Image Recognition as a Service

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:

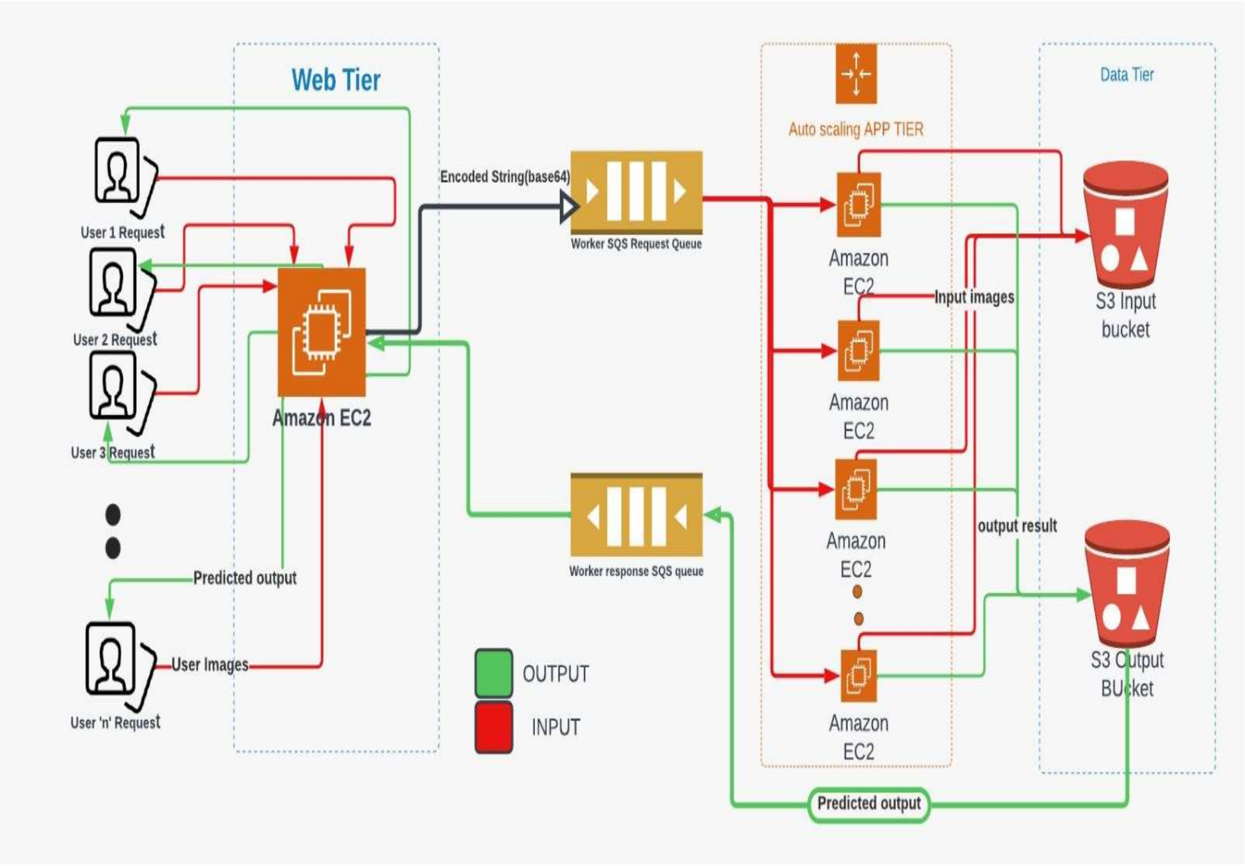


Figure 1- Architecture of the model

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 parallely 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.

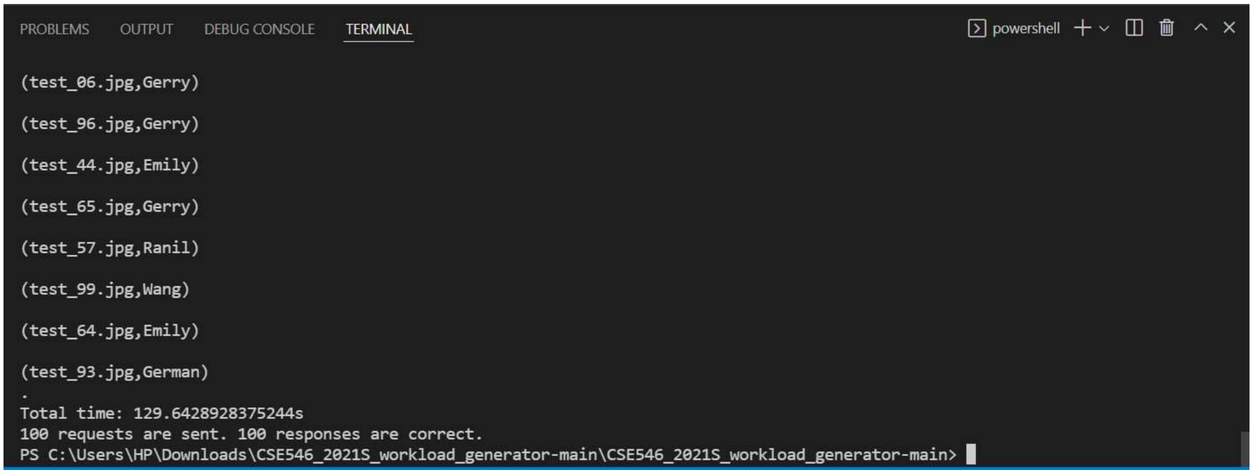
CONTROL FLOW:

- *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.

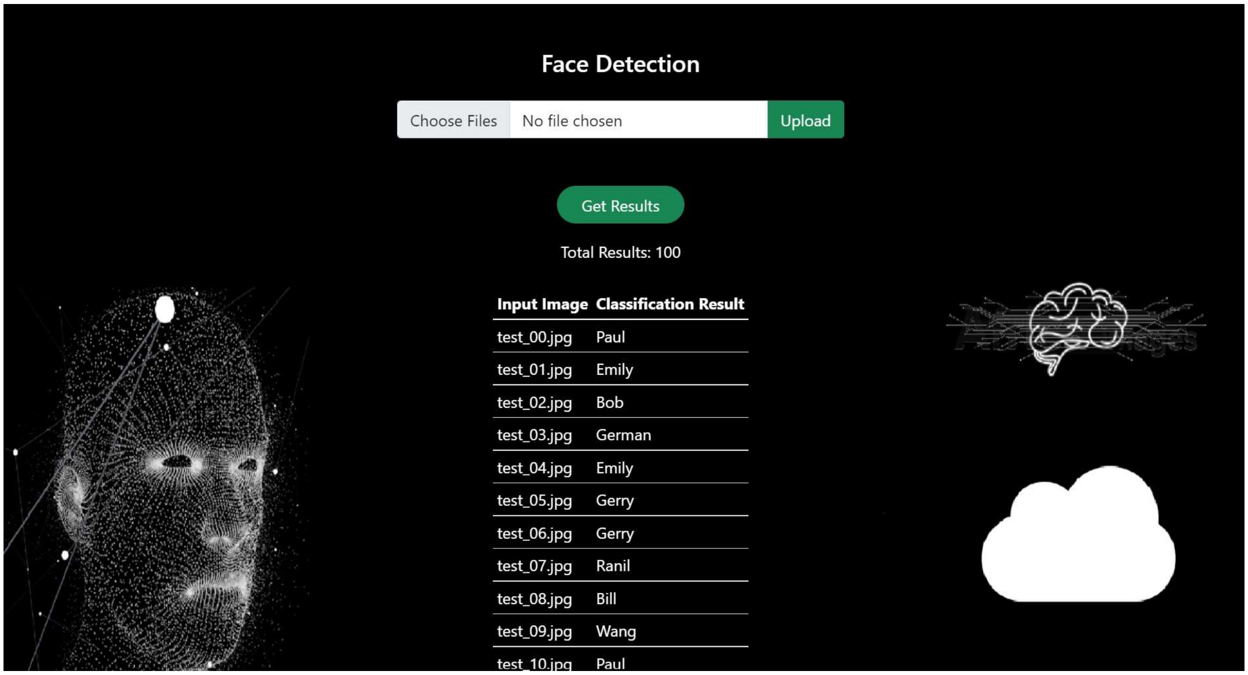
TESTING:

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.



```
(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



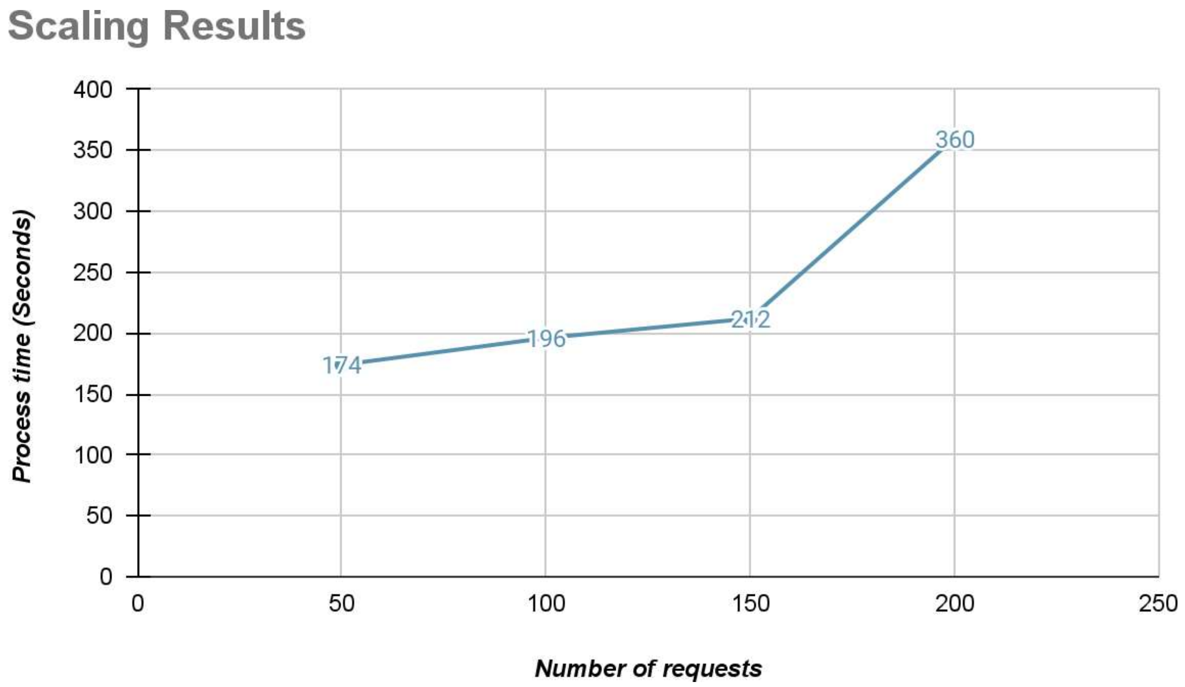
Face Detection

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
Process time (Seconds)
400
350
300
250
200
150
100
50
0
Number of requests
0
50
100
150
200
250
174
196
212
360