CSE 546 — Project 1 Report- Group Skyline Surfers

Atul Prakash - 1225542214

Abhi Teja Veresi- 1225506321

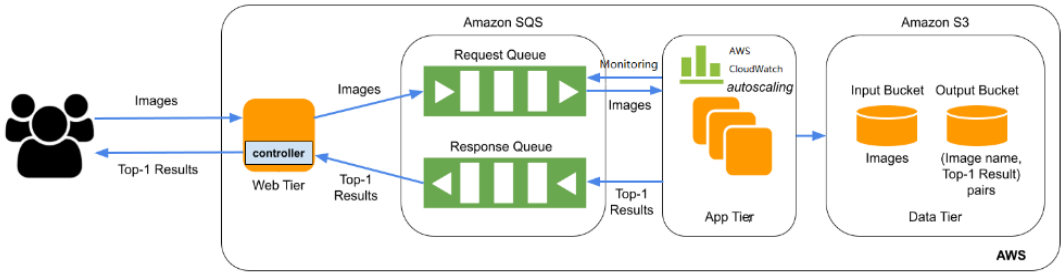
Kedar Sai Nadh Reddy Kanchi - 1225297164

1. Problem statement

Build an elastic cloud application to perform image recognition for the set of images using the deep learning model provided, you should build it using Amazon Web Services(AWS EC2, AWS SQS, AWS S3). The application should take the images from the user and provide classification results to it. It should demonstrate the capacity to efficiently handle numerous concurrent requests, dynamically adjust its scaling (Auto scale) based on the depth of the Request Queue, and store all inputs and outputs in S3 buckets. This application should possess the ability to automatically scale the requests and maintain cost-effectiveness in response to varying demand.

2. Design and implementation

2.1 Architecture



Web Tier: Web tier takes the concurrent images from the user. These images are encoded and are provided to the request queue (SQS) with the attributes for processing. After the images are

processed it retrieves the data from S3 Buckets and deletes the response queue (SQS) after retrieval. The classified results are provided to the user.

Controller: The controller serves as an intermediary for the queues and automatically scales the EC2 instances based on demand. It monitors the volume of incoming requests in the RequestQueue and adjusts the number of active instances using our auto-scaling algorithm. The controller can deploy a maximum of 20 app tier instances when needed.

App Tier: The app tier acts as a processing part of the application. It takes the input from the queues based on the scaling algorithm and uploads the images in the input S3 bucket. The images are provided to the deep learning model to provide the classified results. The results are updated in the output S3 bucket.

SQS: Two SQS queues are used. The user requests are uploaded to the Request queue by the web tier and are scaled by auto scaling algorithms based on the depth of the Request queue. After processing the requests by the app tier they are deleted from the request tier. The auto scaled responses are queued to Response Queue and are deleted by web tier after taking the response of the classification results.

S3 Buckets: Three S3 buckets are used. Input bucket used to store the images and output bucket stores the results after images are classified by the deep learning model and a demo bucket to store all the files necessary, i.e. the config, credentials, controller.py, webTier.py, myscript.sh, imagenet-labels.json and the app.py file which gets executed inside the instances.

2.2 Autoscaling

We have implemented an auto scaling algorithm based on the number requests in the Request Queue and scale the EC2 instances according to the demand. Controller.py is used to autoscale the instances. Initially we get the number of instances running to get the present load and update the current count. We fix the maximum load of instances to 20.

1. We define the global variables maximum count= 20, current count=0 scale out message count=0
2. We get the count in the Request Queue based on the visible messages and invisible messages and initialize the total count (visible msg+ invisible msg) from it.
3. Check using while loop whether to scale out the instances or scale in the instances based on the requests in the queue and
4. Scale out Algorithm: Check if Scale out message count < total count from request queue and (total count > current count && current count < maximum count), we scale out (increase) the instances to the minimum of (maximum count, total count - current count) and increase the current count to the scaleout instances.

5. Scale in Algorithm: Check if total count from request queue < current count scale in (decrease) the instances to the total count.

2.3 Member Tasks

Abhi Teja Veresi (1225506321):

Web tier Implementation: I have implemented the webtier part of the application. Created webTier.py in a flask framework which would be running in webTier EC2 instance. It is a multithreaded environment which gets multiple concurrent requests from the workload generator (workloadgen.py). Created API endpoints to get the images from the post request. Stored the images in a folder and pushed it to the SQS request Queue. Used base64 encoding to push the image to the queue. Designed the logic to run a different thread to get the responses from the response queue by pooling 10 messages at a time and delete the messages after storing the responses in a temporary dictionary. Implemented the logic to get the results from the bucket and return the response to the workload generator. Tested the functionality of web tier by sending multiple concurrent messages in different ranges.

Report and Readme creation: Helped in writing the report using the provided template and also created the README file in the git repository which details the installation requirements and the steps to run the application.

Atul Prakash (1225542214):

Setting up Infrastructure: I have created the infrastructure requirements for the project, which includes setting up all the buckets (input bucket, output bucket and a demo bucket which contains the app.py file which executes in all the running instances when the request is generated, the credentials, config files, controller.py, webTier.py, myscript.sh which contains all the scripts to run before executing the app.py file in the aws console in the instances and the imagenet-labels.json which has the tags possible for the image classification). Creating the queues(Request and Response Queues), the policies required for the IAM instance profile to run the program (all access policy), snapshot of the image provided to us, setting up security groups and also the key pairs. Also created two instances namely webTier and controller which had webTier.py and controller.py respectively.

Controller optimization: Also helping in some logic of controller.py for creating instances automatically for processing the requests (Autoscaling), monitoring the buckets, queues and testing some cases of the application while sending single and concurrent/multiple requests

from workloadgen.py and optimization of the application code accordingly to achieve autoscaling in our project.

Kedar Sai Nadh Reddy Kanchi (1225297164):

Controller and App file implementation: I have designed the logic to the auto scaling algorithm in the controller.py file based on the demand (depth) of the request queue. Scale out function is implemented when the current instances running are less than maximum count and total messages in the queue greater than current instances. Implemented logic for each user data to run in Ubuntu by app.py file for each instance.

Implemented logic for app tier by getting the requests from request queue and storing the images in input buckets. After processing the retrieved images using the deep learning model pushed in the response queue and results in output buckets.

Tested the auto scaling algorithms for different requests concurrently by monitoring the queues and buckets.

3. Testing and evaluation

We have tested the application on different scenarios based on the load in from the queue. Each scenario and its results are detailed below in a tabular form.

Length of Request Queue	Number of Instances running	Auto Scaling action
0	0	No action is required as no instances are running
5	0	Controller will launch 5 instances
5	2	Controller will launch 3 instances
25	0	Controller will launch 20 instances
50	20	Controller takes no action

0	20	Controller terminates all the instances
100	20	Controller takes no action

Evaluation:

1. Evaluated auto scaling algorithms according to the demand from the request queue.
2. Monitored the response queue for the total messages in the flight and available messages in the queue.
3. Check app tier instances and S3 buckets whether the images are loaded in the input bucket, results are inserted in the output bucket.
4. Calculated the total time taken for each number of requests in different scenarios.

Demo Video of the project:

https://drive.google.com/file/d/1GAwEKJp3JS_c5q2EISlsPBtlKDdxtSQF/view?usp=sharing

4. Code

Link to the Github repository:

https://github.com/AtulPrakash1492/IAAS-Project1-CSE546/tree/main

Code and functionality:

webTier.py:

We have used the flask framework and run in a multithreaded environment to get and process the concurrent requests from the workloadgen.py.

1. Initially imported all the libraries (flask, boto3, os etc.) .
2. Created an API endpoint with port 8081 from the flask framework and made it a multithreaded environment.

3. Upload API endpoint which is a post HTTP method, where the images sent from the workloadgenerator.py are retrieved and saved in the upload folder (upload folder is created using filesystems in python).
4. The images from the upload folder are pushed in the request queue by encoding the images using base64. The images are pushed in the queue using image attributes with filename attributes.
5. Run an asynchronous thread with the target function get_messages_from_response_queue() method, which will asynchronously retrieve the response queue (10 messages) with a wait time of 20 seconds. Delete the retrieved messages from the response queue.
6. After getting the responses from the response queue, update the local dictionary which will temporarily store the images in it.
7. After uploading images to the request queue we wait till the dictionary is updated with the response.
8. Update the output S3 bucket with the results and delete the dictionary.
9. Return the results from the S3 bucket to workloadgenerator.py

controller.py:

Controller.py is used to autoscale the EC2 instances based on the number of requests in the Request Queue.

1. Import the required libraries (boto3, time)
2. Initialize the SQS, EC2 instance, incorporate the AWS details like access key, key name, security group , ami id, instance type.
3. Setup the user data commands to install the dependencies in the app tier.
4. Initialize the global variables minimum count, maximum count, current count, additional scale out count.
5. Check the number of current running EC2 instances and initialize the current count.
6. Run a while infinite loop, where it gets the total number of messages in the Request Queue (visible + invisible).
7. Based on the total messages, scale out the responses if there is the possibility of scaling out the instances by checking whether the current count is less than the maximum count.
8. If total messages are less than the current count and maximum count, scale in the instances.
9. If not, then no scaling is needed.

app.py:

App.py is used to get the images from the request queue and provide these to the deep learning model to classify the images.

1. Import the required libraries(torch, numpy, json, boto3 etc).
2. Initialize the SQS Queues, S3 buckets, load the pretrained model.
3. Run a while infinite loop, get a message from the request queue with all the attributes.
4. Encode the received bytes, and create an image path to write the image in ubuntu.
5. Write the encoded image in a file and run it with the deep learning model provided.
6. Get the result from json labels and write the image name and its result in a string.
7. Push the response string to the Response Queue.
8. Add the image to the input bucket and result to the output bucket.
9. Delete the request queue after writing the in S3 buckets.

Installation and running requirements of the application:

Keypair: SkylineSurfers

Security Group: default

IAM instance profile: IAM1

Inside the app-instances:

The shell script (userdata) gets executes while all the instances are created and running and after the execution of the userdata the app.py(appTier) gets executed.

The content of the userdata is as follows -

```
#cloud-boothook
#!/bin/bash
sudo apt update
sudo apt install -y python3
sudo apt install -y python3-flask
sudo apt install -y python3-pip python3 python3-setuptools
pip3 install boto3
sudo apt install -y tmux
```

```
sudo apt install -y awscli
mkdir /home/ubuntu/.aws
cd /home/ubuntu/.aws
aws configure set aws_access_key_id AKIARNMIA37WVGXUZ2M4
aws configure set aws_secret_access_key 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
aws configure set default.region us-east-1
aws s3 cp s3://cc-proj-demo/credentials /home/ubuntu/.aws/
aws s3 cp s3://cc-proj-demo/config /home/ubuntu/.aws
cd ..
sudo aws configure set aws_access_key_id AKIARNMIA37WVGXUZ2M4
sudo aws configure set aws_secret_access_key 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
sudo aws configure set default.region us-east-1
sudo aws s3 cp s3://cc-proj-demo/app.py /home/ubuntu/
sudo aws s3 cp s3://cc-proj-demo/myscript.sh /home/ubuntu/
mkdir /home/ubuntu/classifier
cd /home/ubuntu/classifier
sudo aws configure set aws_access_key_id AKIARNMIA37WVGXUZ2M4
sudo aws configure set aws_secret_access_key 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
sudo aws configure set default.region us-east-1
sudo aws s3 cp s3://cc-proj-demo/imagenet-labels.json /home/ubuntu/classifier
cd ..
sudo chmod +777 /home/ubuntu/app.py
sudo chmod +x /home/ubuntu/app.py
touch /home/ubuntu/log.txt
sudo chmod -R 777 /home/ubuntu
sudo chmod +x myscript.sh
pip3 install torch==1.8.1+cpu torchvision==0.9.1+cpu torchaudio==0.8.1 -f
https://download.pytorch.org/whl/torch_stable.html
/usr/bin/python3 /home/ubuntu/app.py
```

In the terminal/cli:

1. Install python3
2. Install pip

Requirements to run webTier.py:

3. pip install boto3
4. pip install flask
5. pip install Werkzeug

6. pip install requests
7. pip install requests
8. python3 webTier.py

Requirements to run controller.py

9. python3 controller.py

Requirement to run workloadgen.py

10. python3 workloadgen.py --num_request=100 --url=http://127.0.0.1:8081/upload --image_folder=imagenet-100/