CSE 546 — Project Report

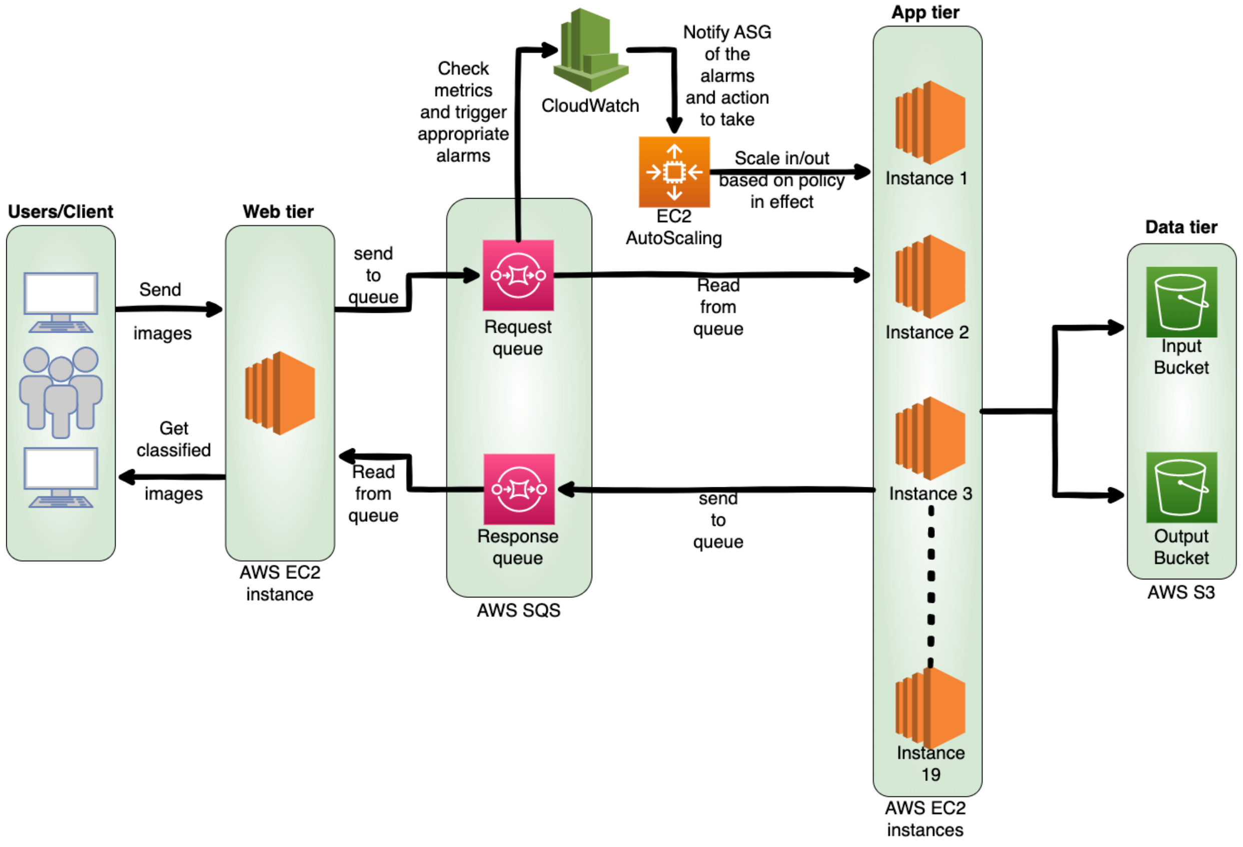
Rahil Vasudev Patel: 1225457025
Krutik Parmar: 1225491150
Rahil Ashish Shah : 1225475355

1. Problem statement

The main goal of this project is to create an IaaS solution utilizing AWS infrastructure for hosting our image classification app. Our success criteria is the efficient scaling in and scaling out of EC2 instances, which will run our image classification code, referred to as the app-tier. The app-tier is already pre-configured with the necessary libraries and code for image classification using deep learning, based on an AMI provided by the course instructor. Our approach will involve integrating S3, SQS, and Cloudwatch to record and log the image classification results, create events that will be consumed by the app-tier, and monitor the status of auto scaling groups (ASG) and alarms. The aim is to optimize the app-tier to fully utilize the resources available and enable it to classify a large number of images efficiently within a short time frame.

2. Design and implementation

2.1 Architecture



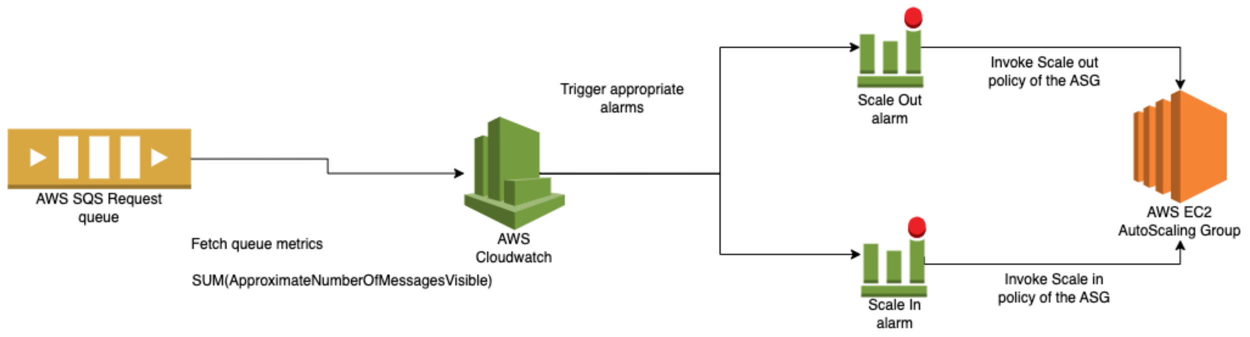
The architecture of our application can be summarized in a high-level diagram. We have a web-tier, which is responsible for hosting the API that sends images for classification. This layer acts as the controller layer. When an image is sent for classification, it is added to the request queue in AWS SQS, and the web-tier waits for the response in the response queue. The app-tier instances fetch the request from the request queue and send it to the S3 input bucket, which acts as the handler layer. The image is classified using the pre-installed code, which functions as the data layer, and the output is sent to the request queue and the output bucket.

If there are many requests to be processed, we scale up (out) the app-tier instances, allowing for parallel processing of multiple requests. The auto scaling group for the instance is configured to limit the number of instances from 1 to 19. Cloudwatch alarms are set up to monitor the thresholds for the instance, and if breached, we can increase or decrease the number of instances accordingly.

2.2 Autoscaling

Autoscaling is a critical component of our project, and we have implemented it using Cloudwatch metrics, alarms, and a dynamic scaling policy linked to the auto scaling group. Our approach is designed to ensure optimal use of AWS resources and minimize costs. Specifically, when we encounter more than two requests, we initiate scaling out of instances. Conversely, when the number of requests drops to less than three, we initiate scaling in instances.

The following diagram illustrates the flow of autoscaling:


AWS SQS Request queue
Fetch queue metrics
SUM(ApproximateNumberOfMessagesVisible)
AWS Cloudwatch
Trigger appropriate alarms
Scale Out alarm
Scale In alarm
Invoke Scale out policy of the ASG
Invoke Scale in policy of the ASG
AWS EC2 AutoScaling Group

The Cloudwatch metrics monitor the app-tier instances, and when the number of requests exceeds a certain threshold, the Cloudwatch alarms are triggered. These alarms, in turn, initiate scaling out of the instances through the dynamic scaling policy linked to the auto scaling group. Similarly, when the number of requests drops below a certain threshold, the alarms trigger scaling in of the instances. The scaling policies determine the specific number of instances to add or remove, ensuring optimal use of AWS resources.

2.3 Member Tasks

2.3.1 Rahil Vasudev Patel(1225457025)

Rahil Patel has contributed in the following ways:

1. Configuring and setting up auto scaling groups, launch templates, and related configurations.
2. Establishing cloudwatch metrics and alarms for monitoring the application.
3. Writing code for the app, web-tier controller and handler layers, and providing insights into the user data script for the launch template to ensure automatic execution of the app-tier code.
4. Implementing the high-level design (HLD) and related code for the application.

2.3.2 Krutik Parmar(1225491150)

Krutik has contributed to the project in the following ways:

1. Set up S3 and ensured its smooth functioning.
2. Developed the controller and handler layers for both the app and web tiers.
3. Wrote the app-tier and web-tier controller code
4. Created the user data script for the launch template to ensure automatic execution of the app-tier code.
5. Conducted integration testing and carried out code refactoring to improve the overall quality of the project.

2.3.2 Rahil Shah(1225475355)

Rahil Shah's contributions to the project are as follows:

1. Setting up SQS, ensuring its smooth functioning.
2. Conducting code cleaning and testing using workload generators.
3. Writing the report.

3. Testing and evaluation

To test our application, we used a Python script with a multithreaded generator to send 100 images concurrently to the web tier. These images were then queued in the request queue and accessed by the app tier for processing. As the number of requests exceeded the threshold, the app tier scaled out automatically, resulting in the creation of 19 instances using CloudWatch and auto scaling group. Conversely, when the request queue was empty, the app tier scaled in automatically, leaving only one running instance.

The outcome of each image classification was sent to the response queue, and the controller retrieved and displayed the results. Additionally, both the images and their corresponding outcomes were stored in S3 buckets for future use. Overall, this testing validated the scalability and functionality of our application, demonstrating its ability to handle multiple requests concurrently and effectively scale up and down as needed.

4. Code

1. **Web-Tier:** The controller layer contains the "web.py" file, which exposes an API for accepting input images, pushing them to the request queue, and waiting for processing by the app tier. It also listens to the response queue for the corresponding output.

   To install and run the web tier, follow these steps:

   a. To install go to the folder and run
      pip3 install -r requirements.txt #this command installs all the required dependencies for to run the web-tier
   b. To start the web server and hit the exposed api run

      python3 web.py

   c. Now using postman or thunder client extension in VS code hit the api.
2. **Workload-generator:** For automated testing of our infrastructure and code, we are using a Python script named "workload-generator.py". This script does not require any additional packages to be installed once the web tier is installed. To run this we type in:
   python3 workload_generator.py --num_request 3 --url 'http://127.0.0.1:8081' --image_folder "imagenet-100/"

   To run a multithreaded workload generator we run:
   python3 multithread_workload_generator.py --num_request 3 --url 'http://127.0.0.1:8081' --image_folder "imagenet-100/"
3. **App-Tier:** This is responsible for processing of the requests. It reads the requests from the request queue, logs the input in an S3 input bucket, then it processes the image through the image classification script given to us. Once processed it clears the message from the request queue, puts it into the response queue and logs it into the S3 output bucket for record keeping. To make sure this script is run automatically once the instance

is launched we have written a userdata script for the launch template which takes care of it.

4. **Userdata-script:** This script is a multi-part script that includes two parts separated by the boundary "//". The first part is a cloud-config script that sets up the instance with some configuration details. The second part is a shell script that will be executed by the instance's user data. This script first appends "Hello World" to the file "/tmp/testfile.txt", lists the files in the directory to "/tmp/test2.txt", clones a GitHub repository using a personal access token, installs the "boto3" Python package, and runs "cloud-based-image-classifier-backend/index.py" script as the "ubuntu" user.

Individual Contributions Report for Rahil Vasudev Patel (1225457025)

The following report details the contributions made by me to the development of a cloud-based image classifier backend application. The application consists of an app tier, web tier, and a controller layer that processes user requests, retrieves image data, and returns image classification results.

Contributions:

I have made significant contributions to the development of the application, including the following:

1. **Configuring and setting up auto scaling groups, launch templates, and related configurations:**

I was responsible for setting up the auto scaling groups, launch templates, and related configurations required for the application to scale dynamically based on user requests. This involved creating launch templates that automatically execute the app-tier code when instances are launched, as well as configuring auto scaling groups to add or remove instances based on workload demands.

2. **Establishing cloudwatch metrics and alarms for monitoring the application:**

I was responsible for configuring and setting up CloudWatch metrics and alarms for monitoring the application's performance. This included monitoring system resources such as SQS latency and queue backlog.

3. **Writing code for the app, web-tier controller and handler layers, and providing insights into the user data script for the launch template:**

I was responsible for writing code for the app, web-tier controller and handler layers. This included creating a multithreaded generator to feed images concurrently to the web tier, as well as processing requests from the request queue, and logging input/output to S3 buckets. In addition, I provided insights into the user data script for the launch template to ensure that the app-tier code is automatically executed once instances are launched.

4. **Implementing the high-level design (HLD) and related code for the application:**

I played a key role in developing the high-level design (HLD) for the application, which provided an overall blueprint for the application's architecture, components, and workflows. This involved designing the app tier, web tier, and controller layer, as well as identifying the tools and technologies required for the application. In addition, I implemented the related code for the application, which involved creating the image classification script, setting up S3 buckets, and creating the response queue.

Conclusion:

In conclusion, I have made significant contributions to the development of the cloud-based image classifier backend application. His contributions include configuring and setting up auto scaling groups, launch templates, and related configurations, establishing CloudWatch metrics and alarms for monitoring the application, writing code for the app, web-tier controller and handler layers, and providing insights into the user data script for the launch template. In addition, I played a key role in developing the high-level design (HLD) and implementing the related code for the application.

Individual Contributions Report for Krutik Parmar (1225491150)

The project involved developing an application that would use image classification to classify images uploaded to a web server. The project was a collaborative effort with several team members contributing to the development of the application. In this report, we will focus on the contributions made by me towards the development of the project.

Contributions: I contributed to the project in the following ways:

1. **Setting up S3:** I was responsible for setting up and configuring S3 for the project. This involved creating the necessary buckets and ensuring their smooth functioning. The input and output of the image classification were logged into the S3 buckets for record-keeping.
2. **Developing and coding the controller and handler layers, i.e., web-tier and app-tier:** Developing and coding the controller and handler layers for the web-tier and app-tier is a crucial aspect of the project. As a team member, I was responsible for this task, which involved writing code that receives input images and manages their processing. The controller layer acted as the entry point for the system, receiving the input images and queuing them for processing. The handler layer, on the other hand, was responsible for processing the input images and generating the output.
3. **Creating user data script:** I developed the user data script for the launch template to ensure automatic execution of the app-tier code. This script would run once the instance was launched and would automatically execute the app-tier code, thus eliminating the need for manual intervention.
4. **Conducting integration testing and code refactoring:** As a project team member, I contributed to the success of the project by conducting integration testing and code refactoring. I ensured that the various components of the application, including the app and web-tier controller and handler layers, and the launch template user data script were properly integrated and worked seamlessly. Additionally, I conducted code refactoring to improve the overall quality of the code, making it more readable, maintainable, and performant. These efforts resulted in a high-quality application that was not only efficient but also easy to maintain and update.

Conclusion:

My contributions were critical to the successful development of the project. His expertise in setting up S3, developing controller and handler layers, and creating the user data script were essential for ensuring the smooth functioning of the application. His contribution to integration testing and code refactoring improved the overall quality of the project. Overall, my contributions were invaluable to the success of the project.