CSE 546: Cloud Computing (2023 Spring) Project 1 Report - IaaS

Anurag Banerjee (ASU ID: 1225497546)

School of Computing and Augmented Intelligence,
Ira A. Fulton Schools of Engineering
Arizona State University
Tempe, United States
abaner40@asu.edu

Abstract— **This document details the contribution of Anurag Banerjee to Project 1 of the course Cloud Computing offered in Spring 2023. The project was to be done in a group of 3. The main idea of the assignment was to create an application that was elastic and could scale out and in based on the demand in a cost-efficient manner with minimal human intervention. This application was to be built using the resources and tools offered by Amazon Web Services (AWS) since it was the most widely used IaaS provided at the time of the project**

Keywords— **Iaas, Amazon Web Services (AWS), Simple Storage Service (S3), Simple Queue Service (SQS), Spring Boot, Cloud Computing**

I. Contribution

The work done in the project was divided into 3 major parts by our group. These parts were as follows: web-tier, app-tier, and setting up of AWS-related functionality like auto-scaling, queues, buckets, etc. The overview application was that the web-tier would receive an image in a POST request from a user. On receiving this image, the web-tier uploaded this image to a Simple Storage Service (S3) bucket which contained all image uploads with the key as the name of the image file. Then this key was uploaded to a Simple Queue Service (SQS) request queue for processing by the app-tier. The app-tier would keep polling the request queue. Once an element was found in the request queue, the app-tier read this key and downloaded the image corresponding to this key from the image uploads S3 bucket. After downloading the image, the app-tier ran the image classification algorithm that was provided to us. On receiving the result from the classification algorithm, the app-tier uploaded the result to another S3 bucket used to store the classification results and pushed the results to an SQS response queue. The web-tier, in the meantime, polled the response queue. On getting an element, the web-tier logged this result on its console for verification. The app-tier had auto-scaling functionality enabled via Auto Scaling Groups in AWS which enabled the app-tier to scale out or in based on the number of messages in the SQS request queue.

Out of the above-mentioned parts, I focused my attention on the web-tier. The server for the web-tier was developed using Java and Spring Boot. For external libraries, awssdk was used to communicate with S3 and SQS resources on AWS. This service exposed an endpoint for posting images. On receiving a request at this endpoint, the service extracted the name of the image file from the request. Furthermore, the file was converted into its byte form since this is required by the awssdk and this image was added to an S3 bucket with the key as the image file name. Once the image was added to S3 successfully, the key was sent to an SQS request queue with the help of awssdk. After this was completed successfully, the key was sent as a response. Another functionality provided by the web-tier was to print the classification results from the SQS response queue. On startup, the web-tier created a separate thread using the ExecutorService framework in Java. This thread was used to poll the SQS response queue. A separate thread was created to create a universal handler for polling requests and also not to block the requests till the classification was done by the app-tier. This thread polled the SQS response queue using awssdk. On receiving a non-empty message, it logged the contents of the message in a prettified format and deleted the message from the SQS so that the same message was not processed multiple times. This service was deployed on AWS Elastic Compute Cloud (EC2). An Amazon Machine Image (AMI) was created with the required dependencies like Java 17 so that the service could be started without the need to go through the installation of dependencies again.