CSE 546 - - — Project Report

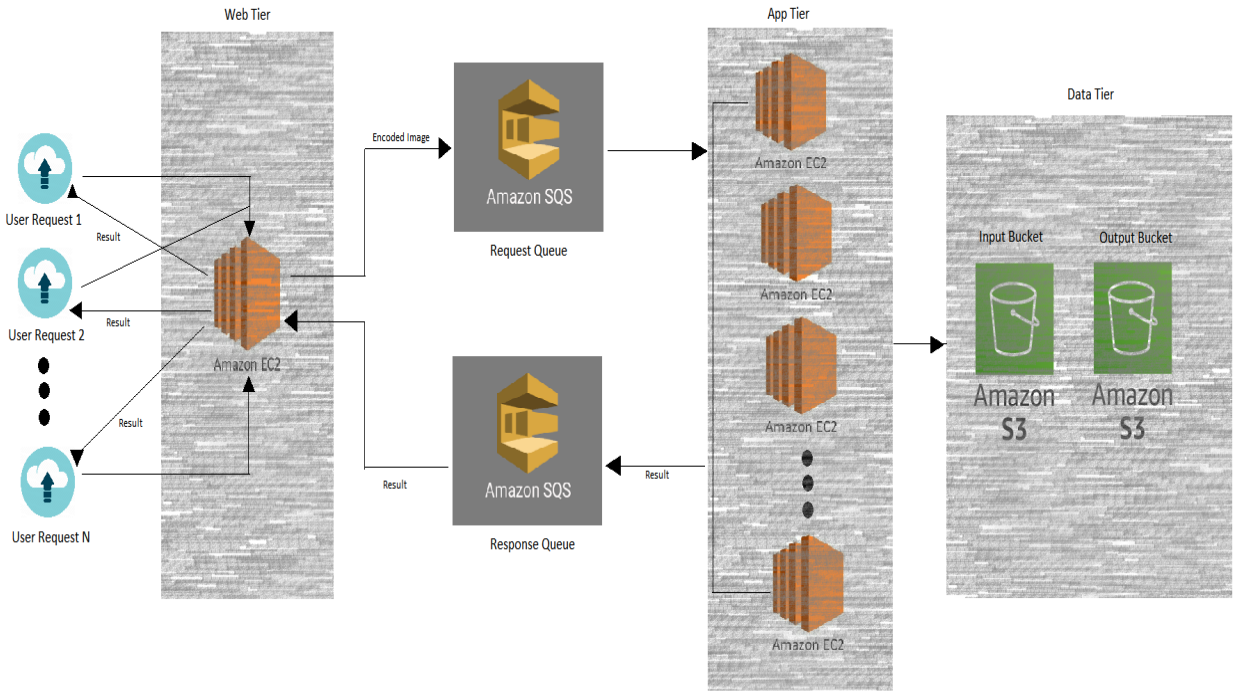
Janam Devang Vaidya (1222377325)
Rhythm Rakeshkumar Patel (1225583281)
Rishabh Krishnakant Pandat (1222158509)

1. Problem Statement

In this project, our aim is to build an elastic web application that automatically scales in and scales out on-demand and cost-effectively using IaaS cloud services. The IaaS resources used in building this application are from Amazon Web Services (AWS). The users are provided with an image recognition service via the cloud app exposed to them as a Rest service to access. The app takes the images and uses a deep learning model that is provided as an AWS image to return the predicted output. The AWS resources are used here for computation, storage, and transportation. This application is important since we are providing a relevant and significant cloud service to users that help them recognize and/or classify any image. Moreover, the tools, techniques, and technologies being used in the project will be beneficial to us and others for building other services in the future.

2. Design and Implementation

2.1 Architecture



AWS Services used in the application:

1. AWS EC2: We create a virtual machine using EC2 in Amazon cloud. For this project, for implementation, we create EC2 instances at the app-tier and web-tier.
2. AWS SQS: For maintaining a request queue and response queue that connects the app and web tier, we are using a standard SQS in this project. Here, the request queue (input queue) stores the request from the web tier and the response queue (output queue) stores the request from the app-tier.
3. AWS S3: We are using this service for storing our results. The user request images are stored in the input bucket and the predicted result (key,value pair) is stored in the output bucket.

As seen in the architecture, initially there is one EC2 instance in the web tier which is continuously running, and that takes multiple user requests. These requests will be then sent to the input queue from which it will be sent to the app-tier which has multiple instances (minimum of 1 and maximum of 19) and these requests will be stored in the input bucket. Scaling in and scaling out is programatically handled by the web tier according to the number of messages present in the input queue. The instances in the app tier run a deep learning algorithm to classify the images that the user had sent and will be fed from the input queue. The resultant output will be stored in the output bucket which is then sent to the output queue and ultimately back to the user. In the web tier, we are maintaining a dictionary so that when there is another request for the same image again, then we dont go the app tier and we just directly output the already stored result which improves the efficiency of the app.

2.2 Autoscaling

Scaling in and scaling out takes place at the web tier itself. We are using a greedy approach to scale out. We have two things, number of instances and number of visible messages in the request queue. We have set the number of app instances to go at a maximum of 19 since 1 will be reserved for the web tier. So, a maximum of 20 instances will be running at a time, 19 max for app instances and 1 for web instance. We create app instances according to the number of visible messages if the number of visible messages will be less than 19. The controller won't allow any more instances to start if the number of messages will be greater than 19 at a given time. For scale in, if the number of app instances become greater than the number of messages in the request queue then we are iteratively stopping the instances that we created programmatically.

2.3 Member Tasks

Janam:

- Created a post api to facilitate upload of images from the user. That api is responsible for sending the message to the request queue and also for listening to the response from response queue.
- Created the image upload flow for SQS and S3 bucket.
- Contributed in creating shell files for automation.
- Worked on report and documentation.

Rhythm:

- Worked primarily on the app tier side of things. Responsible for decoding the image from the request queue and running the image classification model on the input image and again uploading the output to the S3 bucket and sending back the output in the response queue.
- Created the download flow for SQS and S3 bucket.
- Contributed in creating shell files for automation.
- Worked on report and documentation.

Rishabh:

- Responsible for setting up the necessary AWS infrastructure like S3, SQS, EC2 instances.
- Testing and evaluation of the appliaction.
- Worked on report and documentation.

3. Testing and Evaluation

For testing, we used the 100 images given by the TA. During testing, aslo checked whether the application can take more than one images at a given time as an input. Looked at the number of images stored in the S3 bucket is same as the number of images the user had sent. Mathematically checked the correct working of auto scale-in and scale-out. Also checked the input queue to validate the number of messages sent to the SQS queue.

-> For persistent storage, stored the result generated by the image classifier in S3 bucket.
-> Sent the result to the output queue through which the final result is fetched .

4. Code

Web Tier:

-> controller.py : Handles scale in and scale out of number of instances based on the number of requests in the input queues.

-> webv2.py: Contains post API for encoding and uploading multiple images and listen to the output from output queue.

App Tier:

-> index.py: Decodes the image from the input queue and runs the image classification.py file to get the output from the deep learning model provided and stores the output as well as input in respective S3 buckets and also sends back output to the output queue.

Steps to reproduce code

1. We have kept a shell file in web tier ec2 instance to run the webv2.py file. So this is the starting point of our application. This particular file has the post api that the user will use to send user requests. So we need to execute this file by the command sh autoweb.sh. Basically running webv2.py file.

2. We have also created a shell file to run the controller.py file which handles the scale in scale out approach of our application.

3. To run only the app tier ssh or login to the instance and run cd /home/ubuntu/classifier and then followed by python3 index.py