CSE 546 — Project 1 Individual Report

Prasidh Aggarwal

Implementation of Different Tasks:

- **Application design and work distribution:** Sat together with the team to understand the project and jot down important points/concepts. Created rough design diagrams and the independent tasks the team can work on and then distributed the tasks.

- **Web tier java code:** Wrote the entire java application after doing initial research about the AWS SDK. Created a test project to see how to work with the SDK, and CLI, and created/terminated EC2 instances via the test project. I approached the design and development of the application using REST architecture and basic principles of OOPS such as Encapsulation and Abstraction. The 5 parts I divided the application into were:
    - Uploading the image sent by the user to the request S3 bucket.
    - Sending the message with image-name as the body, to the request SQS queue.
    - Polling the response SQS queue and extracting the classification results.
    - Deleting the corresponding messages from the response SQS queue.
    - Sending the results of the classification back to the user/workload-generator

- **Step-scaling policy and CloudWatch alarms:** When the team was blocked due to target tracking policies taking too much time, we researched and agreed to settle on simple/step-scaling policies. Considering the simple scaling policy wasn't allowing us to dynamically add EC2 instances according to our custom m1-m2 metric, I helped the team create step scaling policies after setting up the required metrics, cloud-watch alarms, and their respective alarm actions. Another round of testing was conducted and I fine-tuned the number of data points to be considered before scaling in or out.

- **End-to-end application testing and evaluation locally as well as on AWS:** I conducted thorough testing of the application end-end. I ran the workload generator, one instance of the app tier, and the web tier, locally. I tested both single-threaded and multi-threaded generators for up to 100 requests. After successful testing and evaluation, I uploaded the web tier jar to an EC2 instance and tested the application end-end on AWS using the autoscaling policies in place, and by sending a multithreaded 100-image request from the workload generator running locally. I was able to scale up and see the final results for all 100 images in 3 minutes and 27 seconds.

Major Learnings:

- Use of AWS SDK for Java application development.
- Use of AWS Console and CLI for creating/deleting/modifying resources.
- Working efficiently both independently and as a team.