

# Image Recognition as a Service

Meher Abhilash Komirishetty-1220259144

Sreekar Manchukonda-1224487368

Bharghav Rapolu-1219931960

## PROBLEM STATEMENT:

The project's goal is to leverage Amazon Web Services to create an elastic web application that hosts an image recognition module that can accept several concurrent users' photos as input and outputs the anticipated user image name. The tasks entail creating the model's architecture, which includes using a Web interface to retrieve user images, AWS SQS (Simple Queuing Service) to act as an intermediary between the Web Tier and the App Tier to transport images, AWS S3 (Simple Storage Service) to store the input images and the output result, and AWS EC2 (Elastic Compute Instances) to provide computation power for handling the application. The instances should have the capability to execute the application through the feature of auto-scaling, which minimizes the processing cost by increasing and decreasing the number of EC2 Instances as per the demand.

## ARCHITECTURE:

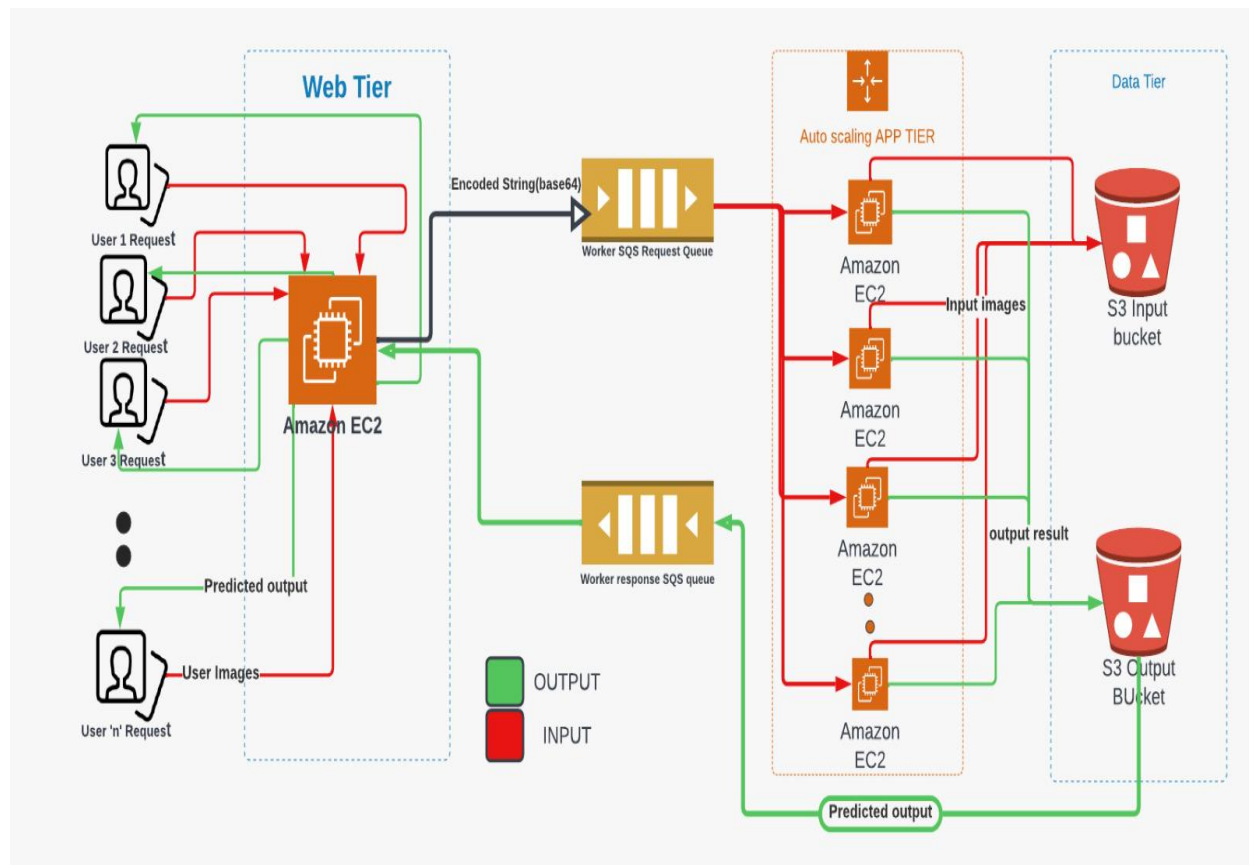


Figure 1- Architecture of the model

## ARCHITECTURE DESCRIPTION:

Multiple user requests [images] are provided to the Web Tier Application through the Workload Generator. The requests are successively provided as an input to the SQS in a format which SQS can handle. Upon the request arrival in the SQS Request Queue, the Controller (present in the web tier) which runs parallelly in the background performs some crucial operations such as keeping track of the number of SQS messages, Creating instances based on the Autoscaling logic and running these instances of default configurations. These newly created Instances exhibit the functionality of a computing engine by receiving the input messages from the SQS Request Queue, Decoding the bytes String back to the Image format to push to the deep learning module and further stores the result in S3 Output bucket. Parallelly the Input images are stored in the S3 Input Bucket for persistence. The predicted result can be viewed by the user along the image name.

## AWS SERVICES USED:

- AWS EC2  
The EC2(Elastic Compute) Service of AWS is used in the App and Web Tiers. App Tier EC2 Instances hosts the Deep Learning Image processing application that fetches images input from the users through the Interface on the Web Tier and the The Web Tier EC2 Instance collects the user images as input.
- AWS SQS  
Simple Queuing Service is used to transfer messages in the form of a string between two endpoints. The SQS Request queue transfers the user input images between the web and app tiers and the SQS Response queue transfers the output result in the form of a string message between the app and the web tier resulting on the Users Interface.
- AWS S3  
AWS S3(Simple Storage Service) stores the data in the form of bucket objects. The S3 Input bucket holds the user input images and the S3 Output bucket holds the predicted result.

## AUTOSCALING:

The maximum number of EC2 Instances used in the application is 20. This comprises 1 Web Instance and 19 App Instances. Scaling is performed upon the Messages Count in the SQS Request Queue. If the Images Count in the SQS Request Queue is greater than 19, the messages are made to wait until the previous input messages have been processed by the App Instances thereby transferring the next batch of images in the Request Queue to the existing Instances without creating new EC2 Instances thus saving the creation time and the processing costs. The EC2 App Tier Instance search for messages in the Request Queue and in the absence of message, it self terminates. UpScaling happens in the Controller Logic of the Web Tier and DownScaling happens due to Self Termination in the App Tier. This methodology reduces the overall budget and computational complexity to run the application as unwanted resources are decommissioned when not required.

## MODULES:

- **WEB TIER**

- *CONTROLLER*

- ☐ **Aws\_properties.py**

- Contains the dynamic variables that are accessed through the python modules. Storage of the AWS User Credential, URLs of AWS Resources and the common parameters is performed.

- ☐ **EC2\_Controller.py**

- This module handles the Scaling Logic that enables the Creation of the EC2 instances using an AMI, Images Count in the SQS Request Queue and the Current Running Instance Count.

- *WEB\_INSTANCE*

- ☐ *UPLOADER*

- **Push\_image.py**

- This module gets the user input .jpg images which is encoded with base64 and passes it as bytes into the SQS Request Queue.

- **Pull\_response.py**

- This module fetches the output responses from the SQS Response Queue and pushes the predicted result to the User where the Input Image is sent.

- ☐ **App.py**

- This module contains the Flask Logic that handles and manages the requests and responses between the User Interface and the Application.

- ☐ *WEB\_APP*

- **Static**

- This folder contains the static webpage content.

- **Templates**

- This folder contains the index.html and response.html pages.

- APP TIER**

- *COMPUTE ENGINE*

☐ **Receive\_message.py**

This module receives the input from the SQS Request queue and pushes it to the Image processing application for prediction. The image input is stored in the Input S3 Bucket.

☐ **Send\_message.py**

This module transfers the result from the image classification model to the SQS Response queue and also stores the resulting output to the S3 Output Bucket.

☐ **Terminate.py**

This module contains the logic for an EC2 Instance to self terminate when the Count of SQS Request Queue messages is Null.

→ **IMAGE RECOGNITION**

This folder contains the image processing logic.

→ **App\_tier.sh**

This bash file consists of a sequence of commands that should be executed by a newly created EC2 App Tier Instance on startup.

→ **Requirements.txt**

This text file contains all the required packages and apis to be installed to run the application.

## CONTROL FLOW:

- **WEB TIER:**

- Concurrent Input Images are retrieved from the user
- Received images are pushed into the SQS Request Queue
- Predicted Results are retrieved from the SQS Response Queue and sent back to the user.
- Controller File performs Auto-Scaling for creation of EC2 Instances in the App Tier

- **APP TIER:**

- App Tier Instances are created and the startup bash file runs the required modules.
- Input Images fetched from the user are stored in the S3 Input Bucket and the Predicted result is stored in the S3 Output Bucket.
- The result is also transferred to the SQS Response Queue through which the user fetches the output.

## TESTING:

The Testing phase is performed by fetching input images from the user and inspecting if the Application can withstand multiple concurrent user requests. Several factors including the Application Scalability, Persistence of data in the S3 Buckets, Count of Running Instances and the Predicted Results are checked and the accuracy of the functionality is determined.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL powershell + - [ ] [ ] ^ X

(test_06.jpg,Gerry)
(test_96.jpg,Gerry)
(test_44.jpg,Emily)
(test_65.jpg,Gerry)
(test_57.jpg,Ranil)
(test_99.jpg,Wang)
(test_64.jpg,Emily)
(test_93.jpg,German)
.
Total time: 129.6428928375244s
100 requests are sent. 100 responses are correct.
PS C:\Users\HP\Downloads\CSE546_2021S_workload_generator-main\CSE546_2021S_workload_generator-main>
```

Figure-2 Results from Multi-Threaded Workload Generator

### Face Detection

Choose Files

No file chosen

Upload

Get Results

Total Results: 100

Input Image	Classification Result
test_00.jpg	Paul
test_01.jpg	Emily
test_02.jpg	Bob
test_03.jpg	German
test_04.jpg	Emily
test_05.jpg	Gerry
test_06.jpg	Gerry
test_07.jpg	Ranil
test_08.jpg	Bill
test_09.jpg	Wang
test_10.jpg	Paul




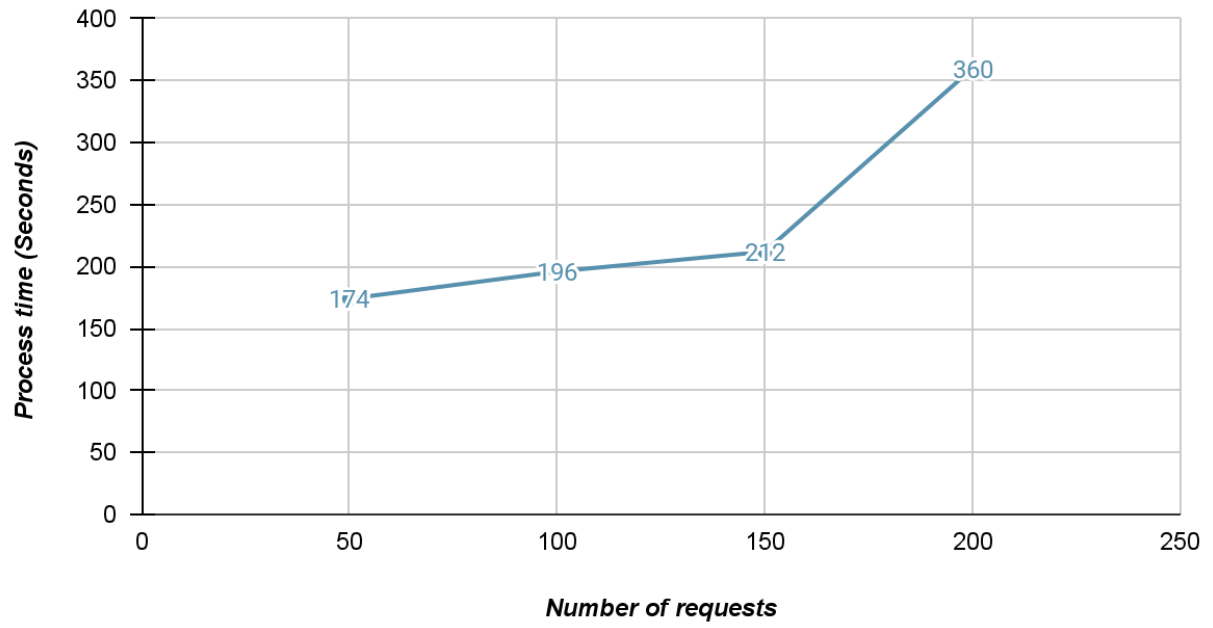


Figure-3 Results from User Interface

## RESULTS:

### Scaling Results



## INDIVIDUAL CONTRIBUTIONS

### ***1. Meher Abhilash Komirishetty(1220259144)***

#### *Design*

- I came up with the final design of the project after conceptualizing several architectures and concepts.
- My role was to implement the flow of the design from the architecture given and implement the design in the form of code.
- I also developed the core task of sending and fetching messages from the SQS.
- I was also involved in the implementation phase which is the core part of the problem statement, developing a manual load balancer by Scaling out EC2 instances based on user demand, which makes the entire service elastic.
- I also solved critical issues in the architecture design such as handling multiple threads of the workload generator and fetching accurate results to the user from the SQS output.
- Creating an algorithm to match the exact result for every input message through various loops and conditions.
- Challenges in the encryption and decryption of the input messages to be sent and received from the SQS Input Queue.
- I am also responsible for evaluating and improving the performance of the entire model and making code changes to reduce the complexity.

#### *Implementation*

- In the Service's Web Tier module, I completed the following tasks:
  - ❖ Retrieval of Concurrent Input Images from the user using Flask
  - ❖ Received images are pushed into the SQS Request Queue based on user Input
  - ❖ Predicted Results are retrieved from the SQS Response Queue and sent back to the user.
  - ❖ The controller module acts as a load balancer to perform Auto-Scaling for the creation of fresh EC2 Instances in the App Tier that runs the application.

#### *Testing*

After implementing the above tasks in python, my job was to conduct unit, integration and end-to-end testing of our Image Recognition Service.

- *Unit Testing:* This part was particularly important as I developed the controller logic and so I ensured that the controller code invoked the correct number of EC2 instances on the App Tier for the maximum utilization of the resources provided.
- *Integration Testing:* After uploading all the modules i.e EC2 Web instance creation and controller code on the AWS, we as a team tested the whole Web Tier module to check if the Controller works fine or not.
- *End to End Testing:* We all participated in end-to-end testing of our developed service as a team to check that the message traveling over SQS from web tier to app tier functioned flawlessly and that the replies from the App tier were valid or not.

## 2. Sreekar Manchukonda (1224487368)

### *Design:*

- I was involved in understanding the architecture of the problem statement and structuring the implementation modules accordingly. We worked together as a group to evaluate the entire design of the application.
- I was primarily involved in the control flow and the implementation of the app tier and implementing it. I was in charge of the app tier's implementation using Python and deployment from the local machines to the server.
- I am also responsible for evaluating the performance of the model and improving it wherever necessary reducing the time and space complexity.
- I managed the security aspects of the AWS resources by configuring the security groups and IAM policies for them.
- I was involved in creating app-tier AMIs and installing the required components on the web and app servers. I am also responsible for creating and managing the AWS resources that are used in the design of the application.

### *Implementation:*

My tasks include:

- ❖ Create a bash script that can install the app tier's dependencies and run all of the code on the creation of the EC2 Instance in the app tier.
- ❖ Fetch all available messages from the SQS Request Queue to every available EC2 by polling for messages and deleting them after storing the result in the SQS Response Queue.
- ❖ Implement the Image Processing module over the input image and transfer the result to SQS Response. Push the input message to the Input S3 and the output to the S3 Output.
- ❖ Terminate the instances in the app tier by checking the available messages in the Input SQS.
- ❖ Deployment and migration of the entire application from local machine to the AWS instances and ensure accurate functioning of the application.

### *Testing*

- Unit Testing: I created each and every code module and tested them individually by keeping different parameters and scenarios in mind.
- Integration Testing: I tested the flow of the app tier after setting up all the code modules on the app tier. I manually invoke bash script to test the flow of the app tier.
- End-to-End Testing: We tested the entire flow of the web tier and app tier in the team. We tried different parameters and uploaded different numbers of images to verify the correctness of our code.



### 3. Bharghav Rapolu (1219931960)

#### *Design*

- The design of the system was implemented in collaboration with my coworkers. In addition, I worked on web tier implementation and configuring the code in the web tier.
- I am responsible for the development and integration of the web tier. I designed a pipeline that mapped user inputs (images) and outputs (data storage in S3, SQS). I worked on the logic for converting a picture to base64 format and transmitting it over SQS and reverting the process. Input and output should be uploaded to an S3 bucket. Interface the code with a workload generator.

#### *Implementation.*

- Implemented the RESTful Web Service for web-tier in flask. This mainly performs four operations:
  - ❖ Storing all uploaded input images in S3 Bucket from a SQS .
  - ❖ Send the {image, result} predicted output to the SQS Queue.
  - ❖ Integrate the workload generator with the code implemented.
  - ❖ Delete the received message {image, result} from the response SQS Queue and store the result in server code.

#### *Testing*

- On my local PC as well as on web-tier, I thoroughly tested the working of the webtier..Unit testing, integration testing, and end-to-end testing were the three phases in which I tested.

*Unit testing:* I tested each module separately (on a local server) to confirm that the functionality was correct. For example, upload files, sendMessage (send photos to the input SQS queue), and receive message were all tested individually.

*Integration Testing:* I uploaded the source code to GitHub and tested each web-tier feature to verify the server did not encounter any issues when performing API queries.

*End-to-End Testing:* We checked outputs in SQS queues and S3 Buckets, as well as the scale-in and scale-out functionalities, to test the end-to-end flow. We put the project through its paces using various parameters to see how well it worked.