CSE 546 — Project 1 Report

Gunda Akshay Kumar (1217179379)
Kusumakumari Vanteru (1217031218)
Tejaswi Paruchuri (1213268054)

1. Problem statement

The main purpose of this project is to implement an Elastic Cloud Application that effectively utilizes the IAAS services (compute, storage and message) of Cloud Computing. Here, the application automatically scales in and out based on the demand i.e. number of requests received to the customer-facing web application. We have used the most widely used resources from Amazon Web Services(AWS) i.e. Amazon Elastic Compute Cloud (EC2), Amazon Simple Storage (S3) and Amazon Simple Queue Service(SQS).

The function of this application is to provide an image recognition service, which uses a deep learning model to predict the images provided by users. The user can upload multiple images in the web-application user interface (web-tier), and then, the cloud application scales automatically according to the number of images uploaded, processes the images using the Deep-Learning based classifier to get the most probable labels and then displays the same on the web-application. This project is important to understand the architecture of auto-scaling and its implementation details followed in large-scale cloud applications, which ensures that our application can always handle the current traffic demand. Also, this helps in better fault tolerance as there is a higher probability of at least one processing App Tier to be always available and hence, better cost management.

2. Design and implementation

The implementation of the system has the following major components -

- **Request SQS Queue:**

  The Request SQS Queue stores the input messages uploaded and sent from the Web-Tier for processing by the App Tier. The number of messages in this queue is used as a metric to scale-up or scale-down the computing resources (App-Tiers) to be started by the Controller.

- **Response SQS Queue:**

  The Response SQS Queue stores the output messages sent from each App-Tier. The Web-Tier reads these messages from this queue to display the predicted output label for each image by the Image classifier (predicted in the App Tier).

- **Input S3 Bucket:**

  The user inputs i.e. the image(s) uploaded by the user in the Web-Application are stored in this Input S3 Bucket by Web-Tier, and is retrieved by an App Tier to run the Deep Learning Model.

- **Output S3 Bucket:**

  The outputs i.e. the predicted image classification labels are stored in this Output S3 Bucket by each App-Tier. These are then read by the Web-Tier to display the final results on the frontend web interface.

- **Web-Tier:**

  The user is provided with a Web Application Interface to upload image(s) into the cloud application. We have used multi threading to upload image(s) into the Input S3 bucket simultaneously, in order to be faster and more efficient. Once the image(s) are uploaded, the service uploads these images into an Input S3 bucket and sends a message corresponding to each image into the Request SQS queue. We have implemented a FIFO (First in First Out) queue so that the requests are handled as they come. We have associated an Elastic IP address with the WebTier Instance, so that users can

access the web interface using the elastic ip address. This web-interface is the entry point for the user to interact with the cloud application.

- **Controller:**

  The Controller mainly reads the messages from the Request SQS message queue and scales up or down based on the number of messages in the queue. Based on the demand i.e. number of input images, an appropriate number of EC2 compute resources (App Tiers) are created and started in running state. Here, since the limit for free-tier usage is 20 instances and we already have 3 instances running - one for the Web-Application, one for App Tier0 and one for the Controller, we make sure that the new instances created never cross the 20 limit. So, in order to maintain the basic and minimum functionality of the system, we make sure that one App-Tier0 instance is always running to process the SQS messages. The controller performs this job of scaling based on the number of messages in the Request SQS queue, every 10 seconds. Once the newly created App-Tier instance finds that there are no more messages left to process, it terminates itself.

- **Application Tier (App Tier):**

  Each Application-Tier instance reads a message from the Request SQS queue, and retrieves an image from the Input S3 bucket to be processed based on the image URL. Then, it runs the Deep Learning algorithm to predict the output and sends the response to the Response FIFO(First in First Out) SQS queue. Also, it uploads the result to an Output S3 bucket. If there are no messages found in the Request SQS queue, then the instance will be terminated. Otherwise it will process the requests until there are messages available in the Request SQS queue. But, the AppTier0 will always be running to serve the basic system functionality.

2.1 Architecture

Given below is a detailed architecture of the elastic cloud system we have designed and implemented -

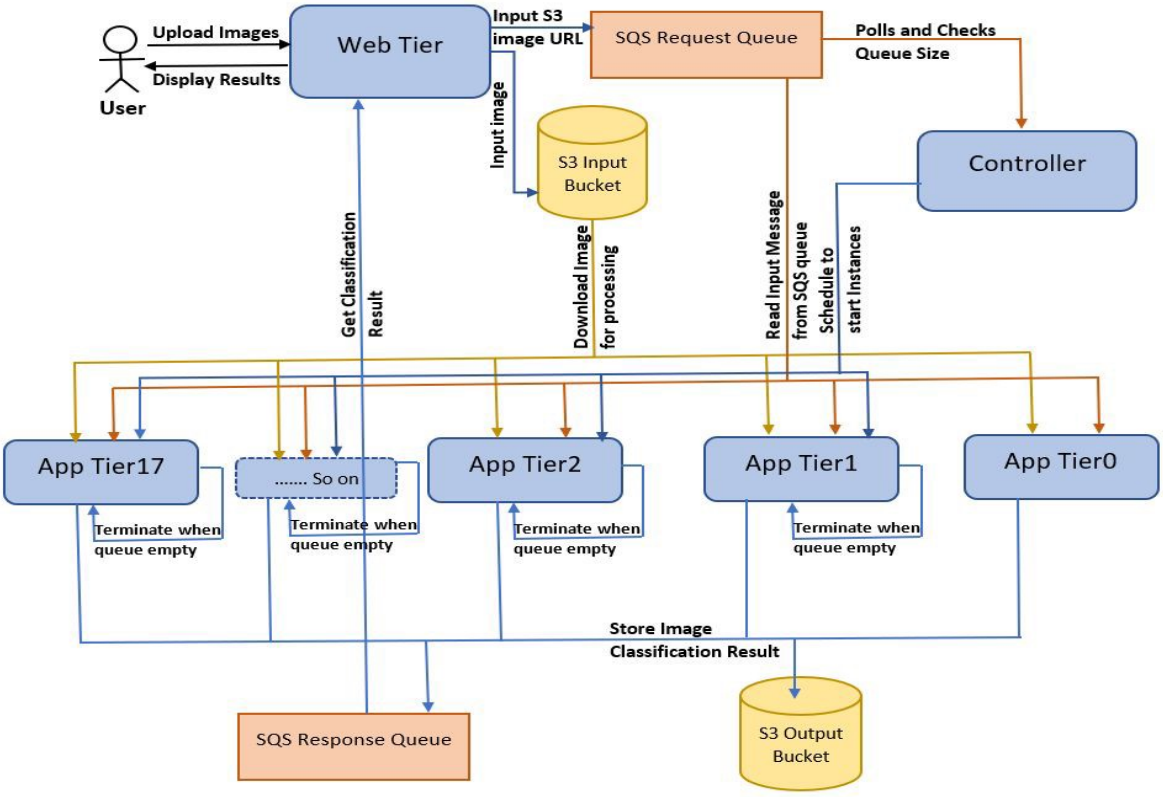


Figure 1: System Architecture

2.2 Autoscaling

In this application, we have implemented an auto-scaling by comparing it to a large scale enterprise application using a message queuing system, where the demand traffic keeps changing rapidly. So here, the App Tiers will be scaled-up or down based on the messages that need to be processed in SQS Request Queue. The process of creation of App Tiers is controlled by the Controller job that is scheduled to run every 10 seconds. So, initially an App Tier i.e "AppTier0" will always be running that will check for available messages in SQS Request Queue (if any) to process them. AppTier0 will never get terminated. At the same time the Controller job will also check for the available messages in SQS Request Queue every 10 seconds and start the App Tiers if any new messages are available to process. The Controller can autoscale from 1 to 17 App Tier instances based on the demand in the SQS Request queue. Each App-Tier instance will continuously poll the Request SQS Queue to see if there are any new messages, process them and send the output to the Response SQS queue. If there are no further messages to process in the SQS Request queue the instances (App Tier 1 to App Tier 17) started by the controller will be terminated.

Considering the real world applications where at least one instance will always be available to process messages and autoscaling will be done when needed, we are following the same idea in this project. Also, after considering the tradeoff between EC2 instance initialization time after creation and the time to process the image classification, always keeping one app tier instance in running state has given better performance. Also, this ensures that the application operates in an asynchronous manner as now, the Web-Tier will not have to wait for each App-Tier to finish its processing to accept new inputs. Similarly, each App-tier can process at their speed, independent of the Web-Tier.

3. Testing and evaluation

First, we have given the user the Public IP Address (`34.236.13.196:5000`) to our web-tier, so that he/she can directly access the URL without the need to access the EC2 instance URL. Below is how our Web-Application interface looks like

Figure 2: Front end Web Interface of the Application for testing

Initially, we have the following existing instances before testing the application:

- Web Tier0 - Running
- Controller0 - Running
- AppTier0 - Running

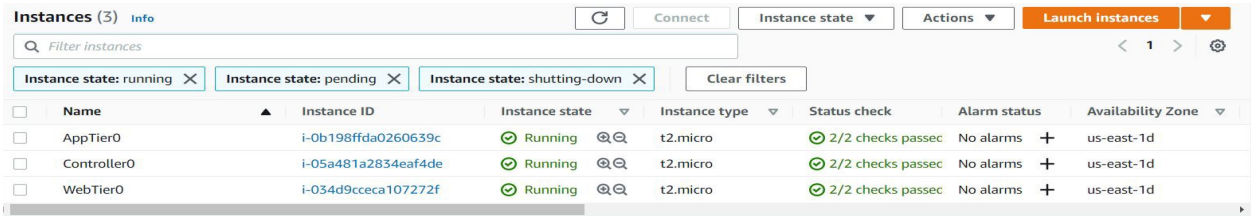


Figure 3: Instances that will always run irrespective of uploaded images

Note: We have appended timestamp and date to the image name on upload because if the same image is uploaded twice at different times we could differentiate them in s3 buckets.

We have tested the system with the help of 2 test-cases:

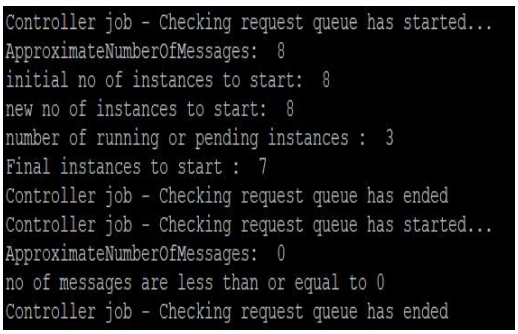
- **Test Case 1: With 10 images**

From the front end Web interface, 10 images were uploaded at a time. Once the images were uploaded we checked the number of App tiers that were autoscaled and could see 7 app tier instances were created by controller as other 3 images are already picked by AppTier0 which is running

- **Test Case 2: With 100 images**

From the front end 100 images were uploaded at a time. Once the images were uploaded we checked the number of App tiers that were autoscaled and could see 17 app tier instances were created by controller as we can only have 20 instances max at a time.

3.1 Evaluation using the Auto-Scaling logic done by controller:



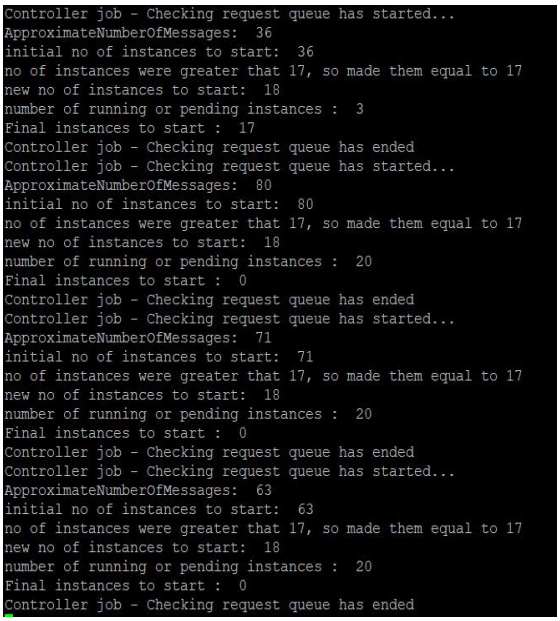


Figure 4: Left image: Autoscaling of 7 AppTier instances by controller when 10 images are uploaded, Right image: autoscaling of 17 App Tier instance by controller when 100 images are uploaded

3.2 Evaluation using App-Tier Instances created in EC2 console:

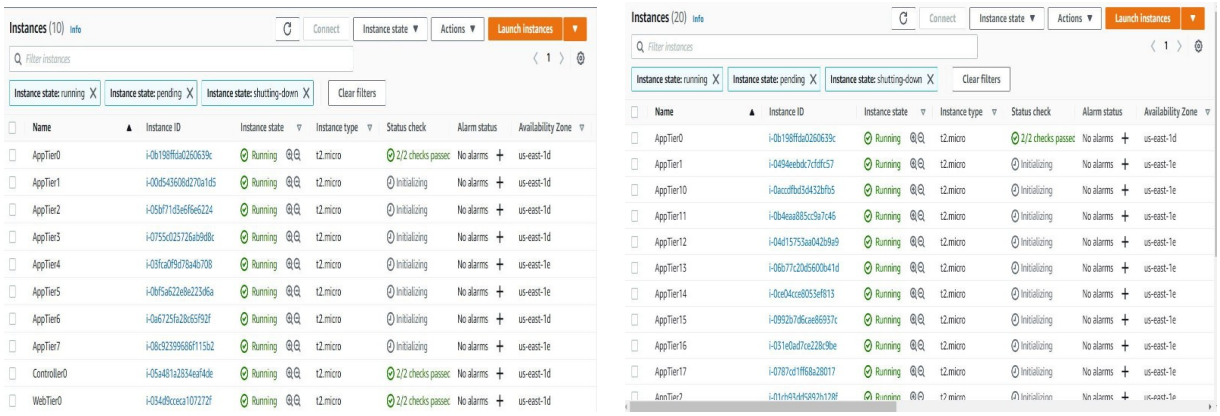


Figure 5: Left Image : AWS Console showing created instances based on autoscaling for 10 images,
Right Image: AWS Console showing created instances based on autoscaling for 100 images

3.3 Evaluation based on the results in Web Tier and Time taken:

Figure 6: Results and Time Taken to process 10 images shown in Web Tier

Figure 7: Results and time taken to process 100 images in Web Tier

Time taken to process 10 images: 15.265690565109253 seconds

Time taken to process 100 images: 140.4356415271759 seconds

3.4 Evaluation using S3 Buckets:

We have also evaluated the application by checking whether all the uploaded images were uploaded in s3 input bucket and their corresponding results are uploaded in s3 output bucket

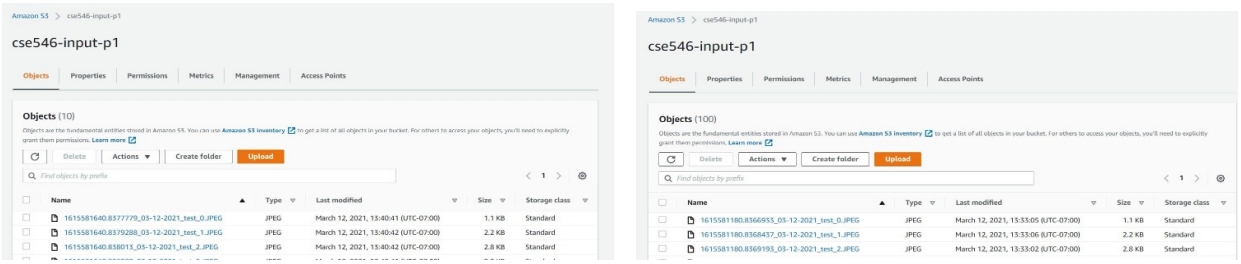


Figure 8: S3 input buckets for 10 and 100 images respectively

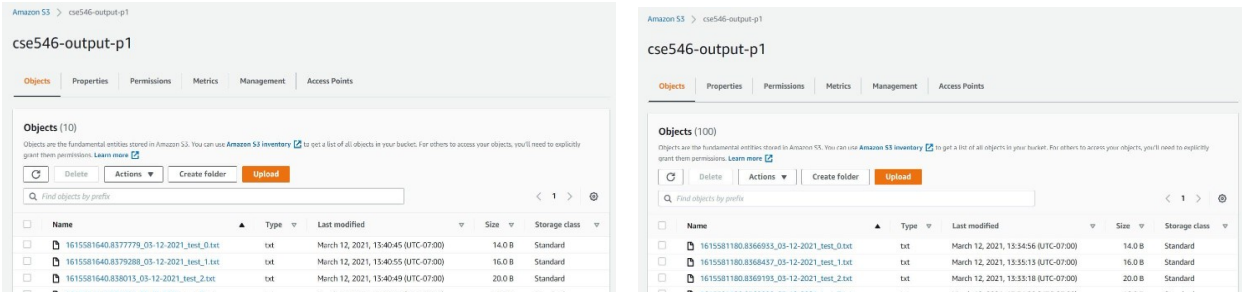


Figure 9: S3 output buckets for 10 and 100 images respectively

3.5 Evaluation using results in Front End:

After we got the results in the front end we evaluated the results with the given classification results.

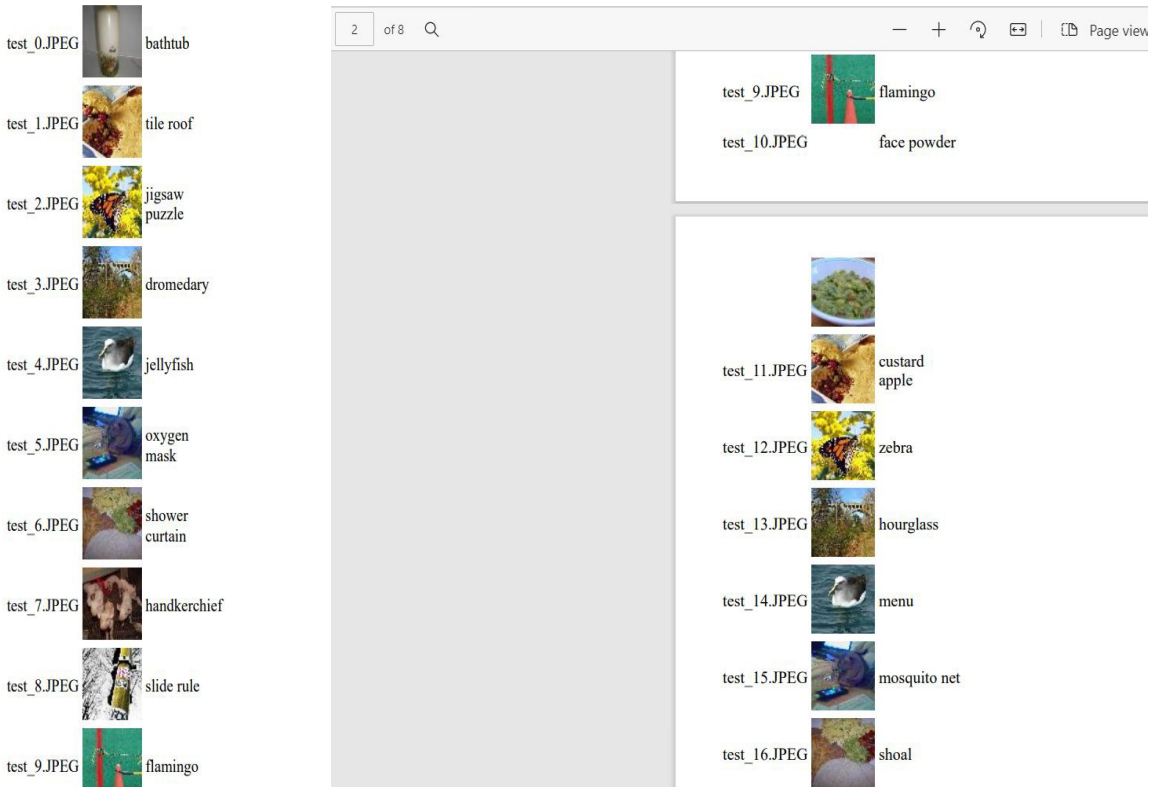


Figure 10: Left Image: Results in front end for 10 images
Right Image: Results in front end for 100 images

Note: All the screenshots are provided in the outputs folder in the submitted folder.

4. Code

4.1 Functionalities of every program in the code

Web Tier:

Web Tier is created using the Flask web framework written in Python. The below are the code files used in Web Tier to store and retrieve data from S3 Buckets and SQS.

- elastic_application.py: This python script is used to start the flask application, it takes inputs from index.html and does function calls from upload_data.py script to upload images to S3 buckets and send SQS messages.
- upload_data.py: This python script has functions written to upload/retrieve images to S3 buckets and send/receive messages from SQS.
- index.html: This html script takes images uploaded by the user and displays results generated by the classifier.
- aws_credentials.py: It has aws login credentials.

Controller:

The Controller is the middle piece of the system which manages the auto-scaling aspect of the application. As discussed before, its main purpose is to continuously poll the Request SQS queue every 10 seconds for messages and create new EC2 instances if the system needs to scale. Here, we have made sure that the number of EC2 instances running at any point of time does not exceed 20, as per the requirements. The main programs are:

- controller.py: This is the main program where the Controller job is run and it is scheduled to run every 10 seconds. In this job, first we get the number of messages in the Request SQS queue. If it is greater than 0, then we make it equal to 18 if greater than 18 because we cannot create more than 20. Then, we send it to another method which calculates the final number of instances to start based on the ones which are already running. Then, using the multi-threading process, we create the App-Tier instances one by one, with the appropriate names using Threads. We always make sure that the total number of instances do not cross 20 in this process.
- requirements: This file has all the required libraries that need to be installed for the Controller to run without any errors. This file is used by pip3 tool of python to install the libraries.

App Tier:

App Tier is started using the private AMI created from the controller based on autoscaling. AMI is created from the AppTier_Controller instance that's in a stopped state in our AWS account. That instance initially is created from the deep learning model AMI (ami-0ee8cf7b8a34448a6) provided. On top of that below code files are added to use the classifier model.

- appInstance.py: This is the main program which will check for any available messages in SQS request queue and download the images from S3 input bucket and runs the given deep learning classifier model and stores the results in S3 output bucket and pushes the result message to SQS response queue. It will again check for any available messages in the SQS request queue otherwise it will terminate.
- classification_result.txt: The result given by the deep learning classifier is stored in this file which will later be read to send the results in SQS request queue and S3 output bucket.

4.2 How to Install and Run the programs

Web Tier:

Start the Flask Application, upload images by accessing the application from a web browser. The results will be displayed on the web page. Before running, create empty upload and result folders in the root directory of the application. The following are the steps to be followed

1. Install all the necessary libraries like boto3, schedule using the commands:
   sudo apt-update
   sudo apt-upgrade
   sudo apt install python3-pip
   pip3 --version
   python3 -m pip install -r requirements
2. Run the script:
   python elastic_application.py
   It runs the application on 5000 port which can be accessed using localhost:5000 if running on the local machine, and on an EC2 instance using <IPV4 DNS>:5000 or <IPV4 Address>:5000, or if an Elastic Ip address is assigned <Elastic IP Address>:5000

Controller:

First, we need to login to the Controller using Putty, by giving the SSH Client URL as - ubuntu@<IPV4 DNS> and the correct ppk file. After logging in, we need to change the directory to /home/ubuntu/controller_cse546 and then follow the below steps:

1. Install all the necessary libraries like boto3, schedule using the commands:
   sudo apt-update
   sudo apt-upgrade
   sudo apt install python3-pip
   pip3 --version
   python3 -m pip install -r requirements
2. Run Controller job using the command (to perform the Controller job, every 20 seconds):
   python3 controller.py

App Tier:

Create an instance using a given deep learning model AMI (ami-0ee8cf7b8a34448a6). Add appInstance.py, classification_result.txt, imagenet-labels.json files in the home/ubuntu folder. After that follow the below steps.

1. chmod 777 classification_result.txt
2. pip3 install boto3
3. pip3 install schedule
4. mkdir .aws
5. add files config and credentials in .aws folder. credentials file will have aws_access_key_id and aws_secret_access_key whereas config file will have region and output
6. sudo crontab -u ubuntu -e. Add @reboot python3 /home/ubuntu/appInstance.py to the crontab.

Create AMI for this instance after doing all the above steps. As cron job to start appInstance.py is added on reboot of the instance as soon as the instance is created from the controller the appInstance.py will automatically run.