CSE 546 — Project Report

Debesh Mishra and Aakash Murari

General requirements:

- Strictly follow this template in terms of both contents and formatting.
- Submit it as part of your zip file on Canvas by the deadline.

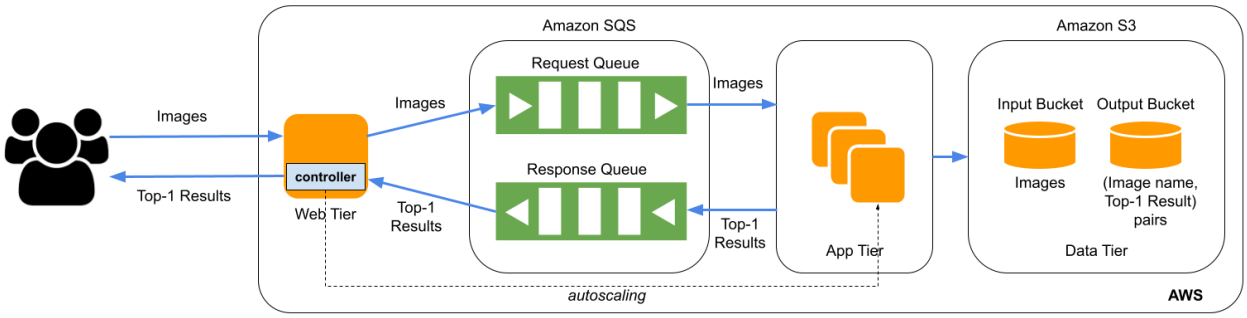
1. Problem statement

To provide an image recognition service by using the cloud resources to perform deep learning on the images provided by the users. It is important because it allows the user to search something directly by image and get the relevant results.

2. Design and implementation

2.1 Architecture

We followed the similar architecture as provided on the assignment with a minor change. Instead of app-tier sending requests to Input S3 bucket, it is the web-tier that does so for the sake of efficiency.



1. Web Tier: This is the web server that we are running which accepts the images from the workload generator. After receiving the input, it stores the image file first in S3 Input Bucket and sends the image name and request id through the Request queue. After app-tier is done processing the image, it receives the results from the Response queue. On receiving the response, the web-tier matches the request and response based on the RequestId and returns the output in the HTTP response.
2. Request Queue: This takes the image name and request id as an input from the web-tier and then sends it to the app-tier.
3. Response Queue: This takes the image name, classification result and request ID from the app-tier as an input and sends it to the web-tier.
4. App Tier: This is a scalable app server which takes the input from SQS Request Queue and uses the image name from it to download the same from the S3 Input Bucket. Then the image_classification python program is run on the image. Afterwards the result is stored in S3

Output Bucket as well as transferred to web-tier by SQS Response Queue for the result to be displayed.

5. Input Bucket: This receives the input to store from the web-tier. App-tier then uses these stored images to download the file on local and run the python command on it.
6. Output Bucket: This receives the input from app-tier about the image classification result obtained from the python command.

2.2 Autoscaling

Even though Professor had made it optional for our group, we decided to still implement it. The web-tier calculates how much instances it needs to create based on the current request queue size. It takes minimum value out of the queue size and 15 then subtracts it from already running instances (excluding web-tier) then creates those many instances to handle the image requests. So, if you only send one image to be processed then only one instance will be created and 15 or more will cap the instances at 15. We kept the limit at 15 to take into account for the instances that might be shutting down after all the requests inside those instances are processed.

2.3 Member Tasks

1. Debesh Mishra
    - Created the S3 Bucket logic which takes input from web-tier and saves it to input bucket and then the same is used to download in app-tier to run the classification program. Also implement the output logic where the app-tier saves the output that it got from the program to the output bucket.
    - Created the logic which runs the python program on the files that was downloaded from the input bucket.
    - Created the logic which can create multiple instances of the app-tier as part of auto-scaling implementation.
    - All the resources were created by me since my aws account is being used, this includes creation of SQS Queues, S3 Buckets, Custom AMI for app-tier EC2 instance, Elastic IPs and the web-tier EC2 instance.
2. Aakash Murari
    - Created the web controller in the web-tier to receive HTTP requests from the workload generator.
    - Setup the initial code base with hosting and startup of web-tier with Spring Web and app-tier with Spring Command Line Runner.
    - Created the functionality to send messages in web-tier and receive messages in app-tier via SQS Queues.
    - Implemented the concurrent response handling logic in web-tier and mapping of requests and responses with the same RequestId.

3. Testing and evaluation

We performed load testing by sending 100 requests several times through both single thread and multi-thread workload generator.

The best times recorded were 1 minute 57 seconds for multi-thread workload generator to get output of all 100 files and 8 minutes 54 seconds for a single thread workload generator to do the same.

This actually inadvertently caused us to use 85% of the free tier limit of S3 requests hence we request you to please keep that in mind while you perform the evaluation.

4. Code

Web-Tier:

Web-tier is a Spring Boot Web Application.

The **ClassifyController** receives an http request on the root path url. It invokes the S3 Repository class to store the input file in S3. It then invokes the **SqsSender** class to send the image name in the Sqs Queue. It generates a random unique Request Id for each message so that it can map the incoming response to the appropriate request. It waits for the response from the response queue.

S3Repository uses the AWS S3 SDK to interact with S3 storage and upload input images to the input bucket.

SqsSender uses the AWS SQS SDK to interact with and send messages and receive responses.

EC2Monitor class keeps checking the SQS Queue Size at regular intervals, and accordingly creates additional app-tier instances if queue size is building up.

The **BuilderUtil** class is a common helper class which helps in construction the S3, SQS and EC2 client by setting the aws properties in the clients.

The aws properties are defined in the config file application.properties and is read into a class **AwsProperties** at startup by Spring Boot.

GeneralSettings class just stores the access key, secret key, keypair and security group id for the EC2 instance builder.

App-Tier:

App-Tier is a Spring Boot Command Line Application.

The **ConsoleApp** class runs at startup since it implements the interface CommandLineRunner. It asks for messages from the **SqsReceiver**. If there are no remaining messages, the app-tier instance shuts down.

The **SqsReceiver** uses the AWS SQS SDK to get messages from the request queue and sends a message to the response queue. It uses a maximum wait time of 20 seconds before exiting.

The **S3Receiver** uses the AWS S3 SDK to interact with S3 storage and download input images from the input bucket, and upload output to output bucket.

5. Running the application

To run the application, just run the python script for either single threaded workload generator or multi-threaded workload generator.

For example - python multithread_workload_generator.py --num_request 100 --url 'http://3.228.132.166:3000' --image_folder "imagenet-100/"

Above is the script we ran to load test the application. The link is the elastic IP of the web-tier which is running the web application jar file. Once the script is run, the web application will create several instances of app-tier depending on the size of the queue and give the output on the terminal where the command is run after a few minutes.