-queue ".
  - The Next Bucket acts as the output bucket named "output-video-hsh" to store the output from the lambda.
- **AWS SQS:**
  - We are using two queues named "event-sqs-queue" and "response_queue-hsh" for setting up the communication between OpenStack and the AWS.
  - Once we get the event notification from the S3 bucket the VM instance in our OpenStack would invoke the lambda function.
- **AWS ECR:**
  - With Amazon ECR, we can take advantage of a fully managed container registry that provides high-performance hosting capabilities, enabling us to deploy application images and artifacts with ease and flexibility to any desired location.
  - We are building our lambda function on a customized container image pre-installed with face_recognition and ffmpeg library.
- **AWS Lambda:**
  - It is the processing part of our Hybrid Cloud application. We are utilizing the Faas Service of AWS so we need not worry about setting up servers, etc. Thus it is called Serverless Computing. It is also very effective in Scaling Out and In automatically.
  - In our application, the lambda function is triggered by the VM instance in Openstack.Once triggered the processing starts for frame extraction and face recognition.

    - Finally It is also responsible for fetching the students academic records from the Amazon DynamoDB,tabulating it to a CSV file and storing it into the Output bucket.
- **AWS DynamoDB:**
    - The NoSQL database is utilized for storing data in both key-value stores and document data structures.
    - In our application, we are storing the student's academic information on the table in DynamoDB which is then retrieved for the identified student which we will use for the Output report generation.

2.2 Autoscaling

The number of EC2 instances spun is automatically decided based on the volume of incoming requests. When a request is made, Amazon Lambda automatically generates a new instance of the requested function. If there are several requests made at the same time, more instances will be created to accommodate the increasing demand. Amazon Lambda tracks the rate of incoming requests and the resources needed for each request in order to dynamically alter the number of function instances to match demand. Amazon Lambda will automatically increase the number of instances to meet the increased demand when the concurrency or requests per second metrics go beyond a predetermined threshold, and it will lower the number of instances to maximize resource use when the load decreases. Reactive scaling is a part of this process, as is proactive scaling, which forecasts future demand based on current consumption patterns. We can also warm up the function instances with AWS Lambda's Provisioned Concurrency feature to avoid cold starts that can delay the process by quite a few seconds sometimes and make sure a function is always prepared to accept incoming requests.

2.3 Member Tasks

Hari Sai charan challa (1225461861)

My work in this project spanned mainly across setting up a VM box, setting up an Ubuntu virtual machine on a VM box, installing openstack on a Ubuntu virtual machine and finally setting up security groups and key-pairs using openstack dashboard for SSH access.

First, I have tried setting up openstack on AWS EC2 instances, but due to severe resource constraints the installation process didn't go through completely and we had to search for other ways to set up openstack. During this process, we found out about VM box and then tried setting up the project environment on VM box. Here, we have first set up an ubuntu VM on a VM box using the ubuntu operating system image. We have allocated 60 GB hard disk space and 6 GB RAM for the ubuntu instance. Then, once the Ubuntu VM was successfully set up. We installed openstack using devstack scripts shared in the course project description. I have also helped the team in setting up security groups and configuring ssh key pairs to login to the instance.

Sathwik Katakam (1225445585)

I played a significant role in designing and implementing a messaging system using Amazon Simple Queue Service (SQS) for communication between a private cloud and public cloud. The system involved Openstack VM as the private cloud, which communicated with S3 buckets through separate SQS queues for input and output. I was responsible for developing the SQS message receiver for requests, which constantly polled for messages and invoked Lambda triggers based on the received payload. Additionally, I ensured that messages were deleted from the queue after invoking Lambda to avoid repeated triggers. I also developed the SQS message receiver for responses, which extracted frames from video, recognized faces, queried Amazon DynamoDB for student data, and wrote the relevant output to the output S3 bucket with a corresponding message in the output SQS.

Hari Murugan Ravindran(1222324688)

I was responsible for finding and installing the Ubuntu Cloud Image as well as setting up the operating system on the virtual machine created using OpenStack. To achieve communication between the OpenStack Private Cloud and the AWS Public Cloud, I also created the event notification queue and a response queue. I also made sure that the VM instance has Python, pip, git, and boto3 installed as well as all other required dependencies. I was also responsible for setting up DynamoDB, creating the table and adding all academic records to the table. Before beginning the project, I read all the relevant information and set up the lambda function to retrieve the student's academic records from DynamoDB.Once all the testing was completed, I manually cross-checked to ensure that the students' names were correctly identified and that their corresponding academic records were saved in the output bucket as a CSV file.

3. Testing and evaluation

We tested the project functionalities in a modular fashion ensuring each individual module is working as expected. To ensure the working of lambda function, we first tested it against a custom event by uploading a single random test file to the S3 bucket. This triggers an event change and successfully invokes the client. We then created an image which contains the code and dependencies using docker and uploaded it to the ECR.

The following steps were followed to test and evaluate the system:

1. There are two test case folders that were given to us. One which contains 8 test videos and the second which contains 100 of them. Workload generator was run against these test cases.
2. We ensured that input videos are being uploaded to S3 bucket.
3. Verified that we poll correctly for input messages from SQS queue and invoke the Lambda function based on the events through OpenStack.
4. Verify that the lambda functions get triggered and invoke the client by automatically scaling based on the demand.
5. Ensure that the student json file is being uploaded to dynamo DB and is queryable based on the given partition key (student name).
6. Also ensure that response queue messages are consumed by OpenStack and also messages are deleted from SQS after processing it.
7. Verify that the output files written to s3 bucket match the count of input bucket and verify the contents of csv file generated for each test case against the expected output.
8. With the 100 test cases being uploaded sequentially to the S3 bucket, it took 2 minutes to process the requests and upload results to the output S3 bucket.

4. Code

- **app_tier**

  This module's purpose is to extract the output from the S3 output bucket and print it on the openstack instance console. We configured an event notification on the s3 bucket which sends a message to the queue when any object is inserted into the corresponding bucket. We fetch the message from the queue and extract required details like bucket-name and file-name from the message. We then download the output file using bucket and file names. Finally, we will be printing the contents of the file on the console and deleting the message in the queue.

- **web_tier**

  This module's purpose is to extract the input from the S3 input bucket and invoke lambda function setup during the second project with required parameters.We configured an event notification on the s3 bucket which sends a message to the queue when any object is inserted into the corresponding bucket. We fetch the message from the queue, invoke a lambda function called 'image-function' and finally delete the message in the input queue.

- **sqs_util**

  This program acts as an interface between SQS and other programs. It contains methods required to send, receive and delete messages SQS queues. We have also added error handling capabilities since network calls are being used underneath.

- **config.py**

  This is used to define all the required constants for the project like queue names, security credentials, bucket names etc.

Explain in detail how to install your programs and how to run them.

1. Setup openstack on a VM, local Ubuntu instance or AWS EC2 instance.
2. Then, download the ubuntu cloud image and launch an openstack instance using the ubuntu cloud image.
3. Configure the security groups and key pairs to allow SSH and Ping traffic into the instance.
4. Log in to the instance and install all the required dependencies like pip, git, boto3 etc.
5. Clone the git repository from the given link (https://github.com/hchalla2/cloud_project_3)
6. Change into the project directory and run **web_tier** python script and then execute **app_tier** python script. (python3 web_tier.py && python3 app_tier.py)
7. Now execute the workload generator (python workload.py)

Individual contributions:

Hari sai charan challa (1225461861)

Requirements analysis and Tasks division:

We have spent a lot of time to clearly understand the project requirements to avoid back and forth. Then, the entire project was divided into three tasks and I was majorly involved in installing openstack and setting up the ubuntu instance on openstack.

Setting up Openstack on AWS EC2 instance

Till now, we have used AWS services to completely build the project. Keeping that in mind, we have first tried installing openstack on AWS EC2 instances. EC2 provides only a few free tier instances . We have cloned devstack project onto the EC2 t2.micro instance and have tried installing openstack 5 times but due to severe resource constraints, the installation process used to get stuck. We then came to the realization that we needed a better instance like t2.large to successfully complete the installation process.

Setting up VM Box:-

Since the AWS EC2 way didn’t work as expected we had to find a new way to complete the installation. So for that, we installed VM Box software to set up a Ubuntu VM on top of the host windows platform. There are many virtual machine technologies like VMware, QEMU etc but ease of use and setup have made us choose VM box software. Oracle VM VirtualBox is a type-2 hypervisor for x86 virtualization developed by Oracle Corporation.

Setting up Ubuntu on VM box:-

After downloading the Ubuntu image from the internet, we have set up the Ubuntu virtual machine on a VM box. We have allocated 60 GB hard disk space and 6 GB RAM for the ubuntu instance. These high numbers are to avoid resource constraint related issues during openstack installation.

Installing openstack on ubuntu instance

After the Ubuntu VM setup, to install openstack on instance we have used instructions given in the project description. First, we have created a user called ‘stack’ and associated necessary permissions to it. Then as a ‘stack’ user, download devstack repository and have then invoked installation scripts to completely install openstack software.

Setting up security groups and key-pairs

I have also helped the team in setting up security groups and creating required key-pairs using the openstack dashboard. These are essential for SSH access. SSH access on an instance is required for many purposes, installing the required dependencies, running relevant python project scripts etc.

Testing

Once the code implementation and instance setup is done, I have tested my setup many times by changing several parameters like Disk size, RAM, difference openstack instance flavors etc. This has been done to avoid any surprises during the project demo session.

Sathwik Katakam (1225445585)

Design:

I was majorly involved in the design and implementation of receiving and requesting messages from SQS queue. After several brainstorming sessions, we have decided to use Amazon Simple Queue Service (SQS) as the means to communicate between the private and public cloud. Since SQS was already used in Project-2, it required minimal changes to configure and setup architecture for this project.

Implementation:

Openstack VM acts as the private cloud that communicates with S3 buckets through separate SQS queues for input and output. It constantly polls for messages in the SQS and invokes Lambda trigger based on these messages. The web tier uploads files to input S3 bucket and also correspondingly also pushes a message to SQS.

1. SQS Request Message Receiver:
    a. There is a main thread that runs to poll messages from the input queue (event-sqs-queue). Once there is any message in the SQS, we retrieve the corresponding S3 key from the message and pass it as the payload to invoke the Lambda function (lambda_client.invoke()). The invocation happens asynchronously so that the function doesn't wait for any response in return.
    b. We also need to ensure that once we invoke the Lambda for a message in SQS, it should be deleted from the queue so that we don't trigger the function repeatedly. This is done using the delete_method() function.
2. SQS Response Message Receiver:
    a. There is a separate thread that runs to poll messages from the output queue named - 'response_queue-hsh'. The app-tier extracts frames from the video and recognizes faces from this using the face-recognition image provided, similar to Project-2. We then query the Amazon DynamoDB to get the student data and write it back to the output S3 bucket, with a corresponding message in the output SQS. These messages get triggered when there is a put event triggered in the S3 bucket.
    b. Once there is a message in the output queue, it downloads the corresponding file from the output bucket and prints the relevant output in the private cloud terminal. Similar to the input queue, we delete the message after processing from the output queue.

Testing:

Each individual module above was tested using unit tests. Similar to other projects, we tested individual modules such as pushing and receiving messages from SQS and invoking Lambda functions based on these events. After this, I integrated the code with other parts and then did end-end testing. We ensured that the number of files in the output S3 bucket and the output matches with content printed on the OpenStack terminal and the requests being sent to the input S3 bucket.

Hari Murugan Ravindran(1222324688)

Design:

We have studied and discussed how to implement this Hybrid Cloud Application for recognizing student faces from the input video by extracting the frame and also to retrieve their academic records. I was responsible for the Operating system setup on the Virtual Machine initiated on the OpenStack(i.e)Downloading and installing the Ubuntu Cloud Image, I have also set up the event notification queue as well as the response queue responsible for the communication between the OpenStack on our personal computer which acts as the Private cloud and the AWS which acts as the Public Cloud. In addition, I was also making sure that all the necessary dependencies are installed on the VM instance like Python, pip, git and boto3. Finally, I was also responsible for setting up the DynamoDB, Table Creation, and Uploading all the necessary data (i.e.) academic records in the Table and the final lambda function part where the student academic record is fetched from the DynamoDB.I have also gone through all the docs, videos, blogs and materials related to OpenStack and AWS Services like DynamoDB, Cloud9, S3, ECR and Lambda before working on this application.I can say that I was responsible for the communication between the OpenStack(i.e)Private Cloud with the AWS(i.e)Public Cloud.I was also responsible for Students Academic Records Data Storage and Data Retrieval from DynamoDB.

Implementation:

- The Base task is Setting up the necessary communication between Public Cloud(i.e.) AWS and Private Cloud(i.e.) OpenStack.It is possible with the help of the Queue named "event-sqs-queue" and "response_queue-hsh".
- Once the OpenStack is installed the download and installation of the Ubuntu Cloud Image is done. Also the necessary dependencies on the VM like Python, pip, git, and boto3.
- Using the AWS Cloud9 I have set up the table in the DynamoDB and loaded the given JSON file to our created table.
- Then once the name of the student is identified using the face recognition part the student's academic records are fetched from the DynamoDB.
- Then the Details are Stored in the CSV format as a file for each of the recognized students and saved in the Output Bucket.

Testing:

There were various phases of testing involved in integrating DynamoDB for the application.

1. **Unit-testing:** Initially all the individual components of the application are checked. The Two queues responsible for the communication between the private and public cloud, the DynamoDB for storing the academic records are checked for their functioning. The Application communication between clouds and the DynamoDB fetching and result generation is checked individually.
2. **Integration-testing:** The three major integrations are the Endpoint of the Lambda which searches the DynamoDB, the communication queue between the public and the public cloud and also the final link to the output bucket. They are checked and verified whether there are no anomalies in their functioning.

3. **End-end testing:** This phase is used for testing the endpoints in the application where the event notification occurs once there is an object in the bucket, working of the queue, DynamoDB data fetching, generation of the report as well as uploading the final results to the output bucket and finally print the academic information on the VM CLI on the OpenStack.