CSE 546 Cloud Computing
Project-1

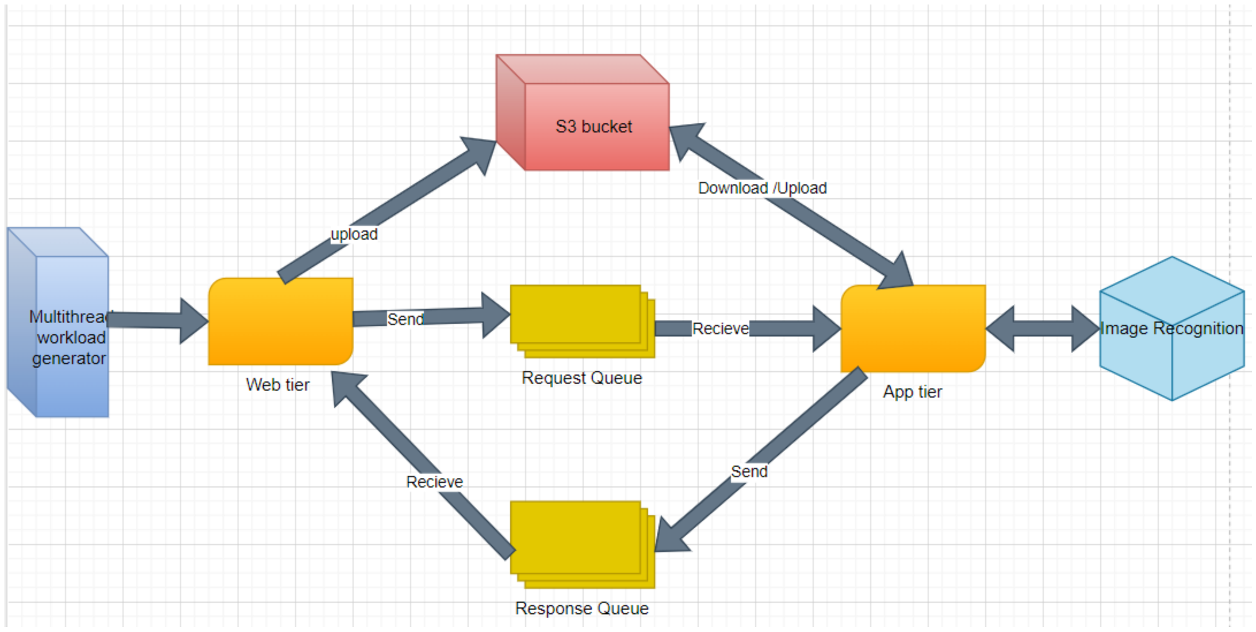
Student Names:
Maharshi Patel
Sahith Doma
Nisarg Patel

1. Problem statement

we will build an image recognition application that can automatically scale out on-demand and cost-effectively by using the IaaS cloud. we will build this application using the IaaS resources from Amazon Web Services (AWS) using EC2, SQS, and S3.

2. Design and implementation

2.1 Architecture



We followed a similar architecture as provided on the assignment,

Our architecture first employs a web-tier EC2 instance that takes input requests from users in the system. These images will be inserted into the Request queue (Amazon

SQS) and passed to the app tier EC2 instances. These instances will be created and terminated with the autoscaling logic of the web tier. All app tier instances will run a deep learning application which will take the image URL as input from SQS Request queue and the predicted output label of that image will be stored in S3 storage for persistency.

We have 1 web tier and 1 app tier initially which will handle all the requests from the user. The web tier will manage to start other app tiers as per the required instances. The app tier will automatically terminate itself when they don’t find input requests from SQS.

The output will be returned to Amazon SQS Response Queue which will push the result to the web tier EC2 instance.

2.2 Autoscaling

The web-tier architecture takes the current request queue size and calculates how many instances it needs to create. We have set the maximum limit to 20. It takes the count of currently running instances, subtracts it from the required instances, and then creates those many instances to handle input requests.

The web tier always checks the number of available messages in the queue and scales out depending on the instance required. When there are no messages we will terminate all the instances when it goes over a certain threshold.

2.3 Member Tasks:

Maharshi:

- Created all the resources in the AWS like creating S3 buckets, SQS Queues, Elastic IPs, and Custom AMI in web-tier and app tiers.
- Developed a code to run the python program that’s downloaded from the input bucket.
- Developed a code to upscale and downscale instances in the app tier as a part of auto-scaling.

Sahith Doma:

- Worked on setting the IAM roles and security groups to allow access over the internet.

- Created an architecture diagram of our project and made a report by testing the application.
- Developed a javascript code to interact with the workload generator, S3, and Queues.

Nisarg Patel:

- Developed the S3 Bucket mechanism that takes input from the web tier and stores it in the input bucket. Later, the same input is used by the application tier to run an image recognition program.
- Worked on integration of S3, SQS, and EC2.
- Helped in testing the application and checking if it is satisfying the given rubrics.

3. Testing and evaluation:

We evaluated our project based on the rubrics given in the assignment description.

Rubrics	**Test Results**
To test if the output from the workload generator is correct.	Yes, the output obtained from the workload generator is correct.
All the input and output files in the S3 buckets are stored correctly.	Yes, all the input files sent and output files generated are stored correctly.
While processing the workload, the number of EC2 instances scaling is correct.	Yes, the app tier scales based on the number of requests generated until it reaches a max cap of 20 instances and scales down back when there are no requests.
All the requests are processed within a reasonable amount of time and test the time taken by 100 concurrent requests.	Yes, all the requests are processed within a few minutes.

4. Code:

Web Tier

webTier.js: This file has functionalities for receiving requests from the workload generator via a POST API request and returning output responses to the workload generator.

App Tier

application.py: This file contains functionalities including getting queue URL, downloading files from an s3 bucket, reading input image files and making predictions with the classifier model, and sending output to the

Classifier: this contains python code to load the classification model and return prediction result on the image with the file path

Auto Controller

AutoScaling.py: this will start web tier instance. It also manages to keep track of the messages in the queue and will start app tier instances as per the traffic.

Resources/main_instance.py: have all functionalities to deal with app tier instances including creating an instance with a given AMI and will pass the script to automatically trigger the application.py file on the instance which is responsible for classifications, terminating the instance with a given id, getting metadata of instances, getting id of all instances with a given state. We are using the boto3 library to perform the different requirements.

Config.ini: This file keeps AWS config parameters

Logger.py: saving info on the log file for debugging purposes

How to install your programs and how to run them.

To start getting to the root location

Start the web tier with the following commands

- node server.js
- python3 autocontroller.py

Start app-tier (This script will be passed automatically with create_instance function)
- pip3 install boto3
- python3 application.py

Start controller (Handles Scaling)
- python3 -m controller.autoscaling.py