CSE 546 --— Project Report

Aditya Goyal (1225689049)

Anshul Lingarkar (1225118687)

Vibhor Agarwal (1225408366)

General requirements:

- Strictly follow this template in terms of both contents and formatting.
- Submit it as part of your zip file on Canvas by the deadline.

1. Problem statement

Create a private cloud using Openstack which will host the Web-tier application. The App-tier for the application is hosted on AWS itself which makes use of AWS EC2, SQS, and S3 services.

The user will upload multiple images on the application running on Web-tier Instance running on Openstack and will be able to get the image processing result for each image he/she has uploaded.

The request from Web-tier will be sent to App-tier which is running on AWS, it will process the image and return the result back to Web-tier running in Openstack.

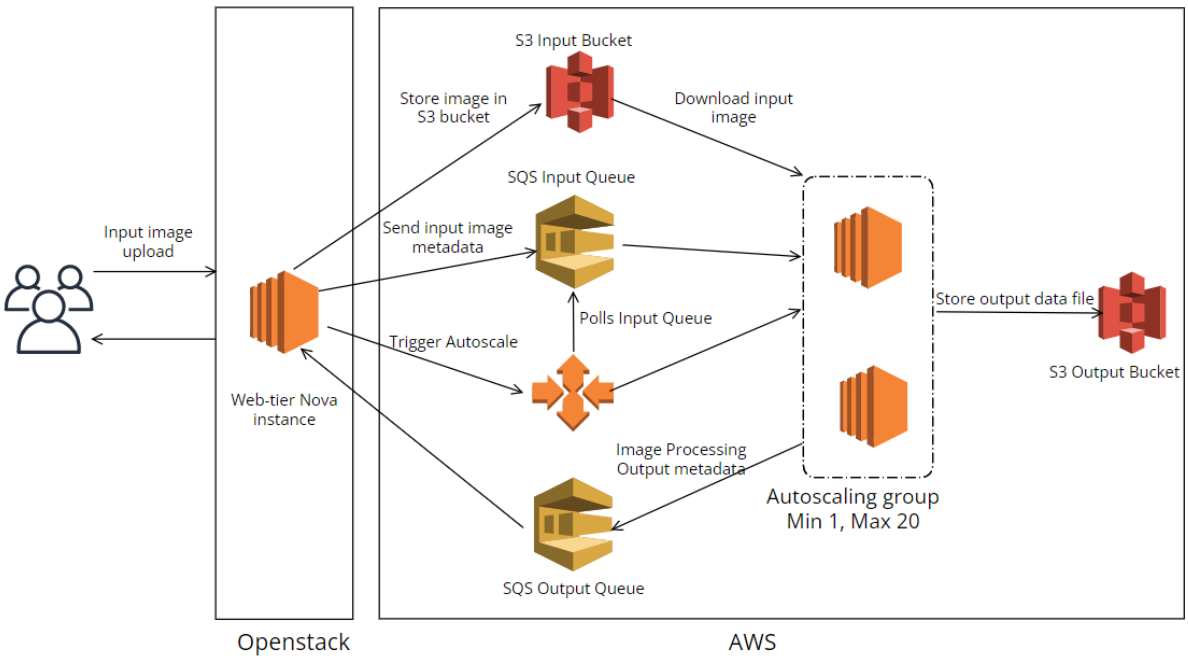
Once the user uploads images on Web Tier, he/she needs to get the result for all the uploaded images without any loss.

Along with this, the user needs to get the result in a minimum acceptable time frame, as well as the App Tier program needs to handle all the requests of the user. For this, we are using the concepts of Autoscaling in EC2, to meet the demands of the user. This is important because users should get the results as quickly as possible, and our architecture should be able to handle the demand in real time.

Web-tier sends the request to auto-scale the App-tier on AWS to meet the demand.

2. Design and implementation

2.1 Architecture



Openstack services used in the project:

1. Nova

   We are using Openstack Nova instance to create a virtual machine in Openstack for implementing our Web Tier. Web Tier will handle all user requests to accept the images uploaded by the user and display the result for image processing on the images uploaded by the user by listening to SQS Queue in AWS.

2. Neutron

   We created a custom network, gateway and router to configure networking part for our Web-tier.

AWS Services used in the project:

1. AWS EC2

   We are using AWS EC2, to create a virtual machine in the Amazon cloud for implementing our App Tiers. App Tier will process the images uploaded by the user, run the NLP algorithm, and store the result in the S3 Output bucket after processing the image and push the output metadata to the SQS.

2. AWS SQS

   We are using SQS Queue for storing input image metadata and NLP processed output data. SQS decouples our architecture components and allows asynchronous processing of concurrent

requests. SQS helps us to address the process of autoscaling where we can scale in and scale out based on the number of messages in the queue and ensure no request from the user is lost.

3. AWS S3

   AWS S3 provides reliable storage services. We are using AWS S3 to store the input images uploaded by the user and the result of image processing for each of the images uploaded by the user.

2.2 Autoscaling

Scaling in and out for the App Tier EC2 instances is managed by the Web Tier from openstack. We are programmatically managing the scaling in and out for App Tier instances. The logic to determine the number of instances is as follows:

Scale Out:

If number_of_messages_in_queue > number of running EC2 instances, then

    Provision a new instance periodically based on a set time interval

Scale In:

If number_of_messages_in_queue = 0 and number of running EC2 instances > 0, then

    Shutdown an instance periodically based on a set time interval

Once the user uploads the image on Web Tier running in openstack, web-tier will upload the images to S3 bucket in AWS.

And the web-tier will also trigger the request to start the App tier which is on AWS, for performing the image processing task on the images uploaded by the user, which are available in the Input S3 bucket.

Web Tier will send the user requests to the Input SQS Queue on AWS and based on messages available in the Input SQS Queue, we are triggering the Scale-out of the EC2 instances for App Tier.

We need to ensure that our logic does not create App Tier instances beyond 20. Our aim is to launch EC2 instances for App Tier so that all our messages in the SQS Queue are consumed and the result after processing is stored in the Output S3 bucket.

Scale in logic is also handled by the Web Tier running in Openstack, so that once all the messages from SQS Queue have been processed by App Tier, then our Web Tier will successfully stop the App Tier instances. This is done after the App Tier has processed the image and stored the result of image processing in the S3 bucket.

2.3 Member Tasks

Aditya Goyal (1225689049)

1. Developed a Flask Application containing Web-tier's code and deployed it on an EC2 instance.
2. Developed the code to connect to the S3 server and upload the user-provided images to it.
3. Wrote the API endpoint to upload the image to the S3 server.

4. Developed the code to connect to the SQS service and push the image metadata to the input queue.
5. Developed a listener that connects to SQS, listens to incoming messages from the output, and displays the output of image processing in the application console.
6. the networking setup for the Nginx server for Web-tier's EC2 instance enabling the users to access the application from anywhere on the internet.
7. Created the web-tier instances and wrote the code to export the AMI to the S3 bucket in 2 different formats (.bin and .vmdk).
8. Created role-policy.json and trust-policy.json to enable the vimport user to read from EC2 AMI and export it to S3.
9. Created the network router, gateway, interface for the web-tier Nova instance.
10. Tested web tier with different test cases.

Anshul Lingarkar (1225118687)

1. Developed the script to instantiate and scale out EC2 instances based on the number of messages in the queue.
2. Developed the code to scale in EC2 instances by stopping them if the number of messages in the queue is 0.
3. Developed the script to startup a pre-configured EC2 instance with App-tier's code already running.
4. Configured Nginx server to enable reverse-proxy services upon Application startup.
5. Set up the Virtual box with the ubuntu image.
6. Configured SSH for the AMI installed in Openstack to run the web-tier instance.
7. Set up the workload generator to test the web-tier and app-tier instances.
8. Created the security groups for Nova instance.
9. Created the Project report for Submission.

Vibhor Agarwal (1225408366)

1. Developed a Flask application containing App-tier's code (NLP algorithm) and deployed it on an EC2 instance.
2. Developed the code to connect to SQS and push the output message to SQS after processing the image.
3. Developed the code to create an output file using python script and write the NLP algorithm's output in it.
4. Developed the code to connect to S3 and upload the output file after processing the image to S3.
5. Completed the networking setup for the Nginx server for App-tier's EC2 instance enabling the users to access the application from anywhere on the internet.
6. Installed openstack in ubuntu and did the initial configuration for installing open stack by making changes to the requirement-constraints file.
7. Created the web-tier instance using Nova by first importing the downloaded AMI as an image in openstack and then launching an instance out of it.
8. Did the setup for floating IP for Nova instance so that it is accessible publicly.
9. Monitored AWS resources (S3, SQS, and EC2) to record the testing results.

3. Testing and evaluation

Using the workload generator provided for the project, we were able to send multiple requests to the Web Tier running in openstack.

Upon receiving the requests from the user, Web tier uploaded the images in the S3 bucket in AWS, and adds the messages in SQS Queue in AWS. Based on the messages available in the SQS, Web Tier initiated the request for launching the App Tier instances to process the messages available in SQS.

For the first stage of our application, we set up Web Tier on openstack and uploaded the images to the S3 bucket. We created an Input bucket for S3, to store all the images uploaded by the user on Web Tier. We then created input SQS and added the messages in the Queue after the images were uploaded to the S3 bucket.

Based on our testing, these are the results for the different number of requests:

# Of Concurrent Requests	Time taken to process the request (This includes time to instantiate app-tier instances)
10	2.3 minutes
50	4.5 minutes
100	6.7 minutes

4. Code

Code files submitted in the zip file:

1. app.py (web-tier) – running on Openstack
2. auto_scale_app_tier.py (auto-scaling)
3. app.py (app-tier)

uploads directory Web-tier:

uploads directory in root directory of web-tier application is used to temporarily hold input image file while uploading to the s3 bucket.

Web-tier app.py file:

1. index() – this method provides the server running status.
2. upload() – this method handles the HTTP post request as well as uploads the file to the input S3 bucket. It also sends the image metadata to SQS.
3. send_message() – this method is called by upload method and is used to send the image metadata to SQS.

4. get_queue_size() – this method returns the size of the queue
5. receive_queue_message() – this method listens to the output queue and receives any incoming message containing the output for the uploaded image.
6. app.run() – this will start the flask application.

Auto Scaling - auto_scale_app_tier.py file :

1. get_input_queue_size() – this method returns the size of the input queue
2. totat_app_instances_running() – this method returns the number of running/pending state EC2 instances.
3. start_instances() – this is used to start an EC2 instance based on the tag value in the image.
4. closeEC2Instance() – this is used to close an EC2 instance based on the tag value in the image.
5. auto_scale_instances() – this is used for scaling in and scaling out the instances based on the number of messages in the SQS.
6. create_instance() – this is used to create an EC2 instance.

uploads directory app-tier:

uploads directory in root directory of app-tier application is used to temporarily hold input image file downloaded from S3.

App-tier app.py file:

1. download_file() – this is used for testing the download file from S3 using an end-point.
2. downloadFileFromS3() – this is used for downloading a file from S3.
3. createOutputFile() – this is used for creating an output file after processing the image using the provided NLP algorithm.
4. get_queue_size() – this is used for getting the size of the input queue
5. process_image() – this is used for processing the input image and getting the output after processing from the NLP algorithm as well as uploading the output to the S3 server.
6. receive_queue_msg() – this is used for receiving the output message from SQS and displaying it in the user console.

Steps to setup Openstack:

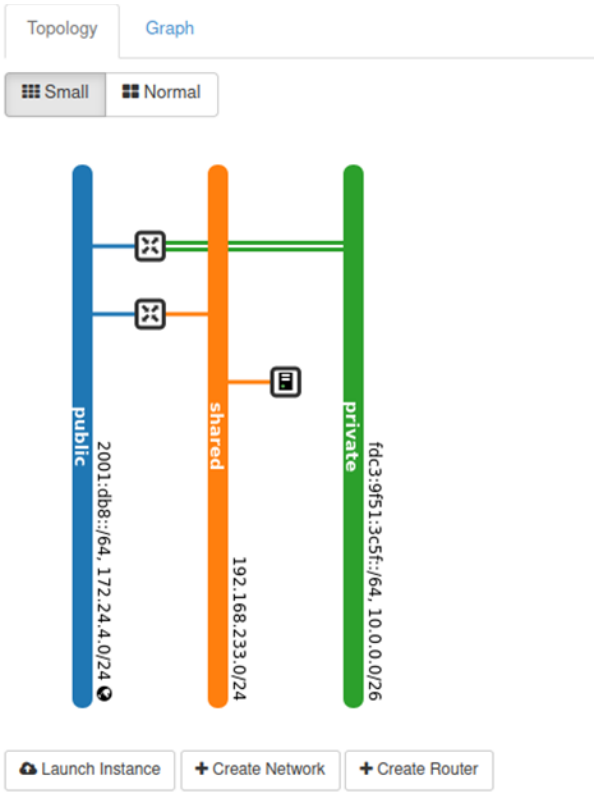
1. Setup Ubuntu 20.04 version on Virtual Box.
2. Configure three network adapters for this Ubuntu OS in virtual box.
3. Install net-tools and update apt-get in Ubuntu.
4. Install Git using command - sudo apt install git -y
5. Download Devstack Scripts:
   git clone https://git.openstack.org/openstack-dev/devstackLinks to an external site.
6. Create devstack configuration file, by creating a local.conf file and adding following details to this file –

```
[[local|localrc]]
```

```
ADMIN_PASSWORD=secret
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
HOST_IP=<HOST IP OF SYSTEM>
HOST_IPV6=<HOST IPV6 OF SYSTEM>
```

7. Install Devstack: using command –
   ./stack.sh
8. This will install the following openstack components – Horizon; Nova; Glance; Neutron; Keystone.
9. Import the downloaded AMI as an Amazon Machine Intteface image using the Openstack horizon dashboard.
10. Once the image is successfully imported, create a new network for it.
11. Create a new router that will be attached to the network created in step 10.
12. Create an interface and attach it to the router.

Your network topology should look like this :



13. Once the new network is created, go to the Instances tab in the Horizon dashboard
14. Click on launch a new instance and use flavor as de512 which will have 512mb RAM and 5gb space.

15. Attach the newly created network to the instance before launching.
16. Now launch the new instance. We need to assign a floating IP to it in order to make our instance publicly accessible. Use the assign floating IP option for that.
17. Now use the IP to SSH into your instance and start the web-tier.
18. Install and configure the nginx for reverse proxying in the web-tier.

Steps to setup and run Web-tier app.py file on Openstack:

1. install python3 preferably 3.8, pip3
2. create a virtual environment, python3 -m .venv venv
3. activate the virtual environment: source .venv/bin/activate
4. install flask using the command: pip3 install flask
5. install Nginx server
6. configure nginx server using the following configuration in the path

   ./etc/nginx/sites-enabled

   ```
   server{   list
     en 80;

     server_name AMI IP;
     location / {

        proxy_pass http://127.0.0.1:8000;

     }

   }
   ```

7. start nginx
8. run flask application server using the command: flask run -h localhost -p 8000
9. Check the server status using the following end point: http://{AMI_IP)}/
10. Use the following web service to upload the file : End Point URL : http://{AMI_IP}/api/v1/upload

    Type: POST

Steps to run auto_scale_app_tier app.py file :

1. Run the following command to start the auto-scale service: python3 auto_scale_app_tier.py

Steps to run App_tier app.py file:

1. install python3 preferably 3.8, pip3
2. create a virtual environment, python3 -m .venv venv
3. activate the virtual environment: source .venv/bin/activate
4. install flask using the command: pip3 install flask
5. install Nginx server
6. configure Nginx server using the following configuration in the path

./etc/nginx/sites-enabled

```
server{   list
   en 80;
   server_name AMI IP;
   location / {
      proxy_pass http://127.0.0.1:8000;
   }
}
```

7. start nginx
8. run flask application server using the command: flask run -h localhost -p 8000
9. Check the server status using the following end point: http://{AMI_IP)}

Google Drive Link for recorded video :

https://drive.google.com/drive/u/1/folders/1Wb3moENyaGnNWBwcQPsmc54dv0HvZwLo