CSE 546 — Project Report 3

HYBRID CLOUD

Shubham Chawla - 1227415724

Anurag Banerjee - 1225497506

Vedant Munjaji Pople - 1225453164

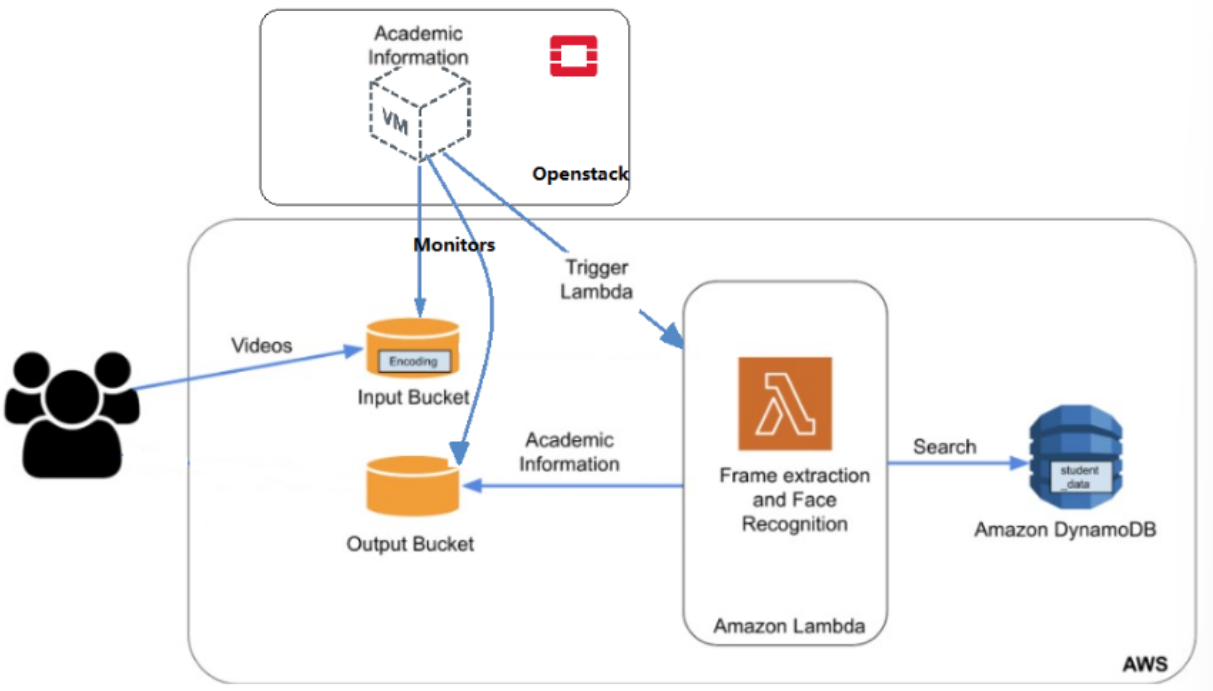
1. Problem statement

In this project, we aim to make an elastic application that can scale in and scale out cost-effectively into the Hybrid cloud environment. This project will be made using resources from AWS and OpenStack. AWS provides a variety of computing, storage, and massage services. OpenStack is a free, open standard cloud computing platform. The application provides face recognition services to users.

2. Design and implementation

2.1 Architecture

The architecture of the project can be seen below. The user uploads videos from the classroom to the input AWS S3 bucket. The bucket is monitored by OpenStack. The academic information, input bucket, and output bucket are monitored from OpenStack. After the upload into the Input bucket is complete, the OpenStack VM will trigger the Lambda function. The Lambda function will then perform frame extraction and face recognition. The lambda function then completes the recognition and uses data from the Dynamo DB to get the classification result. The lambda function returns the output from the classification and the result to the output bucket. This is monitored by OpenStack as well.



2.2 Autoscaling

The autoscaling of the function is overlooked by the Lambda function. The lambda function itself can scale in and scale out when needed. Auto-scaling of this application is looked after by AWS Lambda. AWS Lambda gets many in-flight requests, for each concurrent request. AWS Lambda provisions a separate instance of the execution environment. As the function receives more requests, the Lambda function handles scaling by increasing the number of execution environments until the maximum concurrency limit of execution environments is reached.

2.3 Member Tasks

Anurag Banerjee:

Anurag was involved in setting up the OpenStack for this project. First of all, this involved setting up the basic environment. Anurag created an EC2 t2.xlarge instance to host OpenStack cloud OS. Then he set up OpenStack on this EC2 instance. Furthermore, Anurag set up a VM via OpenStack Horizon UI with CentOS 7 as the OS. Moreover, he managed the configurations so that one could SSH into the VM from the host machine and ping the web from the VM. Anurag also updated and downloaded the required dependencies on CentOS like Python which would be required to run the scripts to monitor the S3 bucket.

Anurag coded the handler.py file for the Lambda function, which is the essence of the complete project. This file will break the input video into frames, then perform image recognition on

these frames. Anurag created a method The method gets the video name and makes a session using Amazon credentials. The method that forms the directories for taking in videos. The images list is created and all the image names are added to it. If the image name is valid, the face_recognition library works and recognizes it. The recognized name is then searched for in the DynamoDB and the results are written to the classification S3 bucket

Anurag also helped to containerize the part for the handler which would we pushed to an ECR from where it was picked up as the base for the Lambda function running on AWS. This involved certain steps like writing an python script, that contains a set of instructions to be executed when the Lambda function would run. Hence, Anurag was the main pillar of the project and looked after the complete execution of the project.

Shubham Chawla:

Shubham's contribution to this project was significant, as he developed a critical component that allowed the project to function seamlessly. The project involved using OpenStack, an open-source cloud management software, to trigger Amazon Web Service's Lambda Function and initiate facial recognition processes.

The team followed a similar pattern to the previous project, which involved breaking down videos into frames using the Python face recognition library. The frames were analyzed for facial recognition, and the recognition results were used to obtain information from Dynamo DB about the recognized faces. However, the team used OpenStack for Lambda triggering and analysis in this project instead of AWS.

Shubham's expertise came into play when he wrote a Python script that ran inside the OpenStack-managed VM on AWS's t2.xlarge instance. The script polled the input S3 bucket and triggered the Lambda function using AWS's Python SDK, allowing the team to complete the facial recognition process.

The rest of Shubham's team members ensured that the Lambda function and other AWS services were synchronized to deliver the final results. The project's success highlighted the benefits of a Hybrid Cloud and how it can help organizations use cloud resources effectively and efficiently.

Vedant Pople:

In this project, Vedant primarily worked on setting up and managing the AWS services part of the project. He started working on setting up the S3 buckets. There are 2 AWS S3 buckets. One

of them is the Input bucket that gets the input from the workload generator. The other S3 bucket is used to store the outputs of the recognized images.

Vedant configured Amazon Lambda to recognize the images from the input bucket. The lambda function uses a docker container image to recognize images from the input. The recognized results are stored in the output bucket. The recognized results are being searched for in the DynamoDB, where Vedant entered the tables containing the academic information of the students. The results along with the academic information about students are stored in the output bucket and then presented to the user.

Apart from this Vedant was instrumental in making the documentation of the complete project and README file that describes the abstract, member contributions, and AWS credentials for the resources used in the project like S3 buckets, Lambda function, and DynamoDB.

3. Testing and evaluation

The code has been tested extensively on the local machine. The handler function is tested with the student_data json file locally. The handler function could use the manual input from the test cases provided. Videos from these inputs were parsed using the ffmpeg library. The still frames obtained were recognized using the recognition function. The recognition function was then tested with a variety of images generated. Iterative changes were made to the function to improve its performance. The ability to fetch data from the json was also tested with recognition.

4. Code

Setup: The most elemental part of the project was the setup. This included setting up OpenStack. OpenStack is set up on a virtual machine. This project uses CentOS as the base OS to run the project. Hence firstly, Openstack was set up on an EC2 instance. Then using OpenStack, we were able to SSH into the CentOS.

Trigger.py: The trigger file contains the code for triggering the lambda function. It takes imports from boto like s3 connection, s3 key, and aws lambda layer. All the constants like table name, access key if, secret key, video upload bucket, lambda function name, and region for the AWS resources are defined. The trigger is defined initially, establishing a connection with the AWS Lambda function, with the given key. The AWS S3 bucket client and lambda client are made. While there exist some entries in the input bucket, the keys are fetched. The contents for each

response object are extracted from the received response. The lambda function is invoked with the key given. The response to this invocation is received from the invocation and stored in a log file. Then the key is used to delete the object from the upload bucket.

Output.py: The output file takes the output from the lambda function stored in the output s3 bucket. The necessary imports are made at the beginning. The constants for the table name, access key id, secret key, classification result bucket, and region for these AWS resources are defined. A method to poll the S3 video classification bucket is defined. The S3 client is made using predefined constants for credentials. While there are objects in the S3 bucket. The response object is received, and the contents of the response object are used to extract the key from the contents. This key is stored in the log file and then, the key is deleted.

Handler.py: The handler.py file is the most important file in this project. This file contains the logic for the classification of the images formed by the ffmpeg library. First of all, all the required packages are imported. This includes packages like boto3, face_recognition, pickle, os, numpy, and urllib. Then the constants like the Input bucket, Output bucket, the path to the files, and image extensions for the generated images are defined. A method, open_encoding is made, this opens the file, and the pickle file for the model is loaded in it. The file is then closed and the pickled model is returned. The next method generates a response from the entered student data json file. The next method, face_recognition_handler is responsible for the recognition of faces. The method gets the video name and makes a session using Amazon credentials. The method then forms the directories for taking in videos. The images list is created and all the image names are added to it. If the image name is valid, the face_recognition library works and recognizes it. The recognized name is then searched for in the DynamoDB. The student_data table uses a recognized name as the key and searches for details in the table. These details are then put into the Output bucket.

MCS PORTFOLIO:

ANURAG BANERJEE

Anurag's contribution to the OpenStack project was critical to its success. He was responsible for setting up the basic environment and creating an EC2 t2 extra large instance to host the virtual machine. With his expertise, Anurag successfully set up CentOS on the instance and was able to SSH into the virtual machine from the host machine.

After setting up the basic environment, Anurag downloaded and updated the required dependencies, including Python and the face_recognition library, on CentOS. This ensured that the project had all the necessary tools to perform image recognition on the frames of the input video.

The handler.py file for the Lambda function was the backbone of the project, and Anurag coded it to perfection. This file broke the input video into frames and then performed image recognition on these frames. The method Anurag created for this function involved getting the video name and making a session using Amazon credentials. The method also formed the directories for taking in videos and created an image list, to which all the image names were added.

When the image name was valid, the face_recognition library worked and recognized it. The recognized name was then searched for in the DynamoDB. Anurag's expertise in programming and his understanding of the Amazon environment played a crucial role in the successful execution of the project.

Anurag's contribution did not stop there; he also contributed to dockerizing the complete application to feed it to the Lambda function. This involved writing an entry bash script containing a set of instructions to be executed when the Lambda function runs. Anurag's attention to detail and technical expertise ensured that the application was correctly dockerized and executed without any issues.

In summary, Anurag was the main pillar in the OpenStack project and looked after the complete execution of the project. His technical expertise and attention to detail played a crucial role in the successful implementation of the project. His contributions in setting up the basic environment, coding the handler.py file for the Lambda function, and dockerizing the complete application were vital to the project's success. Anurag's commitment and dedication to the project made him an essential member of the team, and his contributions were invaluable.

MCS PORTFOLIO:

SHUBHAM CHAWLA:

Shubham's contribution to this project was significant, as he developed a critical component that allowed the project to function seamlessly. The project involved using OpenStack, an open-source cloud management software, to trigger Amazon Web Service's Lambda Function and initiate facial recognition processes.

The team followed a similar pattern to the previous project, which involved breaking down videos into frames using the Python face recognition library. The frames were analyzed for facial recognition, and the recognition results were used to obtain information from Dynamo DB about the recognized faces. However, the team used OpenStack for Lambda triggering and analysis in this project instead of AWS.

Shubham's expertise came into play when he wrote a Python script that ran inside the OpenStack-managed VM on AWS's t2.xlarge instance. The script polled the input S3 bucket and triggered the Lambda function using AWS's Python SDK, allowing the team to complete the facial recognition process.

The rest of Shubham's team members ensured that the Lambda function and other AWS services were synchronized to deliver the final results. The project's success highlighted the benefits of a Hybrid Cloud and how it can help organizations use cloud resources effectively and efficiently.

In conclusion, Shubham's contribution was invaluable to the project's success, and his expertise in developing the OpenStack script helped the team execute the facial recognition process seamlessly. This project demonstrated how the Hybrid Cloud could leverage different cloud computing technologies to achieve project objectives effectively.

MCS PORTFOLIO

VEDANT MUNJAJI POPLE:

As a member of the project team, Vedant's primary role was to set up and manage the AWS services used in the project. Specifically, his focus was on setting up the S3 buckets and configuring Amazon Lambda to recognize images from the input bucket. This task involved working with multiple AWS services, including S3, Lambda, DynamoDB, and Docker. To begin, Vedant created two S3 buckets in AWS. The first bucket, known as the Input bucket, was designed to receive input from the workload generator. The other S3 bucket was used to store the outputs of the recognized images. These two buckets were essential components of the image recognition process, which relied on Lambda functions to detect and recognize images.

Once the lambda function had recognized the images, the results were stored in the output bucket. Vedant then set up a DynamoDB database to store information about the students, such as their academic information. The recognized results were searched for in the DynamoDB database, where he entered the tables containing the academic information of the students. This allowed for the results to be linked to the students who were responsible for the images being recognized.

Finally, the results along with the academic information about students were stored in the output bucket and then presented to the user. This was achieved by setting up a user interface that displayed the recognized results and the academic information about the students in a clear and concise manner. This enabled the user to easily understand the results and their relevance to the students.

Apart from this Vedant was instrumental in the documentation of the complete project. Vedant also made the README file containing the abstract idea of the project, the member's contribution to the project, and AWS credentials of the input bucket, output bucket, dynamo database, and lambda function.

Overall, Vedant's role in the project involved setting up and managing the AWS services used in the image recognition process. This involved working with multiple AWS services, including S3, Lambda, and DynamoDB. By setting up the necessary infrastructure and configuring the services to work together seamlessly, he was able to enable the image recognition process to occur quickly and efficiently. This helped to improve the accuracy and efficiency of the overall project, while also enabling the user to easily understand and interpret the results.