

CSE 546 --- Project Report

Aditya Goyal (1225689049)

Anshul Lingarkar (1225118687)

Vibhor Agarwal (1225408366)

General requirements:

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

1. Problem statement

Using AWS IaaS to create an Image Processing application for processing images uploaded by the user.

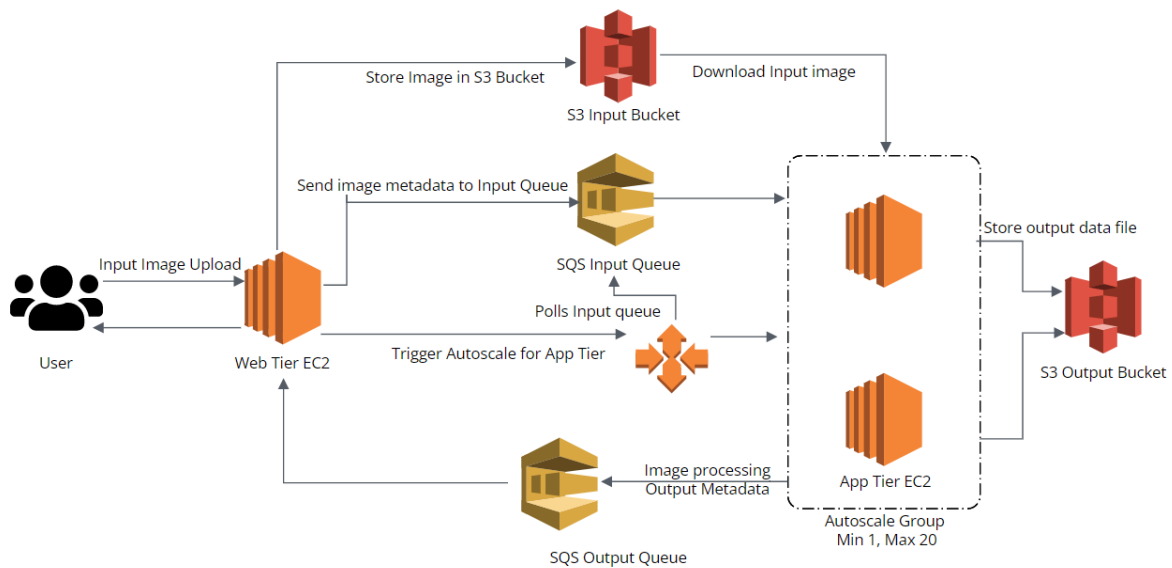
The user will upload multiple images on the application running on EC2 Instance at AWS and will be able to get the image processing result for each image he/she has uploaded.

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

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 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.

2. Design and implementation

2.1 Architecture



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 Web and App Tiers. Web Tier will handle all user requests and display the result for image processing on the images uploaded by the user by listening to SQS. 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. 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, it will trigger the request to start the App tier 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 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, 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. Completed 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.

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.

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.

3. Testing and evaluation

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

Upon receiving the requests from the user, Web tier uploaded the images in the S3 bucket, and adds the messages in SQS. 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 one EC2 instance 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)
1	1.2mins
10	2.2mins

50	3.5mins
100	6 mins

4. Code

Code files submitted in the zip file:

1. app.py (web-tier)
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. total_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 run Web-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{
    listen 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{
    listen 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}/